INSIGHTS INTO
ACCESS PATTERNS OF INTERNET MEDIA SYSTEMS:
MEASUREMENTS, ANALYSIS, AND SYSTEM DESIGN

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the
Graduate School of The Ohio State University

By

Lei Guo, B.S., M.E.

* * * * *

The Ohio State University

2008

Dissertation Committee:
Xiaodong Zhang, Adviser
Robert B. Brown,
Ten H. Lai, Feng Qin,
Prasun Sinha, Dong Xuan

Approved by

_____________________________________________________________________
Adviser
Graduate Program in
Computer Science and
Engineering
© Copyright by
Lei Guo
2008
ABSTRACT

With the dramatic increase of media traffic on the Internet, existing media systems have shown their inefficiencies in resource utilization and performance bottlenecks on high quality media services. Although the inconsistency between the media access patterns and the Zipf-like distributions of Web workloads has been observed by a number of measurement studies, existing media system designs and evaluations still assume that media workload has the same access pattern as that of conventional Web workload. An insightful understanding of media access patterns is essential to guide Internet system design and management, including resource provisioning and performance optimizations.

In this Ph.D. dissertation, we analyze the access patterns of Internet media systems and study effective system designs for large scale media content delivery. With extensive measurements on the Internet, we find current media systems tend to over-supply or over-utilize server hardware and network bandwidth to provide high quality media service, which is not a scalable and effective approach for serving the explosively increasing media traffic on the Internet. We then systematically study the access patterns of different kinds of Internet media systems, in order to exploit the temporal locality among media requests for efficient and high performance system design. Our study shows that the reference ranks of media objects on the Internet follow stretched exponential distribution, despite different underlying systems and delivery techniques used. With this kind of access patterns, the performance of media caching in a client-server model is far less effective than that of Web
content caching. We further analyze the evolution of object reference rank distributions in long duration media workloads, and find that the temporal locality in media systems increases with time. Thus, long term caching is beneficial to improve the performance of media systems. However, a high volume of storage size is required for long term caching, for which peer-to-peer (P2P) model is attractive. Our stretched exponential model lays out an analytical foundation to establish peer-to-peer caching systems for delivering the huge amount of media content on the Internet.

We further conduct a performance study of BitTorrent-like P2P systems for large scale media delivery. Through modeling and analysis, we find although the existing BitTorrent system is effective for addressing the “flash crowd” problem upon the debut of a new file, it has service unavailability and performance instability problems after a period of time, due to the exponentially decreasing peer arrival rate. We then quantitatively analyze the interaction among multiple BitTorrent systems with a graph-based model, and show that inter-torrent collaboration is much more effective than stimulating seed peers to serve longer for addressing the service and performance problems in BitTorrent systems. Finally, we propose PROP, a P2P-assisted media caching system, which utilizes peer-to-peer collaboration to provide service scalability and dedicated servers to provide service reliability.
To my family.
ACKNOWLEDGMENTS

This dissertation is a result of not only the efforts of my six-year work, but also the guidances, supports, and love from many people to whom I am indebted.

First, I would like to thank my advisor, Prof. Xiaodong Zhang, from my deepest heart. During the past six years, Prof. Zhang is the mentor of my academic study and research in College of William and Mary and the Ohio State University. He is my role model of hard working, dedication, and confronting challenges, inspiring my interest on research, encouraging me not to give up, and warning me on my self-satisfaction. He aims very high and is never satisfied with existing achievements, advising me not only on thesis research, but also on the establishment of career goal. Beyond all of these, he also tries his best to help me on every detail in my research life, from communication, presentation to interview skills.

Second, I would like to thank Song Jiang, who is a professor in Wayne State University now. Song was an instructor of one of my master courses in University of Science and Technology of China. It was his recommendation to give me the opportunity to be Prof. Zhang’s student. In the initial period of my Ph.D. study at College of William and Mary, Song helped me a lot on both academic courses and research, and we have coauthored a number of papers. Song and his wife, Shengli, also helped me much on my life in Williamsburg.
I would like to thank my collaborators in academia: Prof. Songqing Chen (George Mason University) and I have collaborated for more than four years. His comments and suggestions are very constructive, and his English writing is really good. Prof. Qun Li (College of William and Mary) and Prof. Haining Wang (College of William and Mary) are very knowledgeable, and I learned a lot from the discussions with them. Prof. Li Xiao (Michigan State University) is another good collaborator.

I would like to thank my collaborators in industry and research labs: Dr. Xin Chen (AskJeeves Inc.), Dr. Oliver Spatscheck (AT&T Labs-Research), Dr. John Tracey (IBM Research), and Dr. Zhen Xiao (IBM Research); and my colleague collaborators in the Ohio State University and College of William and Mary: Mr. Xiaoning Ding, Mr. Shansi Ren, Mr. Enhua Tan, and Mr. Haodong Wang. Without their supports and collaborations, I would not have the achievements so far.

I would also like to thank Prof. William L. Bynum (College of William and Mary), Dr. Wenwei (Theresa) Long (College of William and Mary), and Mr. Robert Richard Painter (College of William and Mary), who have provided many constructive suggestions on my papers. Theresa also gives me many advices on the study, career, and life.

I would also like to thank Joseph Anselmo, for his unselfish help on improving my English communication skills. During my Ph.D. study in College of William and Mary, Joe tutored my spoken English as a volunteer every week. He is always very patient to listen to my speaking and correct my accent. His encouragement helps me to overcome the communication difficulties.

I would also like to thank Prof. Keith W. Ross (Polytechnic University), the shepherd of my first paper in Internet Measurement Conference, and anonymous reviewers of my submissions, who have provided valuable, critical comments on my papers. These comments
have motivated and inspired me to advance my research. I also need to thank numerous researchers who have generously contributed their collected traces for my Ph.D. research.

Finally, I would like to thank other committee members of my Ph.D. dissertation, Prof. Robert B. Brown, Prof. Ten H. Lai, Prof. Feng Qin, Prof. Prasun Sinha, and Prof. Dong Xuan, for their services and helpful advices on this dissertation.
VITA

1996 .................................................. B.S. in Space Physics, University of Science and Technology of China

2002 .................................................. M.E. in Computer Science, University of Science and Technology of China

2001-2005 ........................................ Ph.D. student in Computer Science, College of William and Mary

2006-present ................................. Ph.D. student in Computer Science and Engineering, The Ohio State University

PUBLICATIONS

Measurements and Analysis of Internet Media Traffic


Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. “Measurements, Analysis, and Modeling of BitTorrent-like Systems”. In Proceedings of
Internet Measurement Conference (ACM/USENIX IMC’05), Berkeley, California, USA, October 19-21, 2005, pp. 35-48.


Internet and Wireless Systems for Multimedia Delivery


Peer-to-Peer Systems and Applications


Book Chapters

FIELDS OF STUDY

Major Field: Computer Science and Engineering
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td></td>
<td>ii</td>
</tr>
<tr>
<td>Dedication</td>
<td></td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>Vita</td>
<td></td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td></td>
<td>xvi</td>
</tr>
<tr>
<td>List of Figures</td>
<td></td>
<td>xvii</td>
</tr>
</tbody>
</table>

Chapters:

1. Introduction ........................................... 1
   1.1 Media Delivery on the Internet .................. 1
   1.2 Media Access Patterns and Performance of Internet Media Systems .... 3
   1.3 Research Contributions .......................... 6
      1.3.1 Measurements of Internet media traffic .......... 6
      1.3.2 Access pattern model of Internet media systems .... 7
      1.3.3 Performance analysis of BitTorrent peer-to-peer systems .... 8
      1.3.4 Peer-to-peer assisted media caching system .......... 9
   1.4 Thesis Organization .............................. 9

2. Quality and Resource Utilization of Internet Media Systems .......... 11
   2.1 Introduction ..................................... 11
   2.2 Trace Collection and Processing Methodology ............ 15
   2.3 Traffic Overview .................................. 17
      2.3.1 Streaming traffic by user communities .......... 18
2.3.2 Streaming traffic by media hosting services ............... 21
2.4 Protocol Rollover and User Startup Latency .............. 23
  2.4.1 Startup latency due to protocol rollover .......... 24
  2.4.2 Protocol selection and rollover avoidance ....... 26
2.5 Fast Streaming ........................................... 28
  2.5.1 Fast Cache smoothes bandwidth fluctuation ....... 31
  2.5.2 Fast Cache produces extra traffic ............ 34
  2.5.3 Server response time of Fast Cache ............. 35
  2.5.4 Server load of Fast Cache ..................... 37
  2.5.5 Effectiveness of resource over-utilization .... 38
2.6 Rate Adaptation .......................................... 40
  2.6.1 MBR encoding and stream switch ............... 40
  2.6.2 Stream thinning and video cancellation ....... 43
  2.6.3 Summary of Internet streaming quality ...... 44
2.7 Coordinating Caching and Rate Adaptation ............ 47
2.8 Related Work ............................................ 50
2.9 Conclusion ................................................. 51

3. The Access Pattern Model of Internet Media Systems ........ 52
  3.1 Introduction ............................................. 52
  3.2 Workload Description ................................ 56
  3.3 Related Work ........................................... 61
  3.4 The Reference Rank Model of Internet Media Objects .... 64
    3.4.1 The stretched exponential of Internet media traffic .... 64
    3.4.2 Factors that may affect media access patterns .... 68
    3.4.3 Other media access pattern models ............. 75
  3.5 The Dynamics of Media Reference Rank Distributions .... 78
    3.5.1 Access patterns of different sized media files .... 78
    3.5.2 Non-stationarity of media reference rank distributions .... 80
    3.5.3 Deviation of media access patterns from the Zipf model ........ 85
    3.5.4 Summary ........................................ 88
  3.6 Implications of Stretched Exponential Access Patterns .... 91
    3.6.1 Caching implications of different reference rank models .... 92
    3.6.2 Performance analysis of segment-based streaming media caching .... 96
    3.6.3 The evolution of media caching performance .... 103
  3.7 Conclusion ............................................. 106

4. Performance Analysis of BitTorrent-like Peer-to-Peer Media Systems .... 108
  4.1 Introduction ........................................... 108
  4.2 Related Work ........................................... 111
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 Modeling and Characterization of Single-Torrent Systems</td>
<td>113</td>
</tr>
<tr>
<td>4.3.1 Characterizing file popularity evolution</td>
<td>114</td>
</tr>
<tr>
<td>4.3.2 Torrent evolution and service availability</td>
<td>118</td>
</tr>
<tr>
<td>4.3.3 Client performance variations</td>
<td>123</td>
</tr>
<tr>
<td>4.3.4 Service fairness</td>
<td>127</td>
</tr>
<tr>
<td>4.4 Modeling Multiple Torrents in BitTorrent Systems</td>
<td>128</td>
</tr>
<tr>
<td>4.4.1 Characterizing peer request pattern</td>
<td>129</td>
</tr>
<tr>
<td>4.4.2 Characterizing inter-torrent relations</td>
<td>135</td>
</tr>
<tr>
<td>4.4.3 Reducing downloading failure ratio by inter-torrent collaboration</td>
<td>138</td>
</tr>
<tr>
<td>4.5 Multi-Torrent Collaboration of BitTorrent Systems</td>
<td>140</td>
</tr>
<tr>
<td>4.5.1 Tracker site overlay</td>
<td>140</td>
</tr>
<tr>
<td>4.5.2 Multi-torrent collaboration</td>
<td>141</td>
</tr>
<tr>
<td>4.5.3 Performance evaluation</td>
<td>143</td>
</tr>
<tr>
<td>4.6 Conclusion</td>
<td>145</td>
</tr>
<tr>
<td>5. Peer-to-Peer Assisted Media Caching</td>
<td>146</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>146</td>
</tr>
<tr>
<td>5.2 Related Work</td>
<td>149</td>
</tr>
<tr>
<td>5.3 System Design and Rationale</td>
<td>151</td>
</tr>
<tr>
<td>5.3.1 Infrastructure overview</td>
<td>151</td>
</tr>
<tr>
<td>5.3.2 P2P routing and media streaming</td>
<td>152</td>
</tr>
<tr>
<td>5.3.3 Redundant cache model</td>
<td>156</td>
</tr>
<tr>
<td>5.3.4 Cache management</td>
<td>159</td>
</tr>
<tr>
<td>5.3.5 Streaming task dispatch</td>
<td>164</td>
</tr>
<tr>
<td>5.3.6 Fault tolerance</td>
<td>164</td>
</tr>
<tr>
<td>5.4 Performance Evaluation</td>
<td>165</td>
</tr>
<tr>
<td>5.4.1 Workload summary</td>
<td>166</td>
</tr>
<tr>
<td>5.4.2 Performance results</td>
<td>168</td>
</tr>
<tr>
<td>5.4.3 Implications of our performance evaluation</td>
<td>181</td>
</tr>
<tr>
<td>5.5 Conclusion</td>
<td>182</td>
</tr>
<tr>
<td>6. Concluding Remarks</td>
<td>184</td>
</tr>
</tbody>
</table>

Appendices:

A. Reference Rank Distributions of Other Media Workloads                | 186  |
B. Chi-Square Goodness of Fit Tests for SE and Zipf-like Models         | 189  |
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Home user streaming media workload overview</td>
<td>18</td>
</tr>
<tr>
<td>2.2 Business user streaming media workload overview</td>
<td>18</td>
</tr>
<tr>
<td>2.3 Streaming media traffic by different hosting services</td>
<td>22</td>
</tr>
<tr>
<td>2.4 Streaming traffic with Fast Cache</td>
<td>29</td>
</tr>
<tr>
<td>2.5 Summary of streaming quality</td>
<td>47</td>
</tr>
<tr>
<td>3.1 Summary of media workloads</td>
<td>57</td>
</tr>
<tr>
<td>4.1 Notations and assumptions for the fluid model</td>
<td>123</td>
</tr>
<tr>
<td>5.1 Workload summary of PROP evaluation</td>
<td>166</td>
</tr>
<tr>
<td>B.1 $\chi^2$ hypothesis test results ($\alpha = 0.05$)</td>
<td>190</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>On-demand and live audio distributions in the home and business user workloads</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>On-demand and live video distributions in the home and business user workloads</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>Service providers ranked by traffic volume</td>
<td>22</td>
</tr>
<tr>
<td>2.4</td>
<td>Protocol rollover increases startup latency of streaming sessions</td>
<td>25</td>
</tr>
<tr>
<td>2.5</td>
<td>Features of media delivered by Fast Cache and normal TCP streaming</td>
<td>30</td>
</tr>
<tr>
<td>2.6</td>
<td>The client advertised bandwidth, client requesting rate, and server streaming rate</td>
<td>32</td>
</tr>
<tr>
<td>2.7</td>
<td>Bandwidth fluctuation smoothed by Fast Cache</td>
<td>33</td>
</tr>
<tr>
<td>2.8</td>
<td>Extra traffic produced by Fast Cache</td>
<td>34</td>
</tr>
<tr>
<td>2.9</td>
<td>Average data transmission speed</td>
<td>34</td>
</tr>
<tr>
<td>2.10</td>
<td>The server response time to streaming requests</td>
<td>35</td>
</tr>
<tr>
<td>2.11</td>
<td>Server load comparison between Fast Cache and normal TCP streaming</td>
<td>38</td>
</tr>
<tr>
<td>2.12</td>
<td>Effectiveness of resource over-utilization</td>
<td>39</td>
</tr>
<tr>
<td>2.13</td>
<td>MBR encoding in the home and business user workloads</td>
<td>41</td>
</tr>
<tr>
<td>2.14</td>
<td>Stream switch</td>
<td>43</td>
</tr>
</tbody>
</table>
2.15 Stream thinning ................................................................. 44

2.16 Rebuffering duration for Fast Cache and normal TCP streaming sessions with rebuffering (home user workload only) ........................................ 45

2.17 Comparison of playback durations for sessions with and without quality degradations ................................................................. 46

2.18 Evaluation of Coordinated Streaming ........................................ 49

3.1 Reference rank distributions of media objects in Web systems ............ 65

3.2 Reference rank distributions of media objects in VoD systems .............. 66

3.3 Reference rank distributions of media objects in P2P systems ............... 68

3.4 Zipf-like fit for workloads with extraneous traffic ............................ 69

3.5 References of three kinds of objects in ST-CLT-04 ............................ 71

3.6 References of three kinds of objects in ST-CLT-05 ............................ 71

3.7 Percentage of three kinds of requests in ST-CLT-04 and ST-CLT-05 .......... 71

3.8 Number of requests to each object by the same user ......................... 74

3.9 Number of distinct weekly top $N$ popular objects in Web and media systems 76

3.10 Stretch factor vs. median file size ........................................... 78

3.11 Media birth and access over time (ST-SVR-01) ............................... 81

3.12 Evolution of reference rank distribution (BT-03) .............................. 81

3.13 Evolutions of reference rank distributions (parameter $a$ in the SE model) . 84

3.14 The difference between SE and Zipf-like plots ............................... 86

3.15 Two different observations of YouTube video access pattern ............... 89

xviii
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.16</td>
<td>Caching efficiency of media (SE) and Web (Zipf) workloads</td>
<td>94</td>
</tr>
<tr>
<td>3.17</td>
<td>The reference rank distributions of media objects and segments. Each object has the same length and is divided into 100 segments. There is no partial access to any object.</td>
<td>97</td>
</tr>
<tr>
<td>3.18</td>
<td>The two-mode stretched exponential distribution of segment reference ranks</td>
<td>98</td>
</tr>
<tr>
<td>3.19</td>
<td>The stretched exponential implications on streaming media caching performance</td>
<td>100</td>
</tr>
<tr>
<td>3.20</td>
<td>The accesses to old objects and new objects in client side workloads</td>
<td>104</td>
</tr>
<tr>
<td>3.21</td>
<td>Object age distributions of client side workloads</td>
<td>105</td>
</tr>
<tr>
<td>4.1</td>
<td>The complementary CDF distribution of peer arrival time (time of a peer’s first request to a torrent or time when a meta file was downloaded) after torrent birth for three BitTorrent traces (y-axis is in log scale)</td>
<td>116</td>
</tr>
<tr>
<td>4.2</td>
<td>Fitting deviations of fully-traced torrents in the tracker trace</td>
<td>117</td>
</tr>
<tr>
<td>4.3</td>
<td>The comparison of torrent lifespan: modeling and trace analysis (y-axis is in log scale)</td>
<td>119</td>
</tr>
<tr>
<td>4.4</td>
<td>Torrent population and downloading failure ratio for all fully-traced torrents</td>
<td>120</td>
</tr>
<tr>
<td>4.5</td>
<td>The peers abort downloading voluntarily</td>
<td>122</td>
</tr>
<tr>
<td>4.6</td>
<td>Torrent evolution under the fluid model</td>
<td>124</td>
</tr>
<tr>
<td>4.7</td>
<td>Performance variations in BitTorrent systems</td>
<td>125</td>
</tr>
<tr>
<td>4.8</td>
<td>Fairness of seed service policy in BitTorrent systems (y-axis is in log scale)</td>
<td>127</td>
</tr>
<tr>
<td>4.9</td>
<td>The CDFs of torrent birth, peer request arrival, and peer birth over the trace collection time</td>
<td>129</td>
</tr>
<tr>
<td>4.10</td>
<td>The request pattern of peers</td>
<td>131</td>
</tr>
<tr>
<td>4.11</td>
<td>Torrent involvement of peers (x-axis is in log scale)</td>
<td>133</td>
</tr>
</tbody>
</table>
4.12 The seeding time and sleeping time of peers (y-axis is in log scale)  
4.13 The inter-torrent relation (y-axis is in log scale)  
4.14 Tracker site overlay  
4.15 The performance analysis of our system  
5.1 The requesting and serving of media data  
5.2 Segment distribution  
5.3 Utility variations of media segments  
5.4 The distribution of the reference frequency of media objects  
5.5 Performance evaluation on the REAL workload  
5.6 Performance evaluation on the PART workload  
5.7 Performance evaluation on the WEB workload  
5.8 Proxy load changes  
5.9 The comparison between popularity-based replacement and LRU replacement  
5.10 Replacement policy comparisons on the REAL workload (without proxy caching in the system)  
5.11 Replacement policy comparisons on the PART workload (with proxy caching 5% of the total accessed media data in the system)  
5.12 Replacement policy comparisons on the WEB workload (with proxy caching 5% of the total accessed media data in the system)  
5.13 The comparisons of different adjustment factors for utility function  
5.14 The Adaptiveness of Utility-based Replacement on the REAL workload  
5.15 The Adaptiveness of Utility-based Replacement on the PART workload
5.16 The Adaptiveness of Utility-based Replacement on the WEB workload . . . 181

A.1 More reference rank distributions of media workloads . . . . . . . . . . . . 187

C.1 Zipf-Mandelbrot fitting for workloads with large media files . . . . . . . 192

C.2 Parabolic fractal fitting for workload ST-SVR-01 . . . . . . . . . . . . . . 194
CHAPTER 1

INTRODUCTION

1.1 Media Delivery on the Internet

The dominant traffic on the Internet has changed from text and graphics based Web content to more information-rich media content, such as audio and video. With the dramatic increase of network bandwidth and the advancement of technologies on media authoring, encoding, and distribution, media traffic on the Internet has increased explosively and now accounts for the majority of traffic volume. According to IBLNEWS [66] and comScore [43], video-related traffic is doubling every 3-4 months, while the traffic of popular video sites grows even faster. For example, YouTube nearly doubled its traffic in May 2006, and more than 100 million video streams were served per day in July 2006. According to the measurement report of Alexa Internet [68], the daily reach of YouTube keeps increasing every week, and had ranked the third in November 2007, after Yahoo! and Google.

Different from conventional Web content, media content can be delivered through a variety of approaches. Similar to downloading other files with a browser, Web-based downloading fetches the entire object completely with HTTP protocol [20] before playback. In order to save network bandwidths and CPU cycles, streaming technique has been proposed
and used to transfer media content with the progress of user access, through streaming protocols such as RTSP [102] and MMS [13]. A mixture of downloading and streaming is *pseudo streaming*, which progressively downloads and plays the media content in the media player, with the conventional HTTP protocol [56]. In addition, overlay multicast, such as ppLive [92] and ppStream [93], file exchange based peer-to-peer (P2P) networks, such as Gnutella [1] and KaZaa [2], and swarming based P2P networks, such as BitTorrent [42], are also widely used for large scale media delivery.

Despite these kinds of delivery techniques, existing media services are still far from being satisfactory. Modern media systems are often hosted by expensive, dedicated infrastructures, such as content delivery networks (CDNs). In order to provide quality service, these systems tend to over-supply or over-utilize scarce server resources, causing high CPU and bandwidth consumption [61]. The widely used cost effective P2P-based systems tend to abuse bandwidth resources, while the services are still unstable and even unavailable [54].

The Internet is an open, complex system for a wide variety of applications. Delivering huge amount of media content with high quality to the global scope of end users on the Internet is challenging. First, the delivery of media content has stringent requirements on network bandwidth, delay, and loss rate. However, the Internet has no Quality of Service (QoS) guarantee, and has no management and control for real-time media flows. Second, media traffic is user-behavior driven, and user requests often arrive in bursts. In the peak time, a media server often needs to serve thousands of concurrent streams from diverse clients, including wired and wireless devices with different network connections. Furthermore, users always have increasingly high expectations and demands on the quality of media content, while the performances of media encoding, data compression, and
data communication techniques increase slowly. Due to the highly dynamic network conditions and user behaviors, measurements and modeling are critical for evaluating system performance under the Internet environment, understanding user access patterns in media systems, and providing guidance to media system design and management.

1.2 Media Access Patterns and Performance of Internet Media Systems

Caching is a basic technique to improve the performance of Internet systems with cost effective hardware, by exploiting temporal locality among memory accesses and user requests. The benefit of caching is highly dependent on the access pattern of objects in the system. It is commonly agreed that Web requests follow a Zipf-like distribution, also known as power law, meaning if ranking objects in the system by the descending order of the number of references, then the number of references to an object is proportional to the power of its reference rank [30]:

$$y_i \propto i^{-\alpha},$$

where \(i\) is the rank of the object, and \(y_i\) is the number of references to this object. \(\alpha\) is a constant, called skewness factor. The reference rank of an object reflects the popularity of this object. Zipf distribution characterizes the property of scale invariance for many physical and social phenomena, and has been believed the general model of Internet traffic patterns.

Intuitively, Zipf distribution can be expressed as the so called “80-20” rule, which means 80% of effects comes from 20% of causes. For example, sociology studies have found that 80% of social wealth is owned by 20% people (Pareto law [105]). For Web systems, it has been widely known that 80% Web requests access 20% pages [30]. This means
Web workload has high temporal locality, which is an analytical foundation for improving Web access performance by client-server based proxy caching systems. Today, Web proxies have been widely deployed across the Internet.

However, the Zipf-like access pattern of Web objects is mainly based on the measurement and analysis studies when text-based Web content dominated the Internet. With the dramatic increase of media content on the Internet, it is important to distinguish media traffic from the general Web traffic, since media objects are much larger than conventional Web objects such as html and graphic files, and have accounted for the majority of Internet traffic. Unfortunately, due to the variety of media delivery systems and the diversity of media content, existing studies on media traffic are largely workload specific, and the observed access patterns are often different from or even conflict with each other. For Web media systems, Chesire et al. [38] report that the access pattern of streaming media is Zipf-like in a university campus network, while Cherkasova et al. [37] find that it is not Zipf-like in an enterprise media server. For VoD media systems, Acharya et al. [22] find that it is not Zipf-like in a multicast-based Media-on-Demand server of a campus network, while Yu et al. [123] report it is Zipf-like in a large VoD streaming system of an ISP in China. For P2P media systems, Gummadi et al. [52] report that the access pattern of media workload in KaZaa system collected in a campus network is not Zipf-like, while Iamnitchi et al. [65] report that it is Zipf-like in an ISP network. For live streaming media systems, Veloso et al. [112] report that user interests are object driven and follow a Zipf-like profile for the workload collected in a content service provider of Brazil, while Sripanidkulchai et al. [107] report that the popularity distribution of live media programs hosted by Akamai content delivery network (CDN) is not Zipf-like.
A number of distributions and models have been proposed or used to explain the observed media access patterns, such as Zipf distribution with exponential cutoff effect [31], Zipf-Mandelbrot distribution [98], generalized Zipf-like distribution model [108], “fetch-at-most-once” model [52], two-mode Zipf model [107], and log quadratic distribution model [39], etc. Most of these models are still based on the Zipf distribution, with some heuristic assumptions. However, each of them can only explain a very limited scope of measurement results; and furthermore, the mechanisms of these models are different from or even conflict with each other. A general model of Internet media access patterns is highly desirable for traffic engineering on the Internet and is critical to design, benchmark, and evaluate Internet media delivery systems.

In practice, although a number of caching algorithms and solutions have been proposed, including commercial products, such as Helix Universal Proxy [9] of RealNetworks and Windows Media Proxy [16] of Microsoft, few of them are used in reality. Instead, due to the high quality requirements and resource demands of media delivery, current media systems have to rely on expensive CDN services, or over-supply system resources and over-utilize network bandwidth to ensure the quality of services [61]. An insightful understanding of media access patterns is essential to guide content delivery system design, Internet resource provisioning, and performance optimization.

Establishing the access pattern model of media objects is challenging. Existing studies cannot identify a general media access pattern, due to the limited number of workloads and a constrained scope of media traffic in the study. Furthermore, some biased measurements and noises in the data sets may also affect the observation of media request patterns. A general access pattern model should be accurate, simple, and meaningful. It should characterize the unique properties of media accesses, have clear physical meanings, provide
observable and verifiable predictions, and have impacts on system designs. In order to
evaluate the model, we need not only to verify the model with the goodness-of-fit test,
but also need to reexamine and explain the different observations previously reported, and
reappraise other proposed models.

1.3 Research Contributions

In this dissertation, we have the following objectives: First, to discover a general dis-
tribution model of media object access patterns for different media systems, with com-
prehensive measurements and experiments, rigorous mathematical analysis and modeling,
and insights to media system designs. Second, to analyze the performance of BitTorrent
systems for media delivery, identify the weakness of BitTorrent, model the potential of col-
laboration among different torrents, and propose system facility and incentive mechanism
for multi-torrent collaboration. Finally, we will also design and implement streaming media
systems, for reliable and scalable Internet streaming with P2P techniques, power efficient
wireless media systems, and high performance Internet streaming through WLANs.

1.3.1 Measurements of Internet media traffic

Modern Internet streaming services have utilized various techniques to improve the
quality of streaming media delivery. Despite the characterization of media access patterns
and user behaviors in many measurement studies, few studies have focused on the stream-
ing techniques themselves, particularly on the quality of streaming experiences that they of-
fer end users and on the resources of the media systems that they consume. In order to gain
insights into current streaming services and thus provide guidance on designing resource-
efficient and high quality streaming media systems, we analyze the most commonly used
streaming techniques such as automatic protocol switch, Fast Streaming, multiple-bit-rate
encoding and rate adaptation with streaming media traffic collected in a large ISP. Our measurement and analysis results show that with these techniques, current streaming systems tend to over-utilize CPU and bandwidth resources to provide better services to end users, which is not an effective way to improve the quality of streaming media delivery. We also propose and evaluate a coordination mechanism that effectively takes advantage of both Fast Streaming and rate adaptation to better utilize the server and Internet resources for streaming quality improvement.

1.3.2 Access pattern model of Internet media systems

With the dramatic increase of media traffic on the Internet, the inconsistency between the media access patterns and Zipf-like distributions has been observed in a number of studies. However, due to their case study approach and thus the constrained scope of analyzed media traffic, existing studies have provided various and even conflicting explanations for such inconsistency. A general model of Internet media access patterns is highly desirable.

We have studied a wide variety of media workloads collected from both client and server sides in different media systems with different delivery methods. Through extensive measurements and analysis, we find: (1) The media access pattern of all these workloads follows the stretched exponential (SE) distribution despite their corresponding media systems and delivery methods. (2) Some biased measurements may lead to Zipf-like observations on media access patterns, which may explain previously reported Zipf-like observations. (3) The stretched exponential distribution model of media access patterns has clear physical meanings. The parameters are mainly controlled by its median file size, client request rate, and object birth rate in the system. (4) In general Zipf-like distribution cannot well fit the access patterns of any media workloads. The deviation of media access patterns
from the Zipf model increases with time, due to the popularity decaying of media objects. (5) Previously proposed models are not general to characterize object access patterns in various media systems.

We further analyze the effectiveness of media caching with a mathematical model. Compared with Web caching under the Zipf-like model, media caching under the SE model is far less effective unless the cache size is enormously large. This indicates that many previous studies based on a Zipf-like assumption may have overestimated the benefit of media caching. Analyzing the evolution of object reference rank distributions in long duration media workloads, we find that the temporal locality in media systems increases with time. Thus, long term caching is beneficial to improve the performance of media systems. However, the conventional client-server model of media proxies cannot work well for long term caching, due to the high volume of storage size required. Our study indicates that an Internet infrastructure of distributed or global resource sharing, such as a P2P system, is promising to provide the enormously large and scalable cache space for efficient media content delivery. Thus, the stretched exponential access pattern model lays out an analytical foundation to establish peer-to-peer media caching systems.

1.3.3 Performance analysis of BitTorrent peer-to-peer systems

In order to investigate the potential of P2P collaboration on high quality media content delivery, we conduct a performance study of BitTorrent-like P2P systems by modeling. Existing studies on BitTorrent systems are single-torrent based and assume the process of request arrivals to a torrent is Poisson-like. However, in reality, most BitTorrent peers participate in multiple torrents and the file popularity changes over time. Our measurement and analysis show that although the existing BitTorrent system is effective for addressing
the “flash crowd” problem upon the debut of a new file, due to the exponentially decreasing peer arrival rate, it has service unavailability and performance instability problems after a period of time. Based on the exponentially decreasing peer arrival model, we have further proposed a graph based model for the interactions among multiple torrents, which quantitatively demonstrates that inter-torrent collaboration is much more effective than stimulating seeds to serve longer for addressing the service unavailability in BitTorrent systems. An architecture for inter-torrent collaboration under an exchange based instant incentive mechanism is also discussed and evaluated by simulations.

1.3.4 Peer-to-peer assisted media caching system

To efficiently deliver streaming media on the Internet, we present a novel and efficient design of a scalable and reliable media proxy system assisted by P2P networks, called PROP. PROP utilizes the resources of end users to provide scalability and dedicated proxy servers to ensure reliability. In the PROP system, the clients’ machines in an intranet are self-organized into a structured P2P system to provide a large media storage and to actively participate in the streaming media delivery, where the proxy is also embedded as an important member to ensure the quality of streaming service. We have comparatively evaluated our system through trace-driven simulations with synthetic workloads and with a real-life workload extracted from the media server logs in an enterprise network, which shows our design significantly improves the quality of media streaming and the system scalability.

1.4 Thesis Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents a set of large scale measurements of Internet media traffic and investigate the state of the art of
Internet media delivery from the perspectives of content, quality and resource utilization. Chapter 3 presents the stretched exponential model of Internet media reference rank distribution and the implications of stretched exponential access patterns on media caching. Chapter 4 studies the performance of BitTorrent-like P2P systems. Chapter 5 presents a peer-to-peer based media caching system. Chapter 6 makes concluding remarks, discusses the impacts of media access patterns on the Internet system design and management, and explores possible future research directions.
CHAPTER 2

QUALITY AND RESOURCE UTILIZATION OF INTERNET MEDIA SYSTEMS

2.1 Introduction

The past decade saw the evolution of Internet content from mostly text and images to increasingly more multimedia objects such as audio and video [38, 52]. Three methods are commonly used to deliver multimedia content on the Internet, namely, downloading, pseudo streaming, and streaming. Initially, multimedia objects were distributed in the same way as non-media objects: a client downloads an audio or a video clip from the Web server. This approach is easy to implement and requires no change to the Web server. The drawback is that the client has to finish downloading the entire object before it can start playing the media. This can incur a long startup latency for large media objects or for clients who have limited bandwidth to the Internet (e.g., dial-up clients). Moreover, if a client decides that the content of a large media object is not interesting after playing for a few seconds, most of the traffic in downloading this object is simply wasted.

Pseudo streaming is another delivery method for multimedia objects. It has the same nature of downloading, but provides an option on the client side to play the object while it is being downloaded. Most media players, such as Windows Media Player and Real Player,
support the pseudo streaming mechanism. A major limit of pseudo streaming appears when the downloading connection is slow and cannot catch up with the playback speed. In this case, a client has to stop frequently to wait for new data.

To address the deficiency of downloading and pseudo streaming, researchers have developed streaming as the most efficient technique for delivering multimedia content. With streaming, the playback of a media object can start shortly after the client receives the initial portion of the object from the streaming server. In addition, streaming provides clients with a variety of controls during playback, such as pause, rewind, jump, etc. This allows a client to start or stop the media stream easily at any time. Compared with downloading and pseudo streaming, the amount of data transferred in streaming is closest to what the client really needs. Streaming service is superior in handling thousands of concurrent streams simultaneously, flexible responses to network congestion, efficient bandwidth utilization, and high quality performance [57]. Because of its many advantages, streaming is used in various Internet applications today.

