Efficient Resource Management for Heterogeneous Devices Accessing Internet Streaming Content

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Dongyu Liu
Master of Science
George Mason University, 2003
Bachelor of Engineering
China University of Geosciences, 1997

Director: Dr. Songqing Chen, Professor
Department of Computer Science

Fall Semester 2009
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my parents Rumin Liu and Yanran Liu, and my wife Ying Cao.

Acknowledgments

I would like to thank the following people who made the dissertation possible.

I would like to thank my advisor Dr. Songqing Chen, who has spent so much of his time directing this dissertation work. It would have been impossible without his guidance and support.

My gratitude also goes to Dr. Hakan Aydin, Dr. Fei Li, Dr. Robert Simon and Dr. Chaowei Yang who served on my dissertation committee and gave me their invaluable input. I would also like to thank Pearl for providing proofreading service for my dissertation.

I would also like to thank Dr. Hassan Gomaa, Dr. Daniel Menasce for their advice on my dissertation presentation, as well as Drs Bo Shen, Xinwen Zhang, Shiping Chen, and Eric Setton for their advice and collaboration in several research projects during my Ph.D. study.

I would also express my appreciation to my lab mates Lei Liu, and Yao Liu for their help and collaboration, and to my friends I met at George Mason University: Yixiang Nie, Weijie Cai, Li Li, Fayin Li, Wanting Wang, Leijun Huang, Haidong Lu, Tarek Al-Enawy Hassan, Vinay Devadas, and Shen-Shyang Ho, just to name a few.

As an international student I received a lot of help from people in George Mason University and the local Fairfax community.

I would also like to thank my mother-in-law Peihua Gao, father-in-law Menglan Cao, and my family members: Dongfeng, Dongjie, Chengping, and Li for their support during my study.

Finally I would like to thank my parents Rumin Liu and Yanran Liu who raised me and always supported my decisions and my wife who always understands and fully supports me.

Table of Contents

List of Tables

List of Figures

Abstract

EFFICIENT RESOURCE MANAGEMENT FOR HETEROGENEOUS DEVICES ACCESSING INTERNET STREAMING CONTENT

Dongyu Liu, PhD

George Mason University, 2009

Dissertation Director: Dr. Songqing Chen

The volume of streaming media content has surged on the Internet in recent years. In parallel with the technological advancement in wireless and third generation (3G) networks, more and more Internet users today are turning to their mobile devices, such as smart phones or PDAs, to access the Internet stream content. However, when compared with desktops, mobile devices normally have different display sizes, color depths, bandwidth capacities, CPU speeds, or battery capacities, which are termed Multiple Dimensional Heterogeneity (MDH). Due to the MDH problem, these mobile devices normally cannot directly display the data streamed to desktops. A few solutions based on Multiple Description Coding (MDC) or Scalable Coding (SC) have been proposed, but none of these solutions are practical due to their various limitations.

In this dissertation, novel efficient resource management and transcoding schemes are investigated to address the MDH problem in both Internet on-demand and live streaming systems. First, in on-demand streaming systems, meta-data caching is leveraged during the online transcoding to reduce the overall CPU consumption. A model is further constructed to study the tradeoff between the CPU and the storage consumption for different meta-caching schemes under various conditions. Accordingly, an adaptive scheme that always

outperforms existing schemes is proposed. Second, in live streaming systems, the idle peer computing cycles are leveraged for efficient online transcoding based on an additional transcoding overlay constructed by participating transcoding peers. This proposed scheme effectively explores the tradeoff between the CPU and the bandwidth resources. Third, in consideration of resource contributions made by heterogeneous devices and their limited available resources, e.g., battery capacity, we propose a dynamic rotation scheme in both streaming and meta-transcoding overlays to balance the resource consumption among different peers, aiming to improve the fairness among peers and the overall system robustness. Fourth, a general theoretical framework is constructed to maximize the resource utilization in heterogeneous streaming systems. Accordingly, a distributed algorithm is proposed to achieve near-optimal performance with acceptable control overhead.

Chapter 1: Introduction

Recently the Internet has witnessed surging media content served by various streaming systems. The media contents have attracted many Internet users, among which mobile users account for a large portion due to technology advancements of wireless [1] and the third generation (3G) [2] networks. As mobile devices often differ from desktops in screen sizes, color depths, and bandwidth capacities, which is referred to as Multiple Dimensional Heterogeneity (MDH), the streaming content for desktops cannot be directly rendered and displayed on a PDA or a smart phone due to the format incompatibility. Therefore, the increasing streaming accesses from heterogeneous mobile devices challenge the existing Internet streaming delivery systems.

This dissertation aims to investigate efficient resource management to address the MDH problem in both Internet on-demand and live streaming systems.

1.1 Background

In recent years, streaming traffic on the Internet has increased significantly. According to ComScore [3], 16.8 billion online videos were watched in April 2009 only and that is a 40% increase compared with April 2008. In addition, in April 2009, Youtube.com attracted 107 million viewers who watched 6.8 billion videos [4]. Compared to on-demand streaming services, Internet live streaming has also gained more popularity. The advancements of P2P technology has enabled a significant portion of the streaming traffic delivered by various P2P/overlay streaming systems such as PPLive [5], Coolstreaming [6], PPStream [7], UUSee [8], TVAnts [9], AnySee [10], ESM [11], and MySee [12]. For example, in PPLive and PPStream, hundreds of thousands of users join hundreds of channels and contribute a

large amount of streaming traffic everyday. ESM [11], built upon overlay multicast technology, has been used for over 30 different events, such as SIGCOMM 2002 and 2003, SOSP 2003 and NOSSDAV 2004. Over 10,000 ESM users all around the world contribute a lot of streaming traffic.

At the same time, recent technology advancements in wireless and 3G networks have made Internet accesses more pervasive. For example, 3G wireless networks based on CDMA 2000 [13] (with more than 455 million subscribers worldwide by June 2009 [14, 15]) and HSPDA [16]/UMTS [17] have been widely deployed (AT&T 3G network uses HSPDA/UMTS technology). Through 3G data networks, a peak transmission rate of 2.4 Mbps (EV-DO) [18] or 14.4 Mbps (HSPDA) [16] can be achieved. Thus, it is possible for customers to use PDAs and smart-phones to access streaming videos off the Internet. For example, many Verizon [19] customers use mobile devices to access V-CAST [20] streaming service. For Cingular (now AT&T) [21] customers, they can access Cingular Video (CV) [22] service. According to Nielsen Mobile [23], a leading provider of performance measurement to the mobile industry, more than 40.4 million US mobile subscribers actively accessed the Internet via wireless devices in May 2008 and these subscribers account for 15.6% of the total 254 million US mobile phone subscribers that access the Internet [24]. It is estimated that mobile users worldwide will exceed 1 billion before 2010 [25].

With pervasive Internet accesses from various mobile devices, it is natural that streaming content accesses from mobile devices is rapidly increasing. Due to the MDH problem, a media object that is customized and suitable for a desktop computer cannot be directly rendered and displayed on a mobile device. To fit mobile devices, a media object must be customized appropriately beforehand or at runtime. This type of customization for Quality of Service (QoS) support is often referred to as *content adaptation*.

In general, there are two approaches for content adaptation in the context of multimedia content delivery.

The first approach is *precoding*. Given a media object, precoding either creates multiple provisioned versions or encodes the object with multiple layers, or descriptions. All

the object versions/layers/descriptions are created before they are ready to be delivered. There are three typical precoding methods: Versioning [26], Multiple Description Coding (MDC) [27–29], and Scalable Coding (SC) [30].

- Versioning – The media content provider or server encodes a media object into multiple resolution and modality (media format) versions so that they can be rendered on heterogeneous devices [26]. For example, Apple Inc. [31] provides movie trailers in different bit rates to customers: low quality version (56Kbps) for dial-up clients and high definition version (100+ Kbps) for broadband clients.

- MDC – In a streaming system, the media server encodes a media object, such as a video clip, into multiple descriptions with equal importance. Each description is independent and may be further divided into description chunks of certain length [27]. In MDC, clients receiving different number of descriptions perceive different streaming qualities as any combination of different descriptions can be decoded into a certain streaming quality. The more descriptions the client receives, the better the quality is. According to [32], MDC is an effective method to combat bursty packet losses on the Internet and wireless networks.

- SC – As any combination of descriptions in MDC is decodeable, the coding efficiency of MDC is not high due to overlapped and redundant data in different descriptions [30]. Different from MDC, SC encodes a media object into several layers (a basic layer and a certain number of enhancement layers) with nested dependencies. A higher level of enhancement layer can only be decoded when all the lower layers are available. Therefore, although SC has a higher coding efficiency than MDC, its decoding efficiency is low compared with MDC due to the mutual dependency between different layers. A scheduling mechanism is needed to set different priorities on different layers for delivery to improve decoding efficiency [30].

While the precoding has been used in practice, the precoding approach has several limitations. First, existing solutions based on MDC/SC [33, 34] are not effective in practice

given the fact that currently most devices do not support MDC/SC codecs. Second, although a lot of research has been conducted on precoding approaches [28, 29], it is difficult for these approaches to provide fine-grained media qualities [35] because these approaches are less flexible when finer granular QoS is requested. Third, precoding does not scale well with various media adaptation applications. It may be easy to provide possible bit rate versions that are requested for a streaming application, but it would be difficult to precode content for more generic content adaptation tasks such as streaming personalization. For example, a host server is not likely able to mix an end user's logo into a video stream due to the unavailability of user's logo on the server side. Fourth, in Internet streaming systems, precoding requires streaming of compound objects containing overlapped datawhich implies a waste of storage and bandwidth resources. Although SC content versions require less space than MDC versions [30], it still has data compression redundancy.

The second approach, which is referred to as *transcoding*, offers on-line real-time content adaptation support. Typically, transcoding is conducted on a server which involves a significant amount of computing. The procedure is to decode the original object, perform content customization, and then re-encode it to a different version format. Overall, transcoding is not restricted to content customization for QoS support. This approach offers more flexibility and adapts well with various media adaptation applications. Moreover, no specific codec is required for transcoding. Thus, this approach is suitable to address the MDH problem. However, it is challenging to conduct efficient transcoding for heterogeneous devices in streaming systems due to the following reasons:

(1) In on-demand and live streaming systems, although the MDH problem can be addressed by online transcoding, online transcoding is a computing-intensive task. Traditionally, transcoding can be conducted on a dedicated transcoding server. However, with more and more streaming accesses from heterogeneous devices, a dedicated server cannot scale well to accommodate a large number of concurrent clients. Thus in on-demand streaming systems, how to serve a large number of concurrent requests remains a challenge. Whether the transcoding could be improved or utilized for this purpose needs to be explored.

(2) In P2P/overlay based live streaming systems, the situation is even worse because there is no dedicated infrastructure support (i.e., a transcoding server) for the transcoding in P2P/overlay streaming system. On the other hand, participating mobile devices usually can only contribute limited resources, such as bandwidth resources or computing cycles. Thus, how to provide online transcoding services to serve heterogeneous devices in P2P/overlay streaming systems is challenging, particularly if no additional infrastructure support is available.

(3) In P2P/overlay streaming systems, due to different peer positions in the system, different peers contribute different amount of resources to the system. For mobile devices, the unfairness results in different consumption of their battery sources. Thus, the unbalanced resource contribution among all the clients may exhaust the battery capacity of some clients more quickly than other peers and make them leave the system earlier than expected or even affect the robustness of the system. Therefore, with heterogeneous devices participating streaming system, an efficient solution is needed to ensure the system robustness while providing high QoS to the clients.

(4) Given that Internet streaming for heterogeneous devices demands various resources, such as storage, bandwidth, and CPU cycles, while these resources are commonly limited in a computer, how to efficiently allocate and manage available resources to serve as many clients as possible while maintaining high streaming QoS is critical. This is very imperative for P2P/overlay streaming since there is no infrastructure support. However, so far this problem has not been well studied.

1.2 Contributions

The objective of this dissertation is to investigate efficient resource management in both Internet on-demand and live streaming systems to provide high quality streaming services to heterogeneous devices. In particular, we want to answer the following questions:

1. Whether and how online transcoding could be scalable for on-demand streaming to

heterogeneous devices via efficient resource management?

2. Whether and how online transcoding could be implemented in P2P/overlay streaming systems to serve heterogeneous devices?

3. How to optimally utilize available resources to provide high quality streaming to as many heterogeneous clients as possible?

In this dissertation, we set to systematically investigate efficient resource management and transcoding schemes in both Internet on-demand and live streaming systems to address the aforementioned problems. First, instead of treating the transcoding as a black-box, we study the intermediate results, called meta-data, of transcoding and examine whether caching carefully selected meta-data could effectively reduce the CPU consumption on a transcoding proxy server for on-demand streaming. Furthermore, we seek to leverage the efficient meta-caching in Internet live streaming. But in the typical P2P live streaming systems, there is no dedicated computing servers such as transcoding proxies. Thus we study whether effective peer collaboration in a distributed environment could eliminate such a computing bottleneck. Moreover, since different clients in the live streaming system may contribute different amount of resources, which results in unfair resource contributions and threatens the robustness of the system, we also examine whether we should rotate peer positions in order to maintain fair resource contributions among peers. Lastly, we consider how the limited resources should be used to maximize the system objectives while providing high-quality streaming to clients at the same time. We propose to model this problem and seek an optimal solution to it.

The major contributions of this dissertation are summarized as follows:

1. In on-demand streaming systems, the intermediate transcoding steps are studied, and we find that with meta-caching, the fully transcoded media object can be easily produced with a small amount of CPU cycles and some metadata stored in the cache. Thus, if meta-data caching is leveraged, the CPU consumption during the online transcoding can be reduced. Furthermore, we construct a model to study the tradeoff

between the CPU and the storage consumption for various caching schemes. Compared with other caching schemes, a conditional advantage of meta-caching scheme is shown. Based on this model, an adaptive scheme is designed to leverage the conditional advantage of meta-caching and apply meta-caching based on the client request patterns and available resources. Evaluation results show that our proposed scheme can always outperform existing schemes.

2. In P2P/overlay based live streaming systems, an additional overlay is constructed and idle peer computing cycles are leveraged for efficient online transcoding. The proposed peer-assisted transcoding scheme can effectively explore the tradeoff between the CPU and the bandwidth resources. In addition, the scheme can enable effective online transcoding in a streaming system without additional infrastructure support. To the best of our knowledge, the proposed scheme is the first to combine meta-transcoding and P2P/overlay streaming. The proposed scheme can also be used to design both tree-based and mesh-based transcoding assisted overlay streaming systems. The evaluation results show that the proposed scheme saves up to 58% CPU cycles for online transcoding by consuming 6% extra bandwidth traffic.

3. In streaming systems, heterogeneous devices contribute different amounts of resources due to their positions assigned by the system and the power consumption is directly determined by the resource contribution. Moreover, heterogeneous devices may have limited available resources, e.g., battery power. Thus, unbalanced resource contributions may pre-maturely exhaust the limited battery capacity of mobile devices, and subsequently threaten the robustness of the whole system. If the positions of the peers can be exchanged periodically, the fairness of the system will be improved greatly. Based on this observation, a dynamic rotation scheme is designed to balance multiple types of resource consumption in different clients, aiming to improve the fairness among clients and the overall system robustness. The proposed scheme is evaluated via simulations of a large scale system and the results show that the proposed scheme is effective in balancing the CPU and the bandwidth consumption

in individual clients over other schemes.

4. In streaming systems, although the QoS of heterogeneous clients can be achieved by the novel transcoding schemes by trading storage/bandwidth for computing cycles resources, the system always aims to serve more clients with its limited resources. Thus, transcoding-based streaming systems challenge the coarse admission control and resource allocation policies of existing streaming system. If the available resource utilization could be optimized, there is great potential to further improve the system performance.To fulfill that objective, a generic theoretical framework is proposed to maximize the resource utilization in the streaming system in order to improve the throughput of the system. Based on the framework, a distributed algorithm is proposed to achieve near-optimal performance. Evaluation results show that the proposed resource allocation algorithm can significantly improve the system resource utilization while maximizing the lowest peer's perceived qualities with acceptable overhead.

1.3 Dissertation Organization

The remainder of this dissertation is organized as follows. The design of meta-caching assisted transcoding scheme is presented in chapter 2. Chapter 3 describes the peer-assisted transcoding scheme. A dynamic rotation scheme is described in chapter 4. A general theoretical framework to investigate the resource allocation and a distributed algorithm which can get near-optimal performance is presented in chapter 5. The dissertation is summarized in chapter 6.

Chapter 2: Adaptive Meta-caching for Transcoding

2.1 Introduction

As aforementioned, the MDH problem of heterogeneous devices in today's Internet streaming systems becomes significantly severe. Compared with other schemes, online transcoding is the most appropriate approach to deal with the problem. In this chapter, we will present how to address the MDH problem via efficient online transcoding in a on-demand streaming system by efficiently managing the CPU and the storage resources.

Research on developing efficient online transcoding algorithms has received much attention, especially in video transcoding [36]. From the system's perspective, caching is a viable technique to reduce computing load of transcoding. In detail, the transcoded result for a request can be cached so that identical requests can be served without duplicate transcoding in the future. This kind of transcode-enabled caching designs has been investigated [37] where different precoded contents are cached adaptively for optimal server traffic reduction. The relative advantages of caching precoded versions over layers were evaluated [38–40]. In these studies, the computing load of the proxy is not a concern as no on-line transcoding is involved.

In the on-line transcoding context, an application gateway is proposed for transcoding [41]. In Middleman [42], a video cache proxy server is proposed by leveraging a collection of cooperative proxy servers. Other approaches [43–47] aim to improve the utilization of the CPU, the storage, and the bandwidth resources by applying heuristic solutions. The TeC system [48] is proposed to selectively cache original and/or transcoded objects to increase the performance such as system throughput. In another work, transcoding and scheduling are jointly considered in a network-attached disk architecture [49].

All these existing designs treat transcoding as a black box. The cached data are either the input and/or the output of the transcoding. The possibility of caching intermediate results during transcoding processes is preliminarily investigated in [50]. It has been shown the potential of reducing the aggregated computing load on the proxy for three specific transcoding cases. However, that work is restricted to video transcoding applications.

In this chapter, a meta-caching scheme in which intermediate transcoding steps are identified is proposed so that appropriate intermediate results (also called *metadata*) can be cached and these intermediate results can be reused for future identical transcoding procedures. Shen [50] describes how to extract metadata for different transcoding scenarios as well as how to reproduce the final object by transcoding using the extracted metadata. With the cached metadata, the fully transcoded object can be easily produced with a small amount of CPU cycles. Specifically, if the transcoded version is fully cached (*full-caching*), identical future requests can be directly served without additional transcoding. However, caching every transcoded version may quickly exhaust the cache space. On the other hand, if a transcoded version is not cached (*no-caching*), identical requests will result in repetitive transcoding, which consumes extensive CPU cycles.

In meta-caching, since only metadata is cached, the required cache space is greatly reduced. The saved cache space can be used to store metadata for other transcoding sessions so that the overall computing load can be reduced. Note that the meta-caching scheme allows the system to achieve joint control of the CPU cycles and storage resources. Therefore, it offers a tradeoff between what can be achieved by the full-caching and no-caching schemes.

To precisely characterize the meta-caching scheme and efficiently manage the CPU and the storage resources, an analytical model is constructed, and the conditional advantages of meta-caching scheme over full- or no-caching scheme is investigated. Based on the model-driven analysis, a system called AMTrac, which stands for *A*daptive *M*eta-caching for *Tra*ns*c*oding is proposed. In AMTrac, the meta-caching scheme is adaptively used upon

dynamic client accesses and the available resources in the system. Simulation-based experiments show that AMTrac can effectively improve the system throughput over existing schemes.

The remainder of this chapter is organized as follows. The generalized meta-caching concept is introduced in Section 2.2, and an analytical model is presented in Section 2.3. An adaptive meta-caching design based on the model is provided in Section 2.4 and its performance is evaluated in Section 2.5. This chapter is summarized in Section 2.6.

2.2 Principle of Meta-caching

Meta-caching is proposed to effectively balance the resources used for online transcoding between the CPU cycles and the limited storage space in a transcoding proxy or server. The motivation of meta-caching is as follows. Many existing schemes that aim to optimize transcoding treat the transcoding procedure as a black box, focusing on the optimization of caching transcoded object versions based on the prediction of future requests. However, a typical transcoding procedure involves multiple steps, among which some intermediate results, also called *metadata*, may be saved in the cache to avoid re-computing the same result again for a later identical transcoding request. If the saved metadata only consumes a small storage size while it takes a significant computing load to generate, it is apparently an effective tradeoff between the storage and the CPU cycles. For example, for bit rate reduction transcoding, the requantization scale factor can be stored rather than being re-computed each time the identical data is transcoded. In this case, meta-transcoding requires significantly less CPU cycles to produce the new transcoded object.

To illustrate the principle of meta-caching, we define a storage versus computing space and discuss some practical applications taking advantage of this principle.

2.2.1 Storage-Computing (SC) space

Given any media adaptation process, meta-caching is defined as the caching of intermediate results that are created during the course of the adaptation process. As an example,

Figure 2.1 defines the flow graph of a certain media adaptation process composed of four computing sub-modules. Caching the intermediate result from each of the four sub-modules leads to skipping that sub-module in the next identical transcoding session so that the computing of that sub-module is saved. However, some storage space is required to store the corresponding intermediate result. In general, caching the output of each sub-module maps to one point in a space, called *Storage vs. Computing space*. Suppose Figure 2.2 illustrates such a mapping for this particular process. The vertical axis indicates the amount of the storage required to store the intermediate results from any of the sub-modules, relative to storing the final processed results. The horizontal axis indicates the amount of computing load required to create the final result with the help of the intermediate results from any of the sub-modules. This computing load is relative to the computing load when the intermediate result is not available, i.e., the computing load of the full content adaptation process. Clearly, if the intermediate result of any sub-module is directly available (from a cache, for example) instead of computing it from the input, the computing load required to create the final output is smaller.

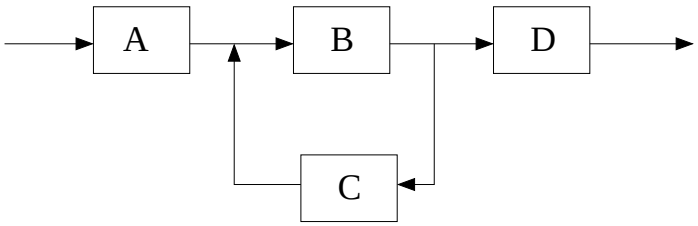

Figure 2.1: Example of processing flow

There are some specific points within the SC space. If nothing from the flow is cached, the point I (no cache) indicates no storage requirement. But 100% of the CPU is required when a final adapted output is requested. On the other hand, if the final output is cached (full cache), 100% of the storage is required while no computing cycles are needed. Note that point D and point I define a shaded area in this SC space. In general, any point that falls outside the shaded area has no advantage over either point D or point I. For example, point A is outside the shaded area, which indicates that storing the intermediate result from

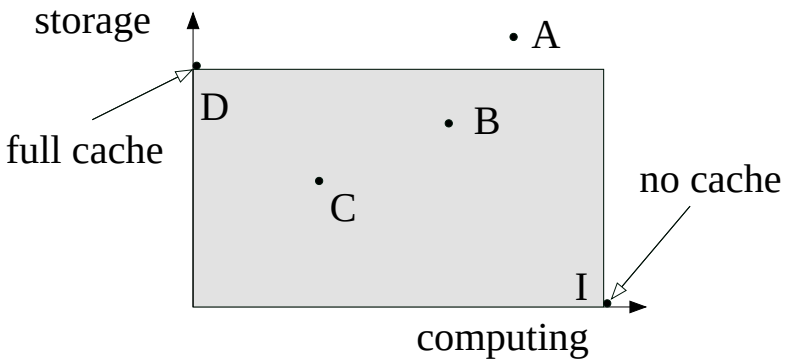

Figure 2.2: Mapping between the Storage and Computing space

A would cost more storage space than storing the result from D. Obviously, this is not as efficient as simply storing the final result.

Point B and C have certain advantages since both points indicate reduced storage requirement at a cost of a certain computing load. In general, the point closer to the origin of the SC space presents better advantages since it indicates less computing and less storage requirement to obtain the final results. In this example, point C is clearly a better choice. In other words, point C indicates that sub-module C is more computing intensive yet its intermediate result requires less storage.

Although this principle is illustrated through an arbitrary example, it is clear that the principle is general. Once any media adaptation process is defined, a map in the SC space that reflects the storage, and computing tradeoff is uniquely obtainable.

2.2.2 Meta-caching Applications

The principle outlined above has many practical applications. In particular, video transcoding is the most common type of media adaptation application.