Today, more than 90% of streaming media traffic on the Internet is delivered either through Windows media services or RealNetworks media services [57]. Different from downloading or pseudo streaming small sized video clips from a Web site such as YouTube [122], streaming long duration and high quality media objects on the Internet has several unique challenges. These commercial streaming services have adopted various techniques to address the above challenges and to satisfy the ever-increasing quality demands of users, such as TCP and HTTP based streaming, Fast Streaming, multiple bit rate (MBR) encoding and rate adaptation. Due to the wide deployment of Network Address Translation (NAT) routers and firewalls that often prevent UDP packet transversal,
TCP-based streaming has been widely used and now accounts for the majority of Internet streaming traffic [57, 107]. Fast Streaming [7] is a group of techniques supported by the Windows media service, which aggressively utilizes the Internet bandwidth by delivering a media object at a rate much higher than its encoding rate, in order to minimize the user perceived startup latency and guard against potential network bandwidth fluctuations. MBR encoding is a technique that encodes a media object with multiple bit rates so that the streaming server can deliver the same content with different quality to clients with different network connections. MBR encoding also enables dynamic stream switch among streams of different rates encoded in the object during a user session, in order to adapt to the current bandwidth, which is called Intelligent Streaming [12] in the Windows media service and SureStream [15] in the RealNetworks media service.

In spite of the wide deployment of these techniques, existing measurement studies of Internet streaming media mainly focus on the characterization of media access patterns and user behaviors, such as [24, 37, 38, 44, 123], which is helpful to the design of media delivery systems such as server clusters and media proxies [36, 119]. However, the mechanisms of the commonly and practically used streaming techniques themselves and their effects on improving Internet streaming quality have not yet been thoroughly studied, which is necessary to understand state-of-the-art of Internet streaming media and to provide guidance on future Internet streaming services. Despite several experimental studies in lab environments on the Windows and RealNetworks media systems [40, 87], to the best of our knowledge, to date, there is no comprehensive study on the delivery quality and resource utilization of these streaming techniques in the Internet environment. It is highly desirable for both streaming service providers and system designers to be guided with an insightful understanding of existing Internet streaming techniques.
In order to investigate Internet streaming quality and the efficiency of resource utilization with the deployment of these techniques, in this work, we have collected a 12-day streaming media workload from a large ISP in the United States. The workload covers thousands of broadband home users and hundreds of business users who access both on-demand and live streaming media. Through extensive analysis of the majority of TCP-based streaming traffic on the Internet, we have the following observations:

- We find that the overhead of protocol rollover plays an important role in user perceived startup latency, and thus may have affected the way that media is served by content providers. When UDP is not supported, the overhead of protocol rollover from UDP to TCP contributes a non-trivial delay to the client startup latency. More than 22% of protocol rollover is longer than 5 seconds.

- By aggressively utilizing the Internet network bandwidth, Fast Streaming shows both positive and negative features. Although Fast Streaming can help smooth re-buffering jitter, it over-supplies media data to end users by about 55%, and consumes more CPU resources, which leads to a longer server response time.

- MBR-encoding is widely used in media authoring, and nearly half of streaming video and audio objects on the Internet are MBR-encoded. However, the rate adaptation functionality of MBR is poorly utilized, particularly when Fast Streaming is used.

- Overall, on the Internet, about 13% of home and 40% of business streaming sessions suffer various quality degradations, such as rebuffering, thinning, or switching to a lower quality stream.

Our measurement and analysis results show that with these techniques, current streaming services tend to over-utilize CPU and bandwidth resources to provide better service
to end users, which may not be a desirable and effective way to improve the quality of streaming media delivery. Furthermore, the Fast Streaming technique does not work with rate adaptation, resulting in even worse user experiences than normal TCP-based streaming upon long-term network congestion. Motivated by these results, we propose *Coordinated Streaming*, a mechanism that effectively coordinates caching and rate adaptation in order to improve streaming quality with an efficient utilization of the server and Internet resources. The potential of such a mechanism in streaming quality improvement is evaluated accordingly.

The remainder of this chapter is organized as follows. Section 2.2 describes our trace collection and processing methodology. Section 2.3 presents an overview of our collected workload. The measurement and analysis of the delivering quality and resource utilization of streaming media services are performed in Sections 2.4, 2.5, and 2.6. The coordinating caching and rate adaptation mechanism is discussed in Section 2.7. Some related work is outlined in Section 2.8. Finally, we summarize our work in section 2.9.

2.2 Trace Collection and Processing Methodology

The prevailing streaming protocols on the Internet are RTSP [102] and MMS [13]. In RTSP streaming, the client and the server exchange streaming commands via RTSP, running on TCP. The media data packets and streaming control/feedback packets are delivered via RTP/RTCP [101] (such as Windows and QuickTime media services) or RDT [14] (RealNetworks media services), running on UDP or TCP. In MMS streaming, all streaming commands and control packets between a client and a server are exchanged via MMS in the same TCP connection, and the media data can be delivered over UDP or TCP. For both RTSP and MMS streaming, when TCP is used to deliver media data, the media and control
packets are interleaved with RTSP or MMS commands in a single TCP connection, instead of using two separate TCP connections. In addition to RTSP and MMS, media can also be streamed through HTTP [10]. Different from HTTP downloading (also known as pseudo streaming [57]), HTTP streaming uses the HTTP protocol to deliver both RTSP commands and media data. In Microsoft HTTP streaming, the RTSP headers are embedded in thePragma headers of HTTP messages. In RealNetworks and QuickTime HTTP streaming, the RTSP commands are embedded in HTTP message bodies with the base64 encoding format.

In this study, we collected streaming media packets in a data center of a major ISP from 2005-04-29 15:00 (Friday) to 2005-05-10 20:30 (Tuesday), using the Gigascope appliance [46]. The data center hosts servers for thousands of business companies, and provides Internet access services for a large cable company. The Gigascope is running on a site close to the end users (broadband home users and business users). To collect streaming packets of RTSP/MMS requests, Gigascope captures all TCP packets from/to ports 554-555, 7070-7071, 9070, and 1755. According to a recent measurement study that collects RTSP/MMS packets based on keyword matching [57], our port number selection covers 97.3% of the RTSP/MMS streaming requests. Meanwhile, we also collected UDP streaming traffic via ports 5004-5005, 6970-6980, and 7000-7010, which are the most popular ports for UDP streaming. For UDP streaming traffic over other port numbers due to network address translation (NAT), we calculate the traffic volume based on the summary information that a client reports to its server when a streaming session is terminated. Compared to existing studies that are based on server logs [37, 44, 107, 113, 123], in which only the summary information of streaming sessions is available, our study is conducted at the packet level.

There are about 2.6% RTSP/MMS/HTTP streaming requests using port 80 or 8080, which are hard to distinguish from regular HTTP downloading traffic. We exclude this traffic in this study.
which facilitates more detailed analysis on the quality and the resource utilization of vari-
ous Internet streaming techniques.

The initial trace processing is as follows. We first grouped TCP packets by TCP con-
nections, based on the IP address, port number, and TCP SYN/FIN/RST flag. Then we
extracted the RTSP/MMS commands from each streaming request. Based on the analysis
of these commands, we identified and parsed media data and streaming control packets
from the TCP or corresponding UDP streams, and dumped the corresponding RTP/RTCP,
RDT, and MMS packet headers. Finally, we identified home users and business users in
our traces based on IP prefix matching.

Our trace collection and processing methodology have been validated by extensive ex-
periments on various media server and player products, including Windows Media Player
and Windows Server 2003, Real Player and Helix Server, QuickTime Player and Darwin
Server. All these products have extensions to the standard RTSP and RTP protocols. Due
to the lack of documentation, we reverse-engineered proprietary protocols by capturing
and analyzing media traffic under different environments, with the help of tools such as
tcpdump/windump, ethereal, and NIST Net network emulator.

2.3 Traffic Overview

We have captured 126 GB of streaming data (compressed in gzip format) during the
12-day period. In our workloads, there are 7,591 home users accessing 1,898 servers in
121,091 requests, and there are 219 business users accessing 911 servers in 18,742 requests.
Both users and servers are identified by their public IPs, and the real number of business
users would be much larger due to the usage of NAT (a business IP may host up to 64 users
as shown in Section 2.4.2).
## 2.3.1 Streaming traffic by user communities

Table 2.1: Home user streaming media workload overview

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Product Type</th>
<th># of Requests</th>
<th>TCP/UDP Traffic (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>on demand</td>
<td>audio</td>
<td>WM</td>
<td>28,210</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RM</td>
<td>9,139</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QT</td>
<td>244</td>
</tr>
<tr>
<td>video</td>
<td>WM</td>
<td>67,002</td>
<td>151.21/20.64</td>
</tr>
<tr>
<td></td>
<td>RM</td>
<td>12,117</td>
<td>6.25/17.31</td>
</tr>
<tr>
<td></td>
<td>QT</td>
<td>113</td>
<td>0.01/0.34</td>
</tr>
<tr>
<td>live media</td>
<td>audio</td>
<td>WM</td>
<td>1,499</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RM</td>
<td>1,164</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QT</td>
<td>4</td>
</tr>
<tr>
<td>video</td>
<td>WM</td>
<td>950</td>
<td>13.50/2.85</td>
</tr>
<tr>
<td></td>
<td>RM</td>
<td>643</td>
<td>5.69/3.09</td>
</tr>
<tr>
<td></td>
<td>QT</td>
<td>6</td>
<td>0.00/0.003</td>
</tr>
</tbody>
</table>

Table 2.2: Business user streaming media workload overview

<table>
<thead>
<tr>
<th>Content Type</th>
<th>Product Type</th>
<th># of Requests</th>
<th>TCP/UDP Traffic (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>on demand</td>
<td>audio</td>
<td>WM</td>
<td>9,725</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RM</td>
<td>1,285</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QT</td>
<td>5</td>
</tr>
<tr>
<td>video</td>
<td>WM</td>
<td>5,762</td>
<td>21.18/3.31</td>
</tr>
<tr>
<td></td>
<td>RM</td>
<td>1,057</td>
<td>3.94/0.42</td>
</tr>
<tr>
<td></td>
<td>QT</td>
<td>8</td>
<td>0.00/0.01</td>
</tr>
<tr>
<td>live media</td>
<td>audio</td>
<td>WM</td>
<td>493</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RM</td>
<td>350</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QT</td>
<td>–</td>
</tr>
<tr>
<td>video</td>
<td>WM</td>
<td>50</td>
<td>0.65/0.00</td>
</tr>
<tr>
<td></td>
<td>RM</td>
<td>7</td>
<td>0.20/0.00</td>
</tr>
<tr>
<td></td>
<td>QT</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
(a) On-demand audio file length  
(b) On-demand audio playback duration  
(c) Live audio playback duration

Figure 2.1: On-demand and live audio distributions in the home and business user workloads

(a) On-demand video file length  
(b) On-demand video playback duration  
(c) Live video playback duration

Figure 2.2: On-demand and live video distributions in the home and business user workloads

Table 2.1 and 2.2 show the traffic breakdowns based on the content types (audio/video, live/on-demand) and media service products in the home user and business user workloads, respectively. In these tables, WM, RM, and QT denote Windows, RealNetworks, and QuickTime media services, respectively. In our workloads, most streaming traffic is delivered over Windows media services (80.7% and 85.5% of the requests in the home and
business user workloads, respectively), and RealNetworks is next (19.0% and 14.4% of the requests in the home and business user workloads, respectively). Only a small fraction of streaming traffic is delivered over QuickTime. These tables also indicate that TCP is responsible for the majority of streaming traffic on the Internet, confirming previous studies such as [57, 107].

In the home user workload, 2.20% and 31.05% of the requests access live and on-demand audio objects, and 1.32% and 65.43% of the requests access live and on-demand video objects, respectively. Video is responsible for the majority of streaming media requested by home users. In contrast, in the business user workload, 4.50% and 58.77% of the requests access live and on-demand audio, while 0.30% and 36.43% of the requests access live and on-demand video, respectively. Audio is responsible for the majority of streaming media requested by business users. In addition, although the volume of live media traffic is far less than that of on-demand media traffic in both workloads, compared to home users, business users are more likely to access live media, most of which is audio.

Figure 2.1(a) and Figure 2.2(a) show the distribution of file length (in terms of playback duration) of on-demand audio and video objects in each streaming session requested by home and business users, respectively. These figures indicate that business users tend to request audio and video objects with longer file lengths. Specifically, for audio objects, as shown in Figure 2.1(a), more than 70% of sessions in the business user workload request objects with a file length between 200–400 seconds, the typical duration of a pop song; in contrast, for home users, more than 50% of audio sessions request files with a length around 30 seconds, most of which are music preview samples.

Figure 2.1(b) and Figure 2.2(b) further compare user playback durations of on-demand audio/video sessions in the home and business user workloads. As shown in Figure 2.1(b),
for more than half of on-demand audio sessions in the business user workload, the user playback duration is about 200–400 seconds, corresponding to the length of a typical pop song as in Figure 2.1(a). Figures 2.1(c) and 2.2(c) show the playback duration of live audio/video sessions in the home and business user workloads. From these figures, we can see that for both live and on-demand audio/video sessions, the playback duration of business users is much longer than that of home users (note that the x-axis is in log scale).

The above results indicate that business users tend to listen more audio on the Internet and tend to stick to the media content being played longer than home users. However, looking into the URLs and the Referer headers of RTSP commands in the trace, we found that the majority of streaming media accessed by home users and business users are both from news and entertainment sites. Thus, the different access pattern of business users is not due to the accesses of business related media content, but rather due to the working environments of business users—audio is more preferred possibly because it attracts less attention when they are working, and the long playback duration might be because their business work prevents them from frequently changing media programs.

2.3.2 Streaming traffic by media hosting services

In general, there are two approaches that a content provider can use to deliver its streaming media. The first is that a content provider can host the streaming server itself. We call this self-hosting. The second is that a content provider can ask help from a third party, such as a commercial content delivery network (CDN) or media delivery network (MDN), to avoid service management and hardware/software investments. We call this third-party hosting. For both self-hosting and third-party hosting, the portals of streaming media are usually on the Web sites of their content providers.
<table>
<thead>
<tr>
<th>Hosting Service</th>
<th>Content Type</th>
<th>Home User</th>
<th>Business User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Third Party</td>
<td>audio</td>
<td>13.82</td>
<td>10.19</td>
</tr>
<tr>
<td></td>
<td>video</td>
<td>126.24</td>
<td>13.71</td>
</tr>
<tr>
<td>Self Hosting</td>
<td>audio</td>
<td>11.41</td>
<td>7.64</td>
</tr>
<tr>
<td></td>
<td>video</td>
<td>95.33</td>
<td>16.00</td>
</tr>
</tbody>
</table>

Table 2.3: Streaming media traffic by different hosting services

We identified service providers in our workloads by their host names and IP addresses. The host names are extracted from the URLs of media objects encoded in RTSP/MMS packets. We have identified 24 third-party media services, including 22 CDNs/MDNs and 2 media services hosted by ISPs (we anonymize the company names due to customer privacy concerns).

Table 2.3 shows the volume and number of requests of streaming traffic served by third-party media services and self-hosting media services in the home and business user workloads, respectively. In the home user workload, third-party media services serve 56.8% of
the traffic and 67.7% of the requests. In the business user workload, third-party media services serve 50.3% of the traffic and 72.6% of the requests. The percentages of audio traffic served by third-party services are similar for home and business users, but business users request more video from self-hosting media services. Our further investigation finds that a substantial amount of such video traffic comes from a news site and a sports site outside United States, which might be due to the foreign employees in these business companies. In general, more than half streaming traffic is served by third-party hosting services in both workloads.

Figure 2.3(a) and Figures 2.3(b), 2.3(c) further show the rank of traffic volume served by different service providers in the home and business user workloads, for third-party hosting services and self-hosting services, respectively. Most streaming traffic is served by the top five CDNs/MDNs and the top two self-hosting commercial sites (one video site and one well-known search engine site).

### 2.4 Protocol Rollover and User Startup Latency

Although traditionally UDP is the default transport protocol for streaming media data transmission, in practice, UDP is often shielded at the client side. Therefore, today streaming media data are often delivered over TCP or even HTTP. Among the three options, generally UDP is tried first upon a client request. If UDP is not supported due to either server side or client side reasons, TCP can be used instead. If TCP is not supported either, HTTP will be used. Such a procedure is called *protocol rollover*, which is conducted automatically by the media player.

Due to the wide deployment of NAT routers/firewalls in home user networks and small business networks, protocol rollover plays an important role in the user perceived startup
latency and may have affected the way that media is served by content providers. In this section, we first analyze the impact of protocol rollover on the user perceived startup latency, then investigate rollover avoidance in these streaming media services.

### 2.4.1 Startup latency due to protocol rollover

Protocol rollover in RTSP works as follows (protocol rollover in MMS is similar). Upon a client request, the media player sends a `SETUP` command with a `Transport` header, specifying the transport protocol it prefers. If UDP is supported, the port numbers for receiving data and sending feedback are also specified. Then in the `Transport` header of the `SETUP` reply message, the streaming server returns the protocol it selects. If it selects UDP, the port numbers for sending data and receiving feedback are also returned. If the player requests UDP but the server does not support it, the server responds with the protocol it supports (i.e., TCP) directly, and the protocol switches without additional overhead. However, if the server supports UDP but the player is shielded by a NAT router/firewall, the incoming UDP traffic may not be able to go through the router. After a timeout, the player has to terminate the current RTSP connection and sends a new RTSP request in a new TCP connection, specifying TCP as the transport protocol in the `SETUP` command. As a result, such a negotiation procedure for protocol rollover takes a non-trivial time.

Thus, the startup latency of a user session can be further decomposed into three parts: (1) protocol rollover time is the duration from the beginning of the first RTSP/MMS request to the beginning of the last RTSP/MMS request in the user session; (2) transport setup time is the duration from the the beginning of the last RTSP/MMS request (or the first request if no protocol rollover) to the time when the transport protocol setup succeeds; (3) startup buffering time is the time to fill the play-out buffer of the media player, starting from the
Figure 2.4: Protocol rollover increases startup latency of streaming sessions

transport setup success time to the playback start time. In our workloads, we have found that although most user sessions with protocol rollover try UDP only once, some sessions may try UDP up to 3 times before switching to TCP. As the negotiation process of protocol rollover may take a non-trivial time, the protocol rollover increases the startup latency a user perceives.

Assuming a five-second play-out buffer [6] is used, Figure 2.4(a) and Figure 2.4(b) show the distribution of startup latency in Windows and RealNetworks media services, respectively, for RTSP sessions with protocol rollover in the home user workload (the distribution for business user workload is similar). In Windows media services, more than 22% of the streaming sessions have a rollover time longer than 5 seconds, in addition to the normal startup latency due to the transport setup and the initial buffering. Figure 2.4(b) shows that RealNetworks media services have an even longer rollover time—more than 67% of the streaming sessions have a rollover time longer than 5 seconds. We also observe that in general the Windows Media service has a much shorter buffering time than
the RealNetworks Media service. This is probably due to the higher buffering rate of Windows media streaming (see *Fast Start* in Section 2.5), since the object encoding rates of Windows and RealNetworks media in our workloads are comparable. Figure 2.4(c) further compares the delay from the session beginning time to the transport setup completion time for sessions with and without protocol rollover in Windows media services in the home user workload. As shown in the figure, about 37% of the sessions with protocol rollover have a delay longer than 5 seconds. In contrast, only about 13% of the sessions without protocol rollover have a delay longer than 5 seconds.

### 2.4.2 Protocol selection and rollover avoidance

In most client side media players, although the transport protocol of streaming media can be specified by a user manually, the default protocol is usually UDP. In Section 2.3, we have also found that the majority of streaming traffic in our workloads is delivered over TCP. Thus, it is reasonable to expect that a large portion of the user sessions experience protocol rollover.

However, in the home user workload, we found that there are only about 7.37% of streaming sessions trying UDP first and then switching to TCP. In the business user workload, only about 7.95% streaming sessions switch from UDP to TCP. These results imply that TCP is directly used without protocol rollover in most streaming sessions, despite the default protocol setting in the player. We analyze this phenomenon as follows.

The Windows streaming service allows the specification of the transport protocol in the URL modifier at either the client side or the server side. In Windows media streaming, the URL passed to a media player is either input by a user (client side action) or exported from a media meta file stored on or dynamically generated by a Web server (server side action).
For example, \texttt{rtsp} means using TCP as the transport protocol while \texttt{rtspu} means using UDP. We extracted URL modifiers from the summary of media playing information, which is sent by the client in a RTSP/MMS command when a session is terminated. In both home user and business user workloads, we found that for more than 70% of the Windows streaming sessions, TCP is specified as the transport protocol by content providers. This explains why TCP-based media streaming is so prevalent on the Internet. Study [107] suggests that the NAT and firewall deployment constrained the usage of UDP in streaming media applications. Our conjecture is that as content providers are generally aware of the wide deployment of NAT and firewalls, they actively use TCP to avoid any possible shielding or protocol rollover to end users. With such a configuration, even if UDP is supported at both the client side and the server side, the streaming media will still be delivered over TCP directly.

To validate our conjecture, we further investigate the NAT usage of home users and business users with the MMS streaming in our workloads. Different from RTSP, in MMS streaming, a client reports its local IP address to its server in clear text. Extracting this information from the MMS workload, we found that most MMS users in the home and business user workloads report private IPs (such as 192.168.1.100), indicating that they access the Internet through NAT. In the home user workload, about 98.3% of the MMS requests are initiated from clients shielded by NAT, and about 99.5% of the MMS clients are shielded by NAT. These two numbers are 89.5% and 88.0% in the business user workload, respectively. A NAT router hosts up to 3 MMS clients in the home user workload, and up to 64 MMS clients in the business user workload. Thus, for these clients, the TCP transmission specified on the server side effectively avoids protocol rollover, and significantly reduces user perceived startup latency.
On the other hand, RealNetworks media services try to avoid protocol rollover by using NAT transversal techniques. By reverse-engineering of the protocol, we find that different from the Windows media service, in which a server sends UDP packets to its client first through the port that the client reports in the SETUP command, in the RealNetworks media service, a client sends UDP packets to its server first, so that the server can figure out the client’s external UDP port number converted by NAT. To distinguish different user sessions shielded by the same NAT, the server uses different UDP port numbers to listen to UDP packets coming from different sessions, which are generated by the server dynamically and sent to its clients in the replies of SETUP commands. As a result, UDP accounts for the majority of streaming traffic in the RealNetworks media service, and protocol rollover is less frequent than that in the Windows media service. However, as indicated by Figure 2.4, once a protocol rollover happens, the rollover time in the RealNetworks media service is generally much longer than that in the Windows Media service. Furthermore, this solution somehow violates the standard RTSP specification because the UDP port number that a client reports to its server in the SETUP command is intentionally discarded.

2.5 Fast Streaming

In early streaming media services, a media object is streamed at its encoding rate and a small play-out buffer is used to smooth the streaming jitter. However, in practice, the play-out buffer may be exhausted, since the available bandwidth between a client and its server may fluctuate from time to time. This is particularly important for TCP-based streaming, in which the congestion control mechanism constrains the streaming rate. As we have shown in Section 2.4, content providers of Windows media services often use TCP-based streaming directly to avoid protocol rollover. In order to provide high quality streaming
<table>
<thead>
<tr>
<th>Delivery Method</th>
<th>home user</th>
<th>business user</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>third-party</td>
<td>self-hosting</td>
</tr>
<tr>
<td>FC</td>
<td>55.00 GB</td>
<td>68.72 GB</td>
</tr>
<tr>
<td>TCP</td>
<td>49.91 GB</td>
<td>15.27 GB</td>
</tr>
<tr>
<td>UDP</td>
<td>35.15 GB</td>
<td>22.74 GB</td>
</tr>
</tbody>
</table>

Table 2.4: Streaming traffic with Fast Cache

experience to end users, Windows Media services use *Fast Streaming* techniques [7], including Fast Start, Fast Cache, Fast Recovery, and Fast Reconnect \(^2\). Both Fast Start and Fast Cache transmit media data at a rate higher than the media encoding rate \(^3\). Fast Start can run over both TCP and UDP while Fast Cache always runs over TCP. Fast Start is enabled by almost all Windows media servers in order to reduce the startup buffering time for clients. Basically, Fast Start transmits data to the client as fast as possible until the play-out buffer is filled. After the Fast Start period, Fast Cache streams media data until the entire object is delivered or the session is terminated by the user. In order to smooth out network bandwidth fluctuations, Fast Cache transmits media data to a client at a speed usually up to 5 times the media encoding rate and the client maintains a growing buffer for the early arrived data.

Table 2.4 shows the total volume of streaming traffic delivered over Fast Cache (FC), normal TCP streaming excluding FC (TCP), and UDP streaming (UDP) in our workloads. As the table shows, Fast Cache is widely used by both third-party hosting services and self-hosting services, accounting for 50.1% and 21.0% of the streaming traffic in the home and

\(^2\)As Fast Recovery and Fast Reconnect events are rare in our workloads, we do not include them in this study.

\(^3\)Similarly, RealNetworks media services can also stream a media object at a rate higher than its encoding rate. Due to page limits, we only present the analysis results of Windows streaming services.
business user workloads, respectively. There is less streaming traffic delivered over Fast Cache in the business user workload than in the home user workload, because the access pattern of business users is different from that of home users. Business users access more audio and live media than home users. Audio media objects have low bit rates and usually do not need Fast Cache, while live media objects cannot be streamed over Fast Cache at all. The on-demand video they access is different too.

Figure 2.5 compares the distribution of file length and encoding rate of objects delivered over Fast Cache supported streaming and normal TCP streaming in on-demand Windows video sessions of the home user workload. As shown in the figure, Fast Cache is more widely used for objects with longer file lengths and higher encoding rates. This is reasonable because these objects are more sensitive to network bandwidth fluctuations and thus Fast Cache can help more.

Due to page limit, in the remainder of this section, we only present our analysis results of Windows media streaming for TCP-based on-demand video sessions with a playback
duration longer than 30 seconds in the home user workload, if not specified particularly (the results for the business user workload are similar).

### 2.5.1 Fast Cache smoothes bandwidth fluctuation

As mentioned before, Fast Cache supported streaming smoothes the fluctuation of network bandwidth by maintaining a growing buffer for the early arrived data. To understand how Fast Cache utilizes the network bandwidth, we extract the Bandwidth header in a client’s PLAY command, and the Speed header in this command and that in the server’s reply. The Bandwidth header contains the client advertised bandwidth, which is either a default value set by a user in the media player, or measured in real time before media streaming with the so called “packet pair” technique. The Speed header in a client’s PLAY command specifies the delivery speed that the client requests, in multiples of the media encoding rate, which is usually equivalent to the client available bandwidth. The speed that a server agrees to offer is specified in the Speed header of the reply to the PLAY command, which is usually not greater than 5 times of the media encoding rate. However, the actual streaming speed may be smaller than the speed that the server claims in the PLAY reply. Thus, we computed the average of actual streaming rate during each user session based on the packet level information in our workload (the startup buffering is excluded).

Figure 2.6(a) shows the distribution of the client advertised bandwidth, the client requested streaming rate (i.e., the product of the Speed value and the media encoding rate), and the actual streaming rate for streaming sessions with Fast Cache support. Figure 2.6(b) shows the advertised bandwidth and the actual streaming rate for normal TCP streaming. Comparing Figures 2.6(a) and 2.6(b), we find that the actual streaming rate in Fast Cache supported streaming is much closer to the client advertised bandwidth than that in normal
TCP streaming\(^4\). So Fast Cache exploits the unutilized network bandwidth under the constraint of TCP congestion control. In other words, for normal TCP streaming, it is possible to deliver the same media at a higher transmission rate, or to deliver the media with a higher encoding rate.

In Windows media streaming, the default play-out buffer size accounts for five seconds media playback [6]. Upon network fluctuations, if the play-out buffer is empty, the client has to stop to buffer data, and a playback jitter occurs. With Fast Cache, the early buffered data could afford a smooth playback much longer before rebuffering is necessary. We define the rebuffering ratio of a streaming session as the total time for rebuffering over the total playback duration, which reflects the streaming quality that a user experiences. Figure 2.7(a) shows the rebuffering ratio of sessions streamed with and without Fast Cache support (i.e. Fast Cache and normal TCP streaming in the figure). To make a fair comparison, we only consider sessions requesting video objects with an encoding rate between

\(^4\)The 2 Gbps client advertised bandwidth corresponds to the player’s connection speed setting “10 Mbps and above”.

---

Figure 2.6: The client advertised bandwidth, client requesting rate, and server streaming rate
200–400 Kbps. About 15% of the normal TCP streaming sessions suffer rebuffering while only about 8.5% of the Fast Cache supported streaming sessions suffer rebuffering. Thus, Fast Cache can effectively eliminate rebuffering in streaming sessions. We also observe that for Fast Cache supported streaming, there are about 1.8% of the sessions with a rebuffering ratio larger than 50%, while for normal TCP streaming, there are only about 0.9% of the sessions with a rebuffering ratio larger than 50%. The reason is that when rebuffering happens, normal TCP streaming may switch to a stream of lower rate if the media object is MBR encoded (see Section 2.6.1), thus avoiding further rebuffering. In contrast, stream switch is disabled in Fast Cache supported streaming, thus the rebuffering may happen repeatedly, resulting a large rebuffering ratio (see Section 2.6.3).

Fast Cache supported streaming also decreases the possibility of encountering network congestion by reducing the data transmission time. Figure 2.7(b) shows the distribution of data transmission duration for Fast Cache supported streaming sessions and normal TCP streaming sessions, respectively. We can see that although in general the media objects
streamed with Fast Cache have higher file lengths and encoding rates as shown in Figure 2.5, the transmission time of Fast Cache supported streaming is much shorter than that of normal TCP streaming (note that the x-axis is in log scale).

2.5.2 Fast Cache produces extra traffic

Fast Cache continuously delivers a media object at a rate higher than (usually up to five times) its encoding rate. In reality, the entire media object can be delivered completely to the user in the middle of the user’s playback. If the user stops in the middle, the pre-arrived data for the remaining part are wasted. Considering the well known fact that most sessions only access the initial part of a video object [119], the extra delivered traffic by Fast Cache would be non-trivial, especially for large media objects such as movies.

In this study, we compute the over-supplied traffic as follows. The packet size and the time that a packet should be rendered can be extracted directly from the RTP packet header. Based on the timestamps of PLAY, PAUSE, and TEARDOWN commands, we get the user playback duration for each session. Assuming a default five-second play-out buffer [6] on the client side, we compute the extra traffic for each session. Figure 2.8 shows the
extra traffic caused by Fast Cache supported streaming and normal TCP streaming, for all Windows streaming sessions in the home user workload. On average, Fast Cache over-supplies about 54.8% of media data to clients due to clients’ early terminations, while the over-supplied traffic is only about 4.6% for normal TCP streaming sessions. Figure 2.9 shows the CDF of the average transmission speed (in multiples of the media encoding rate) of Fast Cache supported and normal TCP streaming, respectively. In the computation of the average transmission speed, we only consider the data transmission after the Fast Start buffering period. The high data transmission speed in Fast Cache supported streaming indicates the reason for the over-supplied data.

2.5.3 Server response time of Fast Cache

In addition to over-utilizing the network bandwidth, by transmitting media at a rate much higher than its encoding rate, a streaming server running Fast Cache may also consume more CPU, memory, disk I/O, and other resources than a streaming server not running Fast Cache. As a result, a request may have to wait for a longer time to be served when a
burst of requests arrive at the server. We define the server response time of a RTSP/MMS request to be the duration from the time instant when a server receives the first command from a client, to the time instant when the server sends its reply. Since our workloads are collected by Gigascope at a site very close to end users, the timestamp of a captured packet can be regarded as the time instant when the client sends or receives that packet. We use the timestamps of TCP handshake packets to estimate the packet round trip time (RTT), and then compute the server response time.

Figure 2.10(a) shows the distribution of the server response time for streaming requests served by third-party media services. We compare the response time of requests to servers with and without running Fast Cache. For servers running Fast Cache, about 43% of the requests have a response time longer than 0.1 second, while for servers not running Fast Cache, only about 9% of the requests have a response time longer than 0.1 second. Figure 2.10(b) shows the corresponding distribution of the server response time for streaming requests served by self-hosting media services. For servers running Fast Cache, about 21% of the requests have a response time longer than 0.1 second, while for servers not running Fast Cache, only about 5% of the requests have a response time longer than 0.1 second. These results indicate that the response time on servers running Fast Cache is statistically longer than that on servers not running Fast Cache. We also observe that the response time of the third-party hosting service is larger than that of the self-hosting service. As a commercial company, a third-party hosting service may want to fully utilize its server resources with many service subscribers. In contrast, self-hosting services are more dedicated and thus are often less heavily loaded.
2.5.4 Server load of Fast Cache

To further investigate the system resources consumed by Fast Cache, we conducted experiments with the Windows server 2003 and the Windows media load simulator [17]. We ran Windows server 2003 on a machine with 2 GHz Pentium-4 CPU and 512 MB memory, and ran a Windows media load simulator on a Windows XP client machine. The server machine and the client machine are connected through a 100 Mbps fast Ethernet switch. We generated two streaming video files with the Windows media encoder, one is encoded with 282 Kbps and the other is encoded with 1.128 Mbps, both of which have a 20-minute playback duration. We duplicated each file with 50 copies, and saved each copy with a different name. We first ran 50 normal TCP streaming sessions for 5 minutes using the simulator, each of which requests a different copy of the 282 Kbps video file simultaneously. Since the simulator does not support Fast Cache, we ran 50 normal TCP streaming sessions requesting the 1.128 Mbps video using the simulator, in order to simulate the streaming with Fast Cache support for the 282 Kbps video (with 4 speed). This experiment was also conducted for 5 minutes, with each session requesting a different file copy simultaneously. In each experiment, the simulator recorded the CPU and memory usage reported by the server every second. The average bandwidth usage was recorded in the server logs. We repeated each experiment 10 times.

Figure 2.11 shows the average usage of CPU and bandwidth of the server over the entire duration of the simulation (the memory usages of Fast Cache and normal TCP streaming are very close and thus are not presented). The bandwidth usage of Fast Cache is 3.67 times of that of the normal TCP streaming, while the CPU load of Fast Cache is 3.57 times of that of normal TCP streaming. This indicates that the CPU consumed by Fast Cache is approximately proportional to the streaming delivery rate. Given that Fast Cache could
deliver a media object at a rate 5 times of its encoding rate, Fast Cache increases server load significantly, and thus limits the scalability of a streaming server. In our workloads, the Windows media servers in the second largest media delivery network and the largest self-hosting media service (a well known search engine site) do not support Fast Cache at all (we anonymize their domain names due to customer privacy concerns), which might be due to the concerns of high resource demands of Fast Cache.

### 2.5.5 Effectiveness of resource over-utilization

Fast Cache delivers a media object to a client faster than the playing speed by over-utilizing the bandwidth and CPU resources. However, streaming a media object at a rate higher than its encoding rate is only possible when the available bandwidth between a client and its server is large enough. Intuitively, when a media object is streamed at its encoding rate, the higher the average bandwidth between a client and its server over its encoding rate,
rate, the lower possibility at which performance degradation occurs during the playback. To understand whether Fast Cache performs better than normal TCP-based streaming when the average bandwidth between a client and its server is large enough, we plot the CDF of rebuffering ratio for Fast Cache based streaming sessions and normal TCP-based streaming sessions in the home user workload in which the media encoding rate of each stream is 200–320 Kbps and the client advertised bandwidth (extracted from the Bandwidth header) is at least 500 Kbps greater than the media encoding rate, as shown in Figure 2.12.

Compared with Figure 2.7(a), the two curves in Figure 2.12 are very close, which means that, although temporary network congestion may occur from time to time, a small play-out buffer performs well enough to smooth out bandwidth fluctuation during streaming, when the average bandwidth is large enough. Thus, aggressively over-utilizing the server and Internet resources is neither performance-effective nor cost-efficient under a high bandwidth condition. The higher speed at which Fast Cache can stream a media object, the lower necessity is this speed for a client. Furthermore, even if no extra traffic generated (assume the media object is played completely in each session), the number of concurrent streams

Figure 2.12: Effectiveness of resource over-utilization
on a server is constrained by the streaming speed, and thus limits the server’s capacity to service bursty requests.

2.6 Rate Adaptation

In order to adapt to bandwidth fluctuations, major media services such as Windows media and RealNetworks media support three kinds of techniques for rate adaptation. *Stream switch* enables a server to dynamically switch among streams with different encoding rates for the same object, based on the available network bandwidth. This technique is called *Intelligent Streaming* in the Windows media service [12] and *SureStream* in the RealNetworks media service [15]. *Stream thinning* enables a server to only send key frames to the client, when no lower bit rate stream is available. If the current bandwidth is not sufficient to transmit key frames, a server can only send audio to client, which is called *video cancellation*.

2.6.1 MBR encoding and stream switch

To enable stream switch, the media object must be encoded with *multiple bit rates* (MBR): the encoder generates multiple streams with different bit rates for the same media content, and encapsulates all these streams together.

Figures 2.13(a), 2.13(b), 2.13(c) and Figures 2.13(d), 2.13(e), 2.13(f) show the distribution of the number of streams encoded in on-demand and live media objects in the home user and business user workloads, respectively. For video objects, we show the number of audio streams and video streams in a file separately (Figures 2.13(b), 2.13(c) for on-demand objects and Figures 2.13(e), 2.13(f) for live objects). Because there are only a small amount of live video objects in the business user workload, we do not present them
in Figures 2.13(e), 2.13(f). These figures show that about 42% of the on-demand video objects in the home user workload are encoded with at least two video streams. The number of video streams in video objects is up to 12, and the number of video and audio streams together in a video object is up to 20. The number of streams in live audio objects is relatively small, but there are still 13% and 28% of the objects in home user and business user workloads encoded with at least two streams, respectively. These results indicate that the MBR encoding technique has been widely used in media authoring, which enables the rate adaptation—dynamically switching among streams based on the available bandwidth.

The stream switch in RTSP protocols works as follows (stream switch in MMS has a similar procedure). When a RTSP session is established upon a client request, the media player sends a `DESCRIBE` command to the server, asking for the description of the
requested media object. In the reply to the DESCRIBE, the server sends the media description using SDP [63], including the description of each video/audio stream encapsulated in the media object. Then the client specifies the stream that it desires in the SETUP (Windows media service) or SET_PARAMETER (RealNetworks media service) command, based on its current available bandwidth. The server delivers the requested stream upon receiving the PLAY command from the client.

If during playback, the available bandwidth drops below the media encoding rate, the play-out buffer will be drained off. In this case, the media player may send a request to ask the server to switch to a lower rate stream. In Intelligent Streaming (Windows media service), the media player sends a SET_PARAMETER with a SSEntry message body via RTSP, specifying the current stream and the stream to switch to. In SureStream (RealNetworks media service), the client sends a SET_PARAMETER command with an UnSubscribe header to cancel the current stream and a Subscribe header to switch to the new stream.

We extracted all related information from RTSP/MMS commands, and analyzed these stream switches. To characterize the overhead and frequency of stream switches, we define the switch latency as the freezing duration between the end of the old stream and the beginning of the new stream, during which a user has to wait for buffering. We also define the low quality duration of a streaming session as the total playback time of streams with lower rates (relative to the highest encoding rate that the content is transmitted in this session).

Assuming a five-second play-out buffer [6], Figure 2.14(a) and Figure 2.14(b) show the distribution of stream switch latency and low quality duration in the home and business user workloads, respectively. As shown in Figure 2.14(a), about 30%–40% of the stream switches have a switch latency greater than 3 seconds, and about 10%–20% of the stream
switches have a switch latency greater than 5 seconds, which is non-trivial for end users. In Figure 2.14(b), we observe that about 60% of the sessions have a low quality duration less than 30 seconds, and 85% of the low quality stream durations are shorter than 40 seconds.

2.6.2 Stream thinning and video cancellation

Stream thinning works in a similar way as stream switch. To characterize the quality degradation due to stream thinning, we define the *thinning duration* as the time duration from the “thinning” command to the “un-thinning” command or the “stop playing” command, which reflects the quality degradation time that a user suffers. We also define the *thinning interval* as the interval between two consecutive stream thinning events, which reflects the frequency of such quality degradations. Figure 2.15(a) and Figure 2.15(b) show the thinning duration and the interval for video sessions longer than 30 seconds in the home and business user workloads, respectively. As shown in Figure 2.15(a), more than 70% of the thinning durations are shorter than 30 seconds. Figure 2.15(b) shows most (70%) in the
home user workload and 82% in the business user workload) thinning intervals are longer than 30 seconds.

When bandwidth is too low to transmit the key frame of video stream, the client may send a **TEARDOWN** command to cancel the video stream, then the server sends audio only. When the bandwidth increases, the client may set up and request the video stream again.

### 2.6.3 Summary of Internet streaming quality

According to our extensive trace analysis and real experiments, Fast Cache does not support rate adaptation in practice. In a streaming session with Fast Cache enabled, the client never requests to switch streams after the initial stream selection in the **SETUP** command, even if there is a more suitable stream matching the decreased/increased bandwidth during playback. Thinning and video cancellation are also disabled when Fast Cache is enabled. As a result, when the bandwidth drops below the encoding rate, Fast Cache supported streaming performs like **pseudo streaming** [57]: the player stops to buffer data for a while, then continues to play the media for about five seconds (the play-out buffer size), and
this procedure repeats. With such a configuration, if a sudden network congestion happens and lasts for a long time, the streaming quality of Fast Cache supported streaming could be even worse than that of normal TCP streaming. Figure 2.16 shows that when rebuffering happens, the rebuffering duration of Fast Cache supported streaming is much longer than that of normal TCP streaming in the home user workload, because it cannot switch to a lower rate stream upon network congestion.

Figure 2.17(a) and Figure 2.17(b) show the CDF of playback duration of TCP-based video streaming sessions that are longer than 30 seconds in the home and business user workloads, respectively. The three curves in each figure denote all sessions, sessions without quality degradations, and sessions with quality degradations (including rebuffering, stream switch, stream thinning, and video cancellation), respectively. We can see that for sessions with longer durations, degradation happens with a higher probability. For example, in the business user workload, 88% of the sessions with quality degradations have a duration longer than 100 seconds, while 58% of the sessions without quality degradations have a duration longer than 100 seconds. Table 2.5 further shows the breakdowns of sessions with and without quality degradations for TCP-based video streaming sessions that
are longer than 30 seconds and longer than 300 seconds, in the home and business user workloads, respectively. We can see that quality degradation happens less frequently in the home user workload than in the business user workload, which may be due to the longer playback duration of business users as shown in Figure 2.17. For sessions longer than 30 seconds, 13%–40% of the video sessions still have quality degradation due to the rebuffering, stream switch, stream thinning, and video cancellation. For sessions longer than 300 seconds, the quality is getting worse. Further investigation shows that in a significant amount of video sessions with rebuffering, the requested media objects are MBR encoded, and the lack of stream switch is largely due to the usage of Fast Cache, which disables rate adaptation.

In conclusion, the quality of media streaming on the Internet leaves much to be improved, especially for those sessions with longer durations.
<table>
<thead>
<tr>
<th>streaming quality</th>
<th>duration &gt; 30 sec</th>
<th>duration &gt; 300 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>home</td>
<td>biz</td>
</tr>
<tr>
<td>smooth playback</td>
<td>87.06%</td>
<td>59.96%</td>
</tr>
<tr>
<td>rebuffering only</td>
<td>8.65%</td>
<td>18.91%</td>
</tr>
<tr>
<td>stream switch</td>
<td>0.83%</td>
<td>16.30%</td>
</tr>
<tr>
<td>stream thinning</td>
<td>1.94%</td>
<td>3.42%</td>
</tr>
<tr>
<td>video cancellation</td>
<td>1.52%</td>
<td>1.41%</td>
</tr>
</tbody>
</table>

Table 2.5: Summary of streaming quality

### 2.7 Coordinating Caching and Rate Adaptation

Fast Cache and rate adaptation are two commonly and practically used techniques that improve the experience of streaming media users from different perspectives. Fast Cache **aggressively** buffers media data in advance at a rate higher than the media encoding rate, aiming to absorb the streaming jitter due to network congestion. In contrast, rate adaptation **conservatively** switches to a lower bit rate stream upon network congestion. As shown in our analysis, both techniques have their merits and limits. Fast Cache has its problems such as increasing server load and producing extra traffic. On the other hand, the latency of stream switch is non-trivial in most sessions, due to the small size of play-out buffer.

Combining the merits of both techniques, in this section, we discuss *Coordinated Streaming*, a mechanism that coordinates caching and rate adaptation. In this scheme, an upper bound and a lower bound are applied to the play-out buffer of the client player. The upper bound setting prevents aggressive data buffering while the lower bound setting eliminates the stream switch latency. When a streaming session starts, the server transmits data to the client as fast as possible until the lower bound is reached. Then the playback begins and the client continues to buffer data with the highest possible rate until the buffer
reaches its upper bound. With a full buffer, the client buffers data at the media encoding rate, and the buffer is kept full. When network congestion occurs, the client may receive data at a rate lower than the object encoding rate, and the buffer is drained off. If the network bandwidth increases before the buffer drops below its lower bound, the client will request data at a higher rate to fill the buffer. Otherwise, the client will switch to a lower rate stream. The selection of the lower rate stream should be based on the following: in a typical bandwidth fluctuation period, the current bandwidth should be able to maintain normal playback of this lower rate stream and transmit extra data to fill the buffer to its upper bound. When the network bandwidth is increased, the client may switch to a higher encoding rate stream.

We conducted an ideal experiment as a proof of concept of this scheme. We set the lower bound of the buffer size as 5 seconds to cover the normal stream switch latency as well as the initial buffering duration, as the default play-out buffer size is 5 seconds, and the average stream switch latency is also about 5 seconds. The upper bound of the buffer size is set to 30 seconds, considering the typical network fluctuation periods that may affect streaming quality, such as low quality duration, thinning duration, and thinning interval (Figures 2.14(b), 2.15(a), and 2.15(b)). In a practical system, the lower and upper bound of buffer size should be adaptively tunable based on these quality degradation events. However, we will show that even with the above simple configuration, the streaming quality can be effectively improved and the over-supplied traffic can be significantly reduced.

We simulated the Coordinated Streaming scheme based on the packet level information of Fast Cache supported streaming sessions, and compared the quality and bandwidth usage of this scheme with that of Fast Cache supported streaming and normal TCP-based streaming. To have a fair comparison, we only consider video sessions that request objects
with 200–400 Kbps encoding rates for a duration longer than 30 seconds in the home user workload. Figure 2.18(a) shows the rebuffering ratio in Fast Cache supported streaming, normal TCP streaming, and Coordinated Streaming. As shown in this figure, the rebuffering ratio of Coordinated Streaming is close to zero. The fraction of normal TCP streaming sessions with large rebuffering ratios is close to or even smaller than that of Fast Cache supported streaming sessions because many of them use rate adaptation to avoid rebuffering. Figure 2.18(b) shows the over-supplied data in the above three schemes, and we can see that Coordinated Streaming reduces 77% over-supplied traffic produced by Fast Cache, although not as good as normal TCP streaming. Figure 2.18(c) shows that the switch handoff latency of Coordinated Streaming is nearly zero, much less than that of normal TCP streaming. Furthermore, the number of stream switches in our scheme is only 33.4% of that in normal TCP-based streaming.
2.8 Related Work

Existing measurement studies have analyzed the Internet streaming traffic in different environments and from different perspectives. Li et al. [79] characterized the streaming media stored on the Web, while Mena et al. [84] and Wang et al. [116] presented an empirical study of Real audio and Real video traffic on the Internet, respectively. These studies characterized the packet size, data rate, and frame rate patterns of streaming media objects. Almeida et al. [24] and Chesire et al. [38] studied the client session duration, object popularities and sharing patterns based on the workload collected from an educational media server and an university campus network, respectively. Cherkasova et al. [37] characterized the locality, evolution, and life span of accesses in enterprise media workloads. Yu et al. [123] studied the user behavior of large scale video-on-demand systems. Padhye et al. [88] and Costa et al. [44] characterized the client interactivity in educational and entertainment media sites, while Guo et al. [58] analyzed the delay of jump accesses for video playing on the Internet. Live streaming media workloads have also been studied in recent years. Veloso et al. [113] characterized a live streaming media workload in three increasingly granular levels, named clients, sessions, and transfers. Sripanidkulchai et al. [107] analyzed a live streaming workload in a large content delivery network.

However, these on-demand and live streaming media measurements mainly concentrated on the characterization of media content, access pattern, and user activities, etc. So far, few studies have focused on the mechanism, quality, and resource utilization of streaming media delivery on the Internet. Chung et al. [40] and Nichols et al. [87] conducted an experimental study in a lab environment on the responsiveness of RealNetworks and Windows streaming media, respectively. Wang et al. [114] proposed a model to study the
TCP-based streaming. In contrast to these studies, we analyzed the delivery quality and resource utilization of streaming techniques based on a large scale Internet streaming media workload.

2.9 Conclusion

In this study, we have collected a 12-day streaming media workload from a large ISP, including both live and on-demand streaming for both audio and video media. We have characterized the streaming traffic requested by different user communities (home users and business users), served by different hosting services (third-party hosting and self-hosting). We have further analyzed several commonly used techniques in modern streaming media services, including protocol rollover, Fast Streaming, MBR, and rate adaptation. Our analysis shows that with these techniques, current streaming services tend to over-utilize the CPU and bandwidth resources to provide better services to end users, which may not be a desirable and effective way to improve the quality of streaming media delivery. A coordination mechanism that combines the advantages of both Fast Streaming and rate adaptation techniques is proposed to effectively utilize the server and Internet resources for building a high quality streaming service. Our trace-driven simulation study demonstrates its effectiveness. Some preliminary results of this work have been presented in [61].
CHAPTER 3

THE ACCESS PATTERN MODEL OF INTERNET MEDIA SYSTEMS

3.1 Introduction

Different from conventional Web content, Internet media content can be delivered through a variety of approaches, such as streaming, pseudo streaming, overlay multicast, and P2P techniques, in addition to the common Web-based downloading. Unlike the widely accepted Zipf-like access pattern of Web traffic [30], where text-based content was dominant, existing studies on media traffic are largely workload specific, and the observed access patterns are often different from or even conflict with each other, due to the variety of media delivery systems and the diversity of media content. For example, Chesire et al. [38] and Yu et al. [123] report that the access pattern of streaming media is Zipf-like in a university campus and in a VoD system, respectively, while Acharya et al. [22] and Cherkasova et al. [37] find that it is not Zipf-like in a multicast-based Media-on-Demand server and in an enterprise server, respectively. For live streaming, Veloso et al. [112] report user interests are object driven and follow a Zipf-like profile, while Sripandkulchai et al. [107] report the popularity of live media programs hosted by Akamai CDN follows a 2-mode Zipf distribution. For P2P media systems, Gummadi et al. [52] report that the access pattern of media
workload in KaZaa system collected in a campus network is not Zipf-like, while Iamnitchi et al. [65] report that it is Zipf-like for KaZaa traffic collected from the network of an ISP. For the access pattern of YouTube video clips, a client-side study by Gill et al. [49] reports it is Zipf-like while a meta-information based study by Cha et al. [31] reports a significant deviation from the Zipf model.

As a result, a number of models have been proposed or used to characterize Internet media access patterns. In [31], Cha et al. fit the thin tail of the reference rank distribution of YouTube video clips by the Zipf with exponential cutoff effect and use the preferential attachment with information filtering model [86] to explain this effect. Gummadi et al. [52] propose that the popularity distribution of P2P media objects deviates from the Zipf model due to the “fetch-at-most-once” property of P2P clients. In study [98], Saleh et al. find that the popularity of P2P objects in KaZaa networks can be empirically modeled by a Zipf-Mandelbrot distribution, which captures the “flattened head” of the popularity distribution of objects in P2P systems. Sripanidkulchai et al. [107] propose a two-mode Zipf model for live media programs hosted by a content delivery network (CDN). Tang et al. [108] propose a generalized Zipf-like distribution model for objects in long duration enterprise media workloads.

The object access pattern has significant impacts on the locality of references and the performance of caching in Internet systems. However, due to a limited number of workloads (typically one or two in each study) and a constrained scope of media traffic (e.g., enterprise server logs or requests from a campus network), the analyses in these case studies are not sufficient to identify a general media access pattern, which is extremely important for traffic engineering on the Internet and is critical to design, benchmark, and evaluate Internet media distribution systems. In reality, although many algorithms and systems for
media caching/proxying have been proposed, including commercial products such as Helix Universal Proxy [9] and Microsoft Windows Media Proxy [16], few of them are practically used. Instead, due to the high quality requirements and resource demands of media delivery, current media systems tend to over-supply or over-utilize hardware and bandwidth resources for good user experience [61]. Thus, it is highly desirable to have an insightful understanding of media access patterns for both media system designers and network administrators.

In this study, we have analyzed a wide variety of media workloads on the Internet, which cover different content types, including entertainment, business, and educational content. The workloads were collected from both the client side and the server side in Web, VoD, and P2P environments between 1998 and 2006, where the media content is delivered via Web/P2P downloading and unicast-multicast streaming, by P2P clients, enterprise servers, and CDNs. The duration of these workloads ranges from a few days to more than two years and the user population ranges from several thousands to more than one hundred thousand. The number of client requests ranges from tens of thousands to hundreds of million, the number of objects in each workload ranges from several hundreds to several million, and the median of file sizes in each workload ranges from a few megabytes to several hundreds of megabytes.

Through extensive analysis, we find that the reference ranks of media objects in all these workloads can be well fitted by the stretched exponential (SE) distribution [78]. This distribution has two parameters. We find that one parameter well characterizes the media file sizes, the other parameter well characterizes the aging of media accesses. We also analyze factors that may affect the observed media access patterns, such as redundant traffic filtered by a cache and extraneous traffic introduced through an ads server, and find a biased
measurement could lead to a Zipf-like observation on media access patterns. We further analyze the evolution of media access patterns in media systems, and find the deviation of media reference rank distribution from the Zipf model increases along with the duration of the workload. Revisiting existing models of media access patterns, we find that the SE distribution is a general model that can fit all existing case studies, while none of other models can.

We have further proposed a mathematical model to analyze the performance of media caching systems with the stretched exponential distribution. Compared with the caching of Web objects, whose access pattern is Zipf-like, caching of media objects is far less effective, unless the cache space is enormously large, and both LRU- and LFU-based replacement algorithms are less helpful to improve the caching efficiency when only a small percentage of media data can be cached. Our study further shows that with more requests to media objects over time, there is a great potential to improve the performance of client-side caching. However, this improvement may take months to years and consumes huge amount of storage. This implies that a performance-effective and cost-efficient media caching system should be capable of scaling its storage size with the increase of its workload size over a long time. Peer-to-peer networks are promising to provide the enormously large and scalable cache space for this purpose. Thus, the stretched exponential model of media access patterns lays out an analytical foundation to establish peer-to-peer based caching systems for efficient media content delivery.

The remainder of the paper is organized as follows. Section 3.2 summarizes the workloads we use. Section 3.3 overviews related works on media access patterns and media caching. We present the stretched exponential model of media reference rank distributions in Section 3.4, and study the physical meanings of this access pattern model in Section 3.5.
We analyze the impact of the stretched exponential distribution on media caching performance in Section 3.6, and make concluding remarks in Section 3.7.

3.2 Workload Description

In this study, we analyze a total of sixteen media workloads collected from both client and server sides in different media systems with different delivery methods, as outlined in Table 3.1. Table 3.1 also summarizes the workload duration, number of requests, number of clients, and number of objects for each workload. Among these workloads, six of them were collected by ourselves. We also have the access to two workloads available in public sources or provided by our research collaborators. The remaining eight, with an asterisk before the workload name in Table 3.1, are extracted from the figures in the published papers due to the unavailability of original data. These sixteen workloads are classified into four categories based on different media delivery environments, summarized as follows.
<table>
<thead>
<tr>
<th>System Type</th>
<th>Workload Name</th>
<th>Delivery Method</th>
<th>Duration</th>
<th>Collection Time</th>
<th>Num. of Requests</th>
<th>Num. of Objects</th>
<th>Num. of Clients</th>
<th>Median File Size</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Media</td>
<td>*HPC-98</td>
<td>streaming</td>
<td>29 months</td>
<td>11/98-04/01</td>
<td>666,074</td>
<td>2,999</td>
<td>131,161</td>
<td>14 MB</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>*HPLabs-99</td>
<td>streaming</td>
<td>21 months</td>
<td>07/99-04/01</td>
<td>14,489</td>
<td>412</td>
<td>2,482</td>
<td>120 MB</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>ST-SVR-01</td>
<td>streaming</td>
<td>122 days</td>
<td>04/01-07/01</td>
<td>169,414</td>
<td>2,260</td>
<td>41,709</td>
<td>15 MB</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>PS-CLT-04</td>
<td>downloading</td>
<td>9 days</td>
<td>08/04</td>
<td>196,621</td>
<td>53,383</td>
<td>6,276</td>
<td>1.5 MB</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>ST-CLT-04</td>
<td>streaming</td>
<td>9 days</td>
<td>09/04</td>
<td>61,889</td>
<td>18,511</td>
<td>4,751</td>
<td>2 MB</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>ST-CLT-05</td>
<td>streaming</td>
<td>11 days</td>
<td>06/05</td>
<td>54,984</td>
<td>18,634</td>
<td>6,238</td>
<td>4.5 MB</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>*mMoD-98</td>
<td>multicast</td>
<td>194 days</td>
<td>08/97-03/98</td>
<td>–</td>
<td>139</td>
<td>–</td>
<td>125 MB</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>*CTVoD-04</td>
<td>streaming</td>
<td>219 days</td>
<td>05/04-12/04</td>
<td>21 million</td>
<td>6,700</td>
<td>150,000</td>
<td>300 MB</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>IFILM-06</td>
<td>streaming</td>
<td>16 weeks</td>
<td>03/06-07/06</td>
<td>62,228,780</td>
<td>11,872</td>
<td>–</td>
<td>2.25 MB</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>YouTube-06</td>
<td>pseudo stream</td>
<td>all-time</td>
<td>10/06</td>
<td>692,343,054</td>
<td>3,981,654</td>
<td>–</td>
<td>3.4 MB</td>
<td>0.17</td>
</tr>
<tr>
<td>VoD Media</td>
<td>*KaZaa-02</td>
<td>exchange</td>
<td>203 days</td>
<td>05/02-12/02</td>
<td>98,997,622</td>
<td>633,106</td>
<td>24,578</td>
<td>300 MB</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>*KaZaa-03</td>
<td>exchange</td>
<td>5 days</td>
<td>01/03</td>
<td>976,184</td>
<td>116,509</td>
<td>14,404</td>
<td>5 MB</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>BT-03</td>
<td>swarming</td>
<td>48 days</td>
<td>10/03-12/03</td>
<td>256,802</td>
<td>2,453</td>
<td>45,058</td>
<td>636 MB</td>
<td>0.52</td>
</tr>
<tr>
<td>P2P</td>
<td>*Akamai-03</td>
<td>live stream</td>
<td>3 months</td>
<td>10/03-01/04</td>
<td>70 million</td>
<td>5,000</td>
<td>–</td>
<td>–</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>*Movie-02</td>
<td>–</td>
<td>1 year</td>
<td>year 2002</td>
<td>–</td>
<td>250</td>
<td>–</td>
<td>–</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>IMDB-06</td>
<td>–</td>
<td>all-time</td>
<td>06/06/2006</td>
<td>–</td>
<td>250</td>
<td>–</td>
<td>–</td>
<td>1.15</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of media workloads
The first category is “Web media”. These workloads contain media requests in the Web environment, where media objects are embedded or linked in Web pages. On these Web pages, the majority are still common Web objects such as text, graphics, and html files, etc. In this category, HPC-98 (extracted from Figure 7(a) of paper [37]) contains server logs collected from the HP Corporate media server, which hosts the streaming media objects accessed through the Web pages of the HP corporation. Similarly, HPLabs-99 (also extracted from Figure 7(a) of paper [37]) contains the logs of streaming servers hosting HP Labs media content, accessed through the HPLabs Web site. Similar to HPC-98, ST-SVR-01 is the log of a large enterprise streaming server, accessed through the company’s Web site. We collected PS-CLT-04 and ST-CLT-04 from a large cable network hosted by a major ISP in the United States. Workload ST-CLT-05 was collected in a subdomain of the same network. PS-CLT-04 is a media workload of Web downloading and pseudo streaming, which includes the first IP packets of HTTP downloading for Windows, RealNetworks, and QuickTime media files. ST-CLT-04 and ST-CLT-05 are RTSP (RFC 2326) and MMS (Microsoft’s proprietary streaming protocol) streaming media traces collected with a similar method to that by Chesire et al. [38].

The second category is “VoD media”. Different from a Web site with a few media objects, a VoD system provides an integrated environment for dedicated media services, though it is often Web-based too. mMoD-98 contains logs of a multicast-based Media-on-Demand video server supporting VCR functions, where the major contents are video lectures and movies (extracted from Figure 6 of paper [22]). CTVoD-04 contains logs collected from streaming servers of a large VoD system deployed by China Telecom, where the major contents are TV shows and movies (extracted from Figure 13 of paper [123]). These two VoD systems host media content themselves. We collected IFILM-06 from the
IFILM Web site [11], which provides the weekly click numbers of Web pages for IFILM video clips, most of which are short movie trailers. These video objects are served by a CDN via streaming. We have also collected YouTube-06 from its Web site [122] by crawling summary pages of YouTube video, where the total number of requests of each clip (for the entire up time of YouTube site) was published (due to its system maintenance during the workload collection time, among 645,941 pages crawled, only 39.2% of them were successfully downloaded). YouTube video is also hosted by a CDN, but is delivered through pseudo streaming.

The third one is “P2P media”, collected from two kinds of P2P systems. In KaZaa networks, users exchange files with each other, while in a BitTorrent swarm, users exchange chunks of the same file. KaZaa-02 is a large file transferring (larger than 100 MB, typically video files) workload over KaZaa networks, collected on a university campus (extracted from Figure 5 of paper [52]). KaZaa-03 is extracted from Figure 5 of paper [65], which includes music files, movie clips, and movie files of different sizes. BT-03 contains data collected from two BitTorrent tracker sites, where most of the files are large video and DVD movies [27].

The above three categories are all on-demand media workloads. We have also analyzed media workloads of live streaming and theater environments, categorized as “other” in our workload set. Workload Akamai-03 consists of the references to live streaming media programs hosted by Akamai CDN, extracted from Figure 3 of paper [107]. Workload Movie-02 is the 2002 U.S. movie box office ticket sales, extracted from Figure 7(d) of paper [52]. Workload IMDB-06 is the cumulative number of votes for top 250 movies in Internet Movie Database (IMDB), which was downloaded from the IMDB Web site [67].
Client side streaming media workloads often contain *extraneous media objects* that are pushed to the user mandatorily no matter the user wants them or not. For example, for advertisement purposes, normally when a user clicks the meta file link of a media object on a Web page, the media server requests a link of *ad clip* from an ad server, inserts it to a dynamically generated meta file, and then sends the meta file to the client player. Furthermore, some media servers may insert the link of *flag clip*—a small video or audio object (usually less than 5 seconds) that plays a silent audio or blank video, a static logo image, or a quick animation—before the URL of a requested media object and/or between two subsequent requested media files, in the dynamically generated meta file. Ad clips are usually served by dedicated ad servers outside the content system, while flag clips are usually served by the same server serving program content. However, both ad and flag media traffic is *extraneous* to users of the media system, and do not reflect the real user access pattern. Our study shows that although ad and flag clips only account for a small percentage of media traffic, they usually have significantly higher access rates than normal objects. In our collected client side streaming workloads ST-CLT-04 and ST-CLT-05, we identified and removed ad and flag clip requests by matching the URLs in RTSP commands with keywords such as “ads” and “logo”, “getnext”, “next”, which indicate the purpose of these objects, as well as by viewing/listening to the video/audio content of these objects. For server side media workloads, the requests of ad media are not recorded in server logs, since media servers and ad servers are usually separate. Meanwhile, not every media system uses flag clips, and the number of flag clips in a media system is very small (usually 1 or 2).
3.3 Related Work

Both Zipf-like and non-Zipf distributions of media access patterns have been reported by a number of measurement studies:

- For Web media systems, Chesire et al. [38] report that the access pattern of streaming media is Zipf-like in a university campus network, with a skewness factor, the minus of the slope of the reference rank distribution in log-log scale, \( \alpha = 0.47 \). However, a direct estimation from Figure 8 in paper [38] leads to \( \alpha \approx 0.67 \). Similarly, Guo et al. [59] report that the reference rank distribution of streaming media is Zipf-like in a broadband residential network, with a skewness factor \( \alpha = 0.61 \). Different from the two client side workload analysis above, Cherkasova et al. [37] report that media access pattern is not Zipf-like in an enterprise media server.

- For VoD media systems, Griwodz et al. [50] report that the probability of movie rentals is roughly Zipf-like by analyzing the records of 250 movies in rental stores. Acharya et al. [22] observe that the media popularity distribution is not Zipf-like in a multicast-based Media-on-Demand server of a campus network. Yu et al. [123] report the video access pattern is Zipf-like in a large VoD streaming system of an ISP. For the access pattern of YouTube video clips, a client-side study by Gill et al. [49] reports it is Zipf-like while a meta-information based study by Cha et al. [31] reports a significant deviation from the Zipf model.

- For P2P media systems, Iamnitchi et al. [65] report that the file download pattern of KaZaa networks is Zipf-like for workloads collected in a large ISP. However, Gum-madi et al. [52] report that KaZaa file download pattern is not Zipf-like for workloads collected in a campus network. Klemm et al. [76] and Guo et al. [60] observe that
the the number of queries in each session and the number of replies provided by each peers in Gnutella networks do not follow the Zipf-like model, respectively. Furthermore, Chu et al. [39] and Saleh et al. [98] report that the popularity distributions of files in Gnutella networks are not Zipf-like, and Zhang et al. [27] report that the popularity of files in BitTorrent networks is not Zipf-like, either.

- For live media systems, Veloso et al. [112] report that the user interest in a given object is Zipf-like for live streaming workloads collected in a large content service provider, while Sripanidkulchai et al. [107] report that the popularity of live streaming programs served by a CDN is not Zipf-like.

However, for most reported Zipf-like media access patterns listed above, the model fitting is quite rough. For example, in studies [50], [59], and [65], the distribution curve does not strictly follow a straight line in log-log scale. In study [123], although the head and waist of the distribution curve in log-log scale are roughly in a straight line, the tail of the distribution curve, which accounts for the majority of media objects, is far from the straight line. The only well fitted Zipf-like workload is study [38]. However, as we will shown in Section 3.4.2, this might be due to the pollution of extraneous media traffic such as ads and flag clips.

A number of models have been proposed to characterize the non-Zipf Internet media access patterns. In [31], Cha et al. fit the thin tail of the reference rank distribution of YouTube video clips by the Zipf with exponential cutoff effect and use the preferential attachment with information filtering model [86] to explain this effect. Gummadi et al. [52] propose that the popularity distribution of P2P media objects deviates from the Zipf model due to the “fetch-at-most-once” property of P2P clients. In study [98], Saleh et al. find that
the popularity of P2P objects in KaZaa networks can be empirically modeled by a Zipf-Mandelbrot distribution, which captures the “flattened head” of the popularity distribution of objects in P2P systems. Sripanidkulchai et al. [107] propose a two-mode Zipf model for live media programs hosted by a content delivery network (CDN). Tang et al. [108] propose a generalized Zipf-like distribution model for objects in long duration enterprise media workloads. Chu et al. [39] propose a log-quadratic model and Saleh et al. [98] propose a Zipf-Mandelbrot model for Gnutella file popularity distributions, respectively. Most of these models are still based on the Zipf-like distribution, with intuitive assumptions and modifications in order to fit data better. However, each of them can only explain the access patterns of a very limited scope of media workloads. We will reappraise these models in Section 3.4.3 and Appendix C.

With an insightful understanding of access patterns of content systems, temporal locality can be well exploited to improve system performance with a low cost. Breslau et al. [30] have studied the Zipf-like distribution of Web accesses and its implication to proxy caching. Fonseca et al. [47] and Vanichpun et al. [111] have further analyzed the effect of caching on the locality of reference for Zipf-like workloads. Although media access patterns have not been comprehensively studied before, a number of media caching algorithms and systems have been proposed, such as [21], [119], [32], [35], [53], and [80]. Commercial media proxy products are also available, such as Helix Universal Proxy [9] and Microsoft Windows Media Proxy [16]. However, few of them are practically used.
3.4 The Reference Rank Model of Internet Media Objects

In this section, through the analysis and modeling of sixteen media workloads, we (1) show that the reference rank distributions of all sixteen media workloads can be well fitted with the stretched exponential model; (2) investigate factors that may affect the media access pattern observations; (3) discuss why Zipf and pow law based models fail to characterize the access pattern of media objects (other previously proposed media access pattern models are reappraised in Appendix C).