Figure 2.3 illustrates the processing flow of the bit rate reduction of a MPEG video, a case of transcoding. With careful selection of the intermediate points within a transcoding process pipeline, meta-caching can be very useful in reducing the aggregated computing load of an adapting server servicing many client requests with different access patterns. In this

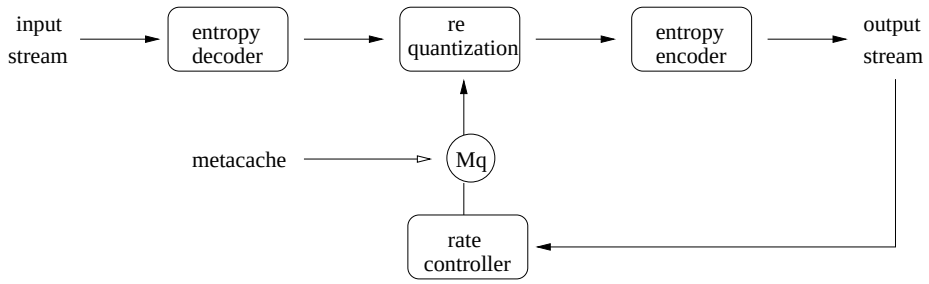

Figure 2.3: Bit rate reduction example

case, caching of the requantization scale factor (M_q) could achieve a good tradeoff between storage and computing resource utilization. For more details about meta-transcoding, please refer to [50–52].

The meta-caching principle is not just restricted to video transcoding applications. To name a few other types of media adaptation applications that can benefit from this principle, the following examples are considered.

- Video to keyframe conversion. Instead of a full length video, a sequence of representative keyframes from the video can be delivered in situations when a client does not have a video player available. Keyframe analysis detects scene changes within a video sequence and identifies frames that are representative of the video. This can be very computing intensive. However, if the intermediate result (e.g., the frame index of the keyframes) of the keyframe analysis module can be stored, which costs very little storage, the computing load can be significantly reduced for future sessions.

- Personalized logo insertion. With a customized logo inserted into each frame of a video, a client can personalize his/her own content. In this process, the compressed video is first adapted with the logo insertion area to be independently coded. Then a logo is inserted. The adaptation from dependently coded original video to independently coded area can be computing intensive. If the converted independent area can be stored, future sessions can be free of significant computing load. On the other hand, an independently coded area (for example, four significantly smaller corner areas) costs less storage than the full logo inserted video.

- Privacy protection. In this scenario, certain features from content (e.g., certain faces in a video) are automatically blocked to protect privacy. The intermediate results from face detection (e.g., face location on each frame) can be stored so that future sessions can be relieved from the computing intensive face detection process.

The meta-caching principle is often useful for adapting multimedia content since this kind of process tends to be computing intensive. However, for any real-time content processing service, if a point in the Storage-Computing (SC) space that can benefit from the meta-caching principle can be identified from its processing flow, it is possible to improve the overall system performance by balancing the use of storage and computing resources. Since overall system performance depends on aggregated client access behavior, certain points in the SC space can be more advantageous than others, given available computing and storage resources.

2.3 Performance Modeling

In this section, the performance of the meta-caching scheme is modeled by comparing it with full-caching and no-caching schemes. In detail, the performances of the meta-caching, full-caching, and no-caching schemes under the available storage and CPU resources are precisely modeled. For this purpose, in the modeling, the throughput of the meta-caching with two other schemes is compared, and is shown that an appropriate scheme can be selected to fully utilize the available CPU and storage resources in order to improve the system's performance. The metric used for comparison is the system throughput. It is represented by the number of concurrent sessions that the proxy or server can serve. Clearly, system throughput is decided by considering the available cache space and CPU cycles jointly.

The goal of the modeling is to quantitatively evaluate the throughput for each scheme, and compare their performance with given CPU and storage resources. The methodology is to calculate the number of sessions that could be served by first considering CPU and storage separately. The smaller one of two session numbers is the maximum number of sessions that

can be supported by the proxy with the corresponding CPU and storage resources.

2.3.1 Model Construction

In the analysis, the following assumptions about client accesses and media objects in the proxy system are made.

1. Object popularity follows a Zipf-like distribution [53,54], and the accessing probability of the k_{th} popular object is denoted as $P_k = \frac{1}{k^\theta} / \sum_{i=1}^{M} \frac{1}{i^\theta}$. Note that $A = (\sum_{i=1}^{M} \frac{1}{i^\theta})^{-1}$, and $P_k = \frac{A}{k^\theta}$. θ is the skew factor in the Zipf-like distribution. For media objects access, according to existing media workload characterizations and measurements [55–57], the value of θ is between 0.47 and 0.73. In the analysis, it is assumed that more popular objects are always cached more favorably than less popular ones.

2. The inter-request arrival pattern follows a Poisson distribution [$p(x, \lambda) = e^{-\lambda}\lambda^x/x!$, x=0, 1, 2,... , λ is the mean arrival rate].

3. The storage space occupied by each original object is one storage unit. Each object has v different versions for client accesses, and each version is accessed by clients uniformly. The storage space needed for caching different versions is different. From the highest quality to the lowest quality, each version occupies a space of 1, $\frac{v-1}{v}$, $\frac{v-2}{v}$, ... , $\frac{1}{v}$. In practice, a system can select an appropriate value of v, and the corresponding output version based on the physical conditions.

4. The CPU used for any transcoding that results in a fully cached version is 1 unit. For meta-caching, it is assumed that the CPU consumed to produce the final version is β ($0 < \beta < 1$). The storage space used to cache metadata is a fraction α ($0 < \alpha < 1$) of the fully cached object [1].

The notations used in the following analysis are summarized in Table 3.1.

[1] If $\alpha = 1$, meta-caching is the same as full-caching, where $\beta = 0$; if $\alpha = 0$, meta-caching is the same as no-caching, where $\beta = 1$. For simplicity, I assume the ratios between metadata and transcoded results for different object versions are the same.

Table 2.1: Notations used in the analysis

S	the total cache size
C	the total amount of computing cycles
N	the total number of accesses to objects in the proxy system
N_i	the number of accesses to all versions of object i, $N_i = N * P_i$
M	the total number of multimedia objects in the proxy system
v	the number of versions of each object
α	the storage required to cache metadata (relative to full result)
β	the CPU used to produce a fully-transcoded version from cached metadata

First, the number of sessions that can be supported by the available storage is considered. It is assumed that among the total M objects, only k most popular objects can be cached in S storage, and all their versions get accessed. Among the N accesses, the total number of sessions to the first k objects is thus $\sum_{i=1}^{k} N_i$.

For full-caching, if v_a is the cache occupied by the v versions of each object, these $\sum_{i=1}^{k} N_i$ sessions use $k * v_a$ cache units. Since different versions of each object are accessed uniformly, the total cache space used is $v_a = 1 + \frac{v-1}{v} + \frac{v-2}{v} + ... + \frac{1}{v} = \frac{v+1}{2}$.

Thus the average storage requirement for each session is

$$s_f = \frac{k \times \frac{v+1}{2}}{\sum_{i=1}^{k} N_i}. \tag{2.1}$$

For meta-caching, since the average storage requirement is α percentage of one unit, the average storage requirement s_m is

$$s_m = \alpha \times \frac{k \times \frac{v+1}{2}}{\sum_{i=1}^{k} N_i}. \tag{2.2}$$

For no-caching, only k units are needed to store original objects in the proxy. Thus these $\sum_{i=1}^{k} N_i$ sessions use k cache units. The average storage requirement s_n for no-caching is

$$s_n = \frac{k}{\sum_{i=1}^{k} N_i}, \tag{2.3}$$

in which $\sum_{i=1}^{k} N_i = N * \sum_{i=1}^{k} P_i = N * \sum_{i=1}^{k} \frac{A}{i^\theta} = N * A * \sum_{i=1}^{k} \frac{1}{i^\theta} = N * A * \frac{k^{1-\theta}-1}{1-\theta} \approx N * A * \frac{k^{1-\theta}}{1-\theta}$.

By replacing $\sum_{i=1}^{k} N_i$ with $N * A * \frac{k^{1-\theta}}{1-\theta}$, Equation 2.1, 2.2, and Equation 2.3 can be rewritten as the following:

$$s_f = \frac{k(v+1)(1-\theta)}{2NAk^{(1-\theta)}} = \frac{k^\theta(v+1)(1-\theta)}{2NA}, \tag{2.4}$$

$$s_m = \alpha\frac{k(v+1)(1-\theta)}{2NAk^{(1-\theta)}} = \frac{\alpha k^\theta(v+1)(1-\theta)}{2NA}, \tag{2.5}$$

$$s_n = \frac{k^\theta(1-\theta)}{NA}. \tag{2.6}$$

Given cache size S, the total number of sessions that can be supported by full-caching, meta-caching, and no-caching is denoted as z_{fs}, z_{ms}, and z_{ns}, respectively. In detail, they are

$$z_{fs} = S/s_f = \frac{2SNA}{(1-\theta)(v+1)k^\theta}, \tag{2.7}$$

$$z_{ms} = S/s_m = \frac{2SNA}{\alpha(1-\theta)(v+1)k^\theta}, \tag{2.8}$$

and

$$z_{ns} = S/s_n = \frac{SNA}{(1-\theta)k^{\theta}}. \tag{2.9}$$

Having considered the storage usage, the CPU usage of full-, meta-, and no-caching schemes are considered. In full-caching, every session needs one unit of CPU for the first access and zero unit for the following accesses. In meta-caching, every session needs one unit of CPU for the first access and β units for each of the following accesses. In no-caching, only accesses to the original objects require zero CPU load. For any other access, one unit of CPU is required.

For full-caching, if the full results of the transcoding of the k_{th} most popular object are cached, the computing load for obtaining all v versions of the k_{th} object is v. Therefore, on average, the computing load for accesses to object i is v, and the average computing load for object i is $\frac{v}{N_i}$. Since there are k objects in the cache, the average computing load for each session is

$$\begin{aligned} c_f &= \left(\frac{v}{N_1} + \frac{v}{N_2} + ... + \frac{v}{N_k}\right)/k \\ &\approx \frac{k^{\theta}v}{NA(1+\theta)}, \end{aligned} \tag{2.10}$$

where $A = (\sum_{i=1}^{M} \frac{1}{i^{\theta}})^{-1}$.

For meta-caching, on average, the computing load for object i is $v + (N_i - v)\beta$. Thus the average computing load for object i is $\frac{v+(N_i-v)\beta}{N_i}$. The average computing load is

$$\begin{aligned} c_m &= \left(\frac{v + (N_1 - v)\beta}{N_1} + \frac{v + (N_2 - v)\beta}{N_2} + ... + \frac{(N_k - v)\beta}{N_k}\right)/k \\ &\approx \frac{k^{\theta}v(1-\beta)}{NA(1+\theta) + \beta}, \end{aligned} \tag{2.11}$$

For no-caching, every access to an object consumes one CPU unit, except for the access to the original version of the objects, which happens exactly once for each object. Therefore, on average, the computing load for the object i is $(N_i - 1)$, and the average computing load for the object is $\frac{(N_i-1)}{N_i}$. Thus the average computing load for no-caching is

$$\begin{aligned} c_n &= \left(\frac{(N_1 - 1)}{N_1} + \frac{(N_2 - 1)}{N_2} + ... + \frac{(N_k - 1)}{N_k} \right) /k \\ &\approx 1 - \frac{k^\theta}{NA(1+\theta)}. \end{aligned} \tag{2.12}$$

Given the available CPU capacity C, the number of sessions that can be supported by the full-, meta-, and no-caching scheme is denoted as z_{fc}, z_{mc}, and z_{nc}, respectively. Specifically, they are

$$z_{fc} = C/c_f = \frac{CAN(1+\theta)}{vk^\theta}, \tag{2.13}$$

$$z_{mc} = C/c_m = \frac{CAN(1+\theta)}{vk^\theta(1-\beta) + AN\beta(1+\theta)}, \tag{2.14}$$

and

$$z_{nc} = C/c_n = \frac{CAN(1+\theta)}{k^\theta + AN(1+\theta)}. \tag{2.15}$$

Now both resource constraints are considered jointly. If sn_f, sn_m, and sn_n are used to denote the total session numbers for full caching, meta-caching, and no-caching, they are

$$sn_f = \min(z_{fs}, z_{fc}) = \min\left(\frac{2SNA}{(1-\theta)(v+1)k^\theta}, \frac{CAN(1+\theta)}{vk^\theta} \right), \tag{2.16}$$

$$sn_m = \min(z_{ms}, z_{mc}) = \min\left(\frac{2SNA}{\alpha(1-\theta)(v+1)k^\theta}, \frac{CAN(1+\theta)}{vk^\theta(1-\beta) + AN\beta(1+\theta)} \right), \tag{2.17}$$

and

$$sn_n = \min(z_{ns}, z_{nc}) = \min\left(\frac{SNA}{(1-\theta)k^\theta}, \frac{CAN(1+\theta)}{k^\theta + AN(1+\theta)}\right). \tag{2.18}$$

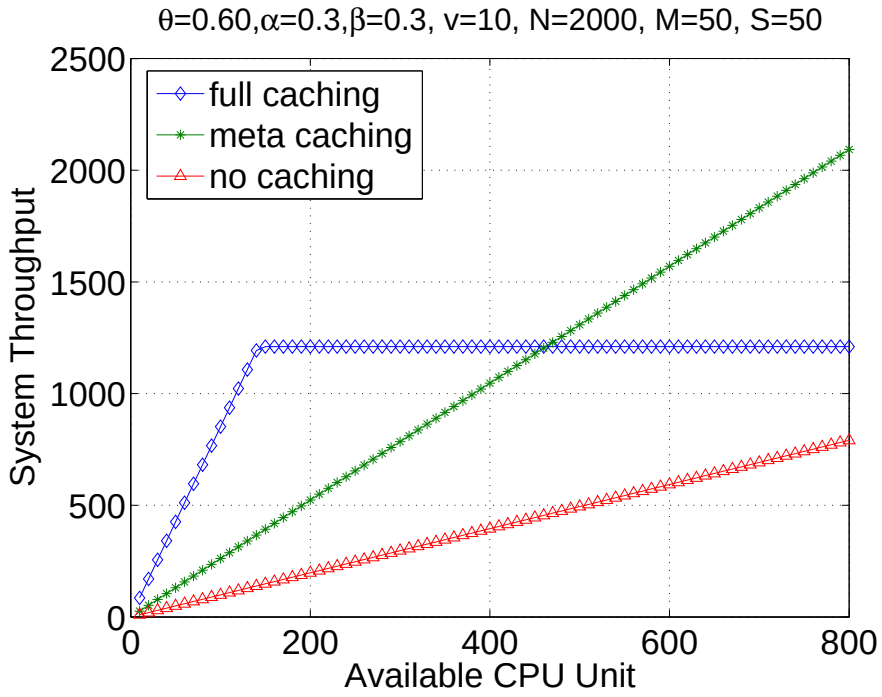

Figure 2.4: Comparisons between three caching methods: 800 available CPU units

Based on the above formulas, Figure 2.4 and Figure 2.5 show the comparison results with some typical values for different parameters when the available CPU resources varies. In these two figures, θ is set to 0.6, and both α and β are set to 0.3. It is assumed that there is a total of 50 objects, and a total of 2000 accesses. Each object has 10 versions.

In Figure 2.4, the available CPU resources ranges up to 800 units. Figure 2.5 changes the available CPU resource to 8000 units. These figures indicate that under certain conditions, one of the three schemes may outperform the other two. Having obtained the quantitative throughput for different schemes, their performance in further detail are compared in order to reveal such conditions. The model quantitatively characterizes the conditional advantages of full-, meta-, and no-caching schemes over other two under different conditions of available resources and client access patterns.

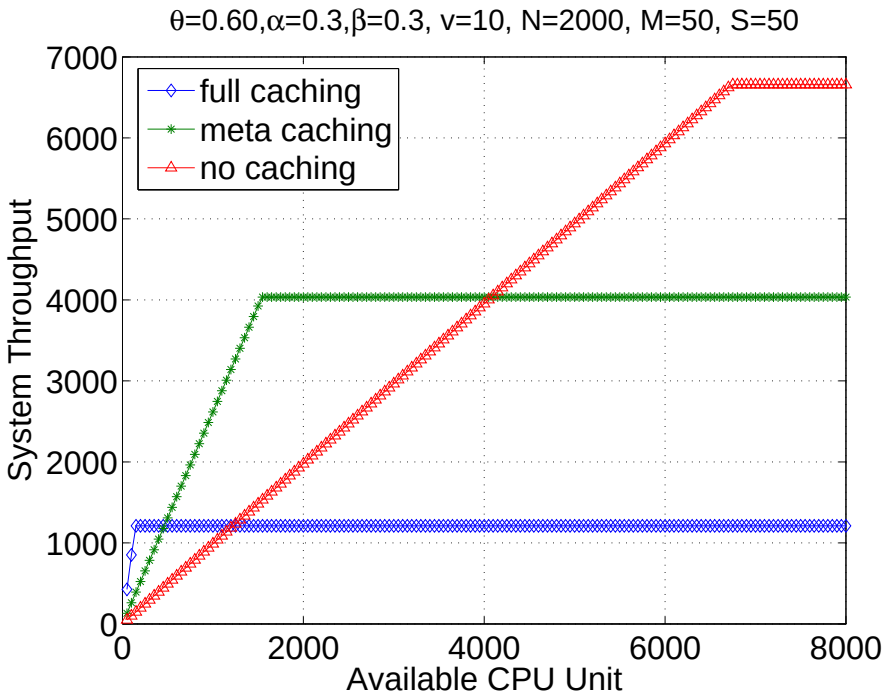

Figure 2.5: Comparisons between three caching methods: 8000 available CPU units

2.3.2 Performance Analysis

According to the model presented above, in this section, the system throughput for the three methods when $0 < \theta < 1$ is discussed.

By equation 2.7, 2.8, and 2.9, and $0 < \alpha < 1$, it is easy to see that

$$z_{fs} < z_{ms} < z_{ns}. \tag{2.19}$$

By equation 2.13, 2.14, and 2.15, it is easy to see that

$$z_{fc} > z_{mc} > z_{nc}. \tag{2.20}$$

When $z_{ns} < z_{nc}$, the following can be derived

$$\frac{SNA}{(1-\theta)k^{\theta}} < \frac{CAN(1+\theta)}{k^{\theta} + AN(1+\theta)}, \tag{2.21}$$

which is equivalent to

$$S < \frac{C(1-\theta)k^{\theta}}{NA}, \tag{2.22}$$

no-caching achieves the largest throughput. In the real context of CPU intensiveness in the transcoding proxy, this case is rare. Otherwise, when $z_{ns} \leq z_{nc}$, no-caching can never be the best method. Thus, in the following context, the comparison between full-caching and meta-caching is emphasized when $z_{ns} \leq z_{nc}$, which is $S \geq \frac{C(1-\theta)k^{\theta}}{NA}$.

1. When $z_{fs} > z_{fc}$, clearly, full-caching has the largest throughput, and the largest throughput is z_{fc}. Such a condition can be represented by

$$\frac{2SNA}{(1-\theta)(v+1)k^{\theta}} > \frac{CAN(1+\theta)}{vk^{\theta}}, \tag{2.23}$$

   which is

$$S > \frac{(1-\theta^2)(v+1)}{2v}C. \tag{2.24}$$

   Since $S \geq \frac{C(1-\theta)k^{\theta}}{NA}$, it is clear that when $S \geq \frac{(1-\theta^2)(v+1)}{2v}C$, full-caching has the largest throughput.

2. When $z_{fs} \leq z_{fc}$, the throughput of full-caching is z_{fs}. The performance of meta-caching needs to be further studied.

   According to the previous analysis, when $z_{fs} \leq z_{fc}$,

$$S \leq \frac{(1-\theta^2)(v+1)}{2v}C. \tag{2.25}$$

   - If $z_{ms} < z_{mc}$, the following can be derived

$$\frac{2SNA}{\alpha(1-\theta)(v+1)k^{\theta}} < \frac{CAN(1+\theta)}{vk^{\theta}(1-\beta)+AN\beta(1+\theta)}, \tag{2.26}$$

the throughput of meta-caching is thus z_{ms}, and meta-caching clearly performs best.

Accordingly, that is

$$\frac{2SNA}{\alpha(1-\theta)(v+1)k^{\theta}} < \frac{CAN(1+\theta)}{vk^{\theta}(1-\beta)}, \tag{2.27}$$

which leads to

$$S \leq \frac{\alpha(v+1)(1-\theta^2)C}{2v(1-\beta)}. \tag{2.28}$$

In a running system, the available storage and CPU resources vary dynamically, which are reflected in the corresponding values of S and C. Considering existing conditions where $S \geq \frac{(1-\theta)k^{\theta}}{NA}C$, $S \leq \frac{(1-\theta^2)(v+1)}{2v}C$, and $S \leq \frac{\alpha(1-\theta^2)(v+1)}{2v(1-\beta)}C$, we have

- When $\alpha+\beta<1$, $\frac{\alpha(1-\theta^2)(v+1)}{2v(1-\beta)}C \geq S \geq \frac{(1-\theta)k^{\theta}}{NA}C$.
- When $\alpha+\beta\geq 1$, $\frac{(1-\theta^2)(v+1)}{2v}C \geq S \geq \frac{(1-\theta)k^{\theta}}{NA}C$.

Under these conditions, meta-caching is the best choice.

- If $z_{ms} > z_{mc}$, the following can be derived

$$\frac{2SNA}{\alpha(1-\theta)(v+1)k^{\theta}} > \frac{CAN}{vk^{\theta}(1-\theta)(1-\beta)+AN\beta}, \tag{2.29}$$

the throughput of meta-caching is thus z_{mc}.

The best method is determined by the relationship between z_{mc} and z_{fs}.

- If $z_{mc} > z_{fs}$, meta-caching performs the best. When $z_{mc} > z_{fs}$, $S \leq \frac{(1-\theta^2)(v+1)}{2v(1-\beta)}C$. Since $S \geq \frac{C(1-\theta)k^{\theta}}{NA}$, $S \leq \frac{(1-\theta^2)(v+1)}{2v}C$, and $S > \frac{\alpha(1-\theta^2)(v+1)}{2v(1-\beta)}C$, we can get that

* When $\alpha + \beta < 1$, $\frac{(1-\theta^2)(v+1)}{2v}C \geq S \geq max(\frac{(1-\theta)k^\theta}{NA}C, \frac{\alpha(1-\theta^2)(v+1)}{2v(1-\beta)}C)$.

  Since $\frac{(1-\theta)k^\theta}{NA}C \approx \frac{(k/M)^\theta}{NM}C$, the following can be derived $\frac{(1-\theta)k^\theta}{NA}C \leq \frac{\alpha(1-\theta^2)(v+1)}{2v(1-\beta)}C$. Thus $\frac{(1-\theta^2)(v+1)}{2v}C \geq S \geq \frac{\alpha(1-\theta^2)(v+1)}{2v(1-\beta)}C$ and $S > \frac{(1-\theta^2)(v+1)}{2(v(1-\beta)+\frac{NA\beta(1+\theta)}{k^\theta})}C$.

* When $\alpha + \beta \geq 1$, no such S exists.

Under the above conditions, meta-caching is the best choice.

– Otherwise ($z_{mc} \leq z_{fs}$), full-caching performs the best. That is $S \geq \frac{(1-\theta^2)(v+1)}{2v(1-\beta)}C$.

Combined with existing conditions that $S \geq \frac{C(1-\theta)k^\theta}{NA}$, $S \leq \frac{(1-\theta^2)(v+1)}{2v}C$, and $S > \frac{\alpha(v+1)(1-\theta^2)}{2v(1-\beta)}C$, the following can be derived

* When $\alpha + \beta < 1$, $\frac{(1-\theta^2)(v+1)}{2v}C \geq S \geq \frac{\alpha(v+1)(1-\theta^2)C}{2v(1-\beta)}$ and $S \leq \frac{(1-\theta^2)(v+1)}{2(v(1-\beta)+\frac{NA\beta(1+\theta)}{k^\theta})}C$.

* When $\alpha + \beta \geq 1$, no such S exists.

Under the above conditions, full-caching is the best choice.

Table 3.3.2 summarizes the result briefly.

Table 2.2: Analysis results when $0 < \theta < 1$

Condition	Best scheme
$z_{ns} < z_{nc} (S < \frac{C(1-\theta)k^\theta}{NA})$	No-caching
$z_{ns} \geq z_{nc}, z_{fs} > z_{fc} (S > \frac{(1-\theta^2)(v+1)}{2v}C)$	Full-caching
$z_{ns} \geq z_{nc}, z_{fs} \leq z_{fc} (S \leq \frac{(1-\theta^2)(v+1)}{2v}C), z_{ms} \leq z_{mc}$	Meta-caching
$z_{ns} \geq z_{nc}, z_{fs} \leq z_{fc} (S \leq \frac{(1-\theta^2)(v+1)}{2v}C), z_{ms} > z_{mc}$ and $z_{mc} > z_{fs}$	Meta-caching
$z_{ns} \geq z_{nc}, z_{fs} \leq z_{fc} (S \leq \frac{(1-\theta^2)(v+1)}{2v}C), z_{ms} > z_{mc}$ and $z_{mc} \leq z_{fs}$	Full-caching

When $\theta = 1$, the analysis leads to the result as shown in Table 2.3.

Table 2.3: Analysis results when $\theta = 1$

Condition	Best scheme
$z_{ns} < z_{nc} (S < \frac{Ck}{NA * \ln k})$	No-caching
$z_{ns} \geq z_{nc}, z_{fs} > z_{fc} (S > \frac{C}{\ln k})$	Full-caching
$z_{ns} \geq z_{nc}, z_{fs} \leq z_{fc} (S \leq \frac{C}{\ln k}), z_{ms} \leq z_{mc} (S \leq \frac{\alpha * C}{(1-\beta) * \ln k})$	Meta-caching
$z_{ns} \geq z_{nc}, z_{fs} \leq z_{fc} (S \leq \frac{C}{\ln k}), z_{ms} > z_{mc} (S \leq \frac{\alpha * C}{(1-\beta) * \ln k})$ and $z_{mc} > z_{fs}$	Meta-caching
$z_{ns} \geq z_{nc}, z_{fs} \leq z_{fc} (S \leq \frac{C}{\ln k}), z_{ms} > z_{mc} (S \leq \frac{\alpha * C}{(1-\beta) * \ln k})$ and $z_{mc} \leq z_{fs}$	Full-caching

2.4 Design of AMTrac

The previous analysis shows that full-, meta- and no-caching schemes perform best under different conditions.

A system aiming at maximizing the throughput should thus adaptively use different schemes upon dynamic client access behaviors and resource availability. To this end, we present the adaptive meta-caching design for transcode-enabled proxy, called AMTrac.

In AMTrac, the proxy provides v different versions of an object to client requests. For each object version, it might be cached with its metadata only or cached with its fully transcoded result. Thus, in AMTrac, the cache space is logically spitted into two parts. One is for caching metadata of different object versions, and the other is to cache the fully transcoded object versions. The size of each part is changing dynamically.

In AMTrac, the system keeps track of the requested object version, even after an object version is evicted. We use the following data structure for each object to record important runtime information to assist the implementation of the proposed strategy.

In this structure, P_s is calculated as the ratio of the *active disk size* over the *total disk size*. The *active disk size* is the sum of the size of all active (being requested) cached objects at present. The *total disk size* is the total available cache size. Based on these, the following policies work together to implement AMTrac. In AMTrac, three modules: *progressive request admission*, *resource-driven replacement policy*, and *reactive cache adjustment* are designed. The detail functionalities of each module are described as follows.

Table 2.4: Fields of data structure

field name	field information
v	version number: the total number of versions of each object
$ms[1..v]$	meta_size: an array to record the metadata size of each version
$fs[1..v]$	full_size: an array to record the full data size of each version
$r[1..v]$	references: an array to record the references to each version
$s[1..v]$	status: whether the object is currently being replaced (0) or being cached (1)
$u[1..v]$	utility value: the utility value of the object version
P_s	storage utilization: current storage utilization (%)
P_c	CPU utilization: current CPU utilization (%)

2.4.1 Progressive Request Admission

Upon a client request for the j^{th} version of an object, there are three cases as follows.

- If the being requested object version j is fully cached, the request is directly served and the corresponding data item, $r[j]$, is increased by one.
- If the being requested object version is cached with its metadata, the request is served with the online meta-transcoding (transcoding based on metadata). The transcoded result is sent to the client. The corresponding data item, $r[j]$, is increased by one. The reactive cache adjustment (see section 2.4.3) is activated to determine whether this fully transcoded object version should be cached or not.
- If the being requested object version does not exist in the cache:
  - If the object version is firstly accessed, the proxy performs online transcoding to produce the object version j. Correspondingly, the metadata gets cached. The corresponding data item, $r[j]$, is increased by 1. $ms[j]$ gets updated. The status of this object version $s[j]$ is set to one. If there is no sufficient cache space, the replacement policy (see section 2.4.2) is activated to make room for its caching.
  - If the object version has been accessed and its status is replaced ($s[j] = 0$), the object version is transcoded again and sent to the client. $r[j]$, is increased by

one. The reactive cache adjustment (see section 2.4.3) is activated to determine whether the corresponding metadata should get cached.

2.4.2 Comprehensive Replacement Policy

When the cache space is not enough, the replacement is activated. To maximize the cache performance, clearly, it is important to select the right victim.

In our design, the utility based policy is used to select the right victim. The utility function is designed as follows:

$$U(i_j) = \begin{cases} \frac{r_j}{S_j} \times \frac{\alpha_j}{\alpha_j+\beta_j} & \text{if } P_c \geq P_s, \\ \frac{r_j}{S_j} \times \frac{\beta_j}{\alpha_j+\beta_j} & \text{if } P_c < P_s. \end{cases} \tag{2.30}$$

In equation 2.30, S_j indicates the occupied cache space of this object version, where $S_j = max(ms[j], fs[j])$; r_j is the reference number to this object version; α_j is the storage unit (in percentage) used for caching the result, while β_j is the CPU unit to transcode to the final version; P_c represents the current CPU utilization, and P_s indicates the current storage utilization. Thus, in this equation, $\frac{r_j}{S_j}$ considers that if an object version is more popular with a unit storage, the object version should have a high utility value and gets a better chance to be cached. If $P_c > P_s$, the current system is CPU constrained, so the utilization of the storage is encouraged. If $P_c < P_s$, the current system is storage constrained, then the utilization of the CPU is encouraged.

Thus, the utility of an object version considers both the object popularity and the current available resources to find the least valuable object version. Each time the replacement policy is activated, all cached object versions must refresh their utility. Based on the utility function, we design the replacement policy as follows.

When the replacement policy is activated, it compares all the object versions in the cache based on their utility values. The one with the minimum value is selected as the

victim and the following procedure loops until sufficient space is found.

- If the selected victim has its fully transcoded data cached in the proxy, the system selects to evict its fully cached data and caches its meta data (it consumes a bit more CPU and it is done when the next time a request is received for this object version). $fs[j]$ is set to 0, while $ms[j]$ is updated accordingly.

- If the selected victim has only metadata cached, the metadata is evicted, $ms[j]$ is set to 0. $r[j]$ is set to 0. The corresponding object version status, $s[j]$, is also set to be replaced.

2.4.3 Reactive Cache Adjustment

According to our design, for each object version, there can be three possible statuses, namely replaced, with metadata cached, or with fully transcoded result cached. To accommodate the dynamic client accesses and thus to maximize the system performance, we design the reactive cache adjustment policy upon different situations. This adjustment is passive since it is always invoked due to new client requests.

- If a currently replaced object version is accessed, the system starts to evaluate whether the current utility of this object version is increased and is large enough to get cached. The new utility is calculated assuming the object version consumes $ms[j]$ for storage. If it is larger than any cached one, the metadata of this object version gets cached and $ms[j]$ and $u[j]$ are set accordingly. The replacement policy is activated if needed.

- If a currently object version is cached with metadata, the system starts to evaluate whether the fully transcoded data of this object version should be cached or not. Assuming the fully transcoded object version is cached using space $fs[j]$, its corresponding utility value is compared with the current cached object versions. If the new utility of this object version is higher than any cached one, the fully transcoded object version gets cached. Its $fs[j]$ and $u[j]$ are updated accordingly. Correspondingly, its

metadata gets evicted and $ms[j]$ is set to 0. Additional space is reclaimed with the assistance of the replacement policy when necessary.

2.5 Performance Evaluation

In this section, four strategies for rate reduction transcoding are evaluated. These four strategies are no-caching, full-caching, meta-caching, and our proposed AMTrac.

2.5.1 Simulation Setup

Bit rate reduction is a typical transcoding process where meta-caching can be applied to cache the requantization factor M_q. The experimental study on such an application is first conducted. To capture the real setup for the storage usage (α) and CPU (β), a full transcoder and a meta transcoder using C language with no special optimizations are implemented. Through transcoder runs on HP X4000 workstation with 2 GHz Intel Xeon CPU, the CPU time used by full transcoding and meta transcoding on MPEG test sequences with spatial resolution 352x240 coded at 25 fps are compared. The original video is coded at the rate of 512 Kbps.

Considering the bit rate reduction transcoding, caching M_q costs storage at 8250 bytes/sec (please refer to [50] for more details). However, bypassing the rate control sub-module enables the transcoder to get the final result with only 45% of the computing load in comparison to a full session. With a target bit rate at 128Kbps, the selection of this meta-caching point represents (α, β) as $(0.5, 0.45)$. Note that in the rate reduction case, the metadata size does not change with the bit rate of the target object. Thus, the corresponding α values for version 1, 2, and 3 are 0.167, 0.25, and 0.5. The β values for different versions are the same.

Based on these parameters obtained from a real transcoding application, experiments are conducted when the cache size varies from 4% to 60% of the total unique object size, and the total amount of CPU capacity is set to 100 units. Correspondingly, θ is set to

0.73 [55–57].

In each workload, there are a total of 1000 objects. Each workload contains 20,000 client requests. The client arrival pattern follows a Poisson distribution with the mean arrival rate as 1 request per 10 seconds, and the maximum arrival interval of 50 minutes. The clients depart randomly after receiving the service for a duration ranging from 5 minutes to 40 minutes. For each object, there are 4 different versions, including the original best quality object – version 0. Version 3 represents the lowest quality version. The storage used by version 1, 2, and 3 is 3/4, 2/4, 1/4 of the original object size (version 0), similar to what is used in the analytical model. Their corresponding encoding rates are 512 Kbps, 384 Kbps, 256 Kbps, and 128 Kbps. The total unique original object size amounts to 89.9 GB. The total traffic is 1205.8 GB, and the access duration lasts about 34 hours.

2.5.2 Experimental Results

There are two evaluation metrics used in the experiments: *Locally Completed Session*, and *Average CPU Load*. Figure 2.6 shows the Locally Completed Session for four different methods when the cache size increases. In this figure, *No*, *Full*, *Meta* represents the no-caching, full-caching, and meta-caching methods as discussed. *AMTrac* represents the proposed scheme. As shown in Figure 2.6, when considering the total completed sessions in the transcoding proxy, the performance of the four methods in decreasing order are *AMTrac*, *Meta*, *Full*, and *No*. The confidence interval is shown in this figure and the confidence level used is 95%. Please note that the confidence level is set to 95% hereafter.

Having examined the overall performance in terms of the total completed sessions in the transcoding proxy, now we examine the proxy's performance at each time unit. Figure 2.7 shows the system throughput along the client accesses when the cache size is 4%. As indicated in the figure, *AMTrac* outperforms all other methods. Among all the four methods, no-caching achieves the worst performance as expected and full-caching also has worse performance compared with meta-caching and AMTrac.

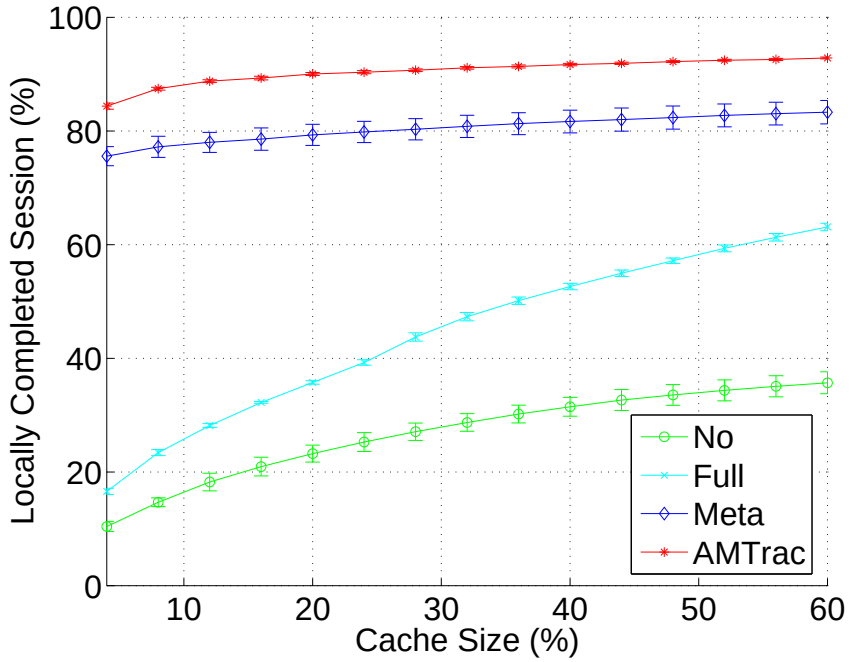

Figure 2.6: Total locally completed session

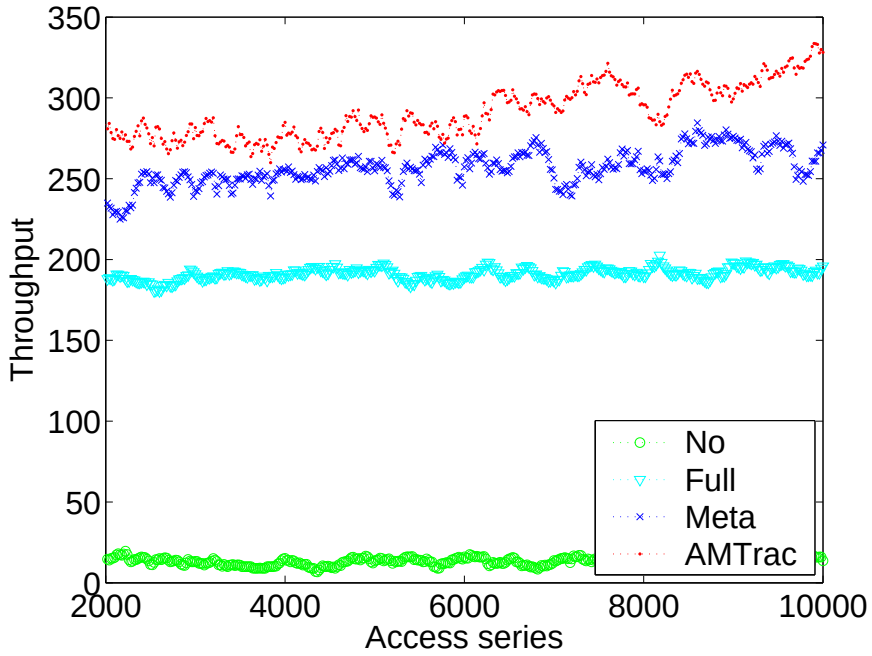

Figure 2.7: System throughput

Figure 2.8 shows the corresponding CPU load along the client accesses (time). Apparently, besides full caching, the other three methods always have a 100% CPU load or close

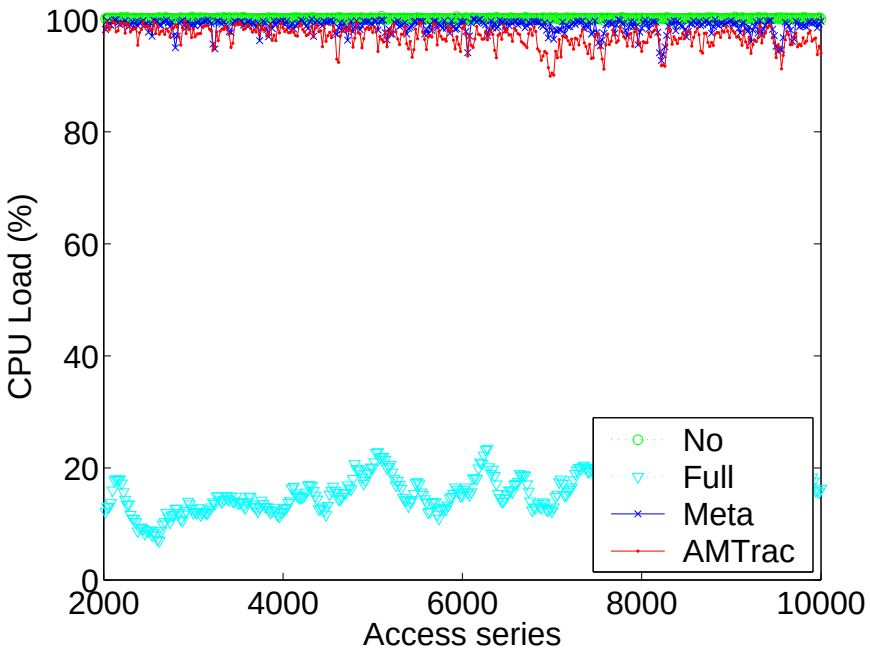

Figure 2.8: CPU load

to 100%. For full-caching, since 4% cache space is far from sufficient, its CPU resource is under utilized.

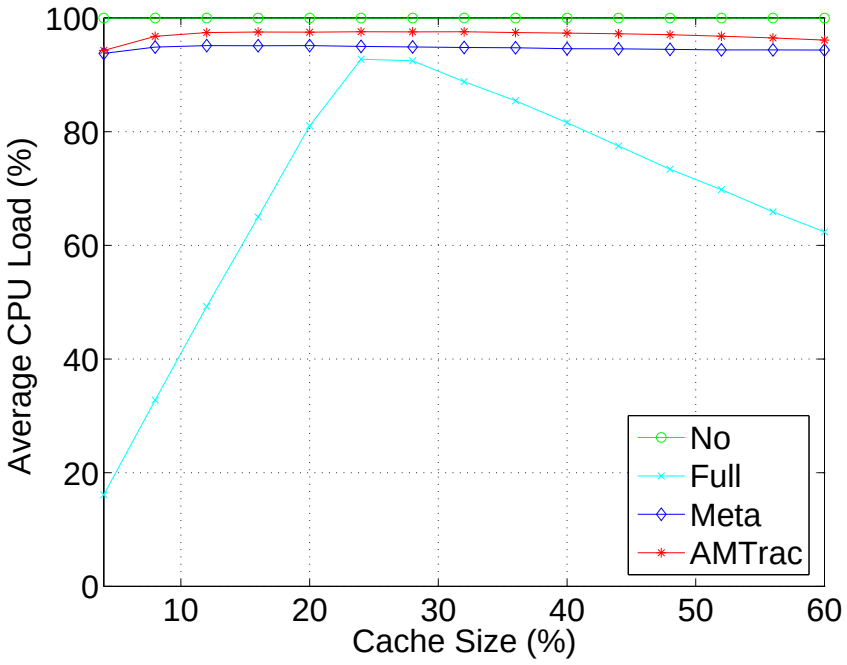

Figure 2.9: Average CPU load

Figure 2.9 further shows the average CPU load for the four schemes when the available cache size varies from 4% to 60%. As expected, the CPU load of no-caching is not affected when the cache size increases. Since full-caching is the most affected by the available cache space, its average CPU load decreases after the cache size increases beyond 28%. Before that, the limited cache space results in frequent replacement upon new client requests, some of which cannot happen because the selected victim objects are being accessed. Thus, although there are spare CPU cycles, they are not utilized. After the cache size is increased beyond 28%, this situation is relieved.

Since full-caching is storage constrained, with larger cache space, more client requests can be served from the cache without repetitive transcoding. This is evidenced by the decreasing order of CPU load of full-caching when the cache size is beyond 28%. From another aspect, although meta-caching and AMTrac have a higher CPU load with the increase of cache size, the throughput of these two approaches is higher than that of full-caching.

The performance is further evaluated when (1) the skew factor (θ) varies from 0.47 to 0.73; (2) β varies from 0.3 to 0.6; and (3) total available CPU varies from 80 to 120 units. Again, experiments are run when the cache size varies from 4% to 60% of the total unique object size.

Based on varying object access behaviors, when θ (for the Zipf-like distribution of object popularity) is changing, the performance is first studied. Second, different content adaptation processes may have different α and β values. The performance of different schemes when β is varying is compared. These experiments are conducted with CPU capacity of 100 units. Since any meta-caching scheme represents a point in the storage-computing space, varying one dimension affects the overall tradeoff in resource utilization. In the following experiments, the total CPU capacity is changed to study the performance of different schemes.

- Impact of Object Popularity (θ)

  Existing research reveals that the skew parameter in the Zipf-like distribution is in

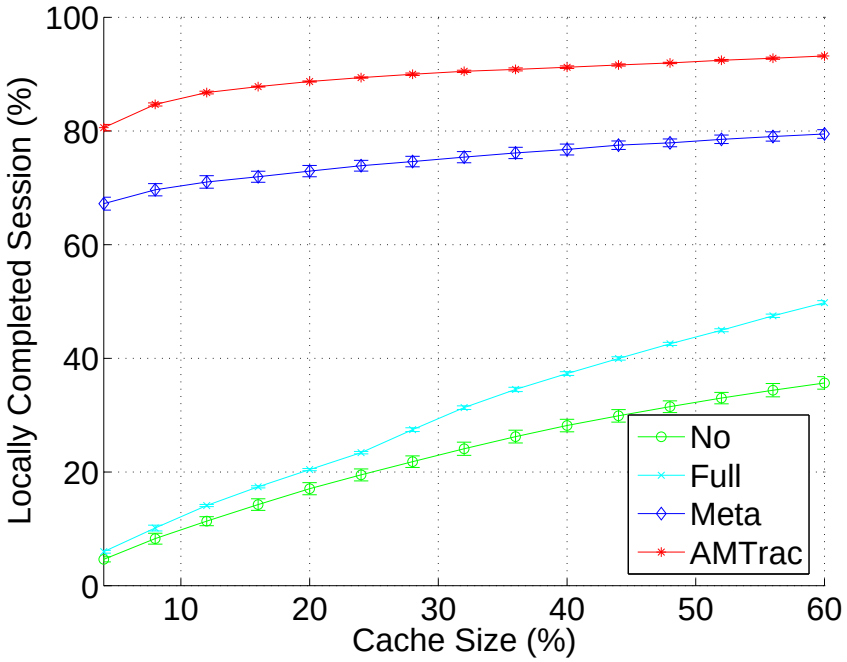

Figure 2.10: Total locally completed session– θ=0.47

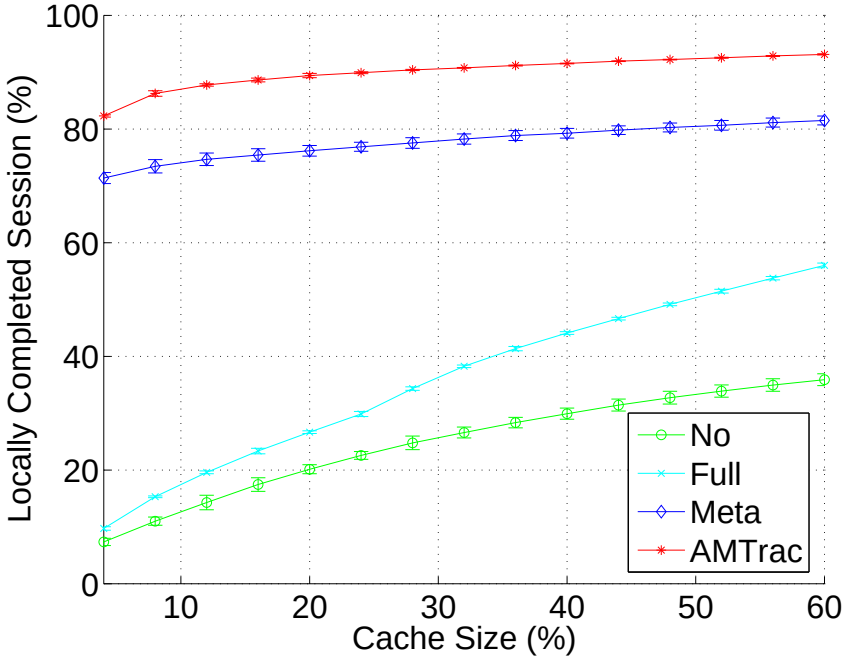

Figure 2.11: Total locally completed session – θ=0.60

the range of 0.47 and 0.73 for media objects [55–57]. In this section, the impact of θ on different methods is evaluated. Three typical values of 0.47, 0.60, and 0.73 are

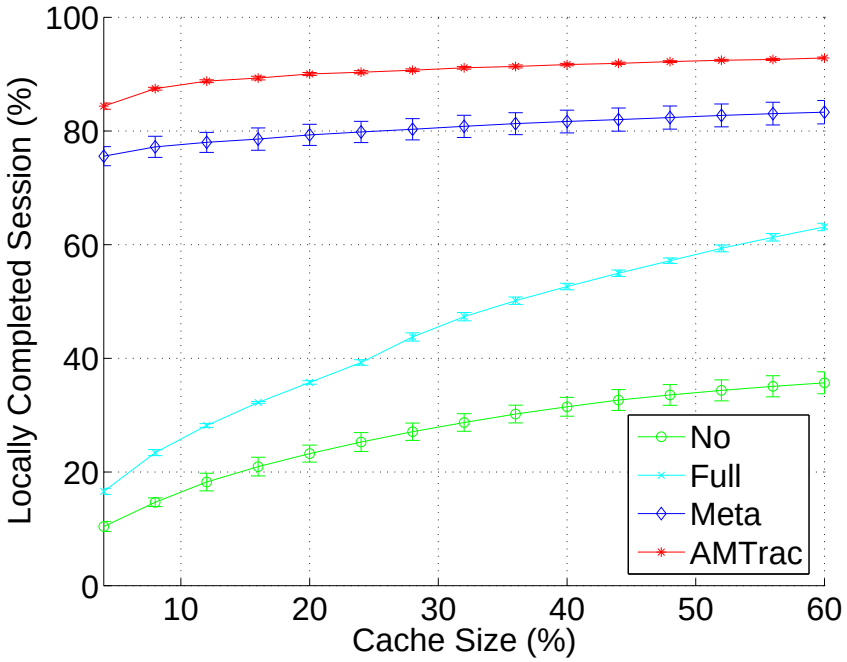

Figure 2.12: Total locally completed session – θ=0.73

tested, while α and β are fixed as 0.5 and 0.45. Note that only the overall system performance reflected by the Locally Completed Session and its transcoding part is shown.