3.4.1 The stretched exponential of Internet media traffic

We use the rank-ordering technique to analyze the Internet media access pattern. Figures 3.1, 3.2, and 3.3 show the reference rank distributions of media objects in Web, VoD, and P2P media systems, respectively. In each figure, the $x$ coordinate represents the reference rank of each object, plotted in log scale, while the $y$ coordinate represents the number of references to this object, plotted in both log scale (marked on the right of $y$-axis) and a powered scale (by a constant $c$, as marked on the left of $y$-axis). We call the combination of log scale in $x$ and powered scale in $y$ as the stretched exponential (SE) scale.

These figures show that in log-log scale, the reference rank distributions of all these workloads have a fat head and a thin tail, which cannot be fitted with a straight line, indicating they are not Zipf-like. In particular, many of them deviate from a straight line significantly, such as Figures 3.2(a), 3.2(b), and 3.3(c). However, by selecting a proper constant $c$, all these workloads can be well fitted with a straight line in SE scale. Such a rank distribution is called a stretched exponential distribution.

\footnote{The extraneous traffic in ST-CLT-04 and ST-CLT-05 has been removed. The figures of workloads HPC-98, HPLabs-99, longer duration of workload IFILM-06, Akamai-03, Movie-02, and IMDB-06 are presented in Figure A.1 of Appendix A.}
Figure 3.1: Reference rank distributions of media objects in Web systems

To evaluate the stretched exponential fit, we compute the coefficient of determination of the fitting result of each workload, $R^2$. As marked in the figures, $R^2$ is very close to 1 for all workloads. For workloads with raw data accesses, $\chi^2$ tests are conducted to check the goodness of fits. The stretched exponential fits are accepted while Zipf-like fits are rejected (see Appendix B). For long term workloads with timestamps of requests, including ST-SVR-01, BT-03, and IFILM-06, stretched exponential fits are further conducted on the reference rank distributions of objects requested in different durations (see Section 3.5.2).
The probability distribution of the stretched exponential distribution can be expressed as

\[ P(X < x) = 1 - e^{-\left(\frac{x}{x_0}\right)^c}, \]  

(3.1)

where \( c \) and \( x_0 \) are constants (this probability distribution is also known as Weibull distribution). If we rank the \( N \) objects in the workload in descending order of their reference numbers \( y_i \) \((1 \leq i \leq N)\), we have \( P(y_n > y_i) = i/N \). So the rank distribution can be

Figure 3.2: Reference rank distributions of media objects in VoD systems
expressed as follows

\[ y_i^c = -a \log i + b \quad (1 \leq i \leq N), \] (3.2)

where \( a = x_0^c \) and \( b = y_N^c \). Since the minimum number of references to an object is 1, we can assume \( y_N = 1 \) when the number of object in the workload, \( N \), is large enough \(^6\). Thus

\[ b = 1 + a \log N. \] (3.3)

An SE distribution curve is a straight line in SE scale. Since \( b \) is a normalization parameter, the shape of an SE distribution is determined by \( c \), the stretch factor of \( y \) coordinate, and \( a \), the minus of the slope of the straight line in SE scale. An SE distribution has a finite mean value (denoted as \( \langle x \rangle \)) \(^78\),

\[ \langle x \rangle = \int_0^\infty xp(x)dx = x_0 \Gamma(1 + \frac{1}{c}), \] (3.4)

where \( p(x) = c e^{x-1} e^{-\left(\frac{x}{x_0}\right)^c} \) is its probability density function and \( \Gamma(\alpha) = \int_0^\infty t^{\alpha-1}e^{-t}dt \) is the gamma function. However, for workloads with a limited number of objects, the deviation of the average number of references to objects (denoted as \( \langle y \rangle \)) from Equation 3.4 can be non-trivial, especially when \( \langle y \rangle \) is small (since the minimum number of references is 1, not zero). A better estimation is

\[ \langle y_{se} \rangle = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \left( 1 - a \log \left( \frac{i}{N} \right) \right)^{\frac{1}{c}} \]
\[ = \lim_{N \to \infty} \int_{\frac{1}{N}}^{1} (1 - a \log x)^{\frac{1}{c}}dx \]
\[ = e^{\frac{1}{c}}x_0(\Gamma(1 + \frac{1}{c}) - \gamma(1 + \frac{1}{c}, \frac{1}{a})), \] (3.5)

where \( \gamma(\alpha, x) = \int_0^x t^{\alpha-1}e^{-t}dt \) is the lower incomplete gamma function. Throughout this paper, the modeling analysis is mainly based on Equation 3.5, while Equation 3.4 is only used as an approximation for simplicity.

\(^6\)As shown in Appendix A, for server side workloads, it is possible that \( y_N > 1 \) due to the small number of objects.

\(^7\)As shown in Appendix A, for server side workloads, it is possible that \( y_N > 1 \) due to the small number of objects.
Figure 3.3: Reference rank distributions of media objects in P2P systems

The stretched exponential distribution has been used to describe many phenomena in nature and economy that do not follow power law [78]. Although it is still empirical, in the subsequent parts of this paper, we will show that the stretched exponential model of Internet media reference rank distributions has clear physical meanings, which further supports the validity of this model.

3.4.2 Factors that may affect media access patterns

In addition to user activities, the computing and networking systems that an entire media delivery procedure involves may affect the measurement results of media object reference rank distributions by filtering redundant traffic (e.g., through a proxy) or introducing extraneous traffic (e.g., through an ad server). In this section, we study how these factors affect the observed media access patterns, and show how a biased measurement can lead to a Zipf-like observation as reported before.
Effect of extraneous traffic

As we have shown in Section 3.2, the client side streaming media workloads often contain extraneous traffic such as ad and flag media clips, which do not reflect the real user access pattern and may significantly affect the object reference rank distribution. Without removing extraneous traffic, Figure 3.4(a) shows that the log-log plot of the reference rank distribution of media objects in workload ST-CLT-04 can be well fitted with a straight line, indicating a Zipf-like distribution (see Equation 3.14 of Section 3.6.1) with a skewness factor $\alpha \approx 0.71$. Similarly, Chesire et al. [38] also find that the reference rank distribution of a streaming media workload collected in University of Washington follows a Zipf-like distribution.

However, as shown in Figure 3.1(c), after removing the extraneous traffic (31% requests in the workload), the distribution can be well fitted with a stretched exponential model, with a skewness factor $\alpha \approx 0.67$ (instead of 0.47 as presented in the paper), quite close to that of workload ST-CLT-04 (0.71).

7 See Figure 8 of paper [38]. The skewness factor (the minus of the slope of log-log plot) estimated from the figure is about 0.67 (instead of 0.47 as presented in the paper), quite close to that of workload ST-CLT-04 (0.71).
while the log-log plot has a clear curvature, indicating it is not a Zipf-like distribution. Since we use a similar method to collect streaming media traffic as that used in study [38], it is possible that the workload used in [38] also contained some extraneous media traffic, which causes the observed Zipf-like distribution. According to [19], streaming video advertising was booming in year 2000-2001, when workload [38] was collected.

Even with extraneous media traffic, it does not seem to be true that a Zipf-like distribution can always capture the access pattern of media objects: we find the extraneous traffic in different workloads varies. As shown in Figure 3.4(b), the raw data in the ST-CLT-05 workload (with extraneous media traffic unchanged) does not follow the Zipf model well. Furthermore, fitting with Zipf-like distribution is rejected by $\chi^2$ test (see Table B.1 of Appendix B).

To illustrate the effect of extraneous traffic on the reference rank distribution, we further investigate the extraneous objects and corresponding requests in workloads ST-CLT-04 and ST-CLT-05, respectively. Analyzing the traces we collected, we find the difference between the reference rank distributions of these two workloads (collected in the same network in different years) is due to the changes of ad and flag media clips in the workloads. As shown in Figures 3.5 and 3.6, the reference numbers of ad and flag clips are both much higher than normal media program objects (denoted as “prog” in the figures). The Zipf-like observation of media reference ranks in workload ST-CLT-04 is caused by those extraneous objects of high reference rates, especially the flag objects of popular video sites.

Compared with workload ST-CLT-04, the amount of ad media traffic from one of the major advertisement media providers, yahoo.com, was significantly reduced in workload ST-CLT-05. Furthermore, the reference rates of flag clips in ST-CLT-05 are smaller than that in ST-CLT-04. We find that in workload ST-CLT-05, several large media providers,
such as MSNBC, merged their flag and logo clips into their media content objects when the objects were authored. For example, in workload ST-CLT-04, a clip of MSN logo was played before every video object of MSNBC site, while in ST-CLT-05, there was no requests to this logo clip at all, because it had been merged into the media programs.

Figure 3.7 shows the percentages of three kinds of objects among all requests. Although extraneous media objects account for a non-trivial portion of all requests in the workload, they only account for a small percentage of media traffic volume, because ad and flag clips are usually small. In our cases, the extraneous traffic account for 31% and 13% of all media requests, but only account for 1.7% and 3.5% transferred bytes, in ST-CLT-04 and ST-CLT-05, respectively\(^8\).

**Caching effect**

Previous studies on Web workloads have also shown that the initial part of the reference rank distribution of objects can be lower than what a Zipf-like model predicts, which looks similar to our SE observation on media workloads. Williamson et al. [118] point out this

\(^8\)The object sizes of ad and flag clips of workload ST-CLT-05 are larger than those of workload ST-CLT-04.
is due to the caching effect: popular Web objects are likely to be cached by Web browsers or proxies, hence, subsequent requests may not reach the server. For Web traffic, usually caching only affects the initial part of the reference rank distribution (depending on the cache size), and the main body of the distribution curve is still a straight line in log-log scale. Similarly, as reported by Cherkasova et al. [37], for short term media workloads, the object reference rank distribution in log-log scale does not significantly deviate from a straight line except the initial part. Thus, it is important to figure out whether the stretched exponential distribution of media access patterns we observed is due to the caching effect of media content.

However, for streaming workloads in this study, the caching effect has been considered. Although a media player can cache media files that have been delivered and played, it still sends a report to the server when the object is re-played in the local cache, in order to let the server generate a log entry [18]. Similarly, both Windows and RealNetworks media proxies also send a report to the server when the cached content is requested, so that the server can collect its object access information and generate a log entry for the request [16, 9]. Thus, server log based workloads include all requests to media objects, even for cached objects. For client side workloads we collected, we have carefully extracted the RTSP/MMS commands for cache-validation and log statistics to include local replay events. Thus, from the perspective of measurement methodology, server side workloads HPC-98, HPLabs-99, ST-SVR-01, CTVoD-04 and client side workloads ST-CLT-04 and ST-CLT-05 reflect the real access patterns of media users. Therefore, caching is not likely to be the reason that causes the observed stretched exponential access patterns in these workloads.
Meanwhile, caching effect is also observed in long duration workloads such as IFILM-06. Workload IFILM-06 records the number of clicks on the page of each media object published on the IFILM Web site. Since the Web page of a media object can be cached by a Web browser or a proxy, a reloaded page may not be counted as a page click. Thus, as shown in Figure A.1(c) (see Appendix A), due to the accumulation of caching effect over time, the initial part (for the first 100 objects at most) of the reference rank distribution deviates from the SE model gradually, but the main body of distribution still follows the SE model. However, for short duration media workloads, the caching effect is trivial. For example, both the 7-day workload of IFILM-06 (Figure 3.2(c)) and the 9-day workload PS-CLT-04 (Figure 3.1(b), pseudo streaming via HTTP) are barely affected by client side caching.

“Fetch-at-most-once” effect

Similar to the caching effect, Gummadi et al. [52] find that P2P traffic is not Zipf-like and attribute this to the “fetch-at-most-once” effect: since typically media objects are immutable (the content of a media object does not change over time), P2P users download them at most once (this effect is equivalent to have an unlimited cache on each client). Assuming the playback activities of P2P users follow a Zipf-like distribution, the authors show that the object reference rank distribution of corresponding downloading activities (workload KaZaa-02) is very close to the real reference rank distribution of P2P workloads by simulations.

However, this effect is still unlikely to be the reason causing the observed deviations from the Zipf model for the access patterns in media workloads. As we have presented in Section 3.4.2, for streaming media workloads, the local replaying events due to caching are
recorded in server logs and have been carefully measured in our client side network measurements. (1) For streaming media workloads of small objects, such as ST-SVR-01, ST-CLT-04, and ST-CLT-05, there is no “fetch-at-most-once” effect, because caching has been considered. These workloads do not follow the Zipf-like distribution, but stretched exponential. Furthermore, as shown in Figure 3.8, for the MMS traces in workloads ST-CLT-04 and ST-CLT-05 (extraneous traffic excluded), it is not rare that an object is requested by the same user multiple times (multiple users shielded by NAT are identified with the technique in [61]). (2) Similarly, for server log based streaming media workloads of large video files, such as CTVoD-04, the “fetch-at-most-once” effect does not exist either. However, the access pattern of workload CTVoD-04 deviates from the Zipf model even more significantly than those small file workloads. Due to privacy concerns, there is no user ID information saved in server logs. Although “fetch-at-most-once” might be the case in P2P workloads, for large video objects, such as files in KaZaa-02, it is also reasonable to assume users

Figure 3.8: Number of requests to each object by the same user
will not repeatedly watch the same video, whether it is rented from a store or downloaded through P2P networks. Thus, it is very likely that user access patterns in P2P workloads of large video files reflect the real user playing activities quite well. This is also evidenced by the similar access patterns found in P2P downloading workload KaZaa-02 and VoD streaming workload CTVoD-04: both the median file sizes and stretch factors of these two workloads are very close. We will further analyze these two workloads and this similarity in Section 3.5.1.

Due to the above reasons and the fact that the stretched exponential distribution can fit both small and large sized media workloads as we have shown, “fetch-at-most-once” effect is unlikely to be the reason causing the SE distribution of media workloads.

3.4.3 Other media access pattern models

A number of models have been proposed to describe or explain the deviation of media access patterns from the Zipf-like distribution. Most of them are still based on the Zipf model. For example, Cha et al. [31] use the Zipf with an exponential cutoff effect to describe the thin tail of media popularity distributions. In the paper, this effect is explained as preferential attachment with information filtering [86]. The basic idea of preferential attachment mechanism is the “rich-get-richer” effect: If $k$ users have fetched an object, the rate of other users fetching it is proportional to $k$. This means that the popularity of a popular object tends to increase rather than to decrease over time with a high probability. For Web workloads, this argument is true. On the Internet, some pages, especially the first pages of Web portals and search engines, such as Yahoo!, Google, and MSN, can have a high popularity rank for a very long time. For example, according to Alexa Internet [69],
the daily traffic rank of Yahoo! keeps number one from 2001 (earlier data are unavailable) until November, 2007 (present).

Nevertheless, this argument is not valid for media objects. Web pages can be updated frequently to attract users and keep their popularity. In contrast, media objects are usually immutable. In a long term, a video object, no matter how popular it is, is unlikely to become more popular or keep its popularity rank with the passage of time and with the creation of new objects. On the contrary, many measurement studies have observed that the popularity of a media object becomes unpopular quickly [31, 52, 54]. For example, study [54] reports media popularity decreases with time exponentially. Thus, the “rich-get-richer” phenomenon reflected in Zipf and power law based models is not present in media objects.

We have compared the weekly top 1, top 2, ..., top 100 popular objects during sixteen weeks in a Web workload (collected by University of Calgary [26]) and a video workload (IFILM-06). Figure 3.9 shows the total number of distinct objects in the union set of sixteen weekly top $N$ objects ($1 \leq N \leq 100$), for the Web and video workloads, respectively. We
can see that the set of 16 weekly top $N$ popular Web objects is much smaller than that of 16 weekly top $N$ popular video objects. Particularly, the top one popular Web object never changes, but the top one popular video object changes every week. This indicates the number of references to a popular media object cannot be linearly accumulated with time. The absence of “rich-get-richer” effect can also be reflected by the evolution of media object popularity distributions in a long duration, which has been reported by study [37]. Study [117] shows that the reference rank distribution of Web objects is still Zipf-like even for workloads of one year. In the next section, we will show that although in a short duration, the popularity distribution of media objects in a system might be highly skewed and look Zipf-like, the skewness tends to weaken rather than strengthen over time, so that finally the distribution looks unlike Zipf at all.

Even considering the information filtering, the Zipf with exponential cutoff model can only explain the thin tail of the distribution curve (in log-log scale). In [31], the authors use the “fetch-at-most-once” effect to explain the fat head. Furthermore, it cannot explain why Gill et al. [49] observe a Zipf-like distribution. In our SE model, the difference between the observations of [31] and [49] can be explained as the different lengths of their workload durations (see Section 3.5.4).

We have also compared the stretched exponential model with other two-parameter media access pattern models, such as Zipf-Mandelbrot model [98] and parabolic fractal model [39]. We find these models can only fit a small number of workloads, while SE fits all (see Appendix C).
3.5 The Dynamics of Media Reference Rank Distributions

In this section, we analyze the reference rank distributions of media objects of different sizes in different durations, in order to further understand the dynamics of stretched exponential access patterns in different media systems. Our analysis shows: (1) parameter $c$ well characterizes the effect of media file sizes; (2) parameter $a$ well characterizes the non-stationarity effect of media access aging; (3) the deviation of media reference rank distribution from the Zipf model increases with the length of workload duration.

3.5.1 Access patterns of different sized media files

Figure 3.10 further shows the stretch factor $c$ of each on-demand media workload (Web, VoD, or P2P) with its median file size (also see Table 3.1). In this figure, each point represents a single workload. Roughly distributed along a straight line, these points can be classified into three groups: for workloads with a median file size $< 5$ MB, the stretch factor is $\leq 0.2$; for workloads with a median file size $> 100$ MB, the stretch factor is $\geq 0.3$; for
other workloads, the stretch factor is between 0.2 and 0.3. As shown in Table 3.1 (column 3, 9, and 10), in general, for media workloads delivered by similar systems and techniques, their stretch factors increase with their median file sizes. For example, the median file size of workload KaZaa-03 is 5 MB, and the median file size of workload KaZaa-02 is 300 MB. Their stretch factors are 0.14 and 0.45, respectively. For media workloads delivered by streaming techniques, the median file sizes of workloads IFILM-06, ST-CLT-05, HPLabs-99, and CTVoD-04 are 2.25 MB, 4.5 MB, 120 MB, and 300 MB, respectively, while their stretch factors in the SE model are 0.15, 0.2, 0.3, and 0.4, respectively.

Meanwhile, for workloads with similar median file sizes, in most cases, the corresponding stretch factors are also similar regardless of the underlying media systems and delivery techniques. For example, workload KaZaa-02 is delivered by P2P networks while CTVoD-04 is delivered by VoD servers. However, they have similar median file sizes (both about 300 MB) and similar stretch factors (0.45 and 0.4, respectively). Workload KaZaa-03 is delivered by P2P networks while IFILM-06 is delivered by a CDN. Their median file sizes are both less than 5 MB, and their stretch factors are 0.14 and 0.15, respectively. Workload PS-CLT-04 is delivered by downloading while ST-CLT-04 is delivered by streaming. Both of their median file sizes are about 2 MB, and both of their stretch factors are 0.2. These cases indicate that in general, despite the different techniques and systems used for media delivery, the larger the median file size of a workload, the greater the stretch factor of its SE reference rank distribution.

Our analysis indicates that the reference rank distribution of a media workload highly depends on the median file size of the workload rather than on the delivery method and the underlying media system. Although the relationship between the median file size and the stretch factor of a media workload shown in Figure 3.10 has not been strictly formulized,
the following factors may help us further understand the trend reflected in this figure. (1) It is unfair to compare files with similar sizes (in bytes) but with different encoding rates and compression ratios. The file length (in seconds) or user playback duration might be a more objective metric, but unfortunately we cannot use this metric because the relevant data are unavailable in some workloads. (2) Video and audio may have different access patterns and thus have different stretch factors. In our workloads, KaZaa-03 is MP3 music dominant, all Web media workloads have a combination of video and audio, and other workloads are video dominant. Due to the lack of related information in the workloads, we cannot separate video and audio in the analysis. (3) Different types of video content (e.g., entertainment, business, and educational content) may have different access patterns. In our study, HPLabs-99 is a business workload, where the content is hosted by an internal media server open for research employees in HP Labs only. Workload mMoD-98 is collected in an educational environment, where the content is a combination of video classes and movies. Other workloads are all entertainment content dominant. HPLabs-99 and mMoD-98 have similar median file sizes but the values of parameter $c$ are 0.3 and 0.55, respectively. As shown in Figure 3.10 (“EDU” for mMoD-98 and “BIZ” for HPLabs-99), both workloads deviate from the straight line greater than other workloads.

3.5.2 Non-stationarity of media reference rank distributions

Previous studies have found that long term media workloads and short term media workloads have different access patterns [37, 108]. In contrast, an analysis of long-term Web workloads shows that the access pattern is still Zipf-like after several months to one year [117]. In this section, we study the stationarity of media reference rank distributions, and show how media access aging affects parameter $a$, the minus of the slope of SE plot.
To characterize the dynamics of a media system, we consider the birth rate of new objects (denoted as \( \lambda_{\text{obj}} \)) and the request rate of all objects (denoted as \( \lambda_{\text{req}} \)) first. We have analyzed media accesses over time for long duration workloads ST-SVR-01, BT-03, and IFILM-06 as well as short duration workloads PS-CLT-04, ST-CLT-04, and ST-CLT-05 (other workloads have no timestamps or temporal information of requests). In this section we focus on long duration workloads in order to study the trend of media access evolution. For workloads BT-03 and IFILM-06, we find that the cumulative number of requests increases with time linearly in a coarse time granularity, indicating that the media request rates in corresponding systems are roughly constants. For workload ST-SVR-01, as shown in the curve of right \( y \) axis of Figure 3.11, the media request rate is almost a constant in the first three months, and then suddenly increases at the beginning of the fourth month due to a system upgrade, but still approximates to a constant.

The curve on the left \( y \) axis of Figure 3.11 shows the cumulative number of objects requested over time for workload ST-SVR-01. The figure shows that the number of objects requested in the workload increases quickly at the beginning, then the increase slows down.
and converges to a linear function asymptotically. This effect can be explained as follows. The initial non-linear increase of the cumulative number of requested objects is due to objects that were created and had been requested before the trace collection. However, the number of requests to these pre-existing objects decreases quickly with time, and the cumulative number of requested objects will be dominated by the objects born after trace collection after a certain duration of time. Thus, the linear part of the curve corresponds to the birth of new objects requested in this workload, indicating that the object birth rate is a constant in a coarse time granularity. We have similar observations on workloads BT-03 and IFILM-06.

The constant media request rates and object birth rates observed in these workloads indicate that these media systems evolve homogeneously over time during the trace collection period. We have analyzed the reference rank distributions of media objects accessed in different weeks and different number of weeks in these three workloads. All of them can be well fitted with the stretched exponential distribution (with $R^2 > 0.95$ for most durations and $R^2 > 0.93$ for all durations). Furthermore, in each workload, we find that parameter $c$ is a constant for different weeks and different number of weeks of that workload. According to its correlation with the median file size in the workload as presented in Section 3.5.1, the time-invariant property of stretch factor $c$ indicates that the median file size of a workload is a constant along time, which is also confirmed by our validation.

Figure 3.12 shows the reference rank distributions of BT-03 in different number of weeks. We can see that the minus of the slope of the fitted line, i.e., the parameter $a$ of the corresponding SE distribution, increases with time gradually. Figure 3.13 further shows the evolution of parameter $a$ over time in these three workloads. This evolution corresponds to the observation in the study of Cherkasova et al. (with workloads HPC-98 and HPLabs-99)
[37]: for monthly workloads, the access pattern looks like a Zipf-like distribution, while for workloads longer than 6 months, it does not. Actually, even for monthly workloads, the Zipf-like fitting of reference rank distribution is quite rough, as shown in the figures of [37]. In contrast, the media reference ranks of a workload in different durations can be well fitted with SE distributions with the same stretch factor $c$ and a different parameter $a$.

Now let us consider a homogeneously evolving media system with a constant media request rate $\lambda_{\text{req}}$ and a constant object birth rate $\lambda_{\text{obj}}$. In a coarse time granularity, this is a reasonable assumption for many systems, such as the three workloads discussed above. According to Equation 3.5, $\langle y_{\text{se}} \rangle$ increases with time since $c$ is a constant and $a$ increases with time (denoted as $a(t)$ in the follows).

The cumulative number of requested objects $N(t)$ in the time duration $[0, t)$ is

$$N(t) = \lambda_{\text{obj}} t + N'(t), \quad (3.6)$$

where $\lambda_{\text{obj}} t$ is the number of requested objects born in time $[0, t)$. $N'(t)$ is the number of “old” objects that are born before $t = 0$ and requested in time $[0, t)$. Denote the cumulative number of requests to object $i$ in the workload as $y_i(t)$ ($1 \leq i \leq N(t)$). Assuming the mean value of the cumulative requests to objects in time $[0, t)$ as $\langle y(t) \rangle$, we have

$$\langle y(t) \rangle = \frac{\lambda_{\text{req}} t}{N(t)} = \frac{\lambda_{\text{req}}}{\lambda_{\text{obj}}} \frac{1}{1 + \frac{N'(t)}{\lambda_{\text{obj}} t}}. \quad (3.7)$$

Intuitively, $\lim_{t \to \infty} \frac{N'(t)}{\lambda_{\text{obj}} t} = 0$, since the number of old objects will be requested less frequently with the passage of time. Assuming the popularity of a media object decreases exponentially with time [54], we have $N'(t) = O(\log t)$ (see Appendix D for a brief proof).

Using Equation 3.4 as an approximation of $\langle y(t) \rangle$, we have

$$a(t) = \left[ \frac{\langle y(t) \rangle}{\Gamma\left(1 + \frac{1}{c}\right)} \right]^c = \left[ \frac{1}{\Gamma\left(1 + \frac{1}{c}\right)\left(1 + \frac{N'(t)}{\lambda_{\text{obj}} t}\right)} \right]^{e} \lambda_{\text{req}}^{c}. \quad (3.8)$$
Thus, if $\lambda_{req}$, $\lambda_{obj}$, and $c$ are all constants, we have

$$
\lim_{t \to \infty} a(t) = \left[ \frac{1}{\Gamma(1 + \frac{1}{c})} \right] \frac{\lambda_{req}}{\lambda_{obj}}^c.
$$

(3.9)

In all workloads, the stretch factor $c$ is less than 2 ($c < 1$ in most cases). When $0 < c \leq 2$, $\frac{1}{\Gamma(1 + \frac{1}{c})}$ increases with $c$. Thus, for the reference rank distribution of a media workload, $a$ increases with stretch factor $c$, the ratio of media request rate to object birth rate, $\frac{\lambda_{req}}{\lambda_{obj}}$, and the duration of workload collection time $t$.

As shown in Figure 3.13(a) and 3.13(b), the increase of $a$ in workloads BT-03 and IFILM-06 slows down with time. For workload ST-SVR-01, as shown in 3.13(a), the increase of $a$ slows down in the first three months, then speeds up at the beginning of the fourth month suddenly, and then slows down again. The sudden increase of $a$ in Figure 3.13(b) is caused by the sudden increase of $\lambda_{req}$ due to a system upgrade, which means $a$ may not converge to a constant in practice.
For short duration workloads PS-CLT-04, ST-CLT-04, and ST-CLT-05, we compute the distribution parameters for workloads of different time intervals. We find that both parameter $c$ and parameter $a$ are constant. The reason is as follows. As shown in Figures 3.20(a), 3.20(b), and 3.20(c), throughout the workload duration, $N'(t)$ increases with time almost linearly. According to Equation 3.8, this means the parameter $a$ is a constant during this time. Thus, the popularity distribution is stationary for short duration workloads.

In summary, for a media system with homogeneous evolution, the media object reference rank distribution is non-stationary, which evolves over time with an increasing parameter $a$ and a constant, time-invariant stretch factor $c$. This evolution is due to the effect of objects born before the workload collection and/or objects pre-existing in the system, which has significant impact on the caching performance of media systems. We will further study this effect in Section 3.6.3.

3.5.3 Deviation of media access patterns from the Zipf model

In Section 3.4, we have shown that the distribution of media reference ranks has a fat head and a thin tail in log-log scale, deviating from the Zipf model. In order to quantitatively measure this deviation, Figure 3.14(a) shows a general stretched exponential distribution curve in log-log scale. In this figure, chord $AB$ on the SE curve corresponds to the Zipf-like distribution (see Equation 3.14 of Section 3.6.1) with the same number of objects and the same number of references to the most popular object as those in the SE distribution. $CD$ is parallel to $AB$ and tangent to the SE curve at point $(X_0, Y_0)$. If we use $|OE|$ to represent the distance from the original point to the chord $AB$ and $|EF|$ to represent the distance between $AB$ and $CD$, $|EF|/|OE|$ reflects the difference between the SE distribution and the corresponding Zipf-like distribution in log-log scale.
In Equation 3.2, let \( X = \log i, Y = \log y_i \), we have

\[
Y = \frac{1}{c} \log(b - aX),
\]

(3.10)

where \( 0 \leq X \leq \log N, 0 \leq Y \leq \frac{1}{c} \log b \).

Since the slope of chord \( AB \) is \( k = -\frac{Y_{\text{max}}}{X_{\text{max}}} = -\frac{\log b}{c \log N} \), chord \( AB \) can be expressed as

\[
Y = kX + Y_{\text{max}}.
\]

(3.11)

Similarly, tangent \( CD \) can be expressed as

\[
Y = kX + Y_0 - kX_0,
\]

(3.12)

where \( X_0 = \frac{1}{a}(b + \frac{a}{ck}), Y_0 = \frac{1}{c} \log(-\frac{a}{ck}) \). Thus,

\[
\frac{|EF|}{|OE|} = \frac{|AC|}{|OA|} = \frac{Y_0 - kX_0 - Y_{\text{max}}}{Y_{\text{max}}} \frac{Y_{\text{max}}}{Y_{\text{max}}} = \frac{Y_0 - kX_0}{Y_{\text{max}}} - 1 = \frac{1}{a \log N} \left[ \frac{1}{\log(1 + a \log N)} - 1 \right] + \frac{1}{a \log N}.
\]

(3.13)
We have $\frac{|EF|}{|OE|}$ increases with $a \log N$. Further, we have $\frac{|EF|}{|OE|} \to 0$ when $a \log N \to 0$ and $\frac{|EF|}{|OE|} \to 1$ when $a \log N \to \infty$. Thus, in log-log scale, the difference between the media reference rank distribution and the Zipf model increases with $a \log N$. Figure 3.14(b) shows $\frac{|EF|}{|OE|}$ with different values of $a$ and $N$.

For a homogeneously evolving media system, $a$ evolves along time until it approaches to a constant (Equations 3.8 and 3.9), and $N$ increases with time linearly (Equation 3.6). Thus, in log-log scale, the deviation of media reference rank distribution from the Zipf model increases with time, causing the “fat head” and the “thin tail” of the distribution curve.

We have compared this deviation for different periods of long duration workloads. For example, for the first week of workload ST-SVR-01, where $a = 0.423$ and $N = 459$, we have $\frac{|EF|}{|OE|} = 0.16$. In contrast, for the entire 4 months of workload ST-SVR-01, where $a = 0.738$ and $N = 2260$, we have $\frac{|EF|}{|OE|} = 0.23$. This evolution is consistent with the observation in the study of Cherkasova et al. [37] (with workloads HPC-98 and HPLabs-99), as we have presented in Section 3.5.2.

Since term $a(t)$ (Equation 3.8) and term $\log N(t)$ increase with time $t$ slowly, it may take a long time to observe a significant deviation between the media popularity distribution and the Zipf model in log-log plot, especially for workloads with small media files, such as Web media workloads in Figure 3.1. However, this does not mean that the Zipf model is a good approximation of media access patterns for workloads with short durations. Since log scale compresses data differences significantly, a small difference in the Zipf model may correspond to a significantly larger difference in the SE model. As shown in Figures 3.1(b) (PS-CLT-04), 3.1(c) (ST-CLT-04), and 3.1(d) (ST-CLT-05), the deviations of these short duration workloads (9–11 days) are non-trivial, though not significant. For
the estimation of system related quantities, such as hit ratios of media caching, using the Zipf model may cause significant errors. We will further study this issue in Section 3.6.1.

Even for short duration workloads, the deviation can still be significant. According to Equation 3.8, parameter $a$ depends on the stretch factor $c$, ratio of $\frac{\lambda_{req}}{\lambda_{obj}}$, and workload duration time $t$. For workloads with large files, usually both the stretch factor $c$ and the ratio of $\frac{\lambda_{req}}{\lambda_{obj}}$ are large. We have presented the correlation between stretch factor $c$ and file sizes in Section 3.5.1. The latter is large because large media files, such as movies, tend to have a high request rate and a low production rate. For example, as shown in Figure 3.13, the parameter $a$ of one-week workload of BT-03 (median file size 636 MB, $c = 0.52$, $\frac{\lambda_{req}}{\lambda_{obj}} = 141.1$) is even much larger than that of the entire duration (4 months) of workload ST-SVR-01 (median file size 15 MB, $c = 0.2$, $\frac{\lambda_{req}}{\lambda_{obj}} = 99.6$), and the difference between its reference rank distribution and the corresponding Zipf-like distribution is significant. For workloads with large files and long durations, the deviation is more significant. As shown in Figures 3.2(a), 3.2(b), 3.3(a), and 3.3(c), the log-log plots of media reference rank distributions of workloads mMoD-98 (median file size 125 MB), CTVoD-04 (median file size 300 MB), KaZaa-02 (median file size 300 MB), and BT-03 (median file size 636 MB), deviate from a straight line remarkably.

3.5.4 Summary

In this section, we have analyzed the dynamics of object reference rank distributions for media systems with homogeneous evolution. We have shown that file size is a key factor affecting media access patterns, and can be well characterized by stretch factor $c$ of the stretched exponential reference rank distribution. For media systems with constant media request rate $\lambda_{req}$ to object birth rate $\lambda_{obj}$, we have also shown that the access pattern
is non-stationary in a long duration: the minus of the slope of the stretched exponential distribution, $a$, depends on stretch factor $c$, the ratio of $\frac{\lambda_{req}}{\lambda_{obj}}$, and the evolution effect due to the aging of media objects. In general, the media access pattern deviates from the Zipf model with the increase of $a \log N$, where $N$ is the number of objects in the system. The file size plays a more important role in affecting this deviation than the workload duration $t$ and the number of objects in the workload, unless $t$ is enormously long.

To summarize our analysis with examples of real systems, we explain the different media access pattern observations of KaZaa networks and YouTube videos as follows. For KaZaa media traffic, Gummadi et al. [52] report that the access pattern of media workload in KaZaa system collected in University of Washington campus network is not Zipf-like. However, Iamnitchi et al. [65] report that it is Zipf-like for KaZaa traffic collected from the network of an ISP in Israel. In Section 3.4, we have replotted the object reference rank distributions of these two workloads, with both log-log scale and SE scale, as shown
in Figures 3.3(a) (workload KaZaa-02, University of Washington) and 3.3(b) (workload KaZaa-03, Israel ISP). Comparing these two figures, we find the access pattern of workload KaZaa-02 deviates from the Zipf model much greater than that of workload KaZaa-03. Actually, in paper [65], workload KaZaa-03 is considered as Zipf-like. Investigating the workload collection methodology of these two studies, we find the duration of workload KaZaa-02 is 203 days, while the duration of workload KaZaa-03 is only 5 days. Furthermore, in workload KaZaa-02, the files smaller than 100 MB were specifically removed from the trace (see paper [52], Section 2.3.3). Thus, the median file size of workload KaZaa-02 is about 300 MB, much greater than 5 MB, the media file size of workload KaZaa-03 (for all KaZaa files). According to our analysis in Sections 3.5.2 and 3.5.3, the different media access pattern observations of KaZaa networks in studies [52] and [65] can be explained.

For YouTube video traffic, Gill et al. [49] collected an 85-day workload of YouTube videos in the campus network of University of Calgary, and found the object popularity distribution is Zipf-like. Cha et al. [31] crawled the meta information of YouTube videos by crawling the index pages on the YouTube Web site (this method is very close to the methodology of our YouTube workload collection). However, they found the popularity distribution of YouTube videos have a significant deviation from the Zipf model. This observation is close to our results. We have also fitted their workload with the stretched exponential distribution, and got similar results. In order to show this difference clearly, we reference the corresponding rank distribution figures of these two papers in Figures 3.15(a) and 3.15(b), respectively. Investigating the collection methodologies of these YouTube workloads, we find that the duration of Gill’s workload is 85 days, while in Cha’s workload and our workload, the durations are much longer: these two workloads were collected with Web crawling on the index pages of YouTube Web site, where the total number of
downloads of each video clip for the entire up time of YouTube is published. For these workloads, the $N'(t)$ in Equation 3.6 should be zero, because there is no “old objects” created before workload collection. Thus, it is not difficult to understand why these Gill’s study and Cha’s study have totally different observations.

Finally, we realize that our analysis above is mainly for “ideal” media systems with simplified assumptions, such as stable file sizes for media objects introduced in the system among time, constant media request rate, and constant object birth rate. For real systems, some assumptions may not be valid and the system evolution may not be homogeneous. For example, the media request rate and object birth rate of YouTube videos have increased significantly with time. In some video systems, the introduction of new content may be bursty and irregular. For example, for workload CTVoD-04, most videos were introduced in the first day of system launch, and new contents were introduced into the system quite randomly. Although the workload is still stretched exponential, the parameter analysis in this section may not be applied. Furthermore, due to the lack of related information, the requests from different unique users have not been analyzed in details. We also realize that other factors, such as the recommendation list by video sites and the reference links among flash videos such as YouTube, may affect the access patterns of these media objects, though we have no enough data to quantitively analyze these effects.

3.6 Implications of Stretched Exponential Access Patterns

The Zipf-like reference rank distribution of Web objects and the widely deployed Web proxies have demonstrated the significance of access patterns on the performance of content delivery systems. Previous studies such as [30] by Breslau et al. have analyzed the

\footnote{Unlike traditional Internet videos, current flash videos support reference links to other similar or related flash videos, in order to recommend and redirect users to these videos.}
Web access distribution and its implication for caching. In this section, we explore the implication of stretched exponential access patterns on media caching, from the aspect of the asymptotic properties of workload size and cache size.

### 3.6.1 Caching implications of different reference rank models

As we have shown in Section 3.5.2, the media reference rank distribution is **stationary** for short duration workloads. If we assume requests in a media workload are **independent** with each other, the temporal locality in the workload can be analyzed with the reference rank model of objects.

**Caching with the Zipf model**

A Zipf-like distribution can be described as

\[
y_i = \frac{A}{i^\alpha},
\]

where \(y_i\) is the number of references to the \(i\)-th popular object (\(1 \leq i \leq N\), \(N\) is the total number of objects in the workload), \(\alpha\) (the skewness factor) is a constant that characterizes the shape of the distribution, and \(A\) is a normalization factor. Thus we have

\[
\log y_i = \log A - \alpha \log i,
\]

which means the distribution function is a straight line in the log-log scale. Assuming \(y_N = 1\) when \(N\) is large enough (the \(N\)-th object gets only one access), we have \(A = N^\alpha\).

When \(\alpha < 1\) and \(N \to \infty\), the mean value of a Zipf-like distribution is

\[
\lim_{N \to \infty} \langle y_{zf} \rangle = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} y_i = \lim_{N \to \infty} N^{\alpha-1} \sum_{i=1}^{N} i^{-\alpha} = \frac{1}{1-\alpha}. \tag{3.16}
\]
With an infinite cache size, cache misses could only come from compulsory misses, thus we get the optimal hit ratio

\[
H_{zf}^{opt}(\alpha) = \lim_{N \to \infty} \frac{N \langle y_{zf} \rangle - N}{N \langle y_{zf} \rangle} = 1 - \frac{1}{\langle y_{zf} \rangle} = \alpha. \tag{3.17}
\]

Clearly, this gives an upper bound when all objects are considered cacheable.

When \( \alpha \geq 1 \), the optimal hit ratio \(\rightarrow 1\) when \(N \to \infty\) since \(\langle y \rangle\) is divergent (proof omitted). For most Web workloads, \(\alpha\) ranges between 0.6 and 0.8 [30]. The above result is consistent with the observation in [30] of the failure of “10/90” rule: the hit ratio of a Web proxy cannot be greater than 90%. In the remainder of this section, we only consider Zipf-like distributions with \(\alpha < 1\).

Now considering a more realistic scenario where the cache size is limited. We study the cache performance when the cache size is proportional to the size of all accessed objects. Assume each object occupies one unit of storage volume and the cache size is \(\eta N\), where \(\eta \leq 1\) and is a constant. With an optimal cache replacement, when \(\alpha < 1\), the number of total cache hits is

\[
\sum_{i=1}^{\eta N} (y_i - 1) \approx \int_1^{\eta N} \left(\frac{N}{x}\right)^\alpha dx - \eta N \\
\approx \left(\frac{\eta^{1-\alpha}}{1-\alpha} - \eta\right)N. \tag{3.18}
\]

Thus, the optimal hit ratio of a workload with Zipf-like distribution is \(^{10}\)

\[
H_{zf}^{opt}(\eta) = \lim_{N \to \infty} \frac{1}{N \langle y_{zf} \rangle} \left(\frac{\eta^{1-\alpha}}{1-\alpha} - \eta\right)N \\
= \frac{\eta^{1-\alpha} - \eta(1 - \alpha)}{1-\alpha}. \tag{3.19}
\]

According to the above derivation, we can estimate the optimal hit ratio. For example, when \(\alpha = 0.8\), caching 25% of all requested data can achieve hit ratio 0.7. This is consistent with the Web caching hit ratios reported in [30].

\(^{10}\)When \(\alpha \geq 1\), \(\langle y \rangle\) is divergent. So we have \(H_{zf}^{opt}(\eta) = 1\).
Caching with the stretched exponential model

Similar to Equation 3.17, since a stretched exponential distribution has a limited mean value (Equation 3.5), for an infinite cache size, the optimal hit ratio is

\[ H_{se}^{opt} = \lim_{N \to \infty} \frac{\langle y_{se} \rangle - N}{N\langle y_{se} \rangle} = 1 - \frac{1}{\langle y_{se} \rangle}. \]  

(3.20)

Assume each object occupies one unit of storage volume and the cache size is \( k = \eta N, \eta \leq 1 \) and is a constant. The optimal hit ratio of SE distribution is

\[ H_{se}^{opt}(\eta) = \lim_{N \to \infty} \frac{\text{hits}}{N\langle y_{se} \rangle} = \lim_{N \to \infty} \frac{1}{N\langle y_{se} \rangle} \sum_{i=1}^{k} (y_i - 1) \]

\[ = \lim_{N \to \infty} \left[ \frac{1}{\langle y_{se} \rangle} \int_{\frac{1}{\langle y_{se} \rangle}}^{\frac{k}{\langle y_{se} \rangle}} (1 - a \log x)^{\frac{a}{c}} \, dx - \frac{k}{N\langle y_{se} \rangle} \right] \]

\[ = \frac{\Gamma(1 + a - \gamma(1 + \frac{1}{c} - \log \eta))}{\Gamma(1 + \frac{1}{c} - \gamma(1 + \frac{1}{c} - \log \eta))} - \frac{\eta}{\langle y_{se} \rangle}. \]  

(3.21)

Figure 3.16(a) shows an exemplified comparison of optimal hit ratios between the Zipf-like and stretched exponential model when \( \eta \) changes. The parameters are selected based
on typical client side Web workloads \((\alpha = 0.8)\) and media workloads \((c = 0.2, a = 0.25,\) same as those in ST-CLT-05). Both distributions have the same hit ratio with an unlimited size of cache. The duration of workload ST-CLT-05 is 11 days, comparable to those of client side Web workloads in Web caching studies such as Breslau et al. [30]. Thus, the comparison is fair. From Figure 3.16(a), it is clear that the caching efficiency of workloads under an SE model is much worse than that under a Zipf-like model: caching 1% Web content can achieve about 40% hit ratio, while caching 1% media content can only achieve 18% hit ratio\(^\text{11}\).

In order to compare the asymptotic caching performance of these two models with a small \(\eta\), we consider the case where the cache size \(k\) is a constant and \(k << N\).

For a Zipf-like distribution with \(\alpha < 1\), we have

\[
H^{\text{opt}}_{zf}(\frac{k}{N}) = \frac{\sum_{i=1}^{k} (y_i - 1)}{\sum_{i=1}^{N} y_i} = \frac{\sum_{i=1}^{k} \left(\frac{y_i}{N}\right)^{\alpha} - k}{\sum_{i=1}^{N} \left(\frac{y_i}{N}\right)^{\alpha}}.
\]

For a stretched exponential distribution, we have

\[
H^{\text{opt}}_{se}(\frac{k}{N}) = \frac{\sum_{i=1}^{k} (y_i - 1)}{\sum_{i=1}^{N} y_i} = \frac{\sum_{i=1}^{k} \left(1 + a \log N\right)^{1 - \gamma} - k}{\sum_{i=1}^{N} \left(1 + a \log N\right)^{1 - \gamma}}.
\]

When \(N \to \infty\), we have

\[
\lim_{N \to \infty} \frac{H^{\text{opt}}_{se}(\frac{k}{N})}{H^{\text{opt}}_{zf}(\frac{k}{N})} = \lim_{N \to \infty} \frac{\langle y_{zf} \rangle}{\langle y_{se} \rangle} \sum_{i=1}^{k} \frac{1}{i^\gamma} N^{1-\alpha} = 0.
\]

Since \(\langle y_{zf} \rangle\) and \(\langle y_{se} \rangle\) are independent of \(k\), this equation means the cache efficiency of a workload following stretched exponential model is asymptotically lower than that of a workload following Zipf-like model when \(\eta\) is small \((k << N)\).

Equation 3.24 is an asymptotic analysis for systems with a very small cache size. Figure 3.16(a) is a comparison between a typical client side, short term Web workload and a

\(^{11}\)The actual hit ratio of this media workload is only 0.6 for unlimited caching. This is because the workload is not large enough so that the tail is well fitted: many objects are accessed only once.
In summary, the analysis above indicates that media caching under a stretched exponential model is far less effective than Web caching under a Zipf-like model, especially when only a small fraction of requested objects can be cached. Thus, in many previous studies advocating Web media caching where a Zipf-like model is assumed, it is highly likely that the caching benefit has been overestimated.

3.6.2 Performance analysis of segment-based streaming media caching

As streaming media objects are generally large and are often partially accessed, segment-based caching has been proposed and applied to better utilize the cache storage.
In this subsection, we study the efficiency of segment-based caching for streaming media workload, in which the reference rank distribution of objects is stretched exponential.

Assume the reference rank distribution of media objects follows the stretched exponential distribution with parameter $c$ and $a$, as shown in Figure 3.17(a). If all media objects have the same length, which can be divided into $m$ segments, and no partial access to these objects, then the reference rank distribution of media segments would have two modes: for the first $m$ segments, i.e., segments of the most popular object, they have the same (highest) number of references; for remaining segments (other objects), the reference ranks and the numbers of references are in a straight line with slope $-a$ under the same SE scale as its object reference rank distribution. Shown in Figure 3.17(b).

We consider streaming media workloads St-SVR-01, ST-CLT-04, and ST-CLT-05 for this analysis. We evenly divide each media object into segments of 5-second playback length, which is the default setting of the play-out buffer size of Windows media player [6].
The same analysis is also conducted with byte-based segmentation, which shows similar results.

Figure 3.18(a), 3.18(b), and 3.18(c) show the distribution of segment reference rank in workloads ST-SVR-01, ST-CLT-04, and ST-CLT-05, respectively. Compared with Figure 3.1(a), 3.1(c), and 3.1(d), we can see that the segment reference rank follows a two-mode stretched exponential distribution with the same $c$ as that in the corresponding object reference rank. As shown in these figures, the minus of the slope of the first mode is much smaller than that of the second mode, but not in a horizontal line. Counting the number of segments in the first mode, we find they are quite close for all three workloads: 193 for ST-SVR-01, 149 for ST-CLT-04, and 199 for ST-CLT-05. Furthermore, although the minus of the slope of the second mode is much higher than that of the first mode, it is still much smaller than that of corresponding object reference rank distribution (the parameter $a$ is smaller). This means, after segmentation, the segments of the top popular objects account for only a smaller percentage of segment requests. Thus, the segment hit ratio of
the workload, which is close to the real byte hit ratio, should be smaller than that of the corresponding object hit ratio.

The shape of each mode in the segment rank distribution depends on the corresponding object rank distribution, the file length distribution, and how each object is accessed. To gain an overall estimate of the performance of segment-based caching, we assume the number of segments in the first mode is a constant $k$ about 200, and $k << N$ ($N$ is total number of segments for all objects in the workload). Assume the distribution equations of the first mode and the second mode are $y_i^{(1)} = (-a_1 \log i + 1 + a_1 \log N_1)^{\frac{1}{c}}, 1 \leq i \leq k$ and $y_i^{(2)} = (-a_2 \log i + 1 + a_2 \log N)^{\frac{1}{c}}, k < i \leq N$, respectively ($a_1 < a_2$ and $N_1 > N$).

Assume the cache size is large enough to hold all segments in the first mode, denoted as $\eta N (\frac{k}{N} \leq \eta \leq 1)$. So the number of request hits is

$$\text{hits} = \sum_{i=1}^{\eta N} (y_i^{(2)} - 1) - \sum_{j=1}^{k} (y_j^{(2)} - y_j^{(1)}), \quad (3.25)$$

and the total number of requests is

$$\text{reqs} = N \langle y_{se}^{(2)} \rangle - \sum_{j=1}^{k} (y_j^{(2)} - y_j^{(1)}). \quad (3.26)$$

Thus the optimal segment hit ratio is

$$H_{seg}^{opt}(\eta) = \frac{H_{se}^{opt2}(\eta) - h_0}{1 - h_0} < H_{se}^{opt2}(\eta), \quad (3.27)$$

where $H_{se}^{opt2}(\eta)$ is the optimal hit ratio of the second mode stretched exponential, and $h_0 = \frac{\sum_{j=1}^{k} (y_j^{(2)} - y_j^{(1)})}{N \langle y_{se}^{(2)} \rangle}$. We have

$$h_0 = \frac{\sum_{j=1}^{k} (y_j^{(2)} - y_j^{(1)})}{N \langle y_{se}^{(2)} \rangle} < \frac{k(1 + a_2 \log N)^{\frac{1}{c}}}{N \langle y_{se}^{(2)} \rangle}, \quad (3.28)$$

and thus $\lim_{N \to \infty} h_0 = 0$.  

99
Figure 3.19: The stretched exponential implications on streaming media caching performance

Figure 3.19(a) and 3.19(b) show the optimal object hit ratio, optimal segment hit ratio, segment hit ratio under the Least Recently Used replacement policy (LRU) and perfect Least Frequently Used replacement policy (LFU), for workloads ST-CLT-04 and ST-CLT-05, respectively. With perfect LFU, all access information can be kept. For optimal hit ratios, when cache size is large enough, the hit ratio reaches a peak value and then remains stable. The reason is that many objects/segments in the workload are rarely accessed, and caching them does not help improve the cache hit ratio. As we expected, the optimal segment hit ratio of streaming media is much lower than corresponding optimal object hit ratio. For example, the maximal segment hit ratio (unlimited cache) is only 0.342 for workload ST-CLT-05, although the object hit ratio can be up to 0.6.

Figure 3.19(a) and 3.19(b) also show the gap between the optimal segment hit ratio and segment hit ratios under LRU and LFU, as they (and their variants) are commonly used in practice for replacement. When the normalized cache size is 0.2, the optimal hit ratio has
reached the peak value 0.342 while the LRU hit ratio is only 0.281 for workload ST-CLT-05. LFU achieves even worse performance, because it cannot capture the sequential access order of segments in an object as LRU.

We further construct a simplified model to quantitatively compare the performance of LRU under stretched exponential with that under Zipf-like. For a workload following a reference rank distribution with a finite mean \( \langle y \rangle \), assume there are a total of \( M \) requests in the workload, so the number of objects requested in the workload is \( N = \frac{M}{\langle y \rangle} \). Assume the probability that object \( i \) gets requested is \( p_i \) and requests to media objects are independent from each other. So on average there is one request to object \( i \) among \( 1/p_i \) consecutive requests. In order to have object \( i \) cached, the cache size must be greater than the aggregate size of distinct objects requested during this time period. Since \( p_i = \frac{y_i}{N\langle y \rangle} \), we have the cache size to keep object \( i \) is \( \frac{1}{p_i} = \frac{N}{y_i} \), or the normalized cache size (in fraction of all requested data)

\[
\eta^{LRU}_i = \frac{1}{y_i}. \tag{3.29}
\]

For a Zipf-like distribution with skew factor \( \alpha < 1 \), we have

\[
\eta^{LRU}_{zf}(i) = \frac{1}{(\frac{y_i}{N})^\alpha} = (\frac{i}{N})^\alpha = [\eta^{opt}_{zf}(i)]^\alpha, \tag{3.30}
\]

where \( \eta^{opt} \) is the minimal cache size to hold object \( i \) (under optimal cache replacement policy).

Similarly, for a stretched exponential distribution with parameter \( a \) and \( c \), we have

\[
\eta^{LRU}_{se}(i) = \frac{1}{[1 - a \log(\frac{i}{N})]^\frac{1}{c}} = \frac{1}{[1 - a \log \eta^{opt}_{se}(i)]^\frac{1}{c}}. \tag{3.31}
\]

In reality, a caching system can have only limited storage space, i.e., \( i << N \) and \( \eta \) is small. Thus, we have

\[
\lim_{N \to \infty} \frac{\eta^{LRU}_{zf}(i)}{\eta^{LRU}_{se}(i)} = \lim_{N \to \infty} \frac{[1 - a \log(\frac{i}{N})]^{\frac{1}{c}}}{(\frac{y_i}{N})^\alpha} = 0. \tag{3.32}
\]
Equation 3.32 indicates that the performance of LRU under a stretched exponential distribution is asymptotically smaller than that under a Zipf-like model with a small cache size. Figure 3.19(c) compares the cache size demand for LRU to achieve the same hit ratio as that of the optimal policy under Zipf-like and stretched exponential models. In this figure, the skew factor $\alpha$ of a Zipf-like model is set to 0.8. The parameters of the stretched exponential model are set based on both the first and the second mode of segment rank distribution of workload ST-CLT-04. Note that the x-axis is $\eta^{opt}$, representing the normalized size of an optimal cache, and the y-axis is $\eta^{LRU} - \eta^{opt}$, representing the additional normalized cache space required for LRU to achieve the same hit ratio as optimal policy with a normalized size of $\eta^{opt}$. Thus y-axis denotes the efficiency of LRU replacement policy for workloads following a Zipf-like or stretched exponential model. As the modeling results in the figure shows, for a small cache size, the efficiency of LRU under a stretched exponential distribution is much lower than that under a Zipf-like distribution. The reason is as follows. The popularity differentiation among popular objects in a stretched exponential model is much smaller than that in a Zipf-like model. During the two successive requests to object $i$, the number of objects requested in a stretched exponential model is larger than that in a Zipf-like model. Hence, object $i$ always gets replaced out of the cache before it is requested again.

Segment-based caching is more efficient in storage usage. Our analysis shows that the hit ratio of segment-based caching, which is close to the corresponding byte hit ratio, is even poorer than that of object-based caching in the analysis Section 3.6.1. Furthermore, LRU replacement is less efficient in improving the caching performance of media workloads (stretched exponential) than that of Web workloads (Zipf-like).
3.6.3 The evolution of media caching performance

As presented in Section 3.5.2, in a homogeneously evolving media system, due to the diminishing accesses to old objects, the average number of requests per object in a media workload gradually increases with time until approaching to a constant. With a higher \(a\) and a constant \(c\) over time, the request concentration of media requests increases, and the performance of media caching can be improved. Now we study how long it takes and how much storage is required for such evolution.

We have extracted the Last-Modified field in the RTSP/HTTP headers of media requests in workloads PS-CLT-04, ST-CLT-04, and ST-CLT-05. We use this field to approximate the birth time of a requested object. Figure 3.20(a), 3.20(b), and 3.20(c) show the cumulative number of objects requested over time in the three workloads. We plot objects born before workload collection (old objects) and object born after workload collection (new objects) separately. In all three workloads, most objects requested in the workload are old objects pre-existing before workload collection, and the numbers of old and new objects both increase with time linearly. Figure 3.20(d) shows the CDF of the ages of old objects when they were requested for the first time in these three workloads. We can see that more than 70% of old objects are at most 500 days old. This figure indicates that the number of old objects will not linearly increase with time forever, and new objects will catch up with and overwhelm old objects after a long time.

Assume all objects are cacheable and have the same file size. According to Equation 3.7, when new objects dominate the workload, the average number of requests to an object in workloads PS-CLT-04, ST-CLT-04, and ST-CLT-05 is 39.7, 30.3, and 40.5, respectively, and the hit ratio for caching 10% objects is 0.85, 0.84, 0.85, respectively. In contrast,
Figure 3.20: The accesses to old objects and new objects in client side workloads
the optimal cache hit ratios for caching 10% objects are only 0.52, 0.48, and 0.54 in the duration of workloads PS-CLT-04, ST-CLT-04, and ST-CLT-05.

However, this process may take a very long time. Even assuming the number of old objects will not increase after the workload collection (an overoptimistic assumption according the trends shown in Figures 3.20(a), 3.20(b), and 3.20(c)), the time for new objects to dominate the workload is 25.3, 54.4, and 103.3 days for PS-CLT-04, ST-CLT-04, and ST-CLT-05, respectively. For server side workloads such as ST-SVR-01 (Figure 3.11), this time can be shorter, because the number of objects in a server is limited. However, the duration of this time is still in weeks.

Considering that the popularity of a media object decreases with time, the request correlation, another source of temporal locality of media systems, could be exploited to further improve the caching performance. However, this process also takes time, since media objects have long life spans. As shown in Figures 3.21(a) and 3.21(b), for media workloads
collected on the client side, most requested media objects are created long time ago, and most media requests are also for objects created long time ago. For example, for workload ST-CLT-05, more than 50% requested objects are created at least 250 days ago, and more than 50% requests are for objects older than 150 days.

Thus, for client side caching systems such as media proxies, a huge storage space is required and the performance can hardly be improved without a long execution time. A distributed caching system with enormously large storage and huge amount of pre-existing media content such as a P2P-based system seems attractive. In a P2P system, the total amount of storage is scalable to the user population and workload size. Furthermore, the content cached in a client’s local machine can be contributed to the system when the client joins for the first time, avoiding the cold misses of new requests and thus significantly reducing the system execution time to achieve the final (maximal) performance.

3.7 Conclusion

In this chapter, we have analyzed sixteen diverse media workloads collected from both the client side and the server side in different media systems with different delivery approaches, and proposed a general model for Internet media access patterns. We have found the reference ranks of media objects in these workloads follow the stretched exponential distribution. For homogeneously evolving media systems, the stretch factor of its reference rank distribution is a constant, which is highly related to the median file size in the workload. Furthermore, the media reference rank distribution evolves over time. Modeling the performance of media caching, we found the performance of media caching under the stretched exponential distribution is far less efficient than Web caching under a Zipf-like
model, indicating previous studies based on a Zipf-like assumption have potentially overestimated the benefit of media caching, even with segment-based partial caching strategies. An efficient media delivery system thus needs to leverage distributed resource sharing to scale its storage space with the increase of its workload size along time, where a media delivery system based on P2P collaborations is very attractive. Some preliminary results of this work have been presented in [62].
CHAPTER 4

PERFORMANCE ANALYSIS OF BITTORRENT-LIKE PEER-TO-PEER MEDIA SYSTEMS

4.1 Introduction

Peer-to-peer collaboration can utilize the huge amount of CPU, bandwidth, and storage resources among the edge clients of the Internet, which is beneficial to resource consuming systems such media delivery. However, first, in a P2P network, although there are some peers providing services voluntarily, in general peers are selfish and always want to get more services but contribute less. Second, in a P2P network, peers are free to come and go at any time. As a result, the quality of P2P services is often unreliable. Although a good incentive mechanism can effectively suppress free riding, the service availability and performance stability of P2P systems cannot be ensured, especially for requests to unpopular media files.

The performance of P2P systems depends on the participation and contribution of clients in the systems. Although the participation and contribution of P2P services can be stimulated by some kinds of incentive mechanism, in general they are driven by the individual interests of peers. With the effective “tit-for-tat” incentive mechanism, BitTorrent performs well during the flash crowd period. However, although the incentive mechanism
of current BitTorrent systems is effective for files with high popularity, it cannot help much for media files in the tails of the reference rank distribution. In this study, we analyze the user request patterns of BitTorrent-like systems, in order to design more efficient P2P collaboration algorithms.

BitTorrent [41] is a new generation of peer-to-peer (P2P) file sharing system that has become very popular recently. According to BigChampagne, there are nearly 7 million BitTorrent online users at the same time in August 2004, and nearly 10 million in August 2005 [3]. According to a recent measurement by CacheLogic, BitTorrent traffic represents 53% of all P2P traffic on the Internet in June 2004 [90]. Unlike traditional P2P systems such as Gnutella [1], KaZaa [2], and eDonkey/eMule/Overnet [4], in which peers sharing different files are organized together and exchange their desired files with each other, BitTorrent organizes peers sharing the same file into a P2P network and focuses on fast and efficient replication to distribute the file. In BitTorrent, a file is divided into small chunks, and a peer can download multiple chunks of the file in parallel. Peers with different file chunks are stimulated to exchange with each other through a “tit-for-tat” incentive mechanism, which enables peers with high uploading bandwidth to have corresponding high downloading bandwidth. In this way, BitTorrent prevents free riding effectively, which is very common in early P2P systems [23]. In contrast, P2P systems for exchanging different files such as KaZaa and eMule use participation levels or credit systems to track the contribution of each peer, and encourage peers to contribute by giving higher service priority to those peers with more contribution. Recently, reputation systems and game theoretic approaches for providing incentive in P2P networks have also been proposed [74, 81]. However, these systems are either too complex and unrealistic or easy to cheat and are misused [8, 25]. Compared to these systems, the direct “tit-for-tat” mechanism of BitTorrent is
simple, effective, and robust. In practice, BitTorrent systems scale fairly well during flash crowd period and have been widely used for various purposes, such as for distributing large software packages [28, 72].

Research has been conducted to study the effectiveness of BitTorrent systems [72, 91, 94, 121]. The most recent work shows the stability of BitTorrent systems through a fluid model, and verifies the effectiveness of its incentive mechanism [94]. However, this fluid model assumes a Poisson model for the downloading request arrival process, which has been shown to be unrealistic in an eight-month measurement study [91]. Consequently, the model can only characterize the performance of the BitTorrent system under stable conditions. In reality, as shown by our trace analysis, this stable period is very short. Furthermore, all existing studies on BitTorrent systems focus on the behaviors of single-torrent systems only, while our trace analysis shows that most peers (> 85%) participate in multiple torrents.

In this chapter, we present a performance study of BitTorrent-like P2P systems by modeling, based on extensive measurements and trace analysis. We first study the evolution of a single-torrent system. We found that although the existing system is effective for addressing the “flash crowd” problem upon the debut of a new file, it has the following limitations:

- Due to the exponentially decreasing peer arrival rate and the limited up time of seeds in a torrent, the service availability of the corresponding file becomes poor quickly, and eventually it is hard to locate and download this file.

- Client performance in the BitTorrent-like system is unstable, and fluctuates significantly with the changes of the number of online peers.
Existing systems could provide unfair services to peers. In current BitTorrent systems, a peer with a higher downloading speed tends to download more and upload less.

Motivated by the results of the single-torrent system study, we further propose a graph-based model to quantitatively analyze the multi-torrent system. In detail, we (1) characterize the peer request pattern in multi-torrent environments; (2) study the service potentials a torrent can provide to and get from other torrents; (3) demonstrate that inter-torrent collaboration is much more effective than stimulating seeds to stay longer for addressing the service unavailability in BitTorrent systems. Guided by the modeling results, we discuss and evaluate a novel architecture to facilitate inter-torrent collaboration with an exchange based instant incentive mechanism, addressing the well-known problem of lacking incentives to seeds.

The remainder of this chapter is organized as follows. Section 4.2 presents related work. In Section 4.3, we demonstrate the limitations of existing BitTorrent-like systems through measurements and trace analysis, and propose an evolution model for single-torrent systems. We present our multi-torrent model in Section 4.4. Section 4.5 discusses an architecture for inter-torrent collaboration. Finally, we summarize our work in Section 4.6.

4.2 Related Work

The amount of P2P traffic and the population of P2P users on the Internet keeps increasing. A lot of studies have been performed on the measurements, modeling, and algorithms of different P2P systems. Saroiu and Gummadi et al. characterized the P2P file sharing traffic over the Internet, including Napster, Gnutella, and KaZaa systems in their measurement studies [100, 99]. Gummadi and Dunn et al. analyzed the popularity distribution of
P2P files over the Internet and characterized the “download at most once” property of P2P clients [52]. Measurements and traffic analysis of BitTorrent systems have also been conducted recently. Izal and Urvo-Keller et al. analyzed a five-month workload of a single BitTorrent system for software distribution that involved thousands of peers, and assessed the performance of BitTorrent at the flash crowd period [72]. In study [28], Bellissimo et al. analyzed the BitTorrent traffic of thousands of torrents over a two-month period, with respect to file characteristics and client access characteristics. In study [91], Pouwelse et al. presented the current infrastructure of BitTorrent file sharing systems, including the Web servers/mirrors for directory service, meta-data distribution, and P2P file sharing. The authors also found that the arrival, abort, and departure processes of downloaders do not follow a Poisson distribution in the eight-month trace they collected, which was assumed in the previous modeling study [94].

A queuing model for P2P file sharing systems was proposed by Ge et al. in [48]. Yang and Veciana analyzed the service capacity of BitTorrent-like systems, and found that multi-part downloading helps P2P systems to improve performance during flash crowd period [121]. Based on their study, Qiu and Srikant further characterized the overall performance of BitTorrent-like systems using a simple fluid model, and analyzed the effectiveness of BitTorrent incentive mechanism using game theory [94]. Massoulie and Vojnovic introduced a probabilistic model of coupon replication systems, and analyzed the performance under an environment where neither altruistic user behaviors nor load balancing strategies (such as rarest first in BitTorrent) are supported [83].

In study [106], Sripanidkulchai et al. proposed an interest-based content location approach for P2P systems. By self-organizing into small groups, peers with the same interest can collaborate more efficiently, which is similar to the BitTorrent networks, where all
peers share the same file. Sherwood et al. proposed a P2P protocol for bulk data transfer, which aims to improve client performance and to reduce server load, by using enhanced algorithms over BitTorrent systems [104].

Different from all studies above, our modeling and trace analysis focus on the evolution of single-torrent systems and the inter-relation among multiple torrents over the Internet, revealing the limitations of current BitTorrent systems. Furthermore, we have proposed an innovative architecture to facilitate inter-torrent collaboration, which represents the first step towards making the BitTorrent-like system a reliable and efficient content delivery vehicle.

4.3 Modeling and Characterization of Single-Torrent Systems

In a BitTorrent system, the content provider creates a meta file (with the .torrent suffix name) for the data file it wants to share, and publishes the meta file on a Web site. Then the content provider starts a BitTorrent client with a full copy of the data file as the original seed. For each data file to be shared, there is a tracker site, whose URL is encoded in the meta file, to help peers find each other to exchange the file chunks. A user starts a BitTorrent client as a downloader at the beginning to download file chunks from other peers or seeds in parallel. As soon as a peer has downloaded a chunk, it is shared to the peer community so that other downloading peers have a new source of this chunk. A peer that has downloaded the file completely also becomes a seed that could in turn provide downloading service to other peers. All peers in the system, including both downloaders and seeds, self-organize into a P2P network, known as a torrent or a swarm. The initial seed can leave the torrent when there are other seeds available, and content availability and
system performance in the future depend on the arrival and departure of downloaders and other seeds.

Although the effectiveness of BitTorrent systems during flash crowds, which normally happen soon upon the debut of a new file, has been widely studied through trace analysis and modeling [72, 91, 94, 121], the overall client performance in the lifetime of a torrent during which the file popularity changes has not been studied. However, the change of file popularity is particularly important for BitTorrent-like systems, where the service availability relies purely on the voluntary participation of peers. This is in contrast to a client-server model where a permanent site (i.e., a server) can provide persistent service. In this section, we propose an evolution model to study the effects of file popularity changes to the performance of a single-torrent system.

4.3.1 Characterizing file popularity evolution

In this study, we analyze and model BitTorrent traffic based on two kinds of traces, one is data file downloading statistics of peers recorded by the tracker sites and the other is meta file downloading activities of BitTorrent users collected on the Internet. The BitTorrent data file downloading traces were collected from two popular dedicated tracker sites (although each torrent can have its own tracker site, there are many dedicated tracker sites on the Internet providing persistent service, each of which may host thousands of torrents), sampled every half an hour for 48 days from 2003-10-23 to 2003-12-10. This trace was collected by University of Massachusetts, Amherst [28] (abbreviated as the tracker trace in the remainder of this paper). We identify different peers and match multiple sessions of the same downloading with the similar methods used in study [72]. The firewalled peers, although cannot accept incoming connections and thus are not provided by the tracker to
allow other peers to connect to, are still included in the tracker statistics. We extract the peer request time, downloading/uploading bytes, the downloading/uploading bandwidth of all peers of each torrent, and the information of each torrent such as torrent birth time and the size of data file. Due to page limit, we only present the analysis results of the larger tracker trace, which includes more than 1,500 torrents (about 550 torrents were fully traced during their lifecycles). The smaller trace has similar results.