Figure 2.10, Figure 2.11, and Figure 2.12 show the total completed sessions by the proxy for the four strategies when θ varies. In general, these figures indicate θ impacts all methods in terms of the Locally Completed Session, and its impact is more pronounced on full-caching, meta-caching, and AMTrac. It is reasonable since a larger θ indicates more clustered client accesses and more opportunities for these methods to use cached data to serve incoming requests.

Correspondingly, Figure 2.13, Figure 2.14, and Figure 2.15 show the Locally Completed Session with Transcoding for the four caching strategies. It is also shown that the increase of θ impacts on all methods. But the improvement is not as large as their corresponding improvement in the total completed sessions, indicating that the increment of θ also impacts the non-transcoded sessions (the figures are omitted for brevity). This is clearly due to the increased skew of the object popularity.

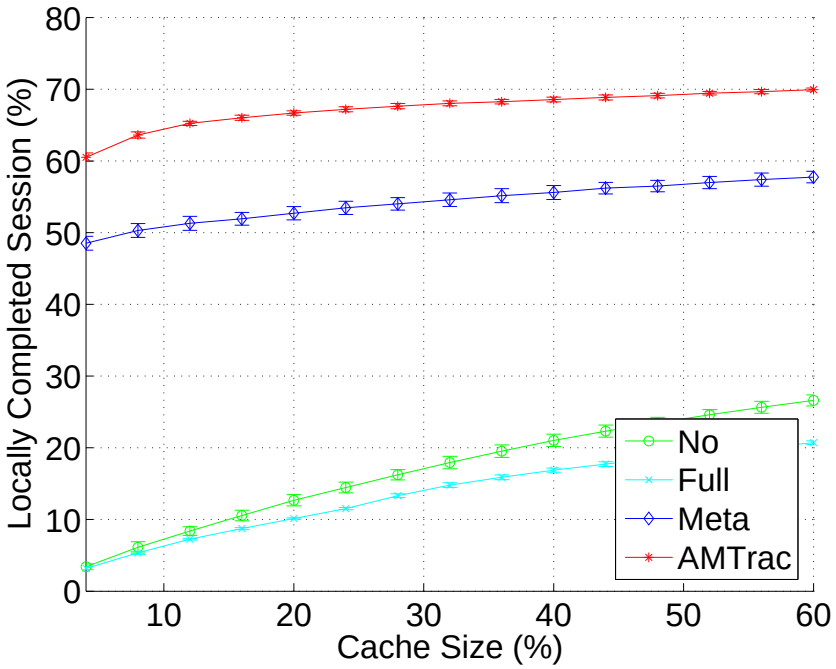

Figure 2.13: Transcoded locally completed session – θ=0.47

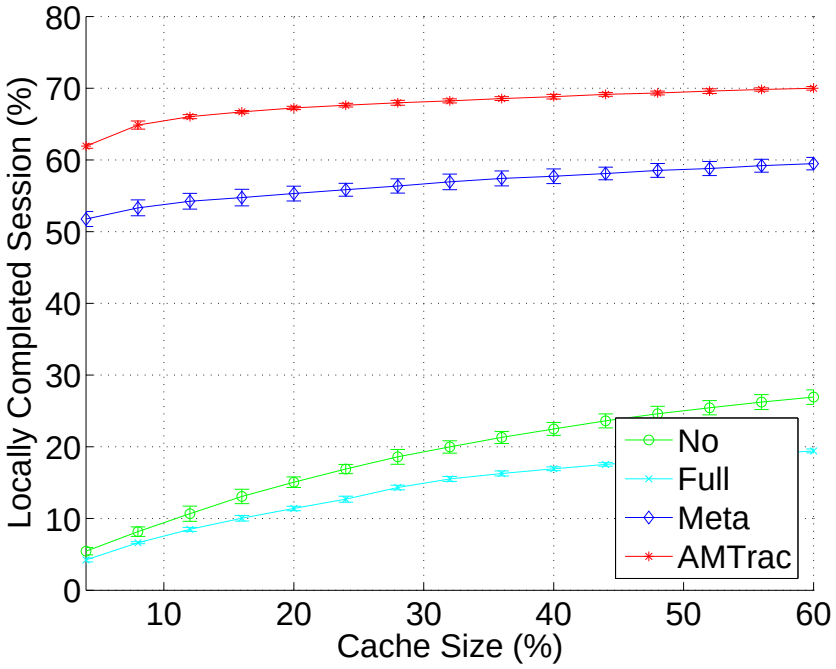

Figure 2.14: Transcoded locally completed session – θ=0.60

- Impact of Meta-Caching Parameters ($\alpha + \beta$)

  The meta-caching parameters determined by real experiments as shown before are

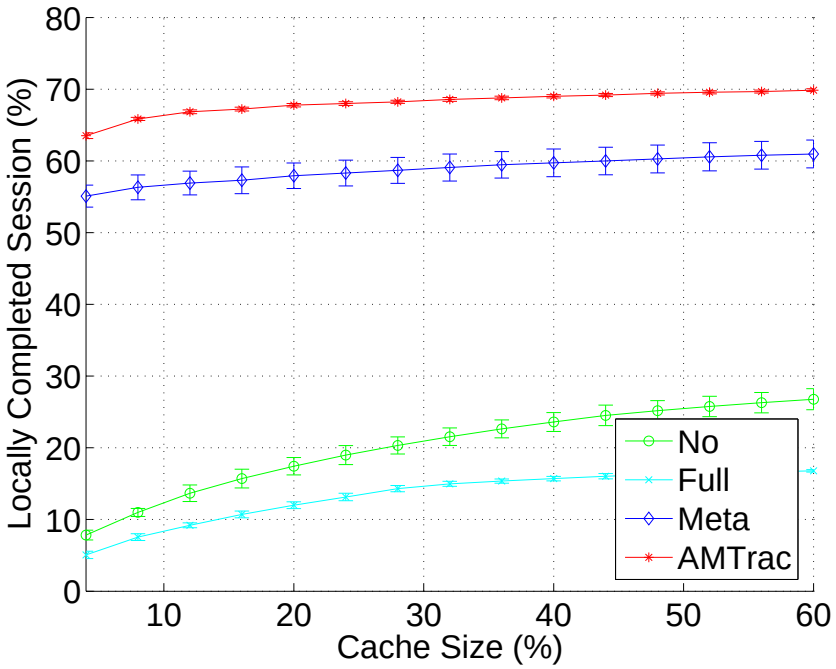

Figure 2.15: Transcoded locally completed session – θ=0.73

machine dependent. For example, β is affected by the CPU performance, and the previous analysis shows that the system performance is related to the sum of α and β. In this section, how the varying sum of α and β affects the performance of different schemes is further studied.

In the following experiments α is fixed at 0.5 and θ is fixed at 0.73 when β varies.

Figure 2.16 , Figure 2.17 (the same as Figure 2.12), and Figure 2.18 show the Locally Completed Session for the four strategies. When β varies from 0.3 to 0.6, meta-caching is significantly affected. The larger the β, the smaller the Locally Completed Session. The impact on AMTrac is trivial. The reason is that meta-caching consumes more CPU, while AMTrac can adaptively balance the usage of CPU and storage, and it is less affected. This confirms the effectiveness of the proposed adaptive AMTrac scheme. Since full-caching and no-caching schemes have nothing to do with β, the performance of these two methods is not affected.

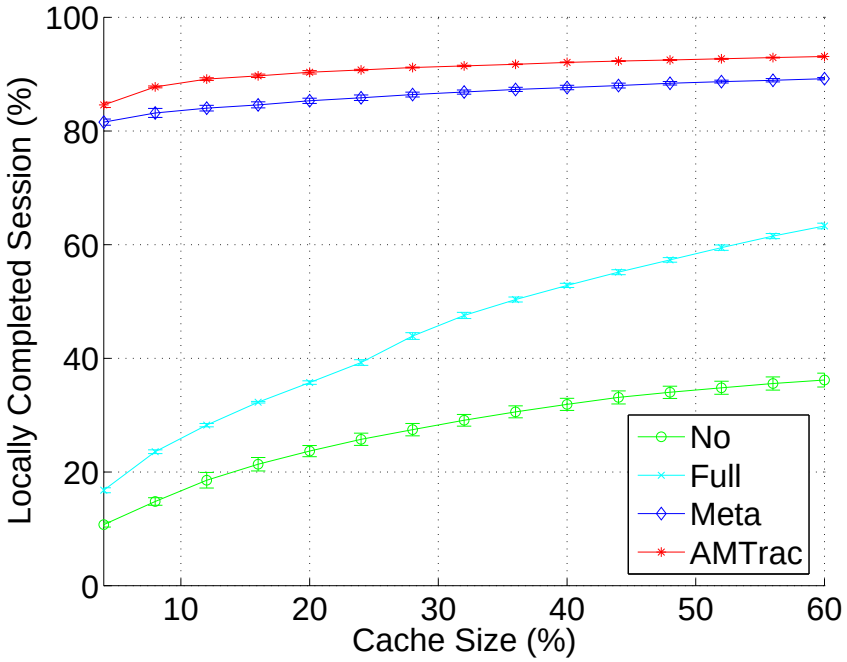

Figure 2.16: Total locally completed session – β=0.3

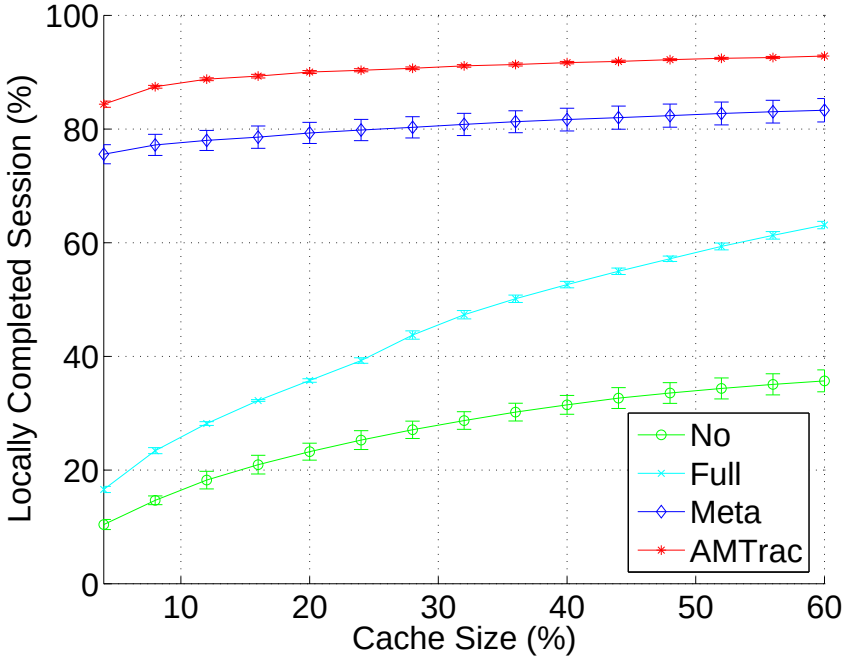

Figure 2.17: Total locally completed session – β=0.45

Figure 2.19, Figure 2.20 (the same as Figure 2.15), and Figure 2.21 show the corresponding Completed Sessions with Transcoding for the four caching strategies. They

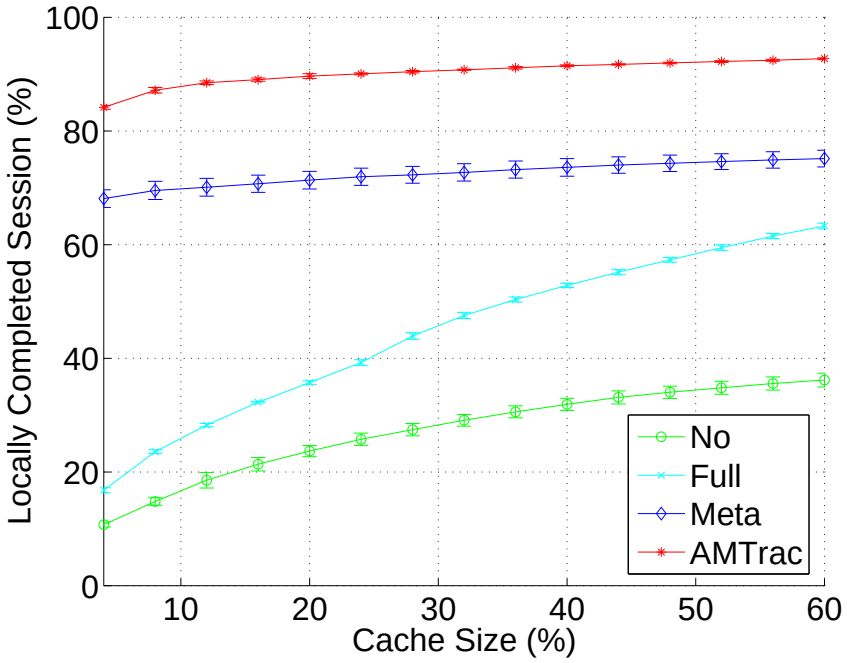

Figure 2.18: Total locally completed session – β=0.6

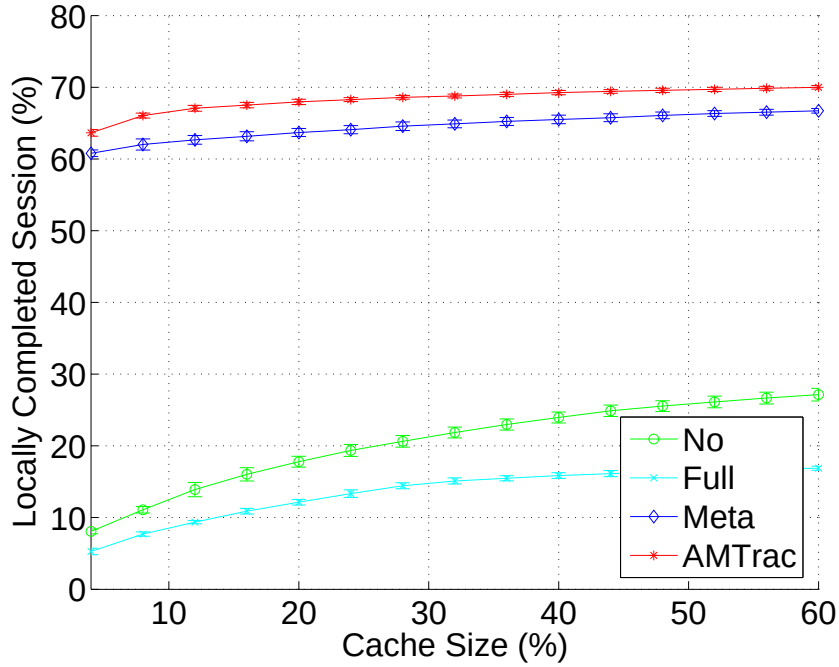

Figure 2.19: Transcoded locally completed session – β=0.3

show similar trends as the Locally Completed Session. This is expected since the variation of β affects the performance of transcoding, while the part without transcoding is

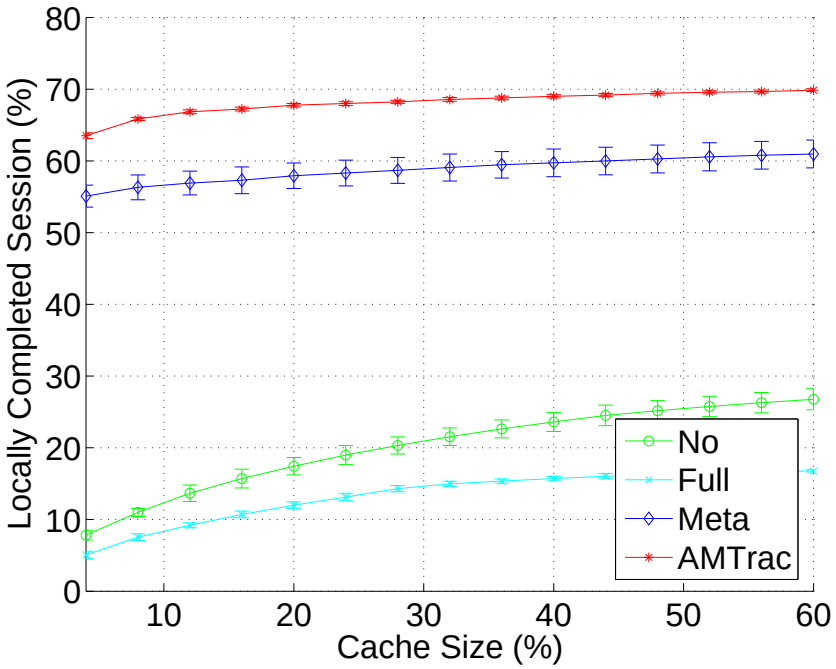

Figure 2.20: Transcoded locally completed session –β=0.45

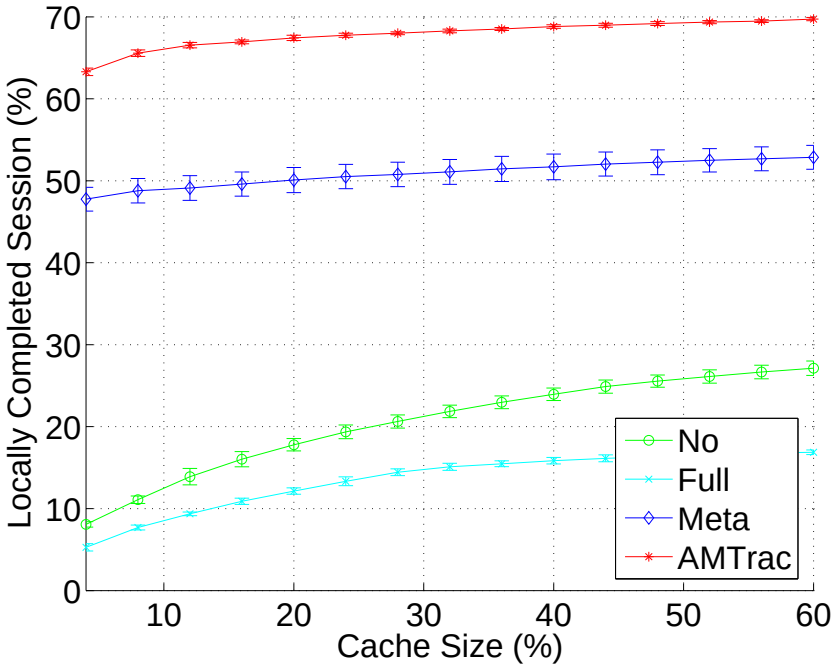

Figure 2.21: Transcoded locally completed session – β=0.6

not affected at all. Overall, these results reflect that a larger demand of CPU resource (β) seriously affects the performance of meta-caching due to its lack of adaptiveness.

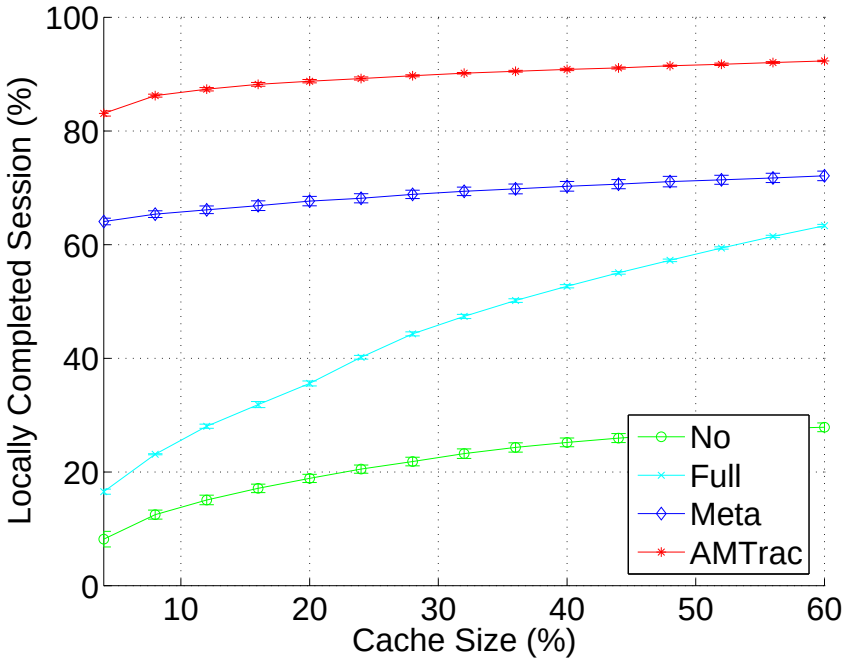

Figure 2.22: Total locally completed session – CPU = 80

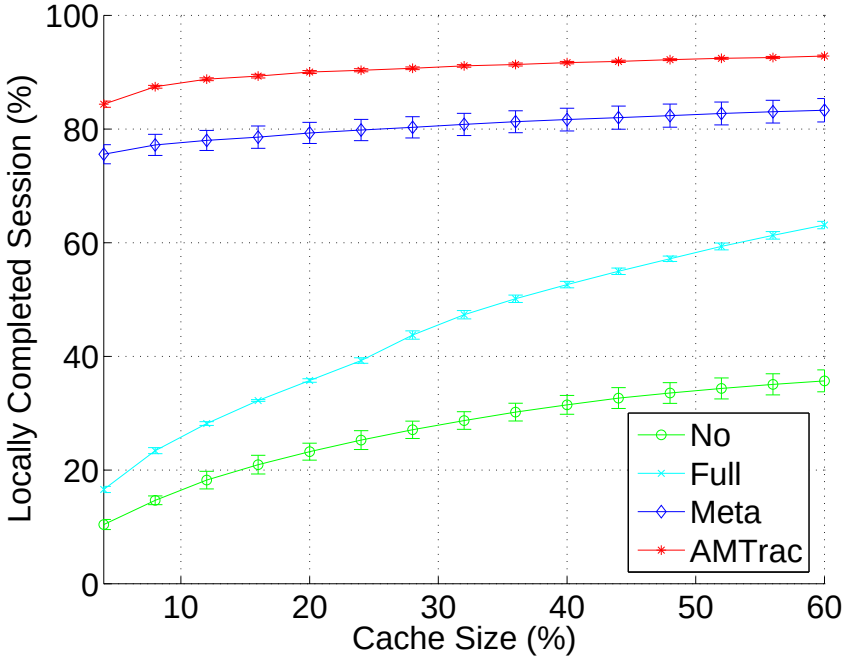

Figure 2.23: Total locally completed session – CPU = 100

- Impact of Total CPU

  Since meta-caching and AMTrac heavily rely on CPU resource, how the CPU resource

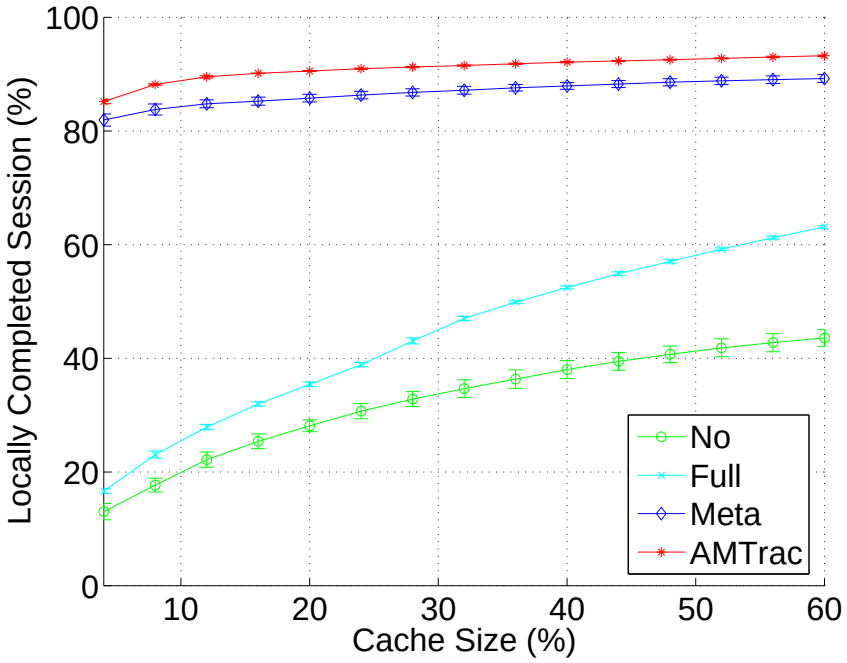

Figure 2.24: Total locally completed session – CPU = 120

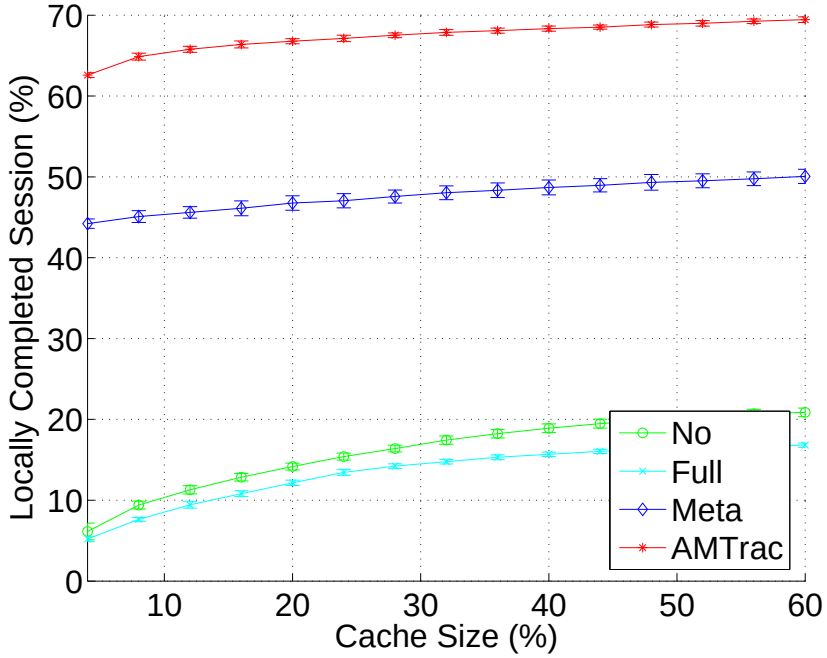

Figure 2.25: Transcoded locally completed session – CPU = 80

affects the performance of different methods is studied. In this set of experiments, the total CPU capacity varies from 80 to 120 units. For this set of experiments, α is set

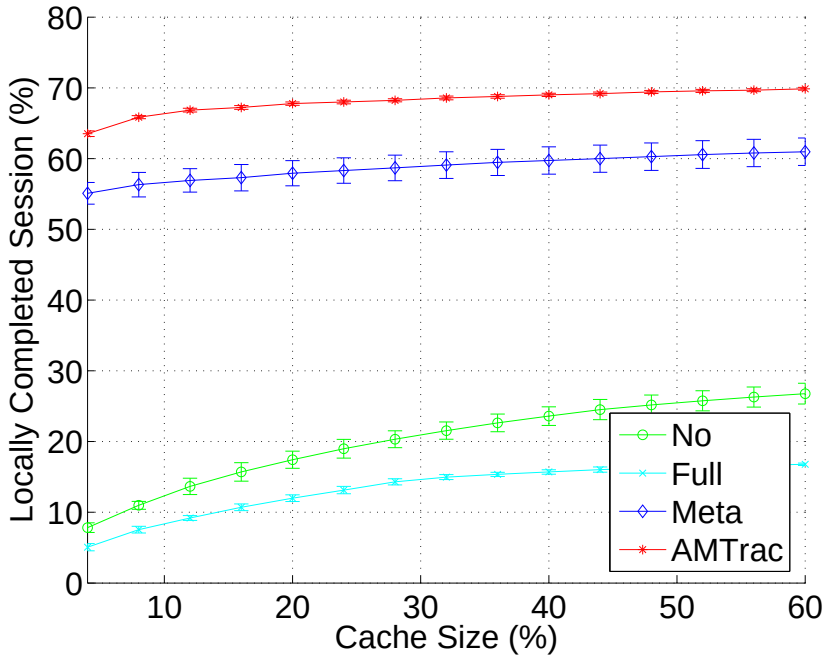

Figure 2.26: Transcoded locally completed session – CPU = 100

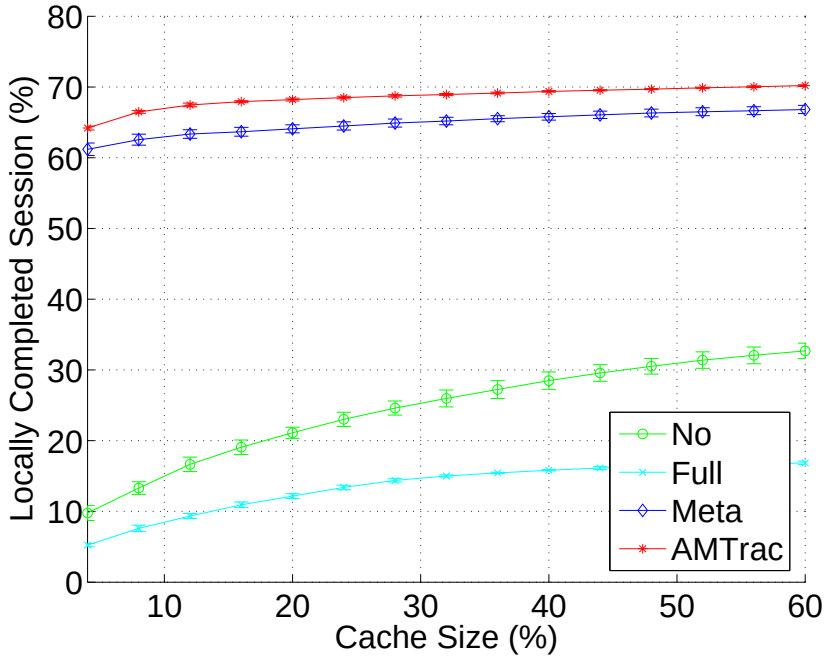

Figure 2.27: Transcoded locally completed session – CPU = 120

to 0.5, β is set to 0.45 and θ is set to 0.73.

Figure 2.22, Figure 2.23 (the same as Figure 2.12), and Figure 2.24 show the corresponding Locally Completed Sessions for the four strategies when the available CPU varies. When the available CPU units increase, the performance of all methods gets improved. But the improvement of meta-caching and no-caching is more pronounced than the other two. The reason is that meta-caching and no-caching are more dependent on the available CPU resources, while full-caching demands more cache space. Only AMTrac is the least affected, particularly when the cache space increases beyond 16%.

Figure 2.25, Figure 2.26 (the same as Figure 2.15), and Figure 2.27 show the corresponding Completed Sessions with Transcoding for the four strategies. These three figures show the same trends as their counterparts for total completed sessions, which indicates that the changes of the Locally Completed Sessions for different methods are largely due to the changes of the completed sessions through transcoding.

These experiments demonstrate that when various conditions, such as available resources and client access pattern, change, the proposed AMTrac can always maintain a balance in using the CPU and storage to maximize system performance under different conditions.

2.6 Summary

The rapid increase of Internet media contents and widespread use of portable devices challenges the existing streaming infrastructure. While transcoding proxy has been proposed and researched extensively, existing strategies generally aim at reducing the server load and server traffic and little attention has been paid to transcoding procedure itself. This leads to less flexibility in addressing the tradeoff between computing and storage resources. By studying a transcoding process itself, we design a new caching strategy with the main focus on computing load reduction. Furthermore, a meta-caching scheme is proposed to offer a new point of control in the computing and storage space. With model-based analysis on the meta-caching scheme, a meta-caching system, which can adaptively use the meta-caching

scheme based on client access pattern and available resources in the system is proposed. Experimental results show that it significantly outperforms existing strategies under various conditions.

Chapter 3: Peer-Assisted Transcoding for Live streaming

3.1 Introduction

In the previous chapter, how to address the MDH problem in on-demand streaming systems by efficiently managing the CPU and the storage resources is discussed. The MDH problem is not only notable in on-demand streaming systems, but also becomes notable in live streaming systems due to the increasing popularity of P2P live streaming systems. However, in P2P/Overlay streaming systems, the MDH problem cannot be addressed as what we have proposed for on-demand streaming since there is generally no dedicated proxy server for online transcoding. In this chapter, we will present how to address the MDH problem in P2P/Overlay live streaming systems by efficiently leveraging the idle peer computing resources with the trade-off to the bandwidth resources (compared with the CPU and the storage trade-off in on-demand streaming systems).

Although existing studies have considered that participating clients may have different capabilities in uploading bandwidth in streaming systems [58], they did not consider the MDH problem when various mobile devices join the system. Some solutions to address the MDH problem based on Scalable/layered Coding (SC) [33,34] or Multiple Description Coding (MDC) [29] may not be effective due to the limitations discussed in chapter 1. Other studies [5, 29, 59] address the MDH problem in live streaming by using new structures in the streaming systems. For example, Padmanabhan et al. [29] used multiple distribution trees and MDC to provide redundant streaming, and enhance the robustness of content distribution in the system. Splitstream [59] is a high-speed content distribution system where streaming content is divided into several strips in order to distribute the forwarding load among all the peers in a fair way. Different from structured streaming systems, unstructured P2P/overlay streaming systems have also been studied. For example, Chainsaw [60]

provides unstructured solutions to high-speed data dissemination systems. A scalable architecture is proposed for congestion controlled multicast real-time communication by using self organized transcoder in [61]. Some studies try to provide different qualities to heterogeneous devices in the streaming systems by transcoding. For example, peers in Dagster [62] are assumed to be able to conduct transcoding when necessary during the data dissemination process, and provide appropriate video qualities to other peers. However, it is not clear how a peer could efficiently conduct transcoding at runtime.

As transcoding solution is more flexible in adapting to heterogeneous environments, some research has been conducted to study online transcoding [41, 42, 45]. But the key challenge remains that such a solution is computing intensive because these studies treat the transcoding procedure as a black box with the final transcoded result as the only caching candidate in the storage. In chapter 2, the idea of meta-caching is introduced to reduce the computing consumption during the transcoding. Meta-caching identifies intermediate transcoding steps from which certain intermediate results can be cached and re-used later so that a fully transcoded object version can be produced from the metadata with relatively smaller amount of CPU cycles. Meta-caching demonstrates an effective tradeoff between the storage and CPU consumption. Such metadata-assisted transcoding is called meta-transcoding.

In this chapter, Peer-Assisted Transcoding (PAT) is proposed to facilitate heterogeneous streaming accesses from various mobile devices by efficiently managing peers' CPU and bandwidth resources in P2P/overlay live streaming systems. PAT scheme relies on client cooperation without extra infrastructure support. In PAT, media content adaptation is fulfilled through meta-transcoding by participating peers. In addition, the sharing of the metadata is facilitated by the construction of an additional overlay, called metadata overlay, which co-exists with the data streaming overlay. This extra overlay helps transcoding peers to instantly share the intermediate results of a transcoding procedure with other transcoding peers, aiming to minimize the total computing overhead in the system. PAT is effective in satisfying diverse demands from heterogeneous participating devices in a

P2P/overlay streaming system. Through extensive real-data-parameterized simulations, it is shown that the client-perceived streaming quality in PAT is significantly improved. With a small amount of additional bandwidth (6%), PAT can reduce the CPU load (up to 58%) for online transcoding.

To the best of our knowledge, PAT is the first to combine transcoding and live P2P/overlay streaming systems. It is generic, and can be used to design both tree- and mesh-based transcoding assisted overlay streaming systems. In this chapter, the tree-based overlay streaming system is used as an example to introduce the scheme for illustrative purposes.

The remainder of this chapter is organized as follows. The system design is described in Section 3.2. The performance modeling is discussed in Section 3.3. The performance evaluation based on simulation is presented in Section 3.4. This chapter is summarized in Section 3.5.

3.2 System Design

In this section, we present how to leverage metadata transcoding for P2P/overlay streaming, where a media stream is disseminated to a large number of clients, utilizing the forwarding capacity of the participating peers. In addition, a peer is able to instantly share the intermediate transcoding results (metadata) with other peers performing the identical transcoding procedure. This approach reduces the overall CPU overhead in the system. Clearly, additional bandwidth is required for sharing metadata between transcoding peers. In the following context, a protocol that can reduce computing load with small bandwidth overhead based on meta-transcoding is presented in detail.

First, the metadata matrix is briefly introduced. Then the design of Peer-Assisted Transcoding (PAT) scheme, which consists of two overlays, is presented. Among these two overlays, one is the data overlay, which is used to deliver streaming data. The other is the metadata overlay, which is used to instantly share metadata to efficiently minimize the total computing overhead for online transcoding by trading a small amount of bandwidth for a large amount of savings on CPU cycles in the system.

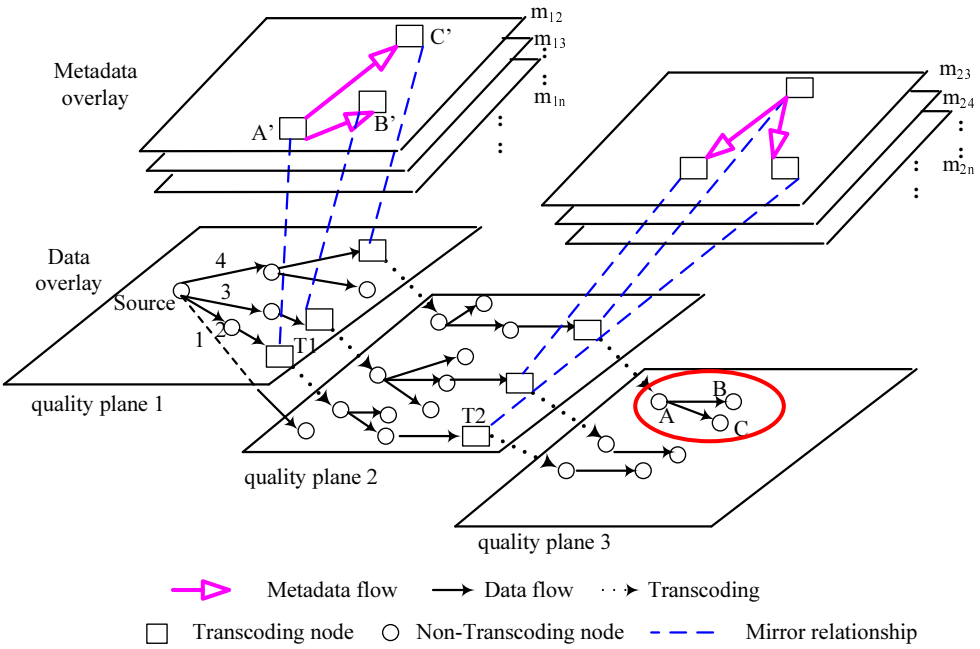

Figure 3.1: A bi-overlay Example

3.2.1 Quality plane and transcoding plane

As shown in Figure 3.1, the proposed system consists of two overlays: data overlay and metadata overlay.

On the data overlay, the peers that request the same streaming quality form a certain quality plane. Formally, *quality plane i (qp_i)* represents the set of peers which request quality i and there are three quality planes shown in Figure 3.1. As shown in Figure 3.1, different quality planes are connected through transcoding nodes and these quality planes form a "quality version falls". With a total of n different quality versions in the system, there are at most n quality planes.

On the metadata overlay, based on the transcoding input and output versions, there are different transcoding planes for transcoding nodes. Formally, *transcoding plane ($tp_{i,j}$)* represents the set of nodes which transcode from quality i to quality j. There are several transcoding planes shown in Figure 3.1. For example, in Figure 3.1, transcoding nodes A', B', and C' conduct the transcoding from quality version 1 to quality version 2 and thus are

on the same transcoding plane (denoted as $tp_{1,2}$). Other three transcoding nodes conduct another type of transcoding so they are on another transcoding plane ($tp_{2,3}$). With a total of n different quality versions in the system, there are at most $\frac{n}{2} \times (n-1)$ different transcoding planes if we ignore the impractical cases of transcoding from a lower quality version to a higher quality version. To represent the availability of the metadata corresponding to transcoding from version i to version j, a matrix of size $n \times n$ is introduced, denoted by *mMatrix*, which is an upper triangle matrix. In addition to providing the original streaming data, the source node of the streaming maintains *mMatrix*. Figure 3.2 shows an example of mMatrix.

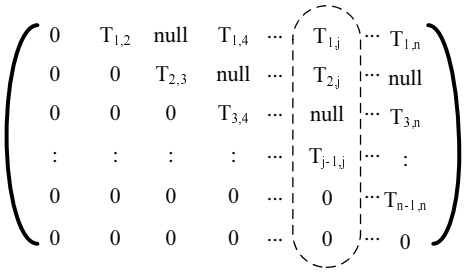

Figure 3.2: A mMatrix Example on the meta-data overlay

In mMatrix, when the corresponding metadata is not available, element $T_{i,j}$ is set to `null`. Otherwise, such metadata is available and $T_{i,j}$ denotes the ID of the node which transcodes the stream from version i to version j. This node is called the *meta-head*. The meta-head can share its metadata to other transcoding nodes which perform identical meta-transcoding.

3.2.2 Data Overlay and Meta-data Overlay Construction

The construction of data overlay in PAT differs from traditional overlays in P2P/overlay streaming systems, mainly because when a node wants to join the streaming system, its parent selection procedure is substantially different from the traditional ones and affects the construction of the metadata overlay. The node departure in PAT affects both the data overlay and the metadata overlay. The node arrival and departure protocols are presented

in this section separately.

Node Arrival

Assume the system starts with a source node (bootstrapping node) that provides the streaming service to the participating nodes. When a node wants to join the streaming service, it first contacts the source node or a bootstrap node with its quality requirement. In this chapter, the quality requirement is represented by the desired frame rate and bit rate of the streaming content. The quality requirement is mapped to a version index. The critical part of node joining is to find an appropriate parent for the joining node. The algorithm is as follows.

- **Case 1.1 – no transcoding required:** Assume the joining node requests object version j, the joining node looks for the node that can directly provide object version j via a candidate parent list containing the nodes that can provide the exact desired object quality. If there are several candidate parents, the joining node selects one of candidate parents with the smallest hop from the source peer compared with all other candidate parents. In this case, only the data overlay is updated.

- **Case 1.2 – transcoding required:** If the desired object version does not exist or if the node having the desired version cannot accept the joining node as its child node, the joining node must look for a parent capable of performing transcoding for it. Assuming that the desired version is j, based on the availability of the metadata for transcoding to version j, the parent selection follows the following steps.

  - **Step 1 – metadata available and meta transcoding**: The joining node checks the mMatrix at the source node or a bootstrapping node by examining the j^{th} column of mMatrix. The possible parent candidates are from $T_{1,j}$ to $T_{j-1,j}$ (the total transcoding candidates number are $j-1$). Based on the type of transcoding (frame rate transcoding or bit rate transcoding), the joining node

starts to send the joining request to the meta-head node indicated by $T_{t,j}$ (t is a variable and $0 < t < j$)

* where $(j - t)$ is the smallest, if the transcoding is for frame rate or spatial resolution reduction. This policy is designed to minimize the computing load since it requires less computing load to transcode from a frame rate or spatial resolution version that is closer to the target frame rate or spatial resolution version.
* where $(j - t)$ is the largest, if the transcoding is only for bit rate reduction. This policy is designed to minimize the quality degradation since the transcoding between versions that are further away (i.e., less number of intermediate versions) from each other in bit rate leads to less generation loss. On the other hand, the computing load for transcoding between different versions in bit rate is almost the same. For example, if we have two choices to get 300kbps video by bit-rate transcoding: 400kbps and 500kbps, the video quality transcoded from 500kbps is better than that of from 400kbps and these two transcodings consume the same computing resources.

Here two policies are used to reduce the usage of resources while maintaining similar quality by considering different characteristics of different transcoding procedures (frame rate or bit rate transcoding).

When the candidate parent P is selected, the candidate parent P needs to decide whether to accept the joining node based on its bandwidth availability. If yes, P needs to conduct meta-transcoding for the joining node and needs to join the corresponding metadata tree. Metadata subtree $T_{t,j}$ is updated as follows: the meta-head of $T_{t,j}$ looks for a parent for P. If the meta-head of $T_{t,j}$ itself has enough bandwidth, it serves as the parent of P; otherwise it checks whether any of its children has enough bandwidth to accept the new child P using a breadth-first search approach. If P can find a parent to join the metadata overlay, the joining process finishes. If none of these nodes can accept P, the joining node will

try another `non-null` element in mMatrix in ascending (or descending) order of $j-t$ value depending on the transcoding type so that it can find a new transcoding parent, and repeat above parent-children relationship establishing process. If all the `non-null` elements in the jth column of mMatrix have been explored without a transcoding parent node being identified, go to the next step.

- **Step 2 – metadata unavailable and full transcoding**: If a transcoding parent cannot be found in the previous step, the joining node must look for a parent with enough bandwidth and CPU resources to perform full transcoding for it. To minimize the overall system transcoding overhead, the joining node starts to probe nodes receiving version t, where $T_{t,j}$ is `null` in mMatrix. The joining node probes the node with the smallest or largest $j-t$ value depending on the type of transcoding as discussed in Step 1. If no suitable transcoding parent is found, the joining node probes the next node in ascending (or descending) order of $j-t$ value until a parent node is found. The chosen parent performs a full transcoding process. The chosen parent can also share the metadata produced during the transcoding session by joining the metadata overlay as follows.
  - After the parent (with its ID denoted as P) that is willing to do transcoding is selected and the joining node joins the data overlay, P needs to join the metadata overlay. Suppose node P needs to do transcoding from version k to version j. As P needs to do full-transcoding, metadata subtree $T_{k,j}$ is updated as follows: node P becomes the meta-head of the $T_{k,j}$ subtree and the corresponding element in mMatrix is updated.

  If a parent node can not be found, go to the next step.
- **Step 3** If version j is not the lowest quality version, go back to Step 1 to request a lower version $j+1$. Otherwise, the joining request is rejected.

During the joining process, the joining node may find several eligible parents and it can save the information about s parents (e.g., s=3) as its backup parents. The reason to do

this is to increase the system robustness: the node can quickly connect to its backup parent when its current parent leaves the system.

Node Departure

In P2P/overlay streaming, participating nodes may depart at any time. Upon node departure, the data and metadata overlay must be adjusted to adapt to the change. These adjustments are performed as follows.

- **Case 2.1:** If the departing node is a leaf node and is receiving version j from its parent node (the parent of the departing node is denoted as P), (1) if P is not a transcoding node, remove the departing node from P's children list and update its available bandwidth information; (2) if P is a transcoding node conducting transcoding from version i to version j, it is not necessary for P to stay in the corresponding metadata overlay and P needs to depart from the metadata subtree $T_{i,j}$ according to the following *META-LEAVE* algorithm.
  - If P is the meta-head in the metadata tree, it tries to find a new meta-head from its children and select the one with the largest available bandwidth. If a child C is selected, $T_{i,j}$ in mMatrix is updated accordingly and the sibling nodes of node C become the children of new meta-head C and parent-children relationship of the related nodes is updated. If the meta-head P leaves the system, the participating nodes' information is still useful to the corresponding meta-data overlay. To avoid the loss of information of all participating peers in one metadata overlay, the meta-head P will transfer the information to the new meta-head. But if P is the meta-head with no children in the metadata overlay, element $T_{i,j}$ in mMatrix is set to `null`.
  - If P is not a meta-head in the metadata tree, the meta-head deletes node P from the metadata subtree. If node P has children, they become the children of the parent of P in the metadata subtree if the parent of P has enough bandwidth.

If the parent of P does not have enough bandwidth, they contact other nodes in the metadata overlay to rejoin the metadata tree. The corresponding children and parent information of these nodes is updated.

- **Case 2.2:** If the departing node is neither a leaf node nor a transcoding node, the children of the departing node need to find new parents. If the parent of the departing node has enough bandwidth, it becomes their new parents. Otherwise, the children of the departing node need to perform a rejoin process as described in Section 4.3.1 and the rejoin process may change the nodes in the transcoding plane but not the nodes in the quality plane because these children do not change their quality requirements. If the parent of the departing node (denoted as P) is a transcoding node, P leaves the metadata subtree following the above *META-LEAVE* algorithm. Finally, the departing node is removed from the meta-data overlay.

- **Case 2.3:** If the departing node receiving version i is not a leaf node and is a transcoding node, the adjustment must be done on both the data overlay and the metadata overlay. We assume the departing node D is in the metadata tree $T_{i,j}$ and the children of node D need version j. All children of D in both the metadata overlay and the data overlay need to find new parents in the corresponding overlays. In the metadata overlay, node D departs the meta-data overlay according to the *META-LEAVE* algorithm in Case 2.1. After the metadata overlay is updated, the first child of D and its siblings in the data overlay find a new parent as follows. In the updated metadata subtree $T_{i,j}$, if there are still some nodes in the metadata overlay $T_{i,j}$ after the departure of node D, they can be selected to be new transcoding parents of these child nodes based on their available bandwidth. If the metadata overlay $T_{i,j}$ is `null` or no node has enough bandwidth, the child nodes of D need to rejoin the streaming system. If these child nodes rejoin the system, these nodes have a chance to be the child node of a node that can directly provide version j within the data tree.

3.3 Performance Modeling

System throughput is an important metric in the streaming systems and it is decided by the available CPU, storage, and bandwidth resources. In this section, the system throughput of different schemes are compared. In PAT, the system throughput is represented by the total number of the peers that the quality plane in the data overlay and the transcoding plane in the metadata overlay can serve.

The transcoding nodes in the transcoding plane have three schemes to select: full caching, meta caching and no-caching transcoding. To conduct a specific transcoding task, different schemes consume different CPU cycles and bandwidth resource. Thus, given available bandwidth resource and CPU cycles resource in the system, the throughput in the transcoding plane is decided by the transcoding scheme selected by the transcoding peer in the corresponding transcoding plane.

For the quality plane, the throughput is only decided by the available bandwidth. For the transcoding plane, the throughput is decided by both available CPU and bandwidth resource jointly. Please note that in the full caching method, a transcoded object version gets cached in full. Meta-caching (meta-transcoding) caches the intermediate transcoding results, which can be used to construct the transcoded object version in full with less CPU resource compared with a full transcoding process. No-caching transcoding does not cache any transcoding result.

The goal is to quantitatively calculate the throughput for full caching, meta caching, and no caching for each quality version. Based on their performance with given CPU and bandwidth resources, a suitable scheme is selected to achieve the largest throughput.

3.3.1 Model Construction

In the analysis, we have the following assumptions about client access distribution and media objects in the streaming system.