The BitTorrent meta file downloading traces were collected from a large commercial server farm hosted by a major ISP and a large group of home users connected to the Internet via a well-known cable company, using the Gigascope appliance [46], from 2004-09-28 to 2004-10-07. The server farm trace includes about 50 tracker sites hosting hundreds of torrents, and the cable network trace includes about 3,000 BitTorrent users (by IP addresses) requesting thousands of torrents on the Internet. Both traces include the first IP packets of all HTTP downloading of the .torrent files, with the timestamp when the packet is captured (the downloading time of the .torrent file). This timestamp represents the peer arrival time to the torrent. We also extract the timestamp encoded in each .torrent file, which is the creation time of the meta file and represents the torrent birth time.

Figure 4.1(a) shows the complementary CDF (CCDF) distribution of the “relative” request arrival time for all fully-traced torrents in the tracker trace. We consider all requests to all torrents in the trace and normalize $x$ and $y$ coordinates as follows. The $x$ coordinate is a “relative time” $t$, which is equal to the request arrival time to a torrent minus the birth time of this torrent, i.e., the age of the torrent when a request arrives at it. For a peer downloading the file in multiple sessions, only the first request is considered. So $t$ denotes the arrival time of a peer to a torrent. The $y$ coordinate at time $t$ denotes the total number of requests to all torrents in the trace minus the cumulative number of requests to these
torrents during time duration \( t \) since the requested torrent is born. The \( y \)-axis in the figure is not normalized to percentage (as normal CCDF plots) to keep the unit of \( y \) coordinates. Similar to Figure 4.1(a), Figures 4.1(b) and 4.1(c) show the CCDF distribution of the time when a .torrent file was downloaded after torrent birth in the server farm and in the cable network, respectively. Note that \( y \)-axis is in log scale in the three figures.

All three curves can be fitted with straight lines. This consistent trend strongly suggests that after a torrent is born, the number of peer arrivals to the torrent decreases exponentially with time. The curves are not straight lines because each data set consists of many torrents, and the number of peer arrivals for different torrents may decrease exponentially with different attenuation parameters. To validate whether this claim holds for each individual torrent, we use the least square method to fit the logarithm of the complementary of the number of peer arrivals to each torrent along the time in the tracker trace. We define the relative deviation of this fitting at time \( t \) for a torrent as \( \frac{\log N_0(t) - \log N(t)}{\log N_0(t)} \times 100\% \), where \( t \) is the age of the torrent when a peer arrives, \( N_0(t) \) is the complementary value of the number
of requests at $t$, and $N(t)$ is the fitting result. Figure 4.2 shows the distribution of average fitting deviation for each fully-traced torrent that has at least 20 peers during its lifetime. In this figure, each point in the $x$-axis denotes a torrent, sorted in non-ascending order of torrent population during the entire lifetime, and the corresponding value in $y$-axis denotes the average of relative fitting deviation of this torrent. We can see that the fitting is more accurate for torrents with larger populations, and the overall average relative deviation is only about 6%. We do not fit the curve for each individual torrent in the server farm and cable network trace, because the data collection duration is short so that they do not cover the whole lifespans of torrents. In the remainder of this paper, we only use the tracker trace for modeling and analysis.

We define the *popularity* of a BitTorrent data file at a time instant as the peer arrival rate of the corresponding torrent at that time, which is the derivative of the peer arrival time distribution of that torrent. Since the derivative of an exponential function is also an exponential function, we assume that the peer arrival rate of a torrent follows an exponential
decreasing rule with time \( t \)

\[
\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}},
\]

(4.1)

where \( \lambda_0 \) is the initial arrival rate when the torrent starts, and \( \tau \) is the attenuation parameter of peer arrival rate (file popularity). This equation characterizes the evolution of file popularity in a single-torrent system over time. In Section 4.3.3, we will use a fluid model to evaluate the file popularity evolution again.

### 4.3.2 Torrent evolution and service availability

We define the **torrent lifespan** as the duration from the birth of the torrent to the time after which there is no complete copy of the file in the system, and thus new arriving peers cannot complete downloading. To simplify our model, we assume that the initial seed exits the system as soon as a downloader has downloaded the file completely. In practice, the initial seed may stay online in the system for a longer time, and some seeds may return to the system to serve the content.

The **inter-arrival time** between two successive arriving peers \( \delta t \) can be approximated as \( \frac{1}{\lambda(t)} \). If we denote the rate at which seeds leave the system as \( \gamma \), then the average service time of a seed can be approximated as \( \frac{1}{\gamma} \). Since \( \frac{1}{\gamma} \) is limited, according to the exponential decrease of peer arrival rate, the inter-arrival time of peers will grow exponentially, and finally there will be only one seed at a time. Thus, when \( \delta t \approx \frac{1}{\lambda(t)} > \frac{1}{\gamma} \), a new peer arrives at time \( t \) cannot complete downloading before the last peer (seed) leaves, and the torrent is dead. Using Equation 4.1, we get the torrent lifespan

\[
T_{life} = \tau \log\left(\frac{\lambda_0}{\gamma}\right).
\]

(4.2)

Equation 4.2 shows the expectation of the real torrent lifespan. To verify Equation 4.2, we compute the initial peer arrival rate \( \lambda_0 \) and the torrent attenuation parameter \( \tau \) for fully
traced torrents in the tracker trace. From Equation 4.1, we have

$$
\log \delta t = - \log \lambda_0 + \frac{t}{\tau}.
$$

Both $\delta t$ and $t$ for each peer arrival can be extracted from the trace and we get $\log \lambda_0$ and $\frac{1}{\tau}$ using linear regression. We also compute the seed leaving rate $\gamma$ as the the reciprocal of the average seed service time, which is extracted from the trace, too. Figure 4.3 shows the comparison of torrent lifespan computed from the tracker trace (indicated by $\text{trace}$) and that from the Equation 4.2 (indicated by $\text{model}$). In this figure, each point in $x$-axis denotes a torrent, while each point in $y$-axis denotes the measurement result or the modeling result of torrent lifespan. The torrents in the $x$-axis are sorted in non-ascending order of the modeling results of torrent lifespans. As shown in the figure, our model fits the real torrent lifespan very well. The average lifespan of torrents is about 8.89 days based on the trace analysis and 8.34 days based on our model. The lifespans of most torrents are between 30 - 300 hours, and there are only a small number of torrents with extremely short or extremely long lifespans.
The total population of a torrent during its lifespan (in the number of peers) is

\[ N_{\text{all}}(t) = \int_{0}^{t} \lambda_0 e^{-\frac{t}{\tau}} dt = \lambda_0 \tau. \] (4.4)

Among them, some peers may not be able to complete downloading due to lack of seeds, which we call failed peers, denoted as follows:

\[ N_{\text{fail}}(t) = \int_{T_{\text{life}}}^{\infty} \lambda_0 e^{-\frac{t}{\tau}} dt = \gamma \tau. \] (4.5)

Thus, the downloading failure ratio of the torrent is

\[ R_{\text{fail}} = \frac{N_{\text{fail}}}{N_{\text{all}}} = \frac{\gamma \tau}{\lambda_0 \tau} = \frac{\gamma}{\lambda_0}. \] (4.6)

Figure 4.4(a) shows the comparison of the torrent population computed from the tracker trace with that computed from our model for each fully-traced torrent. In this figure, each point in \(x\)-axis denotes a torrent, while each point in \(y\)-axis denotes the measurement result or the modeling result of the total population of this torrent during its entire lifespan. The torrents in the \(x\)-axis are sorted in non-ascending order of the modeling results of torrent.
populations. As evidenced by the figure, the modeling result and trace analysis are consistent. In addition, we can see that the distribution of the torrent population is heavily skewed: although there are several large torrents, most torrents are very small, and the average population of torrents is only about 102 peers.

Figure 4.4(b) shows the downloading failure ratio based on trace analysis and on our model (plotted in a manner similar to that of Figure 4.4(a)). This fitting is not as good as that of Figure 4.4(a); the real failure ratio of torrents is lower than what our model predicts because there are some altruistic peers that serve the torrent voluntarily. This also explains why the torrent lifespan in the trace analysis (8.89 days) is slightly higher than that in our model (8.34 days). Furthermore, there are some torrents that have no failed peers in the trace because the seeds leave after the downloaders finish, but cannot be shown in the log scale plot. However, the average downloading failure ratio based on the trace analysis is still about 10%, which is non-trivial for a content distribution system.

Equation 4.5 implies that the number of failed peers in a torrent is independent of the initial peer arrival rate (the initial file popularity). Instead, the number of failed peers depends on the attenuation exponent of peer arrival rate (the attenuation speed of file popularity) and the seed departure rate. Figure 4.4(c) shows downloading failure ratios of torrents and their corresponding populations (plotted in the similar manner as that of Figure 4.4(a) and 4.4(b)). As reflected in the figure and indicated by Equation 4.6, the larger the torrent population, the lower the downloading failure ratio. It is interesting to note that the population of torrents, sorted in non-ascending order of their corresponding downloading failure ratios, forms several clear curves, each of which represents those torrents with similar evolution patterns (the attenuation parameter $\tau$). On the right side of the figure, the failure ratio
of the torrents is zero due to the existence of some altruistic seeds, which always stay until the last downloader completes.

In the above analysis, we assume that peers always complete their downloading unless they cannot. We do not consider peers that abort downloading voluntarily when seeds are still available in the torrent. A peer may abort downloading due to (1) loss of interest to the data file; (2) slow downloading speed or small downloading progress. Figure 4.5(a) shows the distribution of the average downloading speed of peers that voluntarily abort and peers that download the data file completely. Figure 4.5(b) shows the distribution of downloading progress (the percentage of the entire data file that has been downloaded) when peers abort downloading voluntarily. The figures indicate that the probability for a peer to abort downloading voluntarily is almost independent of its downloading speed and the current downloading progress. This is consistent with the study [52], which found that P2P users are patient to wait days to weeks for the entire file downloading. Hence, the
voluntary abort behavior of file downloadings is mainly due to the loss of user interest. Excluding peers that abort file downloading is equivalent to assuming that these peers are uninterested in the data file at the beginning, and thus does not affect our analysis.

### 4.3.3 Client performance variations

Study [94] proposed a fluid model for BitTorrent-like systems with constant peer arrival rate. We use the idea of the fluid model, but assume that peer arrival rate follows Equation 4.1. Assume the downloading bandwidth of a peer is greater than its uploading bandwidth, the basic ODE (ordinary differential equation) set for the fluid model is

\[
\begin{align*}
\frac{dx(t)}{dt} &= \lambda_0 e^{-\tau} - \theta x(t) - \mu (\eta x(t) + y(t)), \\
\frac{dy(t)}{dt} &= \mu (\eta x(t) + y(t)) - \gamma y(t), \\
x(0) &= 0, y(0) = 1,
\end{align*}
\]

(4.7)

where the meanings of the parameters in our fluid model are listed in Table 4.1. These notations are adopted from work [94, 121].

<table>
<thead>
<tr>
<th>$x(t)$</th>
<th>number of downloaders in the system at time $t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y(t)$</td>
<td>number of seeds in the system at time $t$</td>
</tr>
<tr>
<td>$\lambda_0$</td>
<td>the initial value of peer arrival rate</td>
</tr>
<tr>
<td>$\tau$</td>
<td>the attenuation parameter of peer arrival rate</td>
</tr>
<tr>
<td>$\mu$</td>
<td>the uploading bandwidth</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>the rate at which seeds leave the system</td>
</tr>
<tr>
<td>$\theta$</td>
<td>the rate at which downloaders relinquish downloading and exit the system</td>
</tr>
<tr>
<td>$\eta$</td>
<td>the file sharing efficiency, meaning the probability that a peer can exchange chunks with other peers</td>
</tr>
</tbody>
</table>

Table 4.1: Notations and assumptions for the fluid model
When the ODE set has two different real eigenvalues $\psi_1 \neq \psi_2$, the resolution can be expressed as:

\[
\begin{align*}
  x(t) &= ae^{\psi_1 t} + be^{\psi_2 t} + d_1 e^{-\frac{t}{\tau}}, \\
  y(t) &= c_1 ae^{\psi_1 t} + c_2 be^{\psi_2 t} + d_2 e^{-\frac{t}{\tau}},
\end{align*}
\] (4.8)

where $d_1, d_2, c_1, c_2, a, b$ are constant. The value of these constants and the detailed resolution of the fluid model can be found in Appendix E.

The average downloading speed of peers at time $t$ is

\[
u(t) = \mu \frac{\eta x(t) + y(t)}{x(t)} = \mu (\eta + \frac{y(t)}{x(t)}).
\] (4.9)

We use the tracker trace to validate the torrent evolution model. Similar to the peer arrival rate, the modeling results fit the trace better for torrents with larger populations. Figure 4.6(a) shows the torrent evolution by both our fluid model and the analysis results of a typical torrent in the trace. The figure shows that the number of downloaders increases exponentially in a short period of time after the torrent’s birth (the flash crowd period), and then decreases exponentially, but at a slower rate. The number of seeds also increases.

Figure 4.6: Torrent evolution under the fluid model
Figure 4.7: Performance variations in BitTorrent systems

exponentially at first, and then decreases exponentially at a slower rate. The peak time of the number of seeds lags behind that of the number of downloaders. As a result, \( u(t) \) increases until the torrent is dead, and the resources of seeds cannot increase in proportion to service demand. Furthermore, due to the random arrival of downloaders and the random departure of seeds, average downloading performance fluctuates significantly when the number of peers in the torrent is small, as shown in Figure 4.6(b).

Figure 4.7(a) shows the performance variations of the torrent under two kinds of granularities. The *instant speed* represents the mean downloading speed of all peers in the torrent at that time instant, sampled every half an hour. The *average speed* represents the average value of the instant speed over the typical downloading time (the average downloading time of all peers). The figure shows that the client downloading speed at different time stages is highly diverse and can affect client downloading time significantly. The reason is that seeds play an important role in the client downloading performance. However, the generation of
seeds is the same as the completeness of peer downloading, so the random fluctuation of downloading speed cannot be smoothed in the scale of typical downloading time when the number of peers is small.

Figure 4.7(b) shows the number of peers and the average downloading speed for each torrent in the trace at 12:00:01 on 2003-11-15. In this figure, each point in $x$-axis denotes a torrent, while the left $y$-axis denotes the number of peers (the number of downloaders and seeds are represented with different colors and stacked together in the figure) in this torrent, and the right $y$-axis denotes average downloading speed of this torrent. The torrents in the $x$-axis are sorted in non-ascending order of the number of peers (downloaders and seeds) in each torrents. The results at other time instants are similar. In general, peers in torrents with larger populations have relatively higher and more stable downloading speed, while the downloading speed in torrents with small populations disperses significantly. When the number of peers in the torrent is small, the client downloading performance is easily affected by the individual behavior of seeds.

Figure 4.7(c) shows the total number of peers in all torrents (the number of downloaders and seeds are represented with different colors and stacked together in the figure) and the average downloading speed of all downloaders in the trace at different time stages. The average downloading speed of all torrents is shown to be much more stable than that of one torrent. The reason is that the downloader/seed ratio is much more stable due to the large population of the system. This motivates us to balance the service load among different torrents, so that each torrent can provide relatively stable downloading performance to clients in its lifespan.
4.3.4 Service fairness

In a BitTorrent system, the service policy of seeds favors peers with high downloading speed, in order to improve the seed production rate in the system, i.e., to have these high speed downloaders complete downloading as soon as possible and wish they will then serve other downloaders. In this subsection, we investigate the effects of this policy on the service fairness of BitTorrent.

We define the *contribution ratio* of a peer as the total uploaded bytes over the total downloaded bytes of the peer. Figure 4.8(a) shows the peer downloading speed and the corresponding contribution ratio extracted from the trace. In this figure, each point in the $x$-axis denotes a peer, while the left $y$-axis denotes the contribution ratio of this peer, and the right $y$-axis denotes the average downloading speed of this peer. On the $x$-axis, peers are sorted in non-ascending order of their contribution ratios. If the service of BitTorrent
system is fair, the peer contribution ratio and downloading speed should be highly positively correlated. However, the figure shows a rough trend that the peer contribution ratio increases when the downloading speed decreases. That is, the higher the downloading performance peers have, the less uploading service they actually contribute. This indicates that peers with high speed finish downloading quickly and then quit the system soon, which defeats the design purpose of the seed service policy.

Figure 4.8(b) shows the number of torrents that each peer involves and its corresponding contribution ratio (plotted in the similar manner as that of Figure 4.8(a)). The figure shows no distinguishable correlation between the two, indicating that the main reason for seeds to leave old torrents is not to start new downloading tasks.

In summary, we observe that the BitTorrent’s biased seed service policy in favor of high speed downloaders really affects the fairness to peers in downloading, and an incentive mechanism is needed to encourage seeds to contribute.

4.4 Modeling Multiple Torrents in BitTorrent Systems

In the previous section, we have shown that client performance fluctuates significantly in single-torrent systems, but is very stable when aggregated over multiple torrents. Based on this observation, in this section, we study the correlation among multiple torrents through modeling and trace analysis, aiming to look for solutions to enable inter-torrent collaboration.

Although different torrents are independent from each other in the current BitTorrent systems, they are inherently related by peers that request multiple data files. A peer may download a data file, serve as a seed for that torrent for a while, and then go offline to sleep for a period of time. The peer may return sometime later and repeat the activities above.
Thus, a peer’s lifecycle consists of a sequence of downloading, seeding, and sleeping activities. If a peer stops using BitTorrent for a long time that is much longer than its typical sleeping time, we consider the peer as dead.

In the current BitTorrent systems, a peer is encouraged to exchange file chunks with other peers that are downloading the same file instead of serving old data files it has downloaded. Thus, in our model, we assume each peer joins (downloading and seeding) each torrent at most once, and joins one torrent at a time. Having these assumptions, we start to characterize peers in multiple torrents.

### 4.4.1 Characterizing peer request pattern

In the multi-torrent environment, both torrents and peers are born and die continuously. Figure 4.9(a) shows the CDF of torrent birth in the trace (indicated by raw data) and our linear fit. The average torrent birth rate (denoted as $\lambda_t$ in the following context) is about 0.9454 torrent per hour. Figure 4.9(b) shows the CDF of torrent request arrivals (for all
peers over all torrents) and our linear fit. We define the *torrent request rate* as the number of downloading requests for all torrents per unit time in the multi-torrent system, denoted as $\lambda_q$ in the following context. Although the peer arrival rate of a single-torrent system decreases exponentially as shown in Figure 4.1, the torrent request rate in the multi-torrent system is almost a constant, about 133.39 requests per hour.

Since both the torrent birth rate and torrent request rate are almost constant, it is natural to assume that the *peer birth rate* (denoted as $\lambda_p$ in the following context) is also a constant. A peer is born when it appears in the system for the first time. However, as shown in Figure 4.9(c), the peer birth rate is high at the beginning of the trace collection duration, and then converges to a constant rate asymptotically. The reason is that peers appear in the trace for the first time may actually have been born before the trace collection, and the number of such peers decreases quickly after the trace collection starts. Thus, we take the asymptotic birth rate as the real birth rate of peers, which is about 19.37 peers per hour.

The constant peer birth rate and torrent request rate indicate that each peer only joins a limited number of torrents. However, the request rate of a peer might still change over time. We define the *peer request rate* as the number of requests a peer submits for different torrents per unit time. Assume the peer request rate can be expressed as

$$ r(t) = r_0 e^{-\frac{t}{\tau_r}}, \quad (4.10) $$

where $t$ is the time duration after the peer is born, $r_0$ is the initial request rate, and $\tau_r$ is the attenuation parameter of the request rate. When $\tau_r \to \infty$, the peer has a constant request rate; when $\tau_r < 0$, the peer has an increasing request rate.

The inter-arrival time between two successive requests of a peer $\delta t$ is $\frac{1}{r(t)}$ (note this inter-arrival time is different from the inter-arrival time of two successive arriving peers to
a torrent described in Section 4.3.2). Thus, we have

$$\log \delta t = - \log r_0 + \frac{t}{\tau_r}. \quad (4.11)$$

We extract $\delta t$ and $t$ from the trace for each peer requesting multiple torrents, and use linear regression to compute $\log r_0$ and $\frac{1}{\tau_r}$. Figure 4.10(a) shows the number of torrents that each peer requests and the corresponding $\tau_r$, for peers requesting at least 3 torrents. In this figure, each point in the $x$-axis denotes a peer, while the left $y$-axis denotes the $\tau_r$ value of this peer, and the right $y$-axis denotes the number of torrents in which this peer participates. In $x$-axis, peers are sorted in non-ascending order of the number of torrents they join. As shown in the figure, the value of parameter $\tau_r$ in Equation 4.10 is extremely large compared to the typical duration of file downloading, with the mean value of about 77 years, which implies that the average request rates of peers do not change significantly over time. Further, $\tau_r$ is independent of the number of torrents that peers join. Thus, we can
assume that the request processes of peers are Poisson-like with constant average request rates.

Figure 4.10(b) shows the average inter-arrival time of torrent requests for peers requesting multiple torrents (plotted in the similar manner as that of Figure 4.10(a)). As shown in the figure, it is intuitive to find that the upper bound of the number of torrents each peer requests increases with the decrease of inter-arrival time. However, for peers with similar request rates, the number of torrents they request are very diverse, since they stay in the system for different time durations. Figure 4.10(c) further plots the downloading speed versus the number of torrents that each peer joins (plotted in the similar manner as that of Figure 4.10(a)). There is no strong correlation between the two for peers with downloading speed > 1 KB per second. This implies that for peers whose downloading speed is large enough, the numbers of files they download is independent of their downloading speed.

Thus, we assume that a peer joins a new torrent with probability $p$. For $N$ peers in the system, during their whole lifecycles, there are $Np^{n-1}$ peers that request at least $m$ torrents. Ranking peers in non-ascending order of the number of torrents they join, the number of torrents that a peer ranked $i$ joins is

$$m = 1 + \frac{\log i - \log N}{\log p}. \quad (4.12)$$

In addition, a peer has the probability $1 - p$ to download exactly 1 file, probability $p(1 - p)$ to download exactly 2 files, and probability $p^{k-1}(1 - p)$ to download exactly $k$ files. So the mean number of torrents that a peer joins is:

$$\bar{m} = \sum_{k=1}^{\infty} kp^{k-1}(1 - p) = \frac{1}{1 - p}. \quad (4.13)$$

Figure 4.11(a) shows the distribution of the number of files that each peer downloads in the trace. The curve in the figure is a little convex, deviating from what Equation 4.12
predicts (a straight line when $x$-axis is in log scale). The reason is that the number of torrents joined by peers born before the trace collection is under-estimated, since some of these requests cannot be recorded in the trace. A similar situation exists for peers that are still active after the end of trace collection.

Figure 4.11(b) shows the distribution of number of torrents joined by each peer that was born in the middle of the trace collection duration (indicated by raw data) and our linear fit. The curve fits Equation 4.12 very well, and we estimate $p \approx 0.8551$ from the analysis. Thus, the average number of torrents each peer joins is about 7.514.

To verify the probability model we use in the above analysis, we estimate $p$ in another way as follows. Assuming that the peer birth rate is $\lambda_p$ and the torrent request rate is $\lambda_q$, since each peer joins $\frac{1}{1-p}$ torrents during its lifetime in average, we have

$$\lambda_q = \frac{1}{1-p} \lambda_p.$$ (4.14)
Based on the peer request arrival rate and the peer birth rate we derived before (see Figure 4.9(b) and 4.9(c)), we have $p = 1 - \frac{\lambda_p}{\lambda_q} = 0.8548$. This is very close to the value we got from Equation 4.12, 0.8551, meaning that there are more than 85% peers joining multiple torrents.

Having characterized the torrent request pattern of peers, finally we consider the distribution of the seeding time and the sleeping time of peers. According to our fluid model, $\frac{1}{\gamma}$ represents the average seeding time. Figure 4.12(a) and 4.12(b) show the probability distribution functions of the peer seeding time and the peer sleeping time in the system. Note that the $y$-axis is in log scale. Both the peer seeding time and sleeping time roughly follow the exponential distribution with probability density function $f_{sd}(t) = \frac{1}{\tau_{sd}} e^{-\frac{t}{\tau_{sd}}}$, and $f_{sl}(t) = \frac{1}{\tau_{sl}} e^{-\frac{t}{\tau_{sl}}}$, respectively. Based on the trace analysis, we estimate $\tau_{sd} = \frac{1}{\gamma} = 8.42$ hours, and $\tau_{sl} = 58.32$ hours.
4.4.2 Characterizing inter-torrent relations

In this subsection we study how different torrents are connected through peers that download multiple files, based on our previously verified assumptions.

For simplification, we consider a homogeneous multi-torrent environment where all torrents and peers have the same \( \lambda_0, \tau, \mu, \eta, \gamma \), and average sleeping time. Consider all torrents that have been born in the system by the time instant \( t_0 \). We number the torrents and name their birth time as follows: the most recently born torrent by \( t_0 \) is torrent 1, with a birth time \( t_1 \); the torrent born just before torrent 1 is torrent 2, with a birth time \( t_2 \); ... ; and so on and so forth. Thus, for any two torrents \( i \) and \( j \), if torrent \( i \) was born just before torrent \( j \), we have \( i = j + 1 \) and \( t_i < t_j \).

Assume the probability that a peer selects torrent \( i \) at time \( t \) as its \( k \)-th torrent is \( P_{ik}(t) \), \( t \leq t_0 \) and \( 1 \leq i < \infty \). We have \( P_{ik}(t) = 0 \) if \( t < t_i \). We denote \( P_{ik}(t) \) as \( P_i(t) \) for simplicity, and assume that \( P_i(t) \) satisfies

\[
P_i(t) = \frac{e^{-\frac{t-t_i}{\tau}}}{\sum_{j=1}^{\infty} e^{-\frac{t-t_j}{\tau}}},
\]

(4.15)

where \( t_j = t - \frac{j}{M} \), \( 1 \leq j < \infty \). Thus, we have

\[
P_i(t) = \frac{e^{-\frac{1}{M\tau}}}{\sum_{j=1}^{\infty} e^{-\frac{1}{M\tau}}} = (e^{\frac{1}{M\tau}} - 1) e^{-\frac{1}{M\tau}}
\]

(4.16)

\[= (e^{\frac{1}{M\tau}} - 1) e^{-\frac{t-t_i}{\tau}}.
\]

When a peer requests its \( k \)-th data file, the data files that it has requested will not be selected. Assuming

\[
P^k_i(t) = \alpha_k P_i(t),
\]

(4.17)

the peer arrival rate of torrent \( i \) can be expressed as

\[
\lambda_i(t) = \frac{\alpha}{1-p} \lambda_0 P_i(t) e^{\frac{1}{M\tau}} - 1 e^{-\frac{t-t_i}{\tau}},
\]

(4.18)
where \( \alpha = \sum_{k=1}^{\infty} \alpha_k p^{k-1}(1 - p) \). When \( \lambda_t \gg r \), we have \( \alpha_k \approx 1 \) and \( \alpha \approx 1 \). Comparing Equation 4.1 with 4.18, we have \( \lambda_0 = \frac{\alpha}{1-p} \lambda_p(e^{\frac{\alpha}{r\lambda}} - 1) \).

Considering that a peer in a torrent may have downloaded files from other torrents, we can model the relationship among different torrents in the P2P system as a directed graph. Each node in the graph represents a torrent. A directed edge from torrent \( i \) to torrent \( j \) denotes that some peers in torrent \( i \) have downloaded the file from torrent \( j \), and thus have the potential to provide service to peers in torrent \( j \), even though they are not in torrent \( j \) currently. The weight of the directed edge \( W_{i,j} \) represents the number of such peers. For simplicity, we define \( W_{i,i} = 0 \).

The graph changes dynamically over time. Now let us consider the graph at time \( t_0 \). During time \([t, t + dt] \), \( t_j \leq t < t_0 \), there are \( \lambda_j(t)dt \) peers who join torrent \( j \). Let \( k(t) = \lfloor r(t_0 - t) \rfloor \). During time \([t, t_0] \), these peers can download up to \( k(t) - 1 \) torrents completely in addition to torrent \( j \) and may request (or be requesting) the next torrent at time \( t_0 \). Assuming \( \alpha_k \approx 1 \), for a peer who is active at time \( t \), the probability that it is still active at time \( t_0 \), but does not request torrent \( i \) during \([t, t_0] \) is

\[
Q_i(t) = p \times \prod_{l=1}^{k(t)-1} p \times (1 - P_i(t + \frac{l}{r})). \tag{4.19}
\]

When \( i \neq j \), we have

\[
W_{i,j} = \int_{t_j}^{t_0} Q_i(t) \times P_i(t + \frac{k(t)}{r}) \times \lambda_j(t)dt. \tag{4.20}
\]

Therefore, the weighted out-degree of torrent \( i \) represents the total potential capability its peers can provide to peers in other torrents, denoted as \( SP_i \), where

\[
SP_i = \sum_{j=1}^{\infty} W_{i,j}. \tag{4.21}
\]
Correspondingly, the weighted in-degree of torrent $i$ represents the total potentials its peers can get from peers in other torrents, denoted as $SG_j$, where

$$SG_j = \sum_{i=1}^{\infty} W_{i,j}. \tag{4.22}$$

Figure 4.13(a) and 4.13(b) show the weighted out-degree and weighted in-degree at a time instant based on trace analysis and our probability model, respectively. In the figures, each point in the $x$-axis denotes a torrent, sorted in non-ascending order of weighted out-degree or weighted in-degree. The right $y$-axis in the figures denotes torrent size, the number of peers in the torrent at this time instant. In general, torrents with more peers tend to have a larger out-degree and in-degree, though the trend is very rough. The weighted out-degree and in-degree distribution according to our trace analysis follows power law rules roughly. It deviates from our model somewhat because of the heterogeneity of torrents in the real system.
4.4.3 Reducing downloading failure ratio by inter-torrent collaboration

In the multi-torrent environment, old peers that had downloaded the file from a torrent may come back to download other data files, and the lifespan of this torrent can be extended if these old peers are willing to provide service. Assume the request arrival rate of this torrent is $\lambda(t)$ and $\lambda(t) = 0$ when $t < 0$. If we consider both new requesting peers and old returning peers, the peer arrival rate of the torrent is

$$\lambda'(t) = \sum_{l=0}^{k(t)} p^l \lambda(t - \frac{t}{\tau}) = \sum_{l=0}^{k(t)} p^l \lambda_0 e^{-\frac{t}{\tau}}$$

(4.23)

where $k(t) = \lfloor rt \rfloor$ and $q = pe^{\frac{1}{rt}}$ ($q > 1$ according to our trace analysis).

When $\lambda(t) < \gamma$, the torrent is truly dead. The lifespan of a torrent without inter-torrent collaboration is $T_{life} = \tau \log(\frac{\lambda_0}{\gamma})$. Denoting the lifespan of the torrent with inter-torrent collaboration as $T'_{life}$, then $\lambda'(T'_{life}) = \gamma$, we have

$$\log \gamma = \log \lambda_0 - \frac{T_{life}}{\tau} + \log(q^{k(T_{life})} - 1) - \log(q - 1)$$

$$\approx \log \lambda_0 - \frac{T_{life}}{\tau} + (k(T'_{life}) + 1) \log q - \log(q - 1)$$

$$= \log \lambda_0 - \frac{T'_{life}}{\tau} + k(T'_{life}) \log q + \log \frac{q}{q - 1}.$$

It leads to $\log(\frac{\lambda_0}{\gamma} - \frac{q}{q - 1}) \approx \left( \frac{1}{\tau} - r \log q \right) T'_{life}$. Thus

$$T'_{life} \approx \frac{\tau \log(\frac{\lambda_0}{\gamma} - \frac{q}{q - 1})}{1 - r \log q} = \frac{\tau \log(\frac{\lambda_0}{\gamma} - \frac{q}{q - 1})}{\tau r \log \frac{1}{p}}$$

(4.24)

According to the trace analysis and our modeling, $\beta = \frac{1}{\tau r \log \frac{1}{p}} \approx 6$. So we have

$$R'_{fail} = \int_{T'_{life}}^{\infty} \lambda_0 e^{-\frac{t}{\tau}} dt = e^{-\frac{T'_{life}}{\tau}} < R_{fail}^\beta \approx R_{fail}^6.$$ (4.25)

In the single-torrent model, since we cannot change the peer request pattern, the only way to decrease the downloading failure ratio is to decrease the seed leaving rate (i.e., to
(4.26)

Comparing Equation 4.24 and 4.25 with Equation 4.26 and 4.27, we can see that inter-torrent collaboration is much more effective than stimulating seeds to serve longer in order to reduce the downloading failure ratio. Decreasing seed leaving rate can only extend torrent life span by a constant, while inter-torrent collaboration can increase torrent life span multiple times. As a result, for reducing the downloading failure ratio, decreasing seed leaving rate has polynomial effect, while inter-torrent collaboration has exponential effect. For example, if the current downloading failure rate is 0.1, and seeds can be stimulated to stay 10 times longer (i.e., \(\gamma\) will decrease 10 times), then the downloading failure rate will decrease 10 times to 0.01. However, by inter-torrent collaboration, the downloading failure ratio can be as low as 0.1^6 = 10^{-6}. The reason is that extending seed staying time only increases the service time for peers that arrive close to the seed generation time. With the passage of time, the peer arrival rate decreases exponentially, and finally the seed serving time will not be long enough for newly arriving peers. On the other hand, by exploiting inter-torrent collaboration, peers that have downloaded the file may return multiple times during a much longer period, and the downloading failure ratio can be significantly reduced to near zero.
4.5 Multi-Torrent Collaboration of BitTorrent Systems

4.5.1 Tracker site overlay

We propose an architecture where tracker sites of different torrents self-organize into an overlay network to coordinate the collaboration among their peers. Each tracker site maintains a Neighbor-Out Table and a Neighbor-In Table to record the relationship with its neighboring torrents. The Neighbor-Out Table records the torrents that its peers can provide service to. The Neighbor-In Table records the torrents whose peers can provide service to this torrent. When a peer $q$ joins a new torrent $A$, it uploads to its tracker site the information about from which torrents it had downloaded files previously. Then $A$’s tracker site forwards this information to the tracker sites of those torrents where $q$ had downloaded files from. By doing so, the torrents that are created independently by different content providers are connected together to form an overlay network, as shown in Figure 4.14(a). The tracker overlay is actually an implementation of the inter-torrent relation graph presented in Section 4.4.2. Figure 4.14(b) shows that the connectivity degree (unweighted) of the tracker overlay is heavily skewed and similar to P2P overlays like Gnutella networks. Thus, many existing search algorithms can be used in the tracker overlay.

The tracker overlay provides a mechanism for inter-torrent collaboration. In the current architecture of BitTorrent systems, peers in different torrents cannot collaborate because they cannot find and communicate with each other. By using tracker overlay, peers in different torrents can exchange files that they have downloaded and balance their resource sharing. Our simulation shows that the tracker overlay can cover more than 99% torrents in the system.