1. The bandwidth resource used to transfer the original object is one bandwidth unit. The bandwidth resource needed to transfer different versions is different. From the

highest quality (version 1) to the lowest quality (version v), each version consumes bandwidth of 1, $\frac{v-1}{v}$, $\frac{v-2}{v}$, ... , $\frac{1}{v}$ units, correspondingly.

2. The CPU resource used for full transcoding is one unit. The bandwidth resource used to deliver metadata is α $(0 < \alpha < 1)$ unit of the corresponding object size. For meta-caching, it is assumed that the CPU resource consumed to produce the final version is β $(0 < \beta < 1)$ compared with 1 unit of full transcoding.

In this section, the analysis is based on one quality version j. The notations used in the following analysis are summarized in Table 3.1.

Table 3.1: Notations used in the analysis

B	the total bandwidth resource for quality version j
C	the total computing cycles for quality version j
v	the number of quality versions of the original object
N_j	the total number of peers requesting quality version j
p_j	the percentage that the sessions are not served by transcoding in the quality plane qp_j

Bandwidth Constraint

It is assumed that there are v object versions in the system. Among the total N accesses, the number of accesses to each version are $N_1, N_2, ..., N_v$, respectively and $N = \sum_{j=1}^{v} N_j$. In this chapter, it is assumed that the accesses to different versions are uniformly distributed. We can get $N_1 = N_2, ... = N_v = N/v$.

The procedure which processes the requests for version j is used as an example to analyze the system throughput. For quality version j, there is one quality plane (qp_j) and $v - j$ transcoding planes ($tp_{j,j+1}, ..., tp_{j,v}$).

Among all the peers in qp_j (note that both non-transcoding peers and transcoding peers which require quality j locate in quality plane qp_j, some of them can provide the streaming version j without transcoding and some of them can provide the streaming version better than j with transcoding). Assuming the percentage that the sessions are not served by

transcoding in the quality plane (qp_j) is p_j ($0 < p_j < 1$), the number of the sessions that are not served by transcoding is $N_j \times p_j$ and the number of the sessions that are served by transcoding is $N_j \times (1 - p_j)$.

1. For full-caching, the total bandwidth consumption is composed of two parts: non-transcoding bandwidth consumption and transcoding bandwidth consumption. These non-transcoding peers consume $N_j \times p_j \times \frac{v-j+1}{v}$ considering that the size of quality version j is $\frac{v-j+1}{v}$. These transcoding peers use $N_j \times (1-p_j) \times v_a$ (here v_a represents the average size of the versions which are transcoded from version j and v_a is equal to $(\frac{v-j}{v} + \frac{v-(j+1)}{v} + ... + \frac{1}{v})/(v-j)$. That is $v_a = \frac{v-j+1}{2v}$ if different versions are accessed uniformly). Because of the bandwidth constraint of the transcoding head (note that the transcoding head is the node which conducts the full transcoding), the transcoded version is transferred to one child node of the transcoding head and then forwarded to the node which originally requested the transcoded version, thus the total bandwidth consumed by the joining node is $2v_a$.

   Thus the average bandwidth requirement for each session for full-caching b_f is:

$$\begin{aligned} b_f &= \frac{N_j \times p_j \times \frac{v-j+1}{v} + N_j \times (1-p_j) \times 2v_a}{N_j} \\ &= p_j \times \frac{v-j+1}{v} + (1-p_j) \times 2 \times \frac{v-j+1}{2v} = \frac{v-j+1}{v}. \end{aligned} \tag{3.1}$$

2. For meta-caching (meta-transcoding), in addition to the bandwidth used to deliver the transcoded version to the requesting node, meta-data needs to be delivered to the transcoding parent in the transcoding plane, the total bandwidth requirement is $(1+\alpha) \times v_a$ (α is the bandwidth resource required to deliver metadata and $0 < \alpha < 1$).

Thus the average bandwidth requirement b_m is:

$$\begin{aligned} b_m &= \frac{N_j \times p_j \times \frac{v-j+1}{v} + N_j \times (1-p_j) \times (1+\alpha) \times \frac{v-j+1}{2v}}{N_j} \\ &= (p_j + (1-p_j) \times \frac{(1+\alpha)}{2}) \times \frac{v-j+1}{v}. \end{aligned} \tag{3.2}$$

3. For no-caching , every transcoded version is delivered to the requesting client directly after the transcoded version is produced, thus the total bandwidth requirement is v_a. The average bandwidth requirement b_n is:

$$\begin{aligned} b_n &= \frac{N_j \times p_j \times \frac{v-j+1}{v} + N_j \times (1-p_j) \times \frac{v-j+1}{2v}}{N_j} \\ &= (p_j + \frac{(1-p_j)}{2}) \times \frac{v-j+1}{v} = (\frac{1+p_j}{2}) \times \frac{v-j+1}{v}. \end{aligned} \tag{3.3}$$

Given bandwidth capacity B, the total numbers of sessions that can be supported by full-caching, meta-caching, and no-caching schemes are denoted as n_{fb}, n_{mb}, and n_{nb}, respectively. That is

$$n_{fb} = B/b_f = \frac{B \times v}{v-j+1}, \tag{3.4}$$

$$n_{mb} = B/b_m = \frac{B}{(p_j + (1-p_j) \times \frac{(1+\alpha)}{2}) \times \frac{v-j+1}{v}}, \tag{3.5}$$

and

$$n_{nb} = B/b_n = \frac{B}{(\frac{1+p_j}{2}) \times \frac{v-j+1}{v}}. \tag{3.6}$$

CPU Constraint

Having considered the bandwidth constraint, the CPU constraint of full-caching, meta-transcoding and no-caching schemes will be discussed. In full-caching, every session needs one unit of CPU resource for the first access and zero unit for the following accesses requesting the same version. In meta-caching, every session needs one unit of CPU for the first access and stores the metadata and β unit for the following accesses requesting the same version (please note β is the CPU resource used to produce a fully-transcoded version from cached metadata and $0 < \beta < 1$). In no-caching scheme, every session requesting a transcoded version needs one unit of CPU resource.

For full-caching, only $v - j$ versions $(j + 1, ..., v)$ can be provided by version j and total CPU resource required is $v - j$ units. The average computing load c_f is

$$c_f = \frac{v - j}{N_j}. \tag{3.7}$$

For meta-transcoding (meta-caching), only the first $v - j$ sessions need 1 unit CPU resource and other transcoding sessions need β unit CPU resource. The average computing load c_m is

$$c_m = \frac{v - j + (N_j \times (1 - p_j) - (v - j)) \times \beta}{N_j}. \tag{3.8}$$

For no-caching, every transcoding session needs 1 unit CPU resource and the average computing load c_n is

$$c_n = \frac{N_j \times (1 - p_j)}{N_j} = 1 - p_j. \tag{3.9}$$

Given the available CPU capacity C, the numbers of sessions that can be supported by full-caching, meta-transcoding and no-caching schemes are denoted as n_{fc}, n_{mc}, and n_{nc},

respectively. That is

$$n_{fc} = C/c_f = \frac{C \times N_j}{v - j}, \tag{3.10}$$

$$n_{mc} = C/c_m = \frac{C \times N_j}{v - j + (N_j \times (1 - p_j) - (v - j)) \times \beta}, \tag{3.11}$$

and

$$n_{nc} = C/c_n = \frac{C}{1 - p_j}. \tag{3.12}$$

Bandwidth and CPU Constraint

If the system throughput needs to be calculated accurately, the bandwidth and CPU resource constraint needs to be considered jointly. If tn_f, tn_m, and tn_n are used to denote the total session number supported by full-caching, meta-caching, and no-caching, respectively, it is true that

$$tn_f = min(n_{fb}, n_{fc}) = min(\frac{B \times v}{v - j + 1}, \frac{C \times N_j}{v - j}), \tag{3.13}$$

$$\begin{aligned} tn_m = min(n_{mb}, n_{mc}) = min(&\frac{B}{(p_j + (1 - p_j) \times \frac{(1+\alpha)}{2}) \times \frac{v-j+1}{v}}, \\ &\frac{C \times N_j}{v - j + (N_j \times (1 - p_j) - (v - j)) \times \beta}), \end{aligned} \tag{3.14}$$

and

$$tn_n = min(n_{nb}, n_{nc}) = min(\frac{B}{(\frac{1+p_j}{2}) \times \frac{v-j+1}{v}}, \frac{C}{1 - p_j}). \tag{3.15}$$

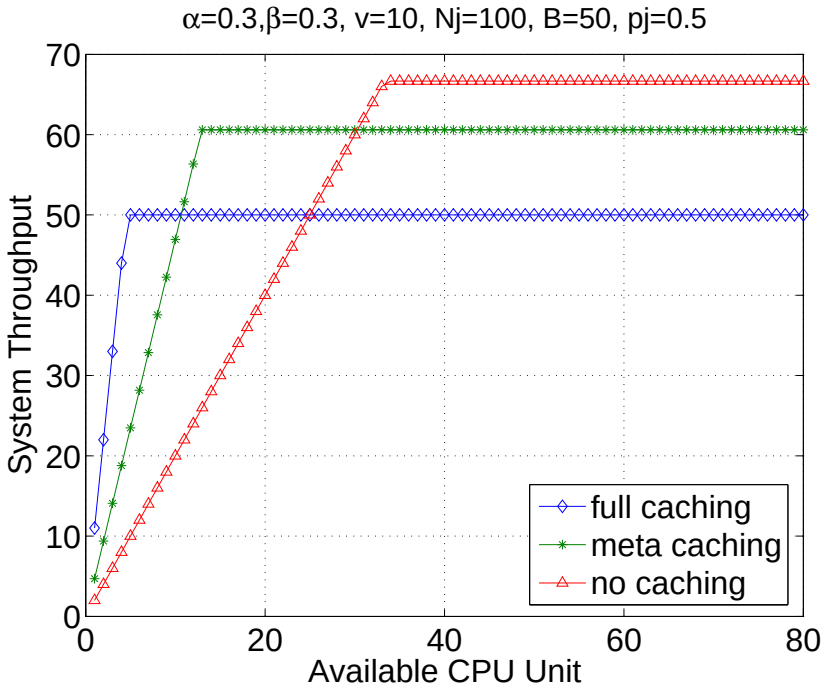

Figure 3.3: Comparisons between three schemes: available CPU ranges up to 80 units

Based on above formulas, Figure 3.3 and Figure 3.4 show the comparison results with some typical values for different parameters when the available CPU and bandwidth resources vary. In these figures, α and β are set to 0.3 and p_j is set to 50%. It is assumed that there is a total of 1000 accesses and the video object has 10 versions in the system. Thus there is a total of 100 accesses for version j (the access number to each version is uniform). Here as an example, take j=1 and N_j=100 (note that version 0 is the highest quality version and version v-1 is the lowest quality version). The total available CPU is varying in Figure 3.3 and the total available bandwidth resource is varying in Figure 3.4.

In Figure 3.3, the available CPU resource ranges up to 80 units. In Figure 3.4, the available bandwidth resource ranges up to 800 units. These two figures indicate that under certain available resource (CPU and bandwidth) conditions, one of the three schemes may outperform the other two. Having analyzed the quantitative throughput for different schemes, their performance are further analyzed in detail to reveal such conditions.

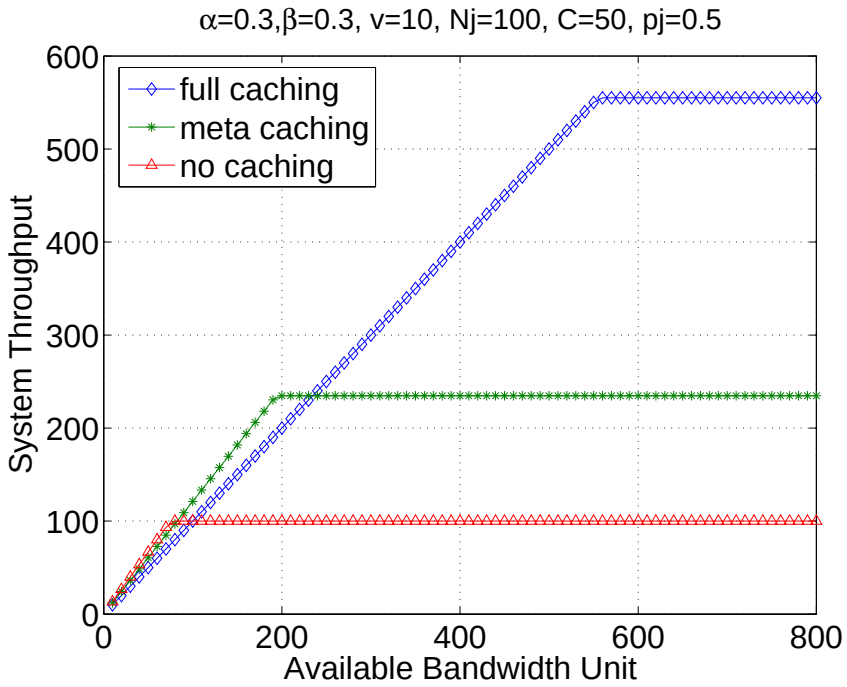

Figure 3.4: Comparisons between three schemes: available bandwidth ranges up to 800 units

3.3.2 Modeling Analysis

Based on the model presented above, the system throughput of these three schemes are compared as follows.

By equation 3.4, 3.5, 3.6, and $0 < \alpha < 1$, it is easy to derive that

$$n_{fb} < n_{mb} < n_{nb} \tag{3.16}$$

By equation 3.10, 3.11, 3.12, and $0 < \beta < 1$, it is easy to derive that

$$n_{fc} > n_{mc} > n_{nc} \tag{3.17}$$

When $n_{nb} \leq n_{nc}$ ($B \leq \frac{C}{1-p_j} \times (\frac{1+p_j}{2}) \times k_1$), no-caching is the scheme that has the largest throughput. Here, k_1 is used to represent $\frac{v-j+1}{v}$. First, when $n_{nb} > n_{nc}$ ($B >$

$\frac{C}{1-p_j} \times (\frac{1+p_j}{2}) \times k_1)$, these three schemes are compared.

1. When $n_{fb} \geq n_{fc}$, full transcoding has the largest throughput and the largest throughput is n_{fc}. If $n_{fb} \geq n_{fc}$, that is

$$\frac{B}{k_1} \geq \frac{C \times N_j}{v - j}. \tag{3.18}$$

Thus, if $\frac{B}{k_1} \geq \frac{C \times N_j}{v-j}$, full-caching has the largest throughput.

2. When $n_{fb} < n_{fc}$, the full-caching result is n_{fb} and that result is compared with the throughput of meta-transcoding.

When $n_{fb} < n_{fc}$, that is

$$B < \frac{C \times N_j \times k_1}{v - j}. \tag{3.19}$$

- $n_{mb} \geq n_{mc}$: If $n_{mb} \geq n_{mc}$, the throughput of meta-transcoding is n_{mc} and n_{mc} needs to be compared with full-transcoding result: n_{fb}.

  - If $n_{fb} \geq n_{mc}$, the full-caching scheme has the best performance. When $n_{fb} \geq n_{mc}$, it is true

$$B \geq \frac{C \times N_j \times k_1}{v - j + (N_j \times (1 - p_j) - (v - j)) \times \beta}. \tag{3.20}$$

  - Combining with 3.19, it is true

$$\frac{C \times N_j \times k_1}{v - j + (N_j \times (1 - p_j) - (v - j)) \times \beta} \leq B < \frac{C \times N_j \times k_1}{v - j}. \tag{3.21}$$

If $n_{fb} < n_{mc}$, meta-transcoding scheme has the best performance. When $n_{fb} <$

n_{mc}, that is

$$\frac{C \times N_j \times (p_j + (1-p_j) \times \frac{(1+\alpha)}{2}) \times k_1}{v - j + (N_j \times (1-p_j) - (v-j) \times \beta} \leq B < \frac{C \times N_j \times k_1}{v - j + (N_j \times (1-p_j) - (v-j)) \times \beta}. \tag{3.22}$$

- $n_{mb} < n_{mc}$: If $n_{mb} < n_{mc}$, the throughput of meta-transcoding is n_{mb} and n_{mb} needs to be compared with n_{fb}. When $n_{fb} < n_{mb}$, that is

$$\frac{B}{k_1} < \frac{B}{(p_j + (1+\alpha) \times \frac{1-p_j}{2}) \times k_1}. \tag{3.23}$$

So when $0 < \alpha < 1$, we can get $n_{fb} > n_{mb}$, full-caching has the best performance.

Table 3.3.2 summarizes the analysis result briefly.

Table 3.2: Summary of analysis result

Condition	**Best scheme**
$n_{nb} \leq n_{nc} (B \leq \frac{C}{1-p_j} \times (\frac{1+p_j}{2}) \times k_1)$	no-caching
$n_{nb} > n_{nc}, n_{fb} > n_{fc} (B > \frac{C \times N_j \times k_1}{v-j})$	full-caching
$n_{nb} > n_{nc}, n_{fb} \leq n_{fc} (B \leq \frac{C \times N_j \times k_1}{v-j}), n_{mb} \geq n_{mc}$ and $n_{fb} \geq n_{mc}$	full-caching
$n_{ns} > n_{nc}, n_{fb} \leq n_{fc} (B \leq \frac{C \times N_j \times k_1}{v-j}), n_{mb} \geq n_{mc}$ and $n_{fb} < n_{mc}$	meta-transcoding
$n_{ns} > n_{nc}, n_{fb} \leq n_{fc} (B \leq \frac{C \times N_j \times k_1}{v-j}), n_{mb} < n_{mc}$	full-caching

3.4 Performance Evaluation

To evaluate PAT, first, real transcoding and meta-transcoding are conducted in the implemented transcoder for both bit rate reduction and frame rate reduction. With parameters obtained from these experiments, the performance of PAT is evaluated based on the simulation on ns2 [63].

3.4.1 Simulation Setup

In the implemented transcoder, bit rate transcoding and frame rate transcoding are conducted. A 1000-second video with a data rate of 500 kb/s and 30 frame/second (denoted as f/s) is used as the original video object for each transcoding experiment. The corresponding Peak Signal and Noise Ratio (PSNR) of the original video is 33.85 dB. For bit rate reduction scenario, there are 5 different versions with the corresponding bit rates decreasing from 500 kb/s to 100 kb/s with a 100 kb/s decreasing interval. Normalizing the CPU load for a full transcoding session as 1 unit, meta-transcoding only consumes 0.4 unit for all bit rate reduction cases. The corresponding metadata size is 10% of the original video file size, regardless of bit rate differences.

For frame rate transcoding, 5 different qualities are used as well. They are 500 kb/s and 400 kb/s at 30 f/s, 300 kb/s and 200 kb/s at 15 f/s, and 100 kb/s at 5 f/s. Normalizing the CPU load for a full transcoding session from 30 f/s to 15 f/s as 1 unit, the CPU load for full transcoding from 30 f/s to 5 f/s and from 15 f/s to 5 f/s are 0.44 unit and 0.36 unit, respectively. For meta-transcoding, the corresponding CPU loads are 0.5, 0.27, and 0.19 unit, and the corresponding metadata size is 20%-25% of the original video file size for transcoding between different frame rates, depending on the source bit rate and target frame rate.

With these parameters, PAT is evaluated over a topology of 1000 peers. Specifically, one source node has all the different versions, and its bandwidth capacity is 5 Mb/s. The distribution of bandwidth capacity for the rest of the peers is as follows: 5Mb/s (11%), 2Mb/s (3%), 896kb/s (9%), 384kb/s (21%), and 256kb/s (56%). Assuming the version index 1 representing the original version with increased index indicating lower quality versions, the nodes with 5Mb/s and 2Mb/s connections are randomly selected to request version 1 to 3, and nodes with other bandwidth capacity are randomly selected to request version 1 to 5. The CPU constraints of peers are randomly selected to set from 1 unit to 2 units. The system starts with one source node capable of serving all versions, and 1000 nodes

dynamically join and depart from the streaming service. Their arrival pattern follows a Poisson distribution with the mean arrival rate as 1 request per three seconds, and the maximum arrival interval of 10 seconds. Nodes depart randomly after receiving the service for a duration ranging from 250 seconds to 1000 seconds.

3.4.2 Experimental Results

The performance of the proposed scheme (*PAT-Meta*) is evaluated by comparing it with two other schemes: *No-Transcoding* and *PAT-Normal*. *No-Transcoding* represents the system in which no node can perform transcoding but the source node can serve precoded versions, and a joining request is accepted only if the requested or a lower quality version is available. *PAT-Normal* represents the PAT system in which all transcoders perform full transcoding without assistance from metadata.

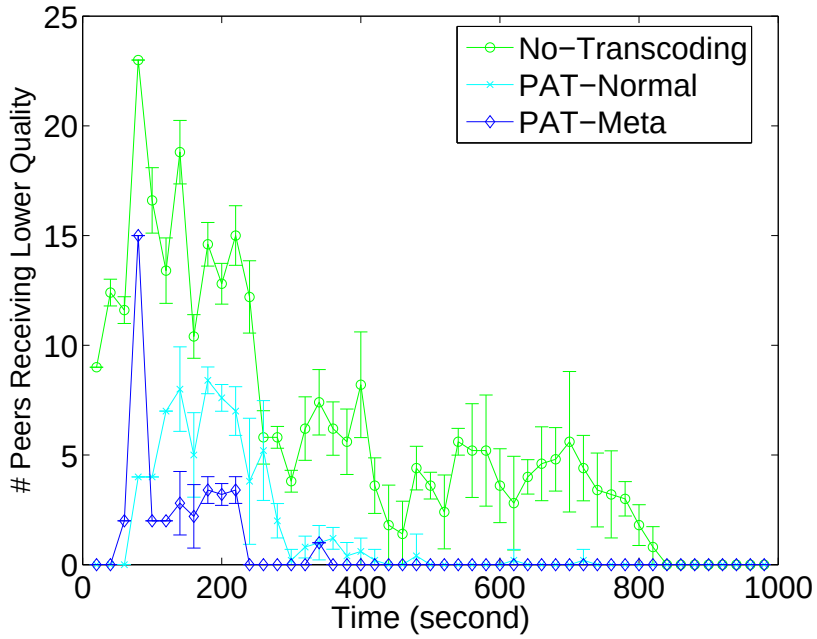

Figure 3.5: Peers receiving lower quality in bit rate transcoding

First, the streaming quality received by the peers is evaluated. Figure 3.5 and Figure 3.6 show the number of nodes that are rejected or accepted but receiving lower than desired quality versions as a function of time for bit rate transcoding and frame rate transcoding,

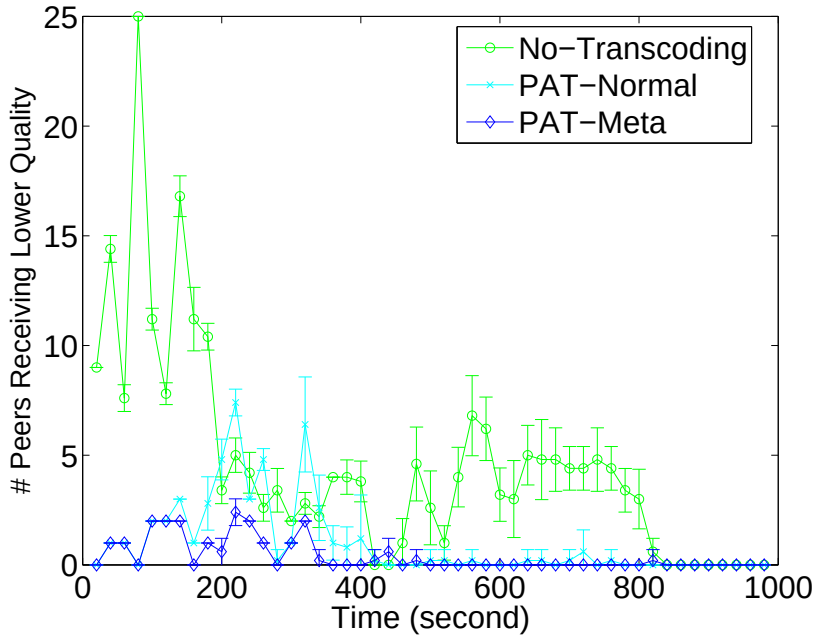

Figure 3.6: Peers receiving lower quality in frame rate transcoding

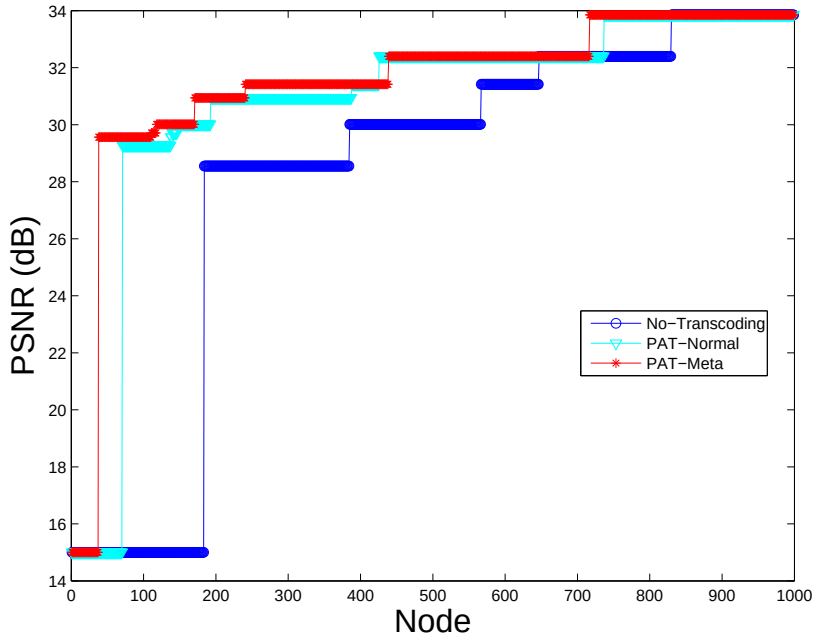

Figure 3.7: Streaming quality in bit rate transcoding

respectively. Both transcoding schemes in PAT significantly outperform No-Transcoding and PAT-Meta performs better than PAT-Normal. Overall, about 30% nodes could not