\(^{12}\)Recently, BitTorrent begins to support trackerless torrents with DHT [5]. The inter-torrent relation graph can be maintained by this DHT in similar way as that in the tracker overlay.
In the tracker overlay architecture, the extra service load on the existing tracker sites is small. Assume a torrent has \( n \) peers at its peak time. Since the average number of torrents each peer involves is \( \frac{1}{1-p} \), the neighbor table size is \( O\left(\frac{n}{1-p}\right) \). Furthermore, the tracker overlay is fully decentralized and has no single point of failure. Tracker overlay has better fault-tolerance and scalability than a central server solution.

Tracker overlay also provides a built-in mechanism to search content among multiple torrents. Currently, BitTorrent users rely on Web-based search engines to look for the content they want to download.

### 4.5.2 Multi-torrent collaboration

BitTorrent assumes each peer is selfish, and a peer exchanges file chunks with those peers that provide it the best service. The incentive mechanism of BitTorrent systems is instant, because each peer must get corresponding benefit immediately for the service it
provides. In contrast, KaZaa and eMule use a participation level or ID stored in the system to trace and identify the contribution of users. Peers with a higher level or higher ID will have a higher priority to be served. Thus, the contribution of a peer under this mechanism will be rewarded in a long term instead of instantly. Although KaZaa and eMule systems are multi-file based and thus the popularity changes of a file have little affect on the service availability of its downloading, their long-term incentive mechanisms are not as effective as the “tit-for-tat” mechanism in BitTorrent. The downloading speed of eDonkey/eMule/Overnet is much slower than that of BitTorrent because peers in the P2P network usually share and download a large number of files, making the bandwidth available to each transfer much smaller than that in BitTorrent [5]. Furthermore, fraud prevention is also a big problem. For example, the participation level system in KaZaa has been cracked and thus one can set its participation level arbitrarily [8]. Compared with systems based on the long-term user reputation, the instant incentive mechanisms like “tit-for-tat” are simple, effective, and robust. Instead of going back to the long term incentive model of KaZaa and eDonkey systems, in this subsection, we propose an exchange based mechanism for instant collaboration among multiple torrents through the tracker site overlay, which still follows the “tit-for-tat” idea.

Our proposed inter-torrent collaboration strategy is as follows. First, if there exists a directed cycle among a number of torrents, such as torrents $A, B$ and torrents $B, C, D, E$ in Figure 4.14(a), then peers in these torrents can exchange file chunks through the coordination of the tracker site overlay. More specifically, a peer that needs service from peers in other torrents on the cycle does not have to serve these peers directly, because its contribution can be transferred to these peers along the cycle with the help of corresponding trackers. Since the contribution of each peer must be rewarded instantly, any fraudulent
behavior will be identified and punished at once. Second, when no such cycles exist for a peer $q$ who wants to get service from peers in other torrents, the peer can construct such a cycle as follows. Peer $q$ may join these torrents temporarily and download some chunks of the files, even if it does not want these files itself. Through the coordination of corresponding tracker sites, the peer can provide uploading service for these chunks only, and attribute its service contribution to the peers it wants to get service from, so that these peers can get benefit from the peers that $q$ serves and offer $q$ the service it needs. Thus, a directed neighboring cycle is constructed. We call this technique *bandwidth trading*. The basic idea is that the bandwidth can only be shared through content downloading/uploading. Since a file chunk can be served to multiple peers in the system, bandwidth trading is efficient and the overhead is trivial.

In such multi-torrent collaboration systems, a peer that has downloaded multiple files can get better service when downloading a new file (no matter it is popular or not) by serving the old files it has downloaded to the peer community, thus addressing the well-known problem of lacking incentives to seeds. Research [25] and [45] present a similar idea of using file exchange as an incentive for P2P content sharing. Different from these studies, our system aims to share bandwidth as well as content across multiple P2P networks.

### 4.5.3 Performance evaluation

We evaluate our system design through simulations with the tracker trace used in previous sections. In the simulations, we assume seeds use a fair service policy that does not prefer any peer in any torrent as long as it can serve. Figure 4.15(a) shows the downloading failure ratio in the current BitTorrent systems (without inter-torrent collaboration) and that in our proposed system (with inter-torrent collaboration), respectively. We only consider
torrents born in the initial period of the trace collection time in order to study the performance in their whole lifetime. In this figure, each point in the $x$-axis denotes a torrent, sorted in non-ascending order of the corresponding downloading failure ratio in the tracker trace (without inter-torrent collaboration). As shown in Figure 4.15(a), under inter-torrent collaboration, the downloading failure ratios in most torrents are actually zero or close to zero. Figure 4.15(b) shows the average downloading speeds of peers in different torrents at a certain time (12:00:01 on 2003-11-15). Each point in the $x$-axis denotes a torrent, sorted in non-ascending order of the average downloading speed of torrents at this time instant. The system with inter-torrent collaboration clearly provides a much better and more stable downloading service to clients. Figure 4.15(c) shows the peer contribution ratio (defined in Section 4.3) in two systems. In our proposed system, the peer contribution ratio is better balanced. These preliminary results demonstrate that our proposed system design, though without complicated credit systems, can enhance the current BitTorrent system significantly.

Figure 4.15: The performance analysis of our system
4.6 Conclusion

BitTorrent-like systems have become increasingly popular for content distribution and file sharing, and have contributed to a large amount of traffic on the Internet. In this chapter, we have performed extensive trace analysis and modeling to study the behaviors of such systems. We found that the existing BitTorrent system provides poor service availability, fluctuating downloading performance, and unfair services to peers. Our model has revealed that these problems are due to the exponentially decreasing peer arrival rate and provides strong motivation for inter-torrent collaborations instead of simply giving seeds incentives to serve longer. We propose the design of a new architecture where the tracker sites of different torrents are organized into an overlay to facilitate inter-torrent collaboration. Our trace-driven simulations have shown promising results. Some preliminary results of this work have been presented in [54, 55].
CHAPTER 5

PEER-TO-PEER ASSISTED MEDIA CACHING

5.1 Introduction

Delivering multimedia contents with high quality and low cost over the Internet is challenging due to the typical large sizes of media objects and the continuous streaming demand of clients. Currently, there are three representative solutions for media streaming on the Internet. First, special content delivery networks (CDNs) have been built to replicate media servers across the Internet to move the contents close to the clients, such as Akamai. This approach is performance-effective but not cost-effective. The second approach is to utilize existing proxies to cache media data, which is cost-effective but not scalable due to limited storages and bandwidths of centralized servers. The third approach is to build client based P2P overlay networks for media content delivery, which is highly cost-effective but does not guarantee the quality of service because the capacities (CPU, storage, and bandwidth) of peers can be heterogeneous and their availabilities can be transient.

Many researchers have delved in the proxy caching approach, a successful approach used for delivering text-based Web content on the Internet, to improve its performance for streaming media delivery. However, a full caching approach of media objects can quickly

\[13\text{http://www.akamai.com/}.\]
exhaust the limited proxy cache space. To handle the large sizes of media objects, researchers have developed a number of segment-based proxy caching strategies (e.g., [35] and [119]) to cache partial segments of media objects instead of their entireties.

Although the segment-based proxy caching technique has shown its effectiveness for media streaming, the quality of service it can provide is still not satisfactory to clients for the following reasons. First, the limited storage capacity of a proxy restricts the amount of media data it can cache for clients. In addition to the large size of media objects, research in [38] shows that the reference locality of multimedia objects is much less than that of Web pages. Thus, the cache space problem is much more significant in media proxy systems than in Web proxy systems. Second, the delivery of streaming media normally requires a dedicated reservation of continuous bandwidths for the clients. However, the highly demanded proxy bandwidths limit the number of clients to be served simultaneously. Furthermore, a proxy not only easily becomes a system bottleneck, but also forms a single point of failure, being vulnerable to attacks. On the other hand, the resources of bandwidth, storage, and CPU cycles are richly available and under-utilized among the clients. Thus, the P2P file sharing model is very attractive. However, directly borrowing this model for media streaming cannot guarantee the quality of streaming service for the following three reasons. First, the availability of demanded media data in each peer can be unpredictable because each peer caches and replaces media content independently without any coordination with other peers. Second, the availability of services can also be dynamic due to the transient nature of peers even though the data are always available. Third, the quality of services provided by multiple collaborative peers may not be sufficient for highly dynamic and bursty media streaming requests.
In order to address the scalability problem of proxy based techniques, and to deliver the high quality media content to clients, we present a novel and scalable segment-based P2P media delivering system by organizing the proxy and its clients in the same intranet into a P2P system. This system is called PROP abbreviated from our technical theme of “collaborating and coordinating PROxy and its P2P clients”. Our proposed system attempts to address both the scalability and the reliability issues of streaming media delivery in a cost-effective way. In such a system, the clients effectively coordinate and collaborate with the proxy to provide a scalable media storage and to actively participate in the streaming media delivery. Media objects are cached in segment units both in peers and in the proxy for the purposes of self-viewing and global sharing. Based on modeling and analysis of cache redundancy in our proposed system, we have designed replacement algorithms, aiming to approximately achieve the optimal distribution of media segments across the whole system. We have comparatively evaluated our proposed system by trace-driven simulations with synthetic workloads and with a real-life workload trace extracted from the media server logs in an enterprise network. Our experimental results show that PROP significantly improves the quality of media streaming and the system scalability. Specifically, the byte hit ratio can be improved up to 2.4 times over client based P2P systems and proxy based P2P systems. Our contributions in this work are threefold.

- The collaboration and coordination between the proxy and its P2P clients in our system address the scalability problem of proxy based technique, and also eliminate the concern of unstable quality of services by only relying on self-organized clients. These two system components are complementary to each other: the proxy provides a dedicated storage and reliable streaming services when peers are not available or
not capable of doing so, while peers provide a scalable storage for data caching and significantly reduce the service load of the proxy.

- To improve the reliability and maximize the utilization of cached data in each peer, we have proposed a model to analyze the cache redundancy in our peer-to-peer caching system where peers are free to come and go. Our modeling results give the optimal replica distribution in such a system, and provide the guidance to cache replacement policy design.

- According to the modeling and analysis, we have proposed segment based replacement policies for proxy and peers, which exploit the data locality and balance the load in the PROP system. Our objective is to keep popular media segments in the proxy for global sharing, and leave a certain space in each peer to cache those relatively unpopular segments. With this arrangement, both popular and unpopular segments are fairly treated, improving the overall hit ratios in the PROP system.

The remainder of the chapter is organized as follows. Section 5.2 discusses other related work. Section 5.3 presents the design and rationale of our proposed media streaming system. We evaluate the performance of our system and its alternatives in Section 5.4, and make concluding remarks in Section 5.5.

5.2 Related Work

Efforts have been made on P2P media streaming recently. Acharya and Smith [21] studied the collaboration among multiple proxy servers in an intranet to provide media streaming service. Chae et al. [32] studied the data replacement techniques for cooperative caching among a number of media proxies. Different from these studies, our PROP system
exploits the potential storages and processing capacities among media clients, and proposes corresponding data placement and replacement policies. Padmanabhan et al. [89] and Tran et al. [110] propose P2P multicast trees for live media streaming. Although multicast can be used for on-demand media streaming as well, the startup latency is non-trivial. Cui et al. [45] and Rejaie et al. [97] propose P2P streaming schemes for layer-encoded media. The quality of layer-encoded media streaming may not be guaranteed if some layers are not delivered in time, which cannot occur in segment-based systems. In weakly coupled P2P systems such as the ones presented in [89] and [97], users collaborate loosely with each other in a peer-to-peer fashion for on-demand media streaming, instead of being organized into a P2P overlay. Ip et al. [70] propose a cooperative proxy-client caching system for on-demand media streaming, which does not consider cache redundancy in the system. Hefeeda et al. [64] propose a structured P2P media streaming system, targeting media sharing on the global Internet, where each peer is the owner and distributor of its cached media, and there are no extra media providers in this system. In contrast, our system targets on-demand media streaming from professional and commercial media providers for clients in a large intranet, where each peer only serves its cached media as a member of the distributed caching system, instead of owning and distributing these media.

Aside from P2P based media streaming, the research on the proxy based and assisted media streaming and delivery techniques has been conducted. Particularly, proxy caching of streaming media has been explored in [21], [29], [57], [75], [103], [109], [119], [124]. Prefix caching and its protocol consideration as well as partial sequence caching are studied in [51], [103]. In video staging [124], a portion of bits from the video frames whose size is larger than a predetermined threshold is cut off and prefetched to the proxy to reduce the bandwidth on the server proxy channel. In [82], the proposed approach attempts to
select groups of consecutive frames by the selective caching algorithm, while in [85], the algorithm may select groups of non-consecutive frames for caching in the proxy. The caching problem for layered encoded video is studied in [75]. The cache replacement of streaming media is studied in [96], [109].

The proxy caching of streaming media has also been integrated with other techniques, to further improve cache efficiency. In [115], the batching, patching and streaming merging are combined with proxy caching. A circular buffer is used in [33], while in [34], a set of existing techniques are evaluated and the running buffer is efficiently utilized together with patching for efficiently delivering the media content.

5.3 System Design and Rationale

5.3.1 Infrastructure overview

The two main components of the PROP system are (1) the proxy and (2) all the client peers receiving the media streaming service, which are self-organized into a P2P overlay network. The proxy is the bootstrap site of the P2P system and the interface between the P2P system and media servers. When an object is requested for the first time or when no peer in the system is able to serve a streaming request, the proxy is responsible to fetch the requested media data from the remote server, divide the object into small segments, and cache them locally.

Each peer in the PROP system has three functionalities. First, a peer is a client that requests media data; second, a peer is a streaming server that provides media streaming service to clients. Each peer caches the media data in segments while its content accessing is in progress, and shares the cached data with other peers in the system. Third, a peer is also an index server that maintains a subset of indices of media segments in the system for
content location. Peers in our system are self-organized into a structured P2P overlay supporting a distributed hash table (DHT), which maps the identifier of each media segment to the index of the segment (see Section 5.3.2 for details). The P2P operations in our system are overlay independent though we use CAN [95] in our simulation.

In our system, the media segments and their corresponding indices are decoupled. In other words, they may be maintained by different peers. The index of a segment contains a location list of peers, each of which caches a copy of the media segment, and the access information of this segment, which is used for replacement operations. The segment locating is conducted in two steps: the first step is to route the request to the peer maintaining the index of the demanded segment, and the second step is to select a peer that caches a copy of the segment. Comparing with the central indexing solution like Napster \(^\text{14}\), distributed indexing is not only scalable but also out of the single-point-of-failure problem. Meanwhile, our approach is efficient and cost-effective for two reasons. First, the selection of a serving peer can be optimized according to the capacities and workloads of peers caching the demanded media data, because the index server maintains all access information of segments. Second, the cost of content locating is distributed over the P2P network so that the burden of P2P routing on each peer is trivial. Once the demanded segment is successfully located, the media streaming between the serving peer/proxy and the requesting peer becomes point-to-point.

### 5.3.2 P2P routing and media streaming

The distributed hash table supported by the P2P overlay stores the \((key, value)\) maps where each \(key\) is the identifier of a media segment and the corresponding \(value\) is the

\(^{14}\text{http://www.napster.com/}.$
index of the segment. The identifier of a media segment is a GUID (globally unique identifier) hashed from the URL of the media object and the offset of the segment in the object with SHA1 algorithm. In our system, each peer is assigned a key space zone when joining the system, and maintains the segment indices mapped to this zone. Joining P2P routing entails getting the key space zone and taking over the corresponding indices from a neighbor while leaving P2P routing entails handing over the segment indices and merging the key space zone to a neighbor [95].

The following operations on the distributed hash table are designed in our system for content locating and data management: publish, unpublish, request, update and notify. All these operations can be built on top of the common functionalities provided by distributed hash tables: put(key, value), get(key), and delete(key).

**Publishing and unpublishing media segments**

The publish(seg_id, location) operation publishes a cached copy of media segment in the P2P system, in which seg_id is the segment identifier, and location is the IP address and port number of the peer that caches the segment copy. Correspondingly, the unpublish(seg_id, location) operation unpublishes the copy of media segment stored in location. To publish or unpublish a segment, a peer routes its location and the seg_id to the target peer that maintains the segment index. Then the target peer put the location into or remove it out of the location list in the segment index. These operations correspond to the put or delete functions in DHT interface. Each segment index is created by the proxy when segmentation, and the index server is responsible for maintaining the consistency between the media segments and corresponding indices. A peer publishes a segment as soon as it caches the full segment and unpublishes a segment as soon as it deletes the segment.
A peer publishes all segments it caches when joining the P2P system and unpublishes all segments it caches when leaving the P2P system.

**Requesting and serving media segments**

A peer requests media data segment by segment, and searches in its local cache first. If the local search fails, it calls the `request(seg_id, URL)` operation, which requests a segment of the object designated by the `URL`. When a peer requests a media object that it does not cache, it routes the `URL` to the target peer that maintains the key space zone that the identifier of the object’s first segment (its `seg_id`) is mapped to. This operation corresponds to the `get` function in DHT interface. If the corresponding index does not exist, meaning the object is requested for the first time, the target peer sends a request to the proxy, which fetches the requested object from the media server, and creates the index and publishes the object. Then the target peer routes the proxy’s location back to the requesting peer, redirecting the peer to the proxy to get the media data. If the target peer finds the segment index, but the location list is empty, the target peer sends a request to the proxy, which fetches the segment and publishes it. The first five steps in Figure 5.1(a) show such
an example. If the location list is not empty, the target peer checks the validation of each location link, then returns the location of the peer with the maximal available bandwidth to the requesting peer. The first three steps in Figure 5.1(b) show such an example. Then the serving peer provides the requested data to the requesting peer, as the last step shown in Figure 5.1(a) and Figure 5.1(b). A client can buffer the next segment when the current segment is played back. If a serving peer wants to leave the P2P system before the current streaming terminates, it must push the rest of the segment to the requesting peer before exiting the P2P system.

**Updating segment popularity and utility values**

PROP uses the popularity and utility values of segments to manage cached data (see Section 5.3.4 in details). These values depend on the access information and number of copies of corresponding media segments in the system. When the proxy or a peer finishes serving a segment streaming task, it calls \texttt{update(seg.id, access.info)} operation, which routes the access information to the target peer maintaining the segment’s index, and then the target peer updates the corresponding information items. When the segment popularity or utility values change, the index server notifies all peers that cache the segment new values by \texttt{notify(peerset, seg.id, value)} operation, where \texttt{peerset} is the peers in the location list of the segment index, and \texttt{value} is the popularity or utility value of the segment designated by \texttt{seg.id}. These operations correspond to the \texttt{put} function in DHT interface.

**Message routing overhead**

In PROP, for each segment a client requests, a \texttt{request} and an \texttt{update} message are generated. For each segment replica that is cached or evicted from cache, a \texttt{publish} or
unpublish message is generated. Although a notify operation may generate multiple messages, it can be postponed if the popularity or utility value changes little. Furthermore, our replacement algorithm can keep the location list of the segment index in a moderate size (see Section 5.3.4 for details). Thus, the routing overhead in PROP is trivial compared to the media data transfered. Our simulation shows it is less than 1% of the streaming media data (including TCP/IP and Ethernet headers) for 100 KB segment size. To further reduce the routing overhead, we can increase the segment size, or use variable-sized segmentation such as exponential segmentation [119] and adaptive and lazy segmentation [35].

5.3.3 Redundant cache model

Due to the dynamical peer arrivals and departures, an object may not be available if it is only cached by a single peer. A solution is to let a peer that wants to go offline transfer the object it maintains to other peers, such as the home mode of Squirrel [71]. However, Squirrel is a peer-to-peer based Web proxy system, where the objects are usually very small. In contrast, the sizes of multimedia objects are much larger. Frequently transferring cached objects among peers is not desired since it consumes more bandwidth and storage. Furthermore, a peer may fail due to many reasons. In our solution, we use two methods to provide the reliable service: (1) the proxy can maintain the most popular objects as a backup service; (2) an object (or a segment) can be cached in multiple peers to provide redundancy.

Assume the popularity (the reference probability) of media segments follows a Zipf-like distribution (for stretched exponential distribution, the analysis is similar). If we rank all media segments in the descending order of their popularities, the popularity of the \( i \)-th media segment, \( p_i \), can be expressed as
where $A = \sum_{i=1}^{m} \frac{1}{i^\alpha}$, $m$ is the number of all media segments, and $\alpha$ is a constant.

Assume there are $N$ peers and $m$ media segments in the system. Each peer contributes a storage of size $C$ and has a failure probability $q$ ($q < 1$) when serving a segment. Assume the size of each segment is 1 unit of storage size. For segment $i$ cached in $r_i$ peers, the cache failure probability is $q^{r_i}$. Our objective is to find the distribution of $\{r_i\}$, which can minimize the total number of failed requests of all media objects, with the constraint of the total cache size

$$\{r_i\} = \arg \min_{\{n_i\}} \{\sum_{i=1}^{m} p_i q^{n_i}\},$$

where

$$\sum_{i=1}^{m} n_i = CN.$$  \hfill (5.3)

We have

$$\sum_{i=1}^{m} p_i q^{n_i} \geq m (\prod_{i=1}^{m} p_i q^{n_i})^{\frac{1}{m}}.$$  \hfill (5.4)

where the minimum value holds when

$$p_1 q^{r_1} = \ldots = p_i q^{r_i} = \ldots = p_m q^{r_m} = B.$$  \hfill (5.5)

For segment $i$, we have

$$p_i q^{r_i} = B,$$  \hfill (5.6)

$$r_i = \frac{\log p_i - \log B}{\log \frac{1}{q}} = \frac{\log A - \alpha \log i - \log B}{\log \frac{1}{q}}.$$  \hfill (5.7)
Since
\[ \sum_{i=1}^{m} r_i = m \log A - \alpha \log m! - m \log B \log \frac{1}{q} = CN, \] (5.8)
we have
\[ \log B = \log A - \alpha \frac{\log m!}{m} - \frac{CN}{m} \log \frac{1}{q} \]
\approx \log A - \alpha (\log m - 1) - \frac{CN}{m} \log \frac{1}{q}, \] (5.9)
\[ r_i \approx \frac{\log p_i}{\log \frac{1}{q}} + \left( \frac{CN}{m} + \alpha \frac{\log m - 1}{\log \frac{1}{q}} - \frac{\log A}{\log \frac{1}{q}} \right) \]
\[ = -\frac{\alpha}{\log \frac{1}{q}} \log \frac{i}{m} + \left( \frac{CN}{m} - \frac{\alpha}{\log \frac{1}{q}} \right). \] (5.10)

When \( \frac{CN}{m} \geq \frac{\alpha}{\log \frac{1}{q}} + 1 \), we have \( r_m = \frac{CN}{m} - \frac{\alpha}{\log \frac{1}{q}} \geq 1 \). That is, when the total cache size of all peers is large enough so that all media segments can be cached, increasing cache size leads to evenly increase the number of copies for each segment. Figure 5.2 shows the number of copies for different segments in the system, where 1,000 peers cache \( 10^6 \) different media segments, with each peer contributing 1,500 storage units (“×” line) and 4,500 storage units (“+” line), respectively.

In reality, it is difficult to get the values of parameters such as \( \alpha, q, \) and \( A \). In Section 5.3.4, we will present a heuristic distributed algorithm to achieve such a distribution without knowing the values of these parameters.
5.3.4 Cache management

In PROP system, the proxy serves as a persistent cache site, but the storage size is limited. On the other hand, the total storage space contributed by peers is huge but the available contents change dynamically because peers come and go frequently. To fully utilize the storage and to improve the streaming service quality, we propose efficient replacement policies for both proxy and peers based on the global information of segment accesses using the following parameters.

- $T_0$, the time when the segment is accessed for the first time;
- $T_r$, the most recent access time of the segment;
- $S_{sum}$, the cumulative bytes that the segment has been accessed;
- $S_0$, the size of the segment in bytes;
- $n$, the number of requests for this segment;
- $r$, the number of replicas of the segment in the system.

**Popularity-based proxy replacement policy**

The proxy takes over the streaming service whenever the requested media segment cannot be served by any peer in the system. Thus, the proxy should hold those popular media objects to minimize the performance degradation due to peer failures. We use a popularity-based replacement policy instead of Least Recently Used (LRU) policy that is commonly used in Web proxies, because LRU is not efficient for file scan operations, which are typical in media streaming services, and can only exploit the locality of reference to the proxy instead of the whole system. We define the *popularity* of a segment as
\[ p = \frac{S_{\text{sum}}}{S_0} \frac{T_r - T_0}{T_r - T_0} \times \min(1, \frac{T_r - T_0}{t - T_r}), \]  

(5.11)

where \( t \) is the current time instant, \( \frac{S_{\text{sum}}}{S_0} \) represents the average access rate of the segment in the past, normalized by the segment size, and \( \min(1, \frac{T_r - T_0}{t - T_r}) \) represents the probability of future access: \( \frac{T_r - T_0}{n} \) is the average time interval of accesses in the past, if \( t - T_r > \frac{T_r - T_0}{n} \), the possibility that a new request arrives is small; otherwise, it is highly possible a request is coming soon. The segment with the smallest popularity is chosen as the victim to be replaced when the proxy cache is full. Considering both the recent access and past access information, the proxy can cache the most useful media data for clients.

Study [35] uses a similar approach to estimate the popularity of objects in proxy caching systems. However, our system architecture and replacement policies are completely different from these systems. In order to quickly collect space for new media objects and to reduce the cache management overheads, the exponential segmentation approach [119] and the adaptive-lazy segmentation approach [35] have been proposed. In contrast, due to the distributed indexing and caching service provided by peers, the load of proxy in the PROP system is much smaller than that in traditional proxy caching systems. PROP uses an evenly sized, fine granulated segmentation with low overheads, and our simulations show that finer granulated data management is more efficient than coarser granulated data management for cache utilization.

**Utility-based replacement policy for client peers**

Independently exploiting reference locality on each client side is neither efficient from the system’s perspective nor effective from the user’s perspective. First, due to the client access patterns, the popular objects get more accesses from peers and thus have more copies...
cached in the system, which are already cached on the proxy side. Keeping those unnecessary copies of popular objects degrades the cache efficiency since the cache space could have been used to cache other objects. Second, the locality on each client side is limited and the cached data is prone to be flushed in a long streaming session if LRU replacement is used. Third, the segments of a media object may be cached in a single peer, thus the data availability is very sensitive to the peer failure and leaving. On the other hand, the reference locality of all clients is much more significant than that of a single client and the difference between the access latency in an intranet and the local disk is not important for media streaming. Thus, although the cached data in PROP system are distributed across all client peers, they should be managed collectively to achieve best utilization.

The purpose of our peer replacement policy is to use an adaptive mechanism to dynamically adjust the distribution of media data cached in the system. We have three objectives:

- The distribution of cached segments should be close to the optimal distribution (described by Equation 5.10) to maximize the cache utilization.
- Different segments of the same object should be cached in different peers. Thus, the failure of individual peers can only affect part of the media object.
- The segment distribution should not involve extra data transferring.

To achieve these goals, we design a heuristic algorithm as follows. Our peer replacement policy is designed to replace both those media segments with diminishing popularities because they rarely get accessed, and those popular media segments with too many copies being cached. As a result, peers accessing media objects completely will cache the latter segments and evict the beginning segments of the objects because they are more popular and have more replicas in the system than the latter segments. Peers that access only the
beginning segments will cache the beginning segments. Thus, naturally a peer will cache only a few segments of each object it has accessed, while the segments of each object are distributed across many peers in the system according to their popularities, reducing the negative effects caused by peer failures. The above operation needs no extra data transferring except the data requested by users.

To achieve the optimal distribution, for the peer replacement policy, we define the utility function of a segment as

$$u = \frac{(f(p) - f(p_{\text{min}})) \times (f(p_{\text{max}}) - f(p))}{r^{\alpha + \beta}}$$

where $p$ represents the popularity of the segment, $p_{\text{min}}$ and $p_{\text{max}}$ estimate the minimum and maximum of segment popularities in the P2P system respectively, $r$ is the number of replicas of this segment in the system, and $f(p)$ is a monotonic non-decreasing function, which is called the adjustment factor of the utility function. The desired distribution of media data in PROP system is $r \propto f(p)$, as described in Equation 5.10. In our system, we choose $f(p) = \log p$ and $\alpha = \beta = 1$, and we will show the performance of different adjustment factors in our evaluation. The values of $p_{\text{min}}$ and $p_{\text{max}}$ can be maintained by the proxy and propagated across the P2P overlay by a flooding when necessary. The term $\frac{f(p) - f(p_{\text{min}})}{p^\alpha}$ captures the segments with small popularities and large numbers of replicas while $\frac{f(p_{\text{max}}) - f(p)}{p^\beta}$ captures the segments with large popularities and large numbers of replicas. These two kinds of segment replicas should be replaced by the replicas of segments with moderate popularities but a small number of copies. So in our system, we choose those segments with the smallest utility value as the victims to be replaced when a peer’s cache is full.

When the cache system reaches its optimal distribution of segment replicas, we have

$$u = (\log \frac{1}{q})^2 (\frac{r_{\text{max}}}{r} - 1)(1 - \frac{r_{\text{min}}}{r}).$$

(5.13)
Let $\frac{du}{dr} = 0$, we have

$$r_0 = \frac{2r_{\text{max}}r_{\text{min}}}{r_{\text{max}} + r_{\text{min}}}.$$ \hspace{1cm} (5.14)

That is, the utility $u$ reaches its maximum when $r = r_0$. Figure 5.3 shows the relation between utility and number of copies. The figures indicates that our replacement policy can effectively reduce the number of popular segment copies that can be accumulated quickly (so as to better utilize the cache space), as well as reduce those unpopular segments when they are cached more than needed. Thus, the data distribution is optimized naturally along with the progress of media accessing, and the efficiency of cache utilization is maximized.

Our simulation shows that the utility-based replacement policy has more benefits for workload with low reference locality than workload with high reference locality, due to the segment distribution it results in the PROP system. This advantage is very attractive for streaming media delivery systems, since the reference locality in streaming media workload is much smaller than that in Web caching workload [38].

Figure 5.3: Utility variations of media segments
5.3.5 Streaming task dispatch

In PROP, a streaming session is divided into a number of streaming tasks in a moderate granularity determined by the segment size (we use 100-500 KB in our simulation). These tasks can be served by different peers and it is the index server’s responsibility to dispatch streaming tasks to different streaming servers. Instead of having multiple peers to collaboratively serve media streaming for a client like [64], only one streaming server is needed at a time in PROP. The failure of the streaming server has little impact on the client, and the media player only needs to buffer one segment for a smooth playback. Furthermore, the streaming tasks can be dispatched fairly and efficiently based on the information in segment indices and the quality of the streaming servers. Currently we only use the available bandwidth of the serving peer as the criterion to dispatch streaming tasks for load balance, and always dispatch streaming tasks to client peers first to decrease the proxy burden. We leave the dispatch optimization and data prefetching as a future work.

5.3.6 Fault tolerance

When a peer fails, both the media data it caches and the segment indices it maintains are lost. In PROP, each peer periodically checks the validation of the replica location links in the segment indices it maintains and simply removes dead links. The loss of segment indices can be recovered by using the recovery mechanism of distributed hash table, e.g., CAN supports multiple realities to improve fault tolerance of routing [95].

When the proxy fails or is overloaded so that it cannot fetch data for clients, the requesting peer connects to the media server directly, fetches data and caches them locally until the proxy is recovered. Since the indices of media segments are distributed in the P2P system, the content locating mechanism still works and the system performance degrades
gracefully. Compared with the solution of maintaining a central index in the proxy like the browser-aware proxy system [120], our system not only removes the single point of failure, but also significantly reduces the burdens of index maintenance and segment locating on the proxy.

5.4 Performance Evaluation

By using trace-driven simulation, we have comparatively evaluated the performance of our proposed system with different proxy cache sizes and different peer cache sizes, and showed the different roles of the proxy and peers. When the total cache size of peers is zero, the system is equivalent to a proxy caching system. When the cache size of the proxy is zero, the system is equivalent to a client based P2P system without proxy server. In the following evaluations, if not specified, the default replacement policy on the proxy is popularity-based, and the default replacement policy on peers is utility-based for PROP system, or LRU for client based P2P system, respectively.

We use the following metrics in our evaluations. The major metric, streaming jitter byte ratio, is defined as the amount of data that is not served to the client in time by the proxy and peers, thus causing the potential playback jitter on the client side, normalized by the total bytes all clients demand. The second metric is the delayed start request ratio, which is defined as the number of requests suffering startup delays, normalized by the total number of requests. These two metrics reflect the QoS of the media streaming to the client. The third metric we use is the byte hit ratio, which is defined as the total bytes of media data served by the proxy and peers, normalized by the total bytes of media data all peers consume. Byte hit ratio represents the storage utilization and outgoing network traffic reduction of the system.
<table>
<thead>
<tr>
<th>Trace Name</th>
<th># of Req.</th>
<th># of Obj.</th>
<th># of Peers</th>
<th>Size (GB)</th>
<th>λ</th>
<th>α</th>
<th>Obj. Length</th>
<th>Trace Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>11559</td>
<td>400</td>
<td>1663</td>
<td>24</td>
<td>-</td>
<td>-</td>
<td>6-131 m</td>
<td>10 days</td>
</tr>
<tr>
<td>PART</td>
<td>15188</td>
<td>400</td>
<td>376</td>
<td>51</td>
<td>4</td>
<td>0.47</td>
<td>2-120 m</td>
<td>1 day</td>
</tr>
<tr>
<td>WEB</td>
<td>15188</td>
<td>400</td>
<td>376</td>
<td>51</td>
<td>4</td>
<td>0.47</td>
<td>2-120 m</td>
<td>1 day</td>
</tr>
</tbody>
</table>

Table 5.1: Workload summary of PROP evaluation

5.4.1 Workload summary

Table 5.1 outlines some properties of the workloads. REAL denotes the workload extracted from the server logs of HP Corporate Media Solutions, covering the period from May 1 through May 10, 2001. WEB and PART are two synthetic workloads for media viewing in the Web environment, where we assume the popularities of media objects follow Zipf-like distribution ($p_i = \frac{f_i}{\sum_{i=1}^{N_i} f_i}$, $f_i = \frac{1}{i^\alpha}$) and the request arrivals follow Poisson distribution ($p(x, \lambda) = e^{-\lambda} \frac{\lambda^x}{x!}$, $x = 0, 1, 2...$). These assumptions and the corresponding parameters for two synthetic workloads are based on the existing workload characterization studies [37, 38], which indicate that Internet media accesses exhibit substantially different characteristics from common Web accesses. Recently we have further confirmed this distribution in [58]. WEB denotes complete viewing scenario while PART denotes partial viewing scenario in the Web environment. In PART, 80% of the requests only view 20% of the objects and then terminate.