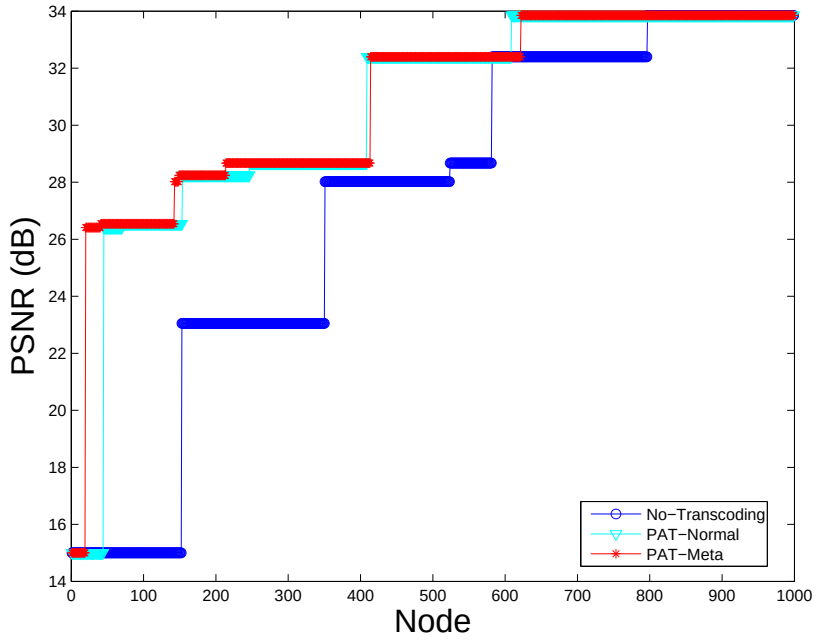

Figure 3.8: Streaming quality in frame rate transcoding

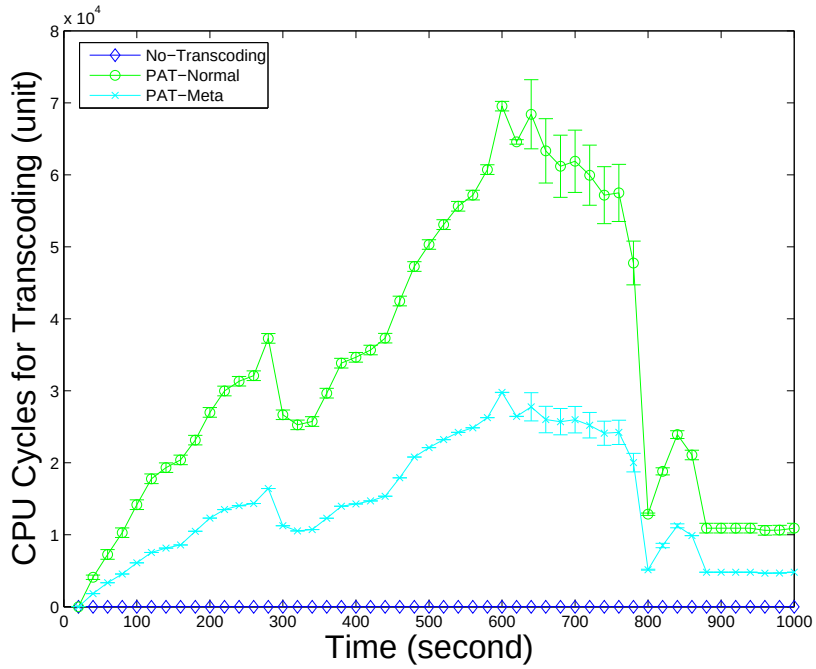

Figure 3.9: CPU cycles consumed in bit rate transcoding

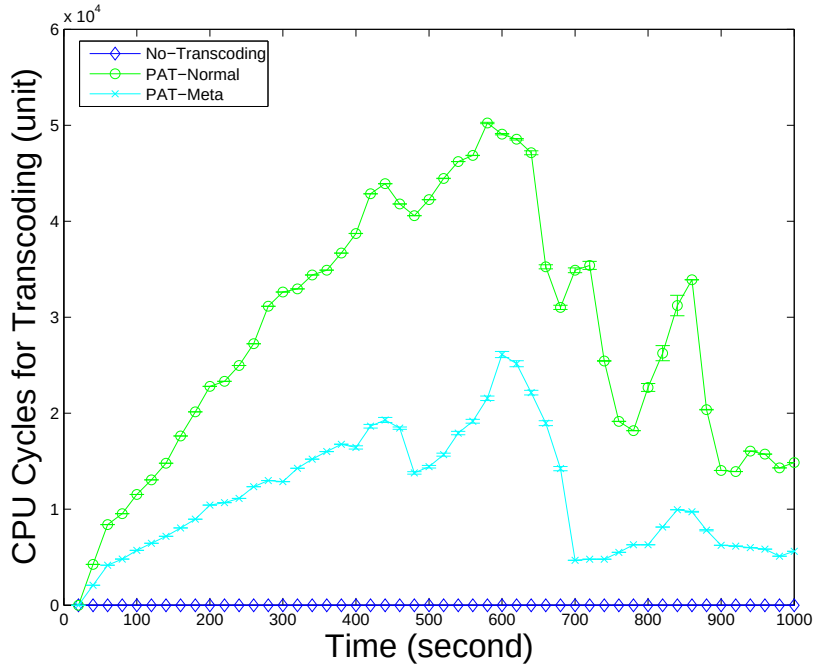

Figure 3.10: CPU cycles consumed in frame rate transcoding

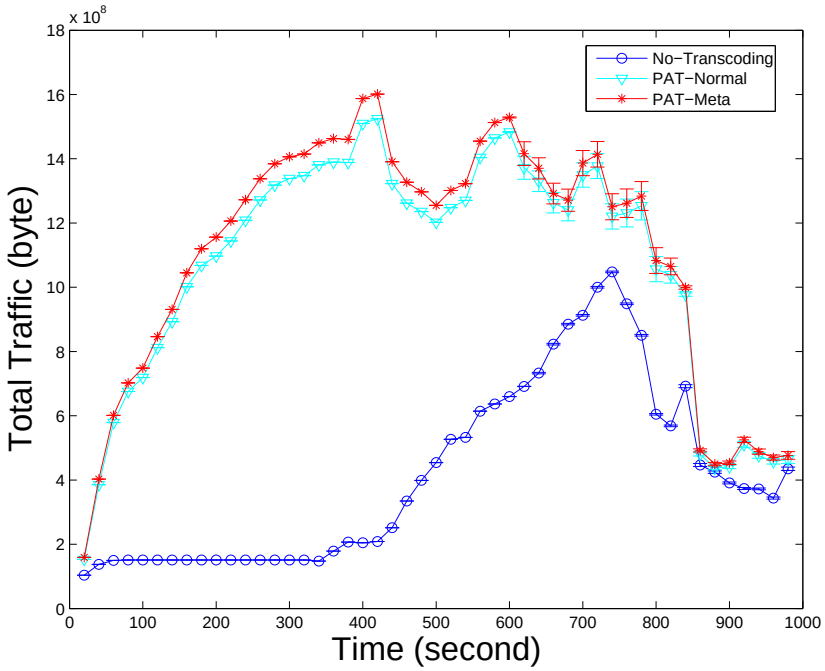

Figure 3.11: Total traffic in bit rate transcoding

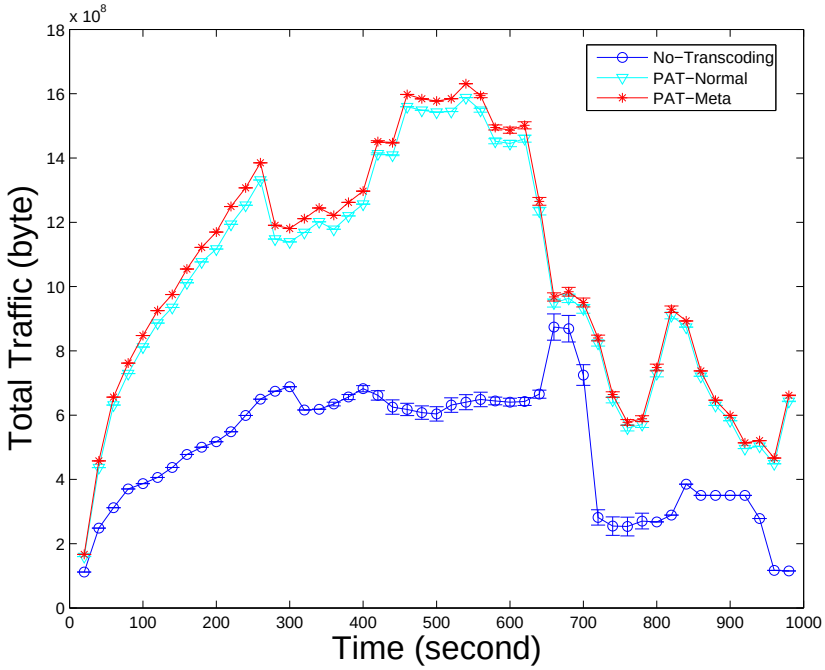

Figure 3.12: Total traffic in frame rate transcoding

receive their desired video qualities in No-Transcoding, while it is only about 4% in PAT-Meta.

Figure 3.7 and Figure 3.8 further show the average PSNR of the received video at each node sorted in increasing PSNR values. To simplify evaluations, it is assumed the PSNR of peer is 15 dB if the node is rejected during the joining process. In Figure 3.7, PAT-Normal and PAT-Meta achieve better overall quality than No-Transcoding. Considering the number of rejected nodes (their PSNR are set to 15 dB), No-Transcoding rejects more nodes than PAT-Normal and PAT-Meta.

Table 3.3 summarizes the average streaming quality that clients perceive in the experiment. Overall, the average video quality experienced by clients in PAT-Meta is the best. Although PAT-Normal achieves similar quality gain over No-Transcoding, it is at the cost of much more computing overhead as shown next.

Second, the CPU and the bandwidth consumptions of three schemes are compared. Figure 3.9 and Figure 3.10 show the normalized CPU load required for bit rate transcoding

Table 3.3: Average PSNR: client perceived streaming quality (dB)

	No-transcoding	**PAT-Normal**	**PAT-Meta**
Bit	28.14	30.86	31.49
Frame	27.19	30.54	30.82

and frame rate transcoding, respectively. As expected, there is no CPU consumption for transcoding in No-Transcoding. With the assistance of metadata, PAT-Meta significantly outperforms PAT-Normal. On average, PAT-Meta can save 58% CPU load in comparison with PAT-Normal for bit rate transcoding. PAT-Meta significantly reduces the CPU load required for transcoding by sharing metadata in the metadata overlay, which leads to some traffic overhead. To evaluate the traffic overhead, Figure 3.11 shows the total traffic for bit rate transcoding, and Figure 3.12 shows the corresponding result for frame rate transcoding.

The total traffic difference in PAT-Normal and PAT-Meta indicates the additional bandwidth consumed by metadata sharing, and the traffic overhead only accounts for at most 6% of the total traffic. Jointly considering the quality improvements and the computing and traffic overhead, PAT-Meta achieves the best quality improvement with the least computing overhead and negligible traffic overhead.

3.5 Summary

To provide appropriate qualities to heterogeneous devices, in this chapter, a system is proposed to leverage idle peer CPU cycles for efficient online transcoding based on an additional transcoding overlay formed by transcoding nodes. To improve the system performance, PAT effectively manages the CPU and bandwidth resources and leverages the meta-transcoding with the support of a metadata overlay that can make all transcoding nodes share their metadata. Driven by parameters obtained from practical transcoding and meta-transcoding schemes, the simulation results demonstrate the effectiveness of the proposed scheme over other schemes.

Chapter 4: Dynamic Bi-Overlay Rotation for Streaming

4.1 Introduction

In the last chapter, we discussed and showed that we could leverage the idle peer resources in P2P/Overlay streaming systems for meta-caching so that online transcoding could be used. However, this may aggravate the unbalanced resource contributions in a P2P/Overlay streaming system. Typically, in a P2P/Overlay streaming system, some peers contribute more resources than others due to their positions in the overlay streaming systems. If these peers use battery powered mobile devices, their battery resources may be consumed faster than other peers who contribute less resources. The unbalanced resource contributions of different peers using heterogeneous devices may pre-maturely exhaust their limited battery capacities, and the entire streaming system may collapse due to the forced departure of these resource-depleted peers. Our proposed PAT scheme could worsen this situation as some peers are required to contribute additional bandwidth and CPU resources. Therefore, we study how to improve the fairness among peers and the robustness of the system in this chapter.

Some research efforts [5, 6, 10, 29, 51, 58, 59, 62, 64, 65] aim to address the robustness and fairness problems for interior peers. For example, Splitstream [59] distributes the forwarding bandwidth load among all the peers, and can accommodate heterogeneous devices. Dagster [62] encourages clients to donate more bandwidth resources to the system by preempting clients that contribute less bandwidth compared with the joining node and introduces transcoding in the system. However, it is not clear how a node could efficiently conduct transcoding at runtime although transcoding is mentioned. These research efforts mainly focus on improving the utilization of bandwidth. In the previous chapters, the tradeoffs between the CPU and the bandwidth resources during online transcoding are systematically

evaluated. Moreover, in order to facilitate the CPU-intensive transcoding at runtime, an additional overlay called metadata overlay is constructed to instantly share intermediate results of the transcoding process to reduce CPU consumption significantly. Nevertheless, the unbalanced use of various types of resources can adversely affect the performance of participating peers and subsequently threaten the robustness of the whole system.

Aiming to address the MDH and fairness problems more efficiently, we propose Dynamic Bi-Overlay Rotation (DOOR) to improve existing P2P/overlay live streaming systems so that heterogeneous devices can receive their desired streaming qualities in a fair fashion by leveraging the online content adaptation without any additional infrastructure support. By considering the resource contribution and consumption of heterogeneous devices, a dynamic rotation scheme is conducted on both data overlay and meta-data overlay to improve the fairness and robustness of the system. Based on the computing load and transcoding quality parameters obtained through real transcoding scenarios, large scale simulations are conducted to evaluate the proposed scheme. The simulation results show clear improvement of the proposed scheme over earlier ones.

The remainder of this chapter is organized as follows. The bi-overlay construction protocol and its associated data structures that are critical to rotation scheme are introduced in Section 4.2. How to construct the overlay is presented in Section 4.3. The design of Dynamic Bi-Overlay Rotation scheme is presented in Section 4.4. The rotation protocol is presented in Section 4.5. Experimental results are presented in Section 4.6. We will conclude this chapter in Section 4.7.

4.2 Bi-overlay Introduction

In this section, the bi-overlay construction, including a data overlay and a metadata overlay is presented. On each overlay, different interchangeable sets are constructed to facilitate the rotation in the streaming service in order to balance resource consumption of heterogeneous devices. An interchangeable set is a peer cluster in which the positions of the peers are interchangeable. In DOOR, there are two interchangeable sets: Data Interchangeable Set

(DIS) and Metadata Interchangeable Set (MIS). The DIS and MIS are formed by the nodes in data overlay and metadata overlay, respectively.

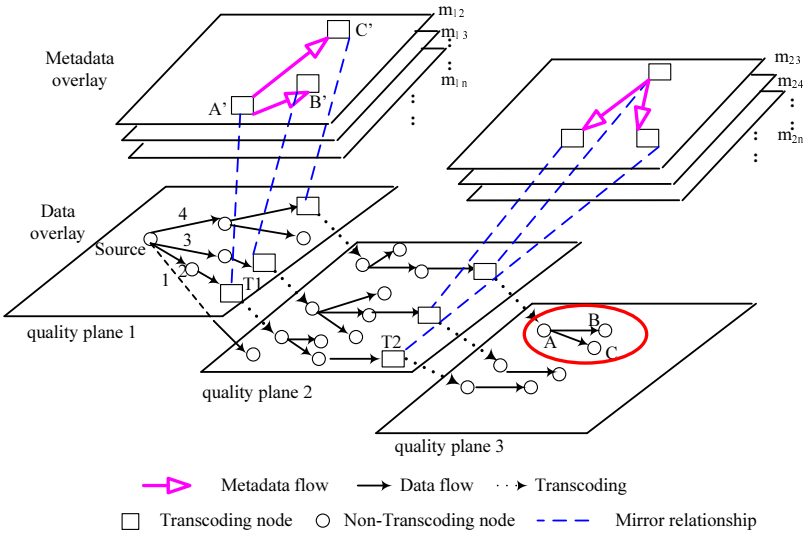

Figure 4.1: A bi-overlay example

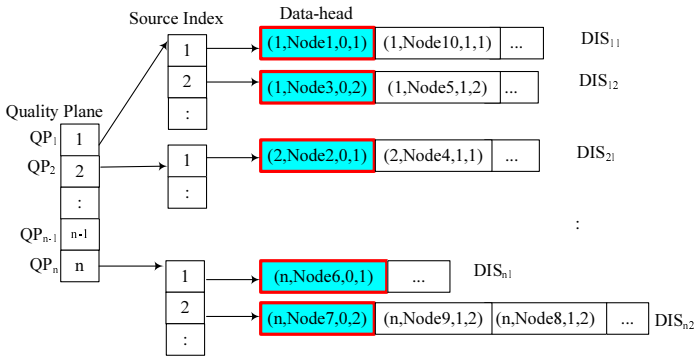

Figure 4.2: A QP-Array example

4.2.1 DIS and QP-Array

On the data overlay, the nodes that demand the same streaming quality form a quality plane. That is, *quality plane i* represents the set of nodes which request the same quality i. Different quality planes are connected through transcoding nodes (If the source node is involved, we can take the source node as a special transcoding node). On a quality

plane, there are different sub-trees. For example, in Figure 4.1, on quality plane 1, there are three sub-trees on different branches originating from the source node. Each sub-tree may contain both transcoding and non-transcoding nodes. To differentiate the peers in the overlays, source index is used to indicate the sub-tree branch originated from the source node. For example, in the quality plane 2 of Figure 4.1, there are 4 sub-trees originated from the source node. Non-transcoding nodes from the same sub-tree on a quality plane form a DIS. For example, nodes A, B, and C on quality plane 3 in Figure 4.1 belong to the same DIS as shown in the circle. In DOOR, the quality plane is managed through an array called *QP-Array*, which is an n-element array if the entire number of different quality versions is n. Figure 4.2 shows an example of *QP-Array*. In Figure 4.2, the fields of an element in DIS-Array are planeID, NodeID, FunctionProperty (0:regular peer, 1:transcoding-capable peer), and source index. Apparently, each element in the array is associated with one quality plane that is represented by an array called DIS-Array. This array may contain several DISs and these DISs have different source indices. The first-element in each DIS is referred to as *data-head*. In Figure 4.2, the data-head is represented by shaded rectangles. The child nodes of a data-head are stored in the depth-first order. For a child node, the source index of that node is the same as that of its parent or is assigned by the source node if its parent is the source node.

4.2.2 MIS and TMatrix

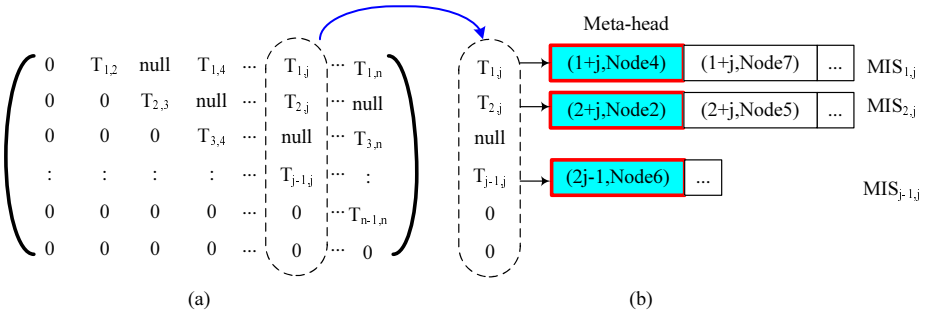

Figure 4.3: A TMatrix example on metadata overlay

On the metadata overlay, similarly, based on the transcoding input and output versions, different transcoding planes are formed. For example, in Figure 4.1, transcoding nodes A', B', and C' perform identical transcoding, thus are on the same transcoding plane. With a total of n different quality versions, there are at most $\frac{n}{2} \times (n-1)$ different transcoding planes without counting the impractical cases (transcoding from a lower quality stream to a higher quality stream). An upper triangle matrix of size $n \times n$ is used and the matrix is denoted by *TMatrix*, to represent the transcoding planes. Figure 4.3(a) shows an example of TMatrix. All transcoding nodes belonging to the same transcoding plane form an MIS if their transcoding processes have the same input and output versions. Figure 4.3(b) shows an example of column j's MIS lists in TMatrix. In Figure 4.3(b), the fields of an element are planeID, and NodeID. Similar to DIS-Array, the first `non-null` element is referred to as *meta-head*. In a MIS, the meta-head performs full transcoding, while other nodes perform the meta-transcoding with the help of metadata provided by the meta-head. As it does not take much space to store the node ID and the connection information of the meta-head, the total space occupied by the mMatrix is not large. In our design, the storage space occupied by the node ID and the connection information are 20 bytes (10 bytes for node ID and 10 bytes for connection information). If the total number of peers is 1000, the total non-null elements are $\frac{1}{2} * 1000 * (1000-1)$ and the occupied storage is $(1000 * (1000-1)/2) * 20 \approx 10M$ bytes. The storage overhead is acceptable to a source peer.

4.3 Bi-Overlay Construction Protocol

Along the node arrival and departure, the bi-overlays are constructed and adjusted dynamically. We present the corresponding node arrival and departure process in section 4.3.1 and section 4.3.2, respectively.

4.3.1 Node Arrival

When a node arrives, it first contacts the source node or a bootstrap node with its quality requirements. In DOOR, the quality requirements of node i are denoted as a tuple of $< fr_i, br_i, cd_i >$, where fr_i, br_i, and cd_i represent the frame rate, bit rate, and color depth requirements, respectively. In the following context, the quality tuple is mapped to a quality index and is referred to as a quality version. It is assumed that if there are n quality versions, the best quality version (original version) is 1, and the lowest quality version is n.

- **Case 1 – no transcoding required and quality plane:** First, the joining node contacts the source node or a bootstrap node to obtain a candidate parent list containing the nodes that can provide the exact desired object quality. The joining node selects one of them with the smallest height (from the source node) in the tree. In this case, only the data overlay is updated. At the same time, this node needs to join a particular quality plane. Suppose the joining node requests streaming quality i, and its candidate parent's source index is j

  - The node contacts the data-head of DIS_{ij}. If the data-head can accept the joining node, it adds the joining node to its children list. Otherwise, the node contacts another candidate parent with a different source index.

  - If all the candidate parents have been tried without any success, and the source node has enough bandwidth, the source node assigns a source index j', and the joining node becomes the data-head of $DIS_{ij'}$. The NodeID and source index are recorded in $DIS_{ij'}$.

- **Case 2 – transcoding required and transcoding plane:** If the desired object version does not exist, the joining node must find a parent node that is able to do transcoding for it. In this case, both the metadata overlay and the data overlay need to be updated. At the same time, the parent node needs to join a particular transcoding plane. Based on whether there is available metadata for transcoding,

– **metadata available and meta-transcoding**: The joining node checks TMatrix from the source node by examining the j^{th} column (note that quality version 1 is the highest quality version). The possible parent candidates are from $T_{1,j}$ to $T_{j-1,j}$ (a total of $j-1$ candidates). The joining node starts to probe nodes that are receiving version t ($0 < t < j$), if $T_{t,j}$ is not `null` in TMatirx. If it is for bit rate transcoding, t is selected where $j-t$ is the largest, otherwise, the smallest [35, 66].

  If the joining node can find a parent (P), P needs to join the metadata overlay by contacting the meta-head of $T_{t,j}$. If P can find a parent node to join the metadata sub-tree, P will receive the shared metadata $m_{t,j}$ within the metadata sub-tree, and can share it with other incoming nodes. If the meta-head cannot accept P into the metadata sub-tree, the joining node needs to probe another `non-null` element in the j^{th} column of TMatrix in ascending (or descending) order of $j-t$ value according to the transcoding type (bit rate or frame rate), and repeat above joining process. If all `non-null` nodes have been tried, go to next step.

– **metadata unavailable and full transcoding**: If a parent is not found in the previous step, the joining node must look for a parent that can perform full transcoding for it. To minimize the total transcoding overhead, the joining node starts to probe nodes that are receiving version t ($0 < t < j$), if $T_{t,j}$ is `null` in TMatrix. The joining node probes nodes where `t-j` is the smallest or the largest, depending on the type of transcoding as discussed above. If a node P accepts the joining node as a child, P starts a full transcoding process. Accordingly, P joins the metadata overlay, and updates the element $T_{t,j}$ in TMatrix if node P needs to do transcoding from quality t to quality j, and node P becomes the meta-head of $T_{t,j}$. The element $T_{t,j}$ of TMatrix is changed from `null` to (t, P). At the same time, the arriving node joins the data overlay with source index i.