The reference frequency of media objects in the three different workloads is shown in Figure 5.4. REAL has a steeper Zipf distribution because the accesses in this workload are limited in the internal media servers of HP Corporation only (paper [37] presents the measurement and analysis of this workload in details). The reference locality of objects in
the PART workload is the same as that in the WEB workload while the reference locality of segments is higher than that in the WEB workload because in the PART workload, the initial parts of media data are accessed more frequently than the latter parts. Thus, these three workloads represent three kinds of access patterns with different reference localities: for media segments, the order of reference localities is WEB < PART < REAL.

For the REAL workload, we set the bandwidth of the link between the proxy and media server as the clients’ available bandwidths in the trace, ranging from 0 bps (unreachable) to 6.248 Mbps. The media encoding rate in the REAL workload ranges from 6.9 Kbps to 2.055 Mbps. For synthetic workloads, we choose the bandwidth of the link between the proxy and media server randomly from 0.5 to 2 times the encoding rate of corresponding media objects. In our simulation, the segment size for the REAL workload is 100 KB, and the segment size for the PART and WEB workload is 500 KB. We assume the bandwidth of the intranet is broad enough for media streaming. We also assume each peer is online randomly, and can serve at most 5 clients at the same time, which is the common configuration in P2P software such as BitTorrent [41]. The caches of the proxy and peers are empty.
at the beginning of our simulation, and each peer’s cache size is proportional to the amount of media content it accesses in the workload. For client based P2P system model without proxy caching, each peer fetches media data individually while it caches its accessed data and publishes them in the system in the same way as our proposed system.

### 5.4.2 Performance results

To simulate the system performance in real environments, in our experiments, the peer arrival and departure to the system are assumed to be random. We believe this reflects (or at least it is close to) the behavior of real peers. Note that different from the Web accesses, in P2P systems, there is no consensus on the peer arrival and departure patterns. We first present the overall performance of PROP under different proxy cache sizes and peer cache sizes, then evaluate the performance improvement of our proposed proxy replacement and peer replacement policy.
Overall performance

Figure 5.5 shows the performance results of the three different system models (PROP, proxy caching, and client based P2P system) on the REAL workload. We selected the proxy cache size as 10%, 3%, and 1% of the total accessed media data in the workload, and ranged the total peer cache size from 0 to about 3 times the total accessed media data. In each figure, “no proxy caching” denotes the performance results of the client based P2P system. The $y$ axis denotes the performance results of the proxy caching system since the total peer cache size is zero. The results in Figure 5.5 show that our proposed system can significantly improve the QoS of media streaming and the byte hit ratio with the increase of peer cache size. For example, compared with the proxy caching system, our system can reduce up to 87% streaming jitter bytes as shown in Figure 5.5(a), reduce up to 82% delayed start requests as shown in Figure 5.5(b), and increase the byte hit ratio up to 2.4 times as shown in Figure 5.5(c). This is because more media data can be cached in the system due to the more storage contributed by peers, and thus improves both the byte hit ratio and QoS of media streaming correspondingly.

Figure 5.5 also shows that the proxy plays an important role in P2P media streaming system. The QoS and the byte hit ratio are improved significantly with the increase of the proxy cache size. Compared with the client based P2P system, our proposed system with a proxy capable of caching 1% of all media data can reduce up to 93% of streaming jitter bytes, up to 88% of delayed start requests, and the byte hit ratio can be as high as 85%, as shown in Figure 5.5(a), 5.5(b), and 5.5(c) respectively. In our system, the streaming performance can be effectively improved as we increase the proxy size. The reason is that a peer is not a dedicated server. It comes and leaves randomly and can only serve a small number of clients simultaneously. Thus, it is possible that a client cannot be served
immediately by peers that cache the requested data. On the other hand, the proxy provides a persistent cache and dedicated service to all clients; it takes over the streaming service whenever the peers are not capable of providing the streaming service, ensuring the quality of media streaming and reducing the outgoing bandwidth cost.

Figure 5.6 and Figure 5.7 show the performance results with the PART workload and the WEB workload respectively. As shown in these figures, the performance results with these two workloads have similar trends to that with the REAL workload. At the first glimpse, it is surprising to find that the performance of workload WEB is slightly better than that of workload PART, which does not seem to be intuitive since the reference locality of media segments in the PART workload is greater than that in the WEB workload. Dissecting the two workloads, we find that in the WEB workload, the total amount of bytes consumed by all clients is about 2.8 times greater than that in the PART workload, while the total amount of media objects is nearly the same as that in PART. Thus, the compulsory misses in the WEB workload are relatively smaller than those in the PART workload. Comparing the performance results of the PROP system evaluated by the three workloads, we can see that
the proxy plays a more important role for workload with smaller reference locality. For example, under the same condition, when proxy cache size increases, the streaming jitter reduction of the WEB workload is up to 4 times greater than that of the PART workload, the delayed startup reduction of the WEB workload is up to 2 times greater than that of the PART workload, and the byte hit increase of the WEB workload is up to 64% greater than that of the PART workload. This confirms the effectiveness of our design of the PROP system: the proxy and its P2P clients should collaboratively provide media streaming service.

Proxy load changes

To further evaluate the collaboration and coordination between the proxy and its P2P clients, we have measured and observed the proxy load changes of our proposed system with different proxy cache sizes and peer cache sizes. The proxy load includes the total amount of bytes it serves peers and the total amount of bytes it fetches from the media server. Figure 5.8(a), 5.8(b), and 5.8(c) show the proxy load for REAL, PART, and WEB,
respectively. In these figures, the proxy load is normalized by twice of the total bytes all peers consume, the maximal load that is possible on the proxy. Increasing the cache size of peers can significantly reduce the proxy load due to the streaming service provided by peers in the system. The proxy load reduction is up to 72% for the REAL workload compared with the proxy caching system, as shown in Figure 5.8(a). We also observe that the proxy load decreases with the increase of the proxy cache size. This is because when the proxy’s cache size increases, more media data can be cached, which reduces the outgoing fetching load on the proxy.

Figure 5.8(b) and Figure 5.8(c) show similar trends when proxy cache size increases or peer cache size increases. Comparing the proxy load on the three different workloads, we also observe that the proxy cache size is more sensitive to workload with smaller reference locality. For example, under same conditions, when proxy cache size increases, the proxy load reduction of the WEB workload is up to 50% greater than that of the PART workload.
Proxy replacement policy

Several proxy caching techniques for media streaming have been proposed, such as exponential segmentation [119] and adaptive and lazy segmentation [35]. Both techniques use variable segment size to improve byte hit ratio and to reduce the cache management overheads. These techniques can be used in PROP as well. However, due to the service load taken by clients, the service load of the proxy in PROP system is much smaller than that in traditional proxy caching systems. Therefore finer granulated segmentation is feasible in the PROP system for better cache utilization. In this section, we compare the performance of popularity-based replacement with that of LRU replacement using uniform segment size to show the advantage of popularity-based replacement. Given the existence of many variants of the LRU based replacement policies, we select LRU for comparisons because LRU is the most commonly used policy in the deployed proxy cache such as SQUID \(^{15}\). Due to this reason, we assume the LRU based replacement policy for peers too in the next subsection.

\(^{15}\)http://www.squid-cache.org/.

Figure 5.9: The comparison between popularity-based replacement and LRU replacement
Figure 5.9(a), 5.9(b), and 5.9(c) show the comparisons of byte hit ratios between popularity-based replacement and LRU replacement for three workloads REAL, PART, and WEB, respectively. For PART, the byte hit ratio of popularity-based replacement is up to 23 percentage points higher than that of LRU (about 3 times the byte hit ratio achieved by LRU). For WEB, the byte hit ratio improvement of popularity-based replacement over LRU is even higher, up to 26.5 percentage points (about 4.8 times byte hit ratio achieved by LRU). The improvement of the REAL workload is not so significant because the reference locality in this workload is relatively high, so that the potential improvement of byte hit ratio is relatively small. PART. LRU replacement is not efficient for media streaming proxy because the data access pattern of media streaming is sequential in most cases, i.e., a user tends to view media from the beginning until the end. This access pattern is similar to file scanning, for which the inefficiency of LRU is well known [73]. With LRU, the latter segments of a media file will replace the initial segments when the cache is full, which is more likely to be accessed in the future. The popularity-based policy, on the other hand, performs replacement operations based on the probability of the future accesses of media segments, maximizing the utilization of proxy cache.

**Peer replacement policy: overall performance**

To further evaluate the efficiency of resource management of our proposed system, we compared the performance of our global caching with utility-based replacement on each peer to that of independent caching with LRU replacement on each peer, and then discussed the performance of different adjustment factors for utility-function.

Figure 5.10, 5.11, and 5.12 show the comparisons between the performance of the system with our utility-based peer replacement policy and that of the system with the LRU peer replacement policy, for the three workloads REAL, PART, and WEB, respectively. For
the REAL workload, we only show the results of the system without proxy caching. For the PART and WEB workloads, we only show the results of the system with proxy caching 5% of the total accessed media data, using the popularity-based replacement policy on the proxy. For other proxy cache sizes, similar performance results are achieved.

From these figures, we can see the streaming jitter bytes and delayed start requests of the utility-based replacement policy decrease much faster than those of the LRU replacement policy, while the byte hit ratio of the utility-based policy increases much faster than that of the LRU policy, indicating our utility-based replacement policy is much more effective than the LRU policy. For the REAL workload, compared with the system using LRU, the system using the utility-based policy can reduce up to 36% streaming jitter bytes and up to 42% delayed start requests, and the improvement of byte hit ratio is as high as 59%, as shown in Figure 5.10(a), 5.10(b) and 5.10(c), respectively. For workload PART, the streaming jitter reduction and the delayed start request reduction is up to 41% and 56%, respectively, and the improvement of byte hit ratio is up to 19%, as shown in Figure 5.11(a), 5.11(b) and
Figure 5.11: Replacement policy comparisons on the PART workload (with proxy caching 5% of the total accessed media data in the system)

5.11(c) respectively. For workload WEB, the streaming jitter reduction and the delayed start request reduction is up to 57% and 61%, respectively, and the improvement of byte hit ratio is up to 24%, as shown in Figure 5.12(a), 5.12(b) and 5.12(c), respectively.

The ranking order of overall performance improvements for the three workloads is REAL < PART < WEB, in the reverse order of reference localities of the three workloads presented in Section 5.4.1, showing significant advantage of utility-based replacement. Meanwhile, although REAL has the highest reference locality in the three workloads, the maximal improvement of byte hit ratio of the REAL workload in Figure 5.10(c) is higher than that of PART workload in Figure 5.11(c) and WEB workload in Figure 5.12(c), because there is no proxy caching for the case of REAL workload in Figure 5.10(c), evidencing the important role of proxy in our PROP system. The utility-based replacement policy can greatly reduce delayed start requests with only a small cache size in each peer because the initial segments of media objects are generally more popular than the latter segments and have greater possibility to be cached. On the contrary, the initial segments
Peer replacement policy: utility function

Figure 5.13(a), 5.13(b), and 5.13(c) show the byte hit ratios of systems with five different utility functions for workload REAL, PART, and WEB, respectively. For REAL, the proxy caches 3% of the total accessed media data. For PART and WEB, the proxy caches 5% of the total accessed media data. With simple utility function, the system tries to cache media data as much as possible. Each segment is cached at most in one place, and the more popular segments have higher priorities to be cached. For other utility functions, the adjustment factors are $p^2$, $p$, $p^{1/2}$ and $\log p$, respectively, in which $p$ is the segment popularity.
The results of all three workloads show the same ranking order for the performance of the five adjustment factors: simple < $p^2 < p < p^{1/2} < \log p$, in terms of byte hit ratio. For example, for WEB, the maximal byte hit ratio improvement with adjustment factor $\log p$ over that with adjustment factor simple, $p^2$, $p$, and $p^{1/2}$ are 18.4, 9.51, 2.61, and 0.56 percentage point, respectively. For REAL and PART, the corresponding byte hit ratio improvements are smaller because the reference localities of these two workloads are greater than that of WEB, but the ranking orders are the same.

The system with simple utility function achieves the worst performance because caching only one copy of each media segment cannot ensure data availability due to the random joining and leaving of clients. Utility functions with adjustment factors $p^2$, $p$, $p^{1/2}$ and $\log p$ cache multiple copies of media segments based on their popularities. Maintaining the number of copies of media segments proportional or polynomial to their popularities will quickly exhaust the available cache size, so that the segments with moderate popularities cannot be cached. Although significantly reducing the proxy’s service burdens, maintaining too much replicas of popular segments is not efficient for storage utilization,
since these popular segments are more than enough in the system. On the other hand, the
order of logarithmic function is lower than that of any polynomial function. PROP uses
log \( p \) as the adjustment factor of the utility function, which reflects the number of seg-
ments replicas in the optimal distribution, as analyzed in Section 5.3.3 and 5.3.4. Thus, the
storage can be utilized more efficiently and the proxy burden can still be relieved with a
small number of redundant popular segment replicas in the system. Our simulation results
confirm the above analysis. As shown in Figure 5.13, lower level polynomial adjustment
factors perform better than higher level polynomial adjustment factors, but still not so good
as logarithmic adjustment factors.

Figure 5.14, 5.15, and 5.16 show the distribution of the number of replicas of media
segments with different popularities under utility-based replacement policy on the REAL,
PART and WEB workload, respectively. For REAL, the proxy caches 3% of total accessed
media data. For PART and WEB, the proxy caches 5% of total accessed media data. The
x axis denotes segments, sorted by the corresponding popularity values. These figures
Figure 5.15: The Adaptiveness of Utility-based Replacement on the PART workload

show that the segment replica distribution resulted from our utility-based replacement is close to the optimal distribution of segment replicas based on Equation 5.10 (a logarithmic function). The number of replicas for popular segments is a little smaller than the optimal one. However, this is not a problem in PROP since the proxy can take over the streaming requests. Furthermore, the popular segments can be quickly replicated in the system due to the high request rate. For replacement with adjustment factor $p$, the number of popular segments is much larger than the optimal value, preventing low popular segments from being cached. These trends are shown consistently in all three workloads. In general, although the data placement in PROP is passive (a client only caches the data it has viewed), our utility-based replacement can adaptively maintain the data cached in the system in the desired distribution for media streaming service.
5.4.3 Implications of our performance evaluation

Our intensive trace-driven performance study provides insights into the PROP system design, and impact of P2P sharing on the streaming system. We briefly summarize the implications of our performance evaluation.

- We have shown that our P2P assisted proxy system significantly improves the quality of streaming service mainly because the caching storage in PROP has been effectively and highly enhanced. Thus, media segments can be timely and smoothly delivered to any end user in the system either by other end users or/and by the proxy collaboratively.

- Our study shows that an increment of proxy cache size can be much more effective to the performance improvement of streaming content delivery than that of text-based Web content delivery. Our P2P approach well responds the high demand of large caching space for streaming contents.
The involvement of P2P segment sharing among users has significantly reduced the traffic to the proxy. This provides us with an opportunity of making data replacement in the proxy at a fine grain level (a segment-based rather than an object-based method). We show that a segment-based replacement method can improves proxy cache utilization.

One important factor to improve the efficiency of the PROP system is the coordination of segment replacement among peers and the proxy. Without such a mechanism, popular media objects/segments can be unnecessarily duplicated in both proxy and many client caches, wasting cache space. We show that segment replacement in each client based on our utility function effectively coordinates segment allocations in the global PROP caching space.

5.5 Conclusion

Existing Internet streaming media delivering techniques are either based on a client-server model, such as proxy caching and server replications by CDNs, or based on a client based P2P structure. The disadvantage of the client-server model is its limited scalability and high cost, while the disadvantage of a client based P2P system is its unreliable quality of streaming media delivery due to the dynamic nature of peers. In this study, we propose P2P assisted proxy systems to address these two limitations. In our system, the proxy is a member of the P2P network managed by the distributed hash table. In addition, the proxy also plays an important and unique role to ensure the quality of media delivery due to its dedicated and stable nature. To improve the cache utilization, we have proposed a model and designed the replacement policies for the collaboration and coordination between the proxy and clients accordingly, making the entire streaming media system both
performance-effective and cost-efficient. We have conducted extensive experiments to evaluate various aspects of our design, and the results show that our system can achieve 2.4 times improvement in byte hit ratio compared to client based P2P systems and proxy based P2P systems.
CHAPTER 6

CONCLUDING REMARKS

Measurements and modeling are critical to understand the highly dynamic Internet systems and Internet traffic. In this dissertation, we have analyzed the access patterns of Internet media systems and studied effective system designs for large scale media delivery through measurements and modeling. We have conducted a series of large scale measurements on the Internet to investigate the effectiveness of current media delivery systems and approaches, from the perspective of delivery quality and resource utilization. We found current media systems tend to over-supply or over-utilize server hardware and network bandwidth to provide high quality media service, which is not a scalable approach for serving the explosively increasing media traffic on the Internet. Motivated by the state-of-the-art of Internet media delivery, we have systematically studied the access patterns of a large variety of Internet media systems to exploit the temporal locality among media requests for efficient and high performance system design. Our study shows that Internet media access patterns follow the stretched exponential distribution with clear physical meanings. With this kind of access patterns, the performance of media caching in a client-server model is far less effective than that of Web content caching. Thus, the widely agreed Zipf model misinterprets media access patterns and overestimates the benefit of client-server based
media caching. We have further analyzed the evolution of media access patterns in homogeneously evolving systems, and found that the temporal locality in media systems increases with time, thus long term caching is beneficial to improve system performance. Our stretched exponential model lays out an analytical foundation to establish peer-to-peer caching systems for delivering the huge amount of media content on the Internet.

In order to enable effective peer-to-peer collaborations for exploiting the huge amount of CPU, bandwidth, and storage resources among the edge clients of the Internet, we have conducted a performance study of BitTorrent-like P2P systems, and analyzed the incentive mechanisms and user access patterns of BitTorrent systems. We found that although the existing BitTorrent system is effective for addressing the “flash crowd” problem upon the debut of a new file, it has service availability and performance stability problems after a period of time, due to the exponentially decreasing peer arrival rate. We have proposed a graph-based model to quantitatively analyze the interaction among multiple BitTorrent systems, and have studied the service potentials a torrent can provide to and get from other torrents. Based on this model, we have demonstrated that inter-torrent collaboration is much more effective than stimulating seeds to stay longer for addressing the service availability and performance stability problems in BitTorrent systems. We have further discussed and evaluated a novel architecture to facilitate inter-torrent collaboration with an exchange-based instant incentive mechanism, addressing the well-known problem of lacking incentives to seeds. We finally proposed PROP, a P2P-assisted media caching system, which utilizes peer-to-peer collaboration to provide service scalability and dedicated servers to provide service reliability. A redundant caching model and an adaptive cache replace algorithm are proposed to maximize the utilization of P2P resources based on media access patterns in the system.
APPENDIX A

REFERENCE RANK DISTRIBUTIONS OF OTHER MEDIA WORKLOADS

Figure A.1(a) and A.1(b) show the reference rank distribution of media objects in workloads HPC-98 and HPLabs-99, respectively. Figure A.1(c) shows the reference rank distributions of workload IFILM-06 in different durations. Similar to Figure 3.12, the slope of the distribution curve increases with time. The head of the reference rank distribution (raw data) deviates from the SE model gradually due to the accumulation of caching effect, as presented in Section 3.4.2. Furthermore, the tail of the distribution (raw data) is “cut off” from the SE model with time gradually. This is because for a server side media workload, the media requests are constrained by the number of objects introduced in the system. If the object birth rate is small, even the least popular objects may still get a certain amount of accesses, and thus Equation 3.3 should be changed to \( b = y_N + a \log N \) where \( y_N > 1 \). However, this does not affect the analysis in the paper since \( b \) is only a normalization parameter.

Figure A.1(d), A.1(e) and A.1(f) show the reference rank distributions in media workloads of the “other” category. As shown in Figure A.1(d), the stretch factor of live streaming media workload Akamai-03 is 0.2, very close to that of most on-demand Web media workloads. Figure A.1(e) and A.1(f) show the reference rank distribution of media objects
Figure A.1: More reference rank distributions of media workloads
in workload Movie-02 (movie box office sales) and IMDB-06 (top 250 movie votes), respectively. The parameter \(a\) of these two workloads are much higher than other workloads, since they have small media birth rate and large media request rate (see Section 3.5.3). The stretch factor \(c\) of Movie-02 is a little bit larger than that of movie dominant workloads such as BT-03 and CTVoD-04, indicating different media environments may affect user access patterns in some extent. The stretch factor \(c\) of IMDB-06 is much larger than all other workloads, possibly because it is a voting rank workload instead of a real media workload.
APPENDIX B

CHI-SQUARE GOODNESS OF FIT TESTS FOR SE AND ZIPF-LIKE MODELS

To have a quantitative evaluation on the goodness of fit for stretched exponential and Zipf-like distributions, we check these statistical assumptions with hypothesis tests. Since the reference rank distribution is a discrete distribution (the number of references must be positive integers), $\chi^2$ test is used.

$\chi^2$ test requires to divide the data set into a number of groups. In order to have a fair comparison, we use the same data binning methods to test the stretched exponential and Zipf-like distributions. We first make sure that the number of points in each group is at least 5, and then divide the domain range of reference numbers as evenly as possible. Assuming the instances of expected distributions have the same number of objects as that in the data set, we use the three-parameter stretched exponential model and two-parameter Zipf-like model (Equations 3.2 and 3.14) to generate expected distributions, due to the consideration of normalization factors. The parameters of Zipf-like model is computed with linear regression fitting the data in log-log scale. We then compute the observed frequency (in number of observations) for each bin of the data set, and the expected frequency (in number of observations) for each bin of the expected distributions. We have
Table B.1: $\chi^2$ hypothesis test results ($\alpha = 0.05$)

\[
\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i},
\]

where $O_i$ is the observed frequency of $i$-th bin, $E_i$ is the expected frequency of $i$-th bin, and $k$ is the number of bins.

Assume the significance level is $\alpha$ (in our test $\alpha = 0.05$), the assumed distribution is rejected when

\[
\chi^2 > \chi_{(\alpha, k-c)}^2
\]

where $k$ is number of bins and $c$ is the number of distribution parameters plus 1.

Table B.1 lists the results of $\chi^2$ tests for our workloads with original data, where the significance level $\alpha = 0.05$. $\chi^2$ test accepts the SE model and rejects the Zipf-like model for all these workloads. For workload ST-CLT-05 with extraneous traffic, the Zipf-like model is also rejected by $\chi^2$ test. We do not present the detailed results of $\chi^2$ tests for weekly and different numbers of weeks workloads ST-SVR-01 and BT-03.
APPENDIX C

REAPPRAISAL OF OTHER MEDIA ACCESS PATTERN MODELS

Many studies have observed that the object popularity distributions of streaming media systems and P2P systems have a “flattened head” in log-log scale, such as [22, 38, 76, 108]. Correspondingly, a number of models have been proposed. Most of them are still based on Zipf or power law model. In Section 3.4.3, we have shown that the “rich-get-richer” phenomenon is not present in media objects, and the assumption of Zipf with exponential cutoff model does not hold for media workloads. In this section, we revisit these models and illustrate why stretched exponential is a better model.

C.1 Zipf-Mandelbrot model

In study [98], Saleh et al. find that the popularity of P2P objects can be empirically modeled by a Zipf-Mandelbrot distribution, which captures the flattened head of the popularity distribution of objects in P2P systems. The Zipf-Mandelbrot distribution can be expressed as

\[ y_i = \frac{A}{(i + q)^\alpha}, \]

where \(i\) represents the reference rank (\(1 \leq i \leq N\), \(N\) is the number of objects), \(y_i\) denotes the number of references to the \(i\)-th object, \(\alpha\) is a constant like that in a Zipf-like distribution, \(q\) is a constant, and \(A\) is a normalization factor. In literature such as [78], it is called
Figure C.1: Zipf-Mandelbrot fitting for workloads with large media files

*shifted linear fractal distribution.* When \(q = 0\), the Zipf-Mandelbrot distribution becomes the Zipf-like distribution.

However, Zipf-Mandelbrot fitting only works for media workloads with small files, such as workloads in study [98], which are mainly small music files collected from Gnutella networks. For such workloads, the tail of the log-log plot of the reference rank distribution is close to a straight line, which is easy to fit by the Zipf-Mandelbrot model with a small \(q\) (compared to the total number of objects in the workload). In contrast, for workloads with large media files, no matter P2P or not, the reference rank distribution cannot be well fitted with Zipf-Mandelbrot model. Figures C.1(a), C.1(b), and C.1(c) show the Zipf-Mandelbrot fitting for workloads KaZaa-02 (median file size 300 MB), BT-03 (median file size 636 MB), and CTVoD-04 (median file size 300 MB), respectively. We have used different values of parameter \(q\) to fit these workloads, by using linear regression. We can see that even if \(q\) is as large as nearly the number of objects in the workload, the reference rank distribution still cannot be well fitted with the Zipf-Mandelbrot model. The reason is that for a media workload with large files, the difference between its reference rank distribution
and the Zipf model is significant, thus the thin tail of the distribution curve in log-log scale becomes important and cannot be fitted with a straight line.

**C.2 Parabolic Fractal Model**

Besides the variants of the Zipf-like model, it is also reported that the popularity distribution of files in peer-to-peer systems can be fitted with a parabola in log-log scale, and is called *log quadratic distribution* [39]. A more commonly accepted name of this model is the *parabolic fractal distribution* [77, 78] with

\[
\log y_i = \log y_1 - a \log i - b \log^2 i,
\]

where \( i \) represents the reference rank (\( 1 \leq i \leq N \), \( N \) is the number of objects), \( y_i \) denotes the number of references to the \( i \)-th object, and \( y_1 \) is the number of references to the most popular object. \( a \) and \( b \) are two positive constants. The Zipf-like distribution is a special case of the parabolic fractal distribution when \( b = 0 \). In the parabolic fractal distribution, the logarithm of the frequency or size of entities in a population is a quadratic polynomial of the logarithm of the rank, which remarkably improves the fit over a simple power-law relationship.

We also fit our workloads using the parabolic fractal distribution. However, only workload PS-CLT-04 can be fitted with a parabolic fractal model approximately. Other workloads can only be partially fitted with a parabola in log-log scale. For example, as shown in Figure C.2, the parabolic fractal model can fit either the head and waist or the tail of the reference rank distribution of workload ST-SVR-01, but not both. Please also note that the y-axis in this figure is in log scale, which can significantly compress the difference between raw data and fitting data from that in the parabolic fractal fitting.
The failure of tail fitting of the parabolic fractal model is due to its infinite mean and infinite moments when the number of events approaches infinity, which is unrealistic for a media workload. Since $y_N = 1$ when $N$ is large enough, from Equation C.2 we have

$$y_i = e^{\alpha \log N + b \log^2 N} = N^{a + b \log N}. \quad (C.3)$$

Thus

$$\langle y_{pb} \rangle \geq \frac{y_1}{N} = N^{a + b \log N - 1}, \quad (C.4)$$

and we have $\langle y_{pb} \rangle \to \infty$ when $N \to \infty$.

### C.3 Generalized Zipf-like Distribution Model

Study [108] analyzed a long term media workload in an enterprise media server and found the reference rank in a short term (e.g., a month) follows a Zipf-like distribution, but in a long term it does not follow a Zipf-like distribution. This observation is explained as due to the grouping effect of sufficient objects with similar popularities in a long term. In
[108], a linear transformation is applied on both $x$ and $y$ axis to transform the distribution into Zipf-like, called the \textit{generalized Zipf-like distribution}. Although in this case the generalized Zipf-like fitting looks good, workloads with different durations are not examined, and the selection of transformation parameters is not explained. As illustrated in Section 3.5.2, according to the stretched exponential model, the non-Zipf observation on the reference rank distributions of long term workloads is due to the cumulative effects of media access evolution.

\section*{C.4 Two-Mode Zipf Model}

Sripanidkulchai et al. [107] show that the popularity of events of the live streaming media workload Akamai-03 follows a \textit{two-mode Zipf distribution}. However, as shown in the log-log plot of Figure A.1(d), the transition between the two modes is not sharp. Similarly, workload CTVoD-04 also shows a non-strict two-mode Zipf-like property, although it is reported as Zipf-like (see the log-log plot of Figure 3.2(b)). As shown in the SE plot of these two figures, both workloads actually follow the stretched exponential model: the two-mode Zipf-like property is only a straightforward approximation. Other examples include workload BT-03 (Figure 3.3(c)) and workload Movie-02 (Figure A.1(e)).
APPENDIX D

AN ANALYSIS OF PRE-EXISTING OBJECTS IN MEDIA WORKLOADS

We use a simple fluid model to qualitatively analyze the magnitude of the number of pre-existing objects accessed in the system during the collection of a media workload. Consider a homogeneous media system where all media objects have the same popularity when they are created, and their popularities attenuate with time exponentially in the same way (note this assumption cannot be extended to describe media reference rank distribution). The request arrival rate of each object can be described with the following function

$$q(t) = q_0 e^{-\frac{t}{\tau}};$$  \hspace{1cm} (D.1)

where $t$ is the time after the object is born, $\tau$ is a constant characterizing popularity attenuation rate, and $q_0$ is a constant characterizing the initial request rate to this object.

Thus, the total number of requests to an object during its lifetime is

$$N_0 = \int_0^\infty \lambda_0 e^{-\frac{t}{\tau}} dt = \lambda_0^0 \tau.$$ \hspace{1cm} (D.2)

Since the total number of requests to an object decreases with time, the inter-arrival time of requests to the object, i.e., the time duration between two consecutive requests to the same object, $\delta t$, increases exponentially with time:

$$\delta t = \frac{1}{\lambda_0(t)} = \frac{1}{\lambda_0^0} e^{-\frac{t}{\tau}}.$$ \hspace{1cm} (D.3)
Assume a constant media request rate and a constant object birth rate. The cumulative number of requested objects \(N(t)\) in the time duration \([0, t)\) is:

\[
N(t) = \lambda_{\text{obj}} t + N'(t),
\]

where \(\lambda_{\text{obj}} t\) is the number of requested objects born in time \([0, t)\). \(N'(t)\) is the number of pre-existing objects that are born before \(t = 0\) and requested in time \([0, t)\). The oldest object among them should be requested only once during \([0, t)\). Assume the birth time of this object is \(-t_0\) \((t_0 > 0)\). Objects born after \(-t_0\) should be requested at least once during \([0, t)\). Thus the number of requested objects that are born before the workload collection is \(N'(t) = \lambda_{\text{obj}} t_0\). Assume each object is requested immediately after it is born, the number of requested objects that are born after workload collection is \(\lambda_{\text{obj}} t\).

To estimate \(t_0\), we consider the beginning of workload collection time, i.e., \(t = 0\). Let \(t_0 = 0\) when \(t = 0\). The request inter-arrival time of this oldest object at this time is thus \(t + \frac{1}{\lambda_0^\tau}\). We have

\[
\delta t = \frac{1}{\lambda_0^\tau} e^{\frac{4}{\tau}} = t + \frac{1}{\lambda_0^\tau},
\]

thus

\[
t_0 = \tau \log(\lambda_0^\tau t + 1).
\]

So

\[
N'(t) = \lambda_{\text{obj}} t_0 = \lambda_{\text{obj}} \tau \log(\lambda_0^\tau t + 1),
\]

and

\[
N(t) = \lambda_{\text{obj}} t + \lambda_{\text{obj}} \tau \log(\lambda_0^\tau t + 1).
\]
APPENDIX E

THE RESOLUTION OF THE BITTORRENT FLUID MODEL

Equation 4.7 is a non-homogenous ODE equation system. A particular solution for 4.7 is

\[
\begin{aligned}
x &= d_1 e^{-t/\tau}, \\
y &= d_2 e^{-t/\tau}, \\
\end{aligned}
\]

(E.1)

where

\[
\begin{aligned}
d_1 &= \frac{-\lambda_0}{\mu + \gamma - \psi^2}, \\
d_2 &= \frac{\lambda_0}{\mu (\psi^2 - \psi)}.
\end{aligned}
\]

(E.2)

The eigen equation is

\[
\psi^2 + (\mu \eta + \theta + \gamma - \mu) \psi + \mu \eta \gamma + \theta (\gamma - \mu) = 0.
\]

(E.3)

When the corresponding homogenous equation system has two different real eigenvalues \( \psi_1 \neq \psi_2 \), the resolution can be expressed as

\[
\begin{aligned}
x &= ae^{\psi_1 t} + be^{\psi_2 t} + d_1 e^{-t/\tau}, \\
y &= c_1 a e^{\psi_1 t} + c_2 b e^{\psi_2 t} + d_2 e^{-t/\tau},
\end{aligned}
\]

(E.4)

where

\[
\begin{aligned}
c_1 &= \frac{\psi_1 + \theta + \mu \eta}{\psi_2 + \theta + \mu \eta}, \\
c_2 &= \frac{\psi_2 + \theta + \mu \eta}{\psi_1 + \theta + \mu \eta}, \\
a &= -\frac{c_2 d_1 + d_2 - 1}{c_1 d_1 + d_2 - 1}, \\
b &= -\frac{c_1 d_1 + d_2 - 1}{c_1 d_1 + d_2 - 1}.
\end{aligned}
\]

(E.5)
When the corresponding homogenous equation system has two equal real eigenvalues \( \psi_1 = \psi_2 \), the resolution can be expressed as

\[
\begin{align*}
x &= (a + bt)e^{\psi_1 t} + d_1 e^{-t/\tau}, \\
y &= (ac_1 + bc_2 + bc_1 t)e^{\psi_1 t} + d_2 e^{-t/\tau},
\end{align*}
\]

(E.6)

where

\[
\begin{align*}
c_1 &= \frac{\psi_1 - \theta + \mu n}{\mu}, \\
c_2 &= \frac{1}{\mu}, \\
a &= -d_1, \\
b &= \frac{1 - d_2}{c_2}.
\end{align*}
\]

(E.7)

When the corresponding homogenous equation system has a pair of conjugate complex eigenvalues \( \alpha \pm \beta i \), the resolution can be expressed as

\[
\begin{align*}
x &= e^{\alpha t}(c_1 \cos \beta t + c_2 \sin \beta t) + d_1 e^{-t/\tau}, \\
y &= -se^{\alpha t}(c_1 \cos(\beta t + \phi) + c_2 \sin(\beta t + \phi)) + d_2 e^{-t/\tau},
\end{align*}
\]

(E.8)

where

\[
\begin{align*}
s &= \frac{1}{\mu \sqrt{(\alpha + \theta + \mu n)^2 + \beta^2}}, \\
\phi &= \tan^{-1}\left(\frac{\beta}{\alpha + \theta + \mu n}\right), \\
c_1 &= -d_1, \\
c_2 &= \frac{1 - d_2 + d_1 \cos \phi}{\sin \phi}.
\end{align*}
\]

(E.9)
BIBLIOGRAPHY


[27] A. Bellissimo, B. Levine, and P. Shenoy. Exploring the use of BitTorrent as the basis for a large trace repository. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 2004.


201