The data-head of DIS_{ji} accepts the joining node as its member, and updates the DIS_{ji} list.

If a parent node still cannot be found, and the version j is not the lowest quality version, the node starts to request version $j+1$ following the above procedure. Otherwise, the joining request is rejected.

4.3.2 Node Departure

In P2P/overlay streaming systems, participating nodes may leave the system at any time. When a node departs, DIS-Array and TMatrix must be updated to reflect the change on the data overlay and metadata overlay. The procedures are as follows.

- **Case 2.1:** If the departing node is a leaf node, and is receiving version j, assuming its parent is P,

  (1) **DATA-DEPART**: if P is not a transcoding node, remove the departing node from P's children list, and the departing node leaves the data tree as follows.

  - If the departing node is the data-head of DIS_{jk} (k is the source index of departing node), and it is the only node in the list of DIS_{jk}, remove the departing node from the list and set DIS_{jk} to *null*.
  - If the departing node is the data-head and is not the only node in the list of DIS_{jk}, the data-head selects the node with the largest bandwidth as the new data-head. In addition, the corresponding data-head element of DIS_{jk} in DIS-Array is updated.

  (2) **META-DEPART**: if P is a transcoding node conducting transcoding from quality i to quality j, P leaves the metadata sub-tree $T_{i,j}$ as follows.

  - If P is the meta-head and the only node in the sub-tree, the element $T_{i,j}$ of TMatrix is set to *null*.

– Otherwise, P selects the node with the largest available bandwidth to be the new meta-head in the transcoding plane, and the meta-head element of $T_{i,j}$ in TMatrix is set to the newly selected node.

In addition, the departing node leaves the data tree following the **DATA-DEPART** algorithm.

- **Case 2.2:** If the departing node is neither a leaf node nor a transcoding node, the children of the departing node need to find a new parent. The parent of the departing node is a good candidate if the parent is not a transcoding node, and has enough bandwidth. If the children of the departing node cannot find an appropriate parent, the children of the departing node need to perform a rejoining process, and the rejoining process may change the transcoding plane but not the quality plane because the children do not change their quality version requirement. If the parent (denoted as P) of the departing node is a transcoding node, P leaves the metadata sub-tree $T_{i,j}$ following the **META-DEPART** algorithm, and the corresponding transcoding plane is updated. If the departing node is on the DIS list of quality plane j, the departing node is removed from the system following the **DATA-DEPART** algorithm.

- **Case 2.3:** If the departing node receiving version i is not a leaf node and is a transcoding node, the adjustment must be done on both the corresponding quality plane and the transcoding plane. Assume the departing node D is in the metadata tree $T_{i,j}$, and the children of node D need quality version j. The departing node leaves the metadata sub-tree $T_{i,j}$, and the transcoding plane is updated. Following the **META-DEPART** algorithm, the departing node leaves the metadata sub-tree. If the departing node is on the DIS list of quality plane j, node D is removed from the system according to the **DATA-DEPART** algorithm. In addition, the children of node D on the data overlay need to find another transcoding parent having the same metadata in the metadata sub-tree that the departing node is originally in. If they cannot find transcoding parents through metadata sub-tree $T_{i,j}$, they need to rejoin

the system.

4.4 Dynamic Bi-overlay Rotation

Although metadata sharing through the metadata overlay can significantly reduce the total amount of CPU cycles demanded for online content adaptation, the overhead would be charged on fixed nodes, which not only is unfair, but also may exhaust the resources on these nodes quickly than expected and leads to the collapse of the system. In addition, on the data overlay, different nodes may contribute different amounts of bandwidth due to the different numbers of children they serve. In order to deal with these heterogeneity and fairness issues, Dynamic Bi-overlay Rotation is proposed. The rotation criteria for selecting candidate nodes, when to rotate, and how to rotate are presented in Section 4.4.1, Section 4.4.2, and Section 4.5, respectively.

4.4.1 Rotation Criteria

The rotation on the metadata overlay aims to reconcile CPU and bandwidth consumption for online content adaptation, while the rotation on the data overlay is to balance the bandwidth contribution of different nodes. Thus, to select nodes for rotation, it must be based on nodes' contribution on different overlays. It is nature that the rotation criteria on these two overlays are different.

Node contribution on the data overlay

On the data overlay, different nodes make different contributions due to the different numbers of children they serve. To characterize the bandwidth contribution of one node, one metric: Contribution Index (CI) is defined to reflect how much bandwidth the node contributes to the system. The CI of node i on the data overlay is defined as

$$CI_{bi} = \sum_{1}^{K} ub_{ik} \times t_{ik}, \tag{4.1}$$

where t_{ik} is node i's uploading service time for child k, K is the total number of children that node i serves on the data sub-tree, and ub_{ik} is the bandwidth consumption of node i for child k.

Node contribution on the metadata overlay

Different from data overlay, the Contribution Index of a node on the metadata overlay includes two portions: CPU and bandwidth contribution. The amount of the CPU cycles is dependent on full transcoding or meta-transcoding the node performs, while the bandwidth consumption on the metadata overlay is dependent on the size of metadata and the number of children the node serves.

Accordingly, without considering the scarcity of different resources, the contribution index of node i is $CI_i = a \times CI_{ci} + b \times CI_{mbi}$, where a and b represent the normalization factors for the CPU and bandwidth consumption. If there are k nodes in a metadata sub-tree, $a = 1/\sum_1^k CI_{ci}$ and $b = 1/\sum_1^k CI_{mbi}$. The CI_{mbi} is calculated by

$$CI_{mbi} = \sum_{1}^{K} umb_{ik} \times t_{ik}, \tag{4.2}$$

where t_{ik} is node i's uploading service time for child k, K is the total number of children that node i serves in the metadata sub-tree, and umb_{ik} is the metadata bandwidth load of node i for child k. In addition, CI_{ci} is calculated by: $CI_{ci} = c_{iv1,iv2} \times t_i$, where $c_{iv1,iv2}$ represents the node i's transcoding quality distance from version $v1$ to version $v2$ and t_i is the time during which node i is an active transcoder.

Fragile edge and rotation pair selection

Based on the definitions of CI in section 4.4.1, CI of each node on these overlays can be calculated periodically. Note that the contribution of each node is accumulative from the beginning. Based on their contributions, the protocol selects pairs of nodes as candidates

to exchange their positions in the topology. The selection works as follows. First, with the contribution of each node, each link is tagged with the absolute contribution difference value of the two neighboring nodes in each DIS and MIS. Second, the data-head or meta-head selects the link with the largest difference as a *fragile edge*. The two nodes connected by a fragile edge form a rotation pair. Thus, each rotation would only affect a small number of nodes in the system. Furthermore, a node having contributed much resources can exchange its position with a node having contributed little resources in a recursive fashion to balance their resource consumption.

4.4.2 Rotation Schedule

Having identified the rotation pairs, the system needs to determine when to perform rotation. The frequency of rotation should be considered carefully. If the sub-tree is rotated too frequently, it keeps the overlays busy with adjusting new neighborhood relationships. If the rotation happens with a very low frequency, the resource contributions of different nodes could be very skew. Thus, an appropriate rotation interval should be determined and the interval selection can be contribution-driven or time-driven. A time-driven approach is straightforward. That is to rotate the sub-tree with a fixed time interval I_r. However, the contribution is not only affected by the time period, it is also affected by the node arrivals and departures since these dynamics would affect the number of children each node serves. Therefore, the interval is determined by considering both aspects. That is, by monitoring the node arrivals and departures to the sub-tree, the data-head on the data overlay and the meta-head on the metadata overlay can determine the rotation interval dynamically as follows. Each arrival or departure is taken as an event. If the event rate, denoted as E_r, is less than E_α, a system threshold, the rotation interval is I_r. Otherwise, the rotation interval is $I_r \times E_r$. The purpose of such a design is to give extra time for nodes in a metadata sub-tree to adapt to the dynamics.

4.5 Rotation Protocol

Although rotation can balance the resource consumption among participating nodes, it is important to guarantee that the rotation would not disrupt the streaming delivery, and would not degrade user perceived experience (Quality of Service). That is, during the rotation, users should not experience additional glitches caused by rotations.

To fulfill these objectives, a technique that has been commonly used in practice [67] is leveraged. That is, during a playback session, if the playback speed is slightly changed (increased or decreased) within a certain range, users would not perceive any difference compared with a normal playback. This technique has been used by TV channels to insert additional advertisements during the normal broadcasting. The practice indicates that this range should be controlled under 20%. With the support of this technique, the buffers of the involved nodes for rotation are carefully managed to guarantee the unchanged size of the buffers before and after rotation.

4.5.1 Data Overlay Rotation and Buffer Management

Rotation is performed simultaneously for the node pairs connected by fragile edges. For a pair of nodes (a, b) to be rotated, assume node a is the parent of node b on the sub-tree. Without loss of generality, it is assumed that the child set of node a (except b) is clustered, denoted as C_a, and the child set of node b is denoted as C_b. The parent of node a is denoted as P_a.

It is assumed that by default, each active peer maintains a playback buffer that can sustain T second playback at the normal playback speed v. It is assumed that there is a playback delay of ΔT between node a and node b. This delay applies to any parent and child pair. The rotation procedure consists of two phases, and it is assumed the duration for them is T_1 and T_2, respectively. Phase 1 starts at time 0, and ends at time T_1, and phase 2 starts at time T_1, and ends at time $T_1 + T_2$. Based on these assumptions, the initial buffer size of node a, and b is $T \times v$, and the stream content in related nodes before phase 1 is shown in Figure 4.4(c). The goal is to guarantee that after the rotation, the buffer size

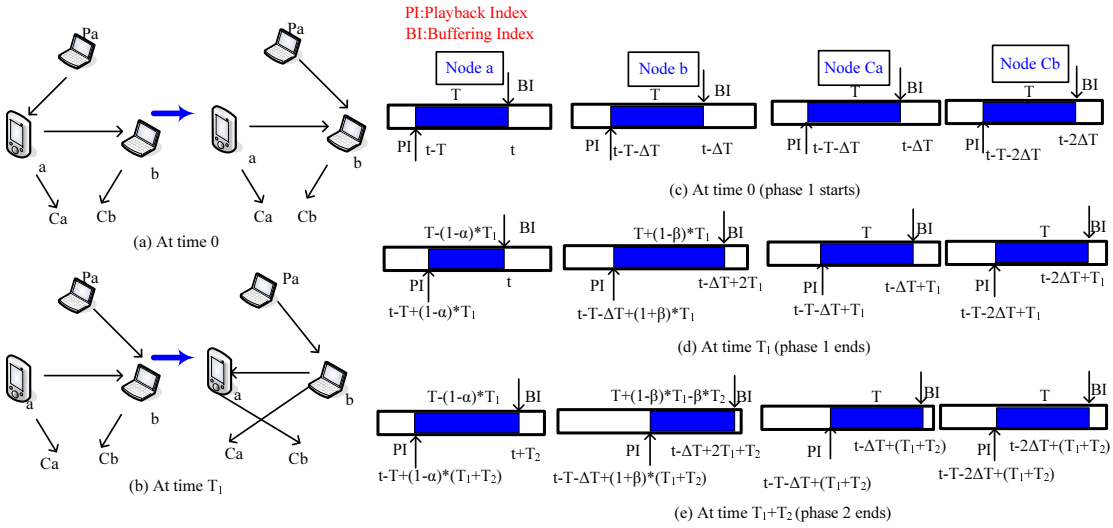

Figure 4.4: The rotation process on the data overlay

of the involved nodes remains the same based on the following procedure:

1. Figure 4.4(a) shows the original situation at time 0 (when phase 1 starts). When rotation starts, node P_a stops streaming to node a, and starts to stream to node b. Simultaneously, node a 1) reduces its playback rate by α $(0 < \alpha < 1)$, thus its playback rate is $(1 - \alpha) \times v$; 2) node a continues to stream to node b. At the same time, node b increases its playback speed by β $(0 < \beta < 1)$, thus its playback rate is $(1 + \beta) \times v$. Hence, the buffering speed of node a is 0. The buffering speed of node b is $2 \times v$ because it receives streaming from node a, and node P_a simultaneously. This phase continues till T_1. When it is T_1, the buffer size of node a is decreased from $T \times v$ to $(T - (1 - \alpha) \times T_1) \times v$, and the buffer size of node b is increased from $T \times v$ to $(T + (1 - \beta) \times T_1) \times v$. While for node C_a and C_b, as long as T_1 is less than ΔT, their buffer size is not changed. The index of playback, buffering, and the buffers of different nodes after phase 1 are shown in Figure 4.4(d).

2. Figure 4.4(b) shows the rotation situation at time T_1 (the time when phase 2 starts). At time T_1, node b starts to stream to node a and node C_a. Node a starts to stream to node C_b. At the same time, node a stops streaming to node C_a and node b stops

streaming to node C_b. From time T_1 to T_2, the buffer size of node a is increased to $(T - (1-\alpha) \times T_1 + \alpha \times T_2) \times v$, and the buffer size of node b is decreased to $(T + (1-\beta) \times T_1 - \beta \times T_2) \times v$.

Through the entire rotation procedure, the buffering and playback speed of node C_a and C_b are not changed. Thus, their buffer sizes are not affected by the rotation.

Assume for node a and node b, they resume the normal playback after T_{2a} and T_{2b} in Phase 2. For node a, at time $T_1 + T_{2a}$, its buffer size should be equal to or larger than $T \times v$, the buffer size before the rotation. Thus,

$$(T - (1-\alpha) \times T_1 + \alpha \times T_{2a}) \times v \geq T \times v => T_{2a} \geq \frac{1-\alpha}{\alpha} \times T_1 \tag{4.3}$$

For node b, it is expected that when it is $T_1 + T_{2b}$, its buffer size should be equal to or larger than $T \times v$, and node b can resume normal playback speed v. That is

$$(T + (1-\beta) \times T_1 - \beta \times T_{2b}) \times v \geq T \times v => T_{2b} \leq \frac{1-\beta}{\beta} \times T_1. \tag{4.4}$$

To minimize $T_1 + T_2$ (the duration of the entire rotation), where $T_2 = max\{min\{T_{2a}\}, max\{T_{2b}\}\}$, that is

$$T_1 + T_2 = \begin{cases} \frac{1}{\alpha} \times \Delta T & \text{if } \alpha \leq \beta, \\ \frac{1}{\beta} \times \Delta T & \text{if } \alpha > \beta. \end{cases} \tag{4.5}$$

As it has been mentioned, α, β should be controlled under 20%. The rotation time thus depends on ΔT, which is one-hop delay on the overlay. It varies depending on the underlying physical topology. However, with an average estimate of 100 ms of ΔT, the rotation can be finished in 0.5 seconds if the α and β are set to 20%. Via careful buffer management, the related nodes can be rotated without interfering with the playback of the

stream. The same conclusion can be drawn for the rotation in metadata overlay, and the detail is omitted here.

4.6 Performance Evaluation

In this section, to examine effectiveness of the rotation scheme, a large scale overlay streaming system is evaluated based on ns2 [63] simulator.

4.6.1 Simulation Setup

Both bit rate reduction and frame rate reduction on the transcoder are conducted with a 1000-second video clip, which has a data rate of 500 kb/s, and a frame rate of 30 frame/second (denoted as f/s hereafter).

Four other versions have been transcoded from the original version for each type of transcoding. In bit rate reduction, these versions have 400 kb/s to 100 kb/s with a difference of 100 kb/s in successive versions. If it is assumed that CPU consumption for a full transcoding session is 1 unit, meta-transcoding only demands 0.4 units for transcoding to these four versions with a metadata size of 10% of the original video file size regardless of different versions. In frame rate reduction, the other four versions are 500 kb/s and 400 kb/s at 30 f/s, 300 kb/s and 200 kb/s at 15 f/s, and 100 kb/s at 5 f/s. If CPU load for a full transcoding session from 30 f/s to 15 f/s takes 1 unit, the CPU load for full transcoding from 30 f/s to 5 f/s and from 15 f/s to 5 f/s takes 0.44 units and 0.36 units, respectively. While for meta-transcoding, the corresponding CPU loads are 0.5, 0.27, and 0.19 units, and the corresponding metadata size is 20%-25% of the original video file size for transcoding between different frame rates, depending on the source bit rate and the target frame rate.

With the setting of these parameters in ns2, *DOOR* is evaluated by comparing it with other three schemes. Among them, *No-Transcoding* represents the system in which no client can perform transcoding but the source node can serve precoded versions. A joining request is accepted only if the requested or a lower quality version is available. *Full-Transcoding*

represents the system in which all transcoders perform full transcoding without metadata. *Meta-Transcoding* indicates the system with the metadata overlay, but without dynamic bi-overlay rotation, which is the scheme as proposed in [35].

In the experiments, there are 2000 nodes dynamically joining and leaving the system. The source node has all the different versions, and its uploading bandwidth is 5 Mb/s. The distribution of the uploading bandwidth for other nodes is as follows: 5Mb/s (11%), 2Mb/s (3%), 896kb/s (9%), 384kb/s (21%), and 256kb/s (56%), which is based on the experimental conditions used in [68]. The CPU constraints of nodes are randomly distributed from 1 unit/second to 2 units/second. Nodes with 5Mb/s and 2Mb/s uploading bandwidth randomly request quality from version 1 to 3, and nodes with other bandwidth randomly request quality from version 1 to 5.

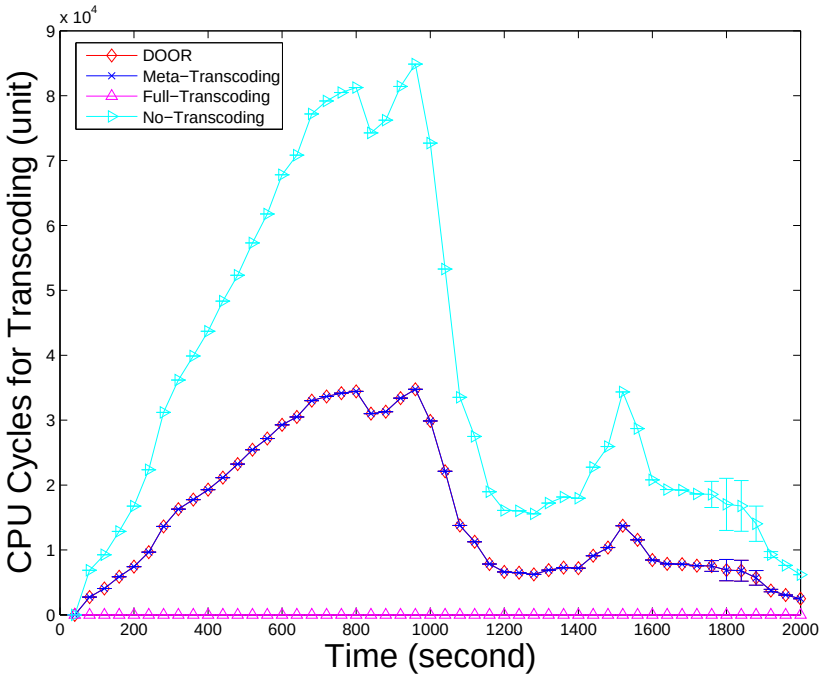

Figure 4.5: CPU cycles consumed in bit rate transcoding

The node arrivals follow a Poisson distribution with the mean arrival rate as 1 request per three seconds. The maximum arrival interval is 10 seconds. A node stays in the system receiving streaming for a duration ranging from 250 seconds to 1000 seconds. For rotation,

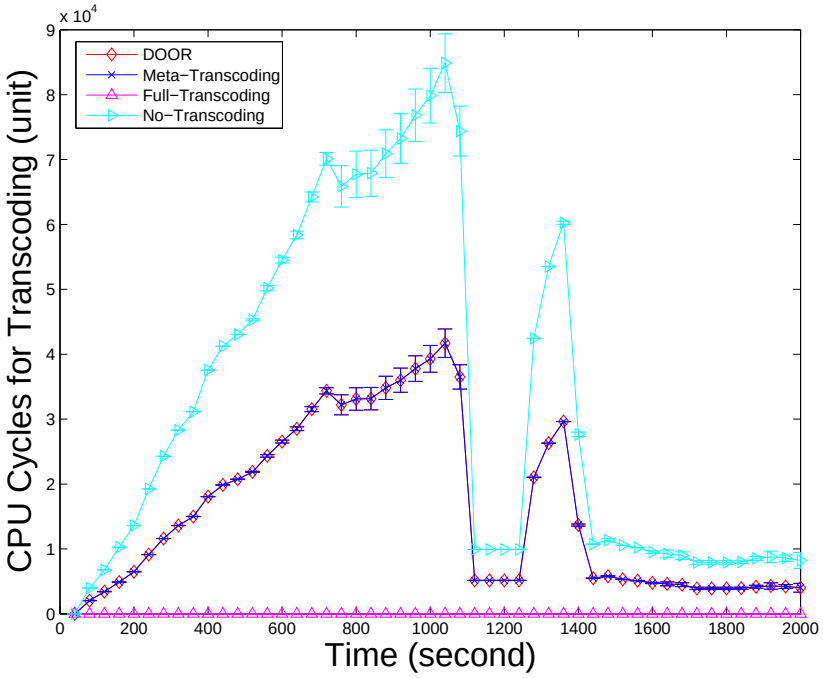

Figure 4.6: CPU cycles consumed in frame rate transcoding

the default rotation interval is set to 20 seconds. The system rotation interval threshold is set to 0.25.

4.6.2 Experimental results

First, the total CPU and bandwidth consumption of different schemes are evaluated. Figure 4.5 and Figure 4.6 show the normalized CPU load required for bit rate transcoding and frame rate transcoding, respectively. As expected, there is no CPU consumption for transcoding in No-Transcoding. Both Meta-Transcoding and DOOR consume the same amount of CPU cycles. Both schemes significantly outperform Full-Transcoding as Full-Transcoding does not utilize the metadata to reduce the CPU consumption. On average, metadata assisted schemes can save 58% CPU load compared with Full-transcoding for bit rate transcoding.

Second, the bandwidth contribution of each node is studied. After evaluating the total amount of resources consumed in different schemes, the resource consumption on individual

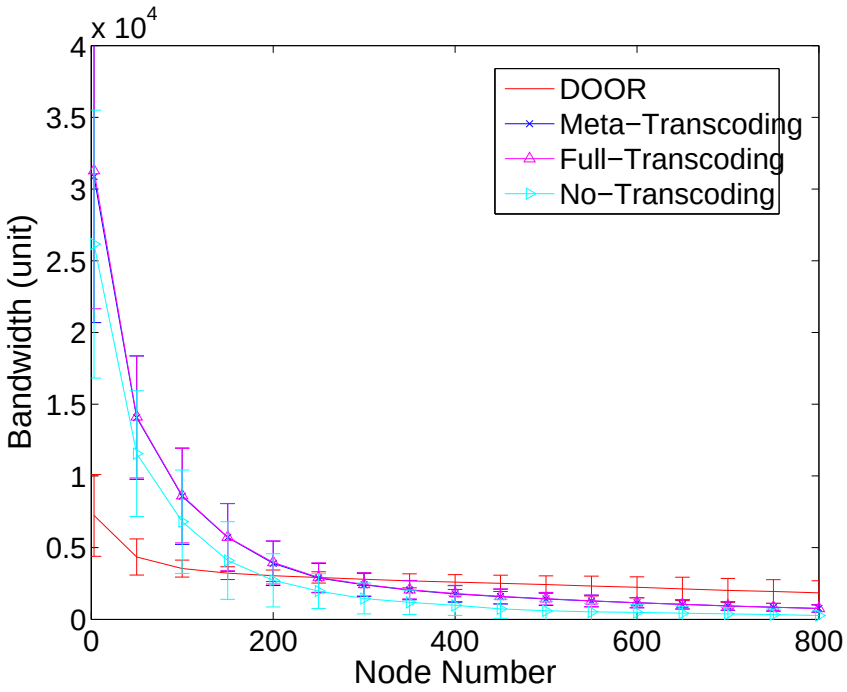

Figure 4.7: Data overlay rotation in bit rate transcoding

nodes is measured to verify whether dynamic rotation can balance their resource consumption or not. As the major bandwidth consumption is on the data overlay, this portion is focused on studying whether or not the fairness is improved with the dynamic rotation scheme. Figure 4.7 and Figure 4.8 show the peer bandwidth contribution for bit rate transcoding and frame rate transcoding, respectively. Note that the contribution values of these nodes in these figures are sorted in descending order. As shown in Figure 4.7 and Figure 4.8, compared with Full-Transcoding and Meta-Transcoding, DOOR can significantly balance the bandwidth consumption among peers. Overall, nodes in other schemes have their contributions varying from 540 to 27700 with a standard deviation of 4056, while in DOOR, it is in the range from 1280 to 5100 with a standard deviation of 745.

Third, how many nodes are forced to leave the streaming system once their battery energy is exhausted is measured. How many nodes need to leave the system when a predetermined amount of bandwidth or CPU cycles are used up is measured. Figure 4.9 and

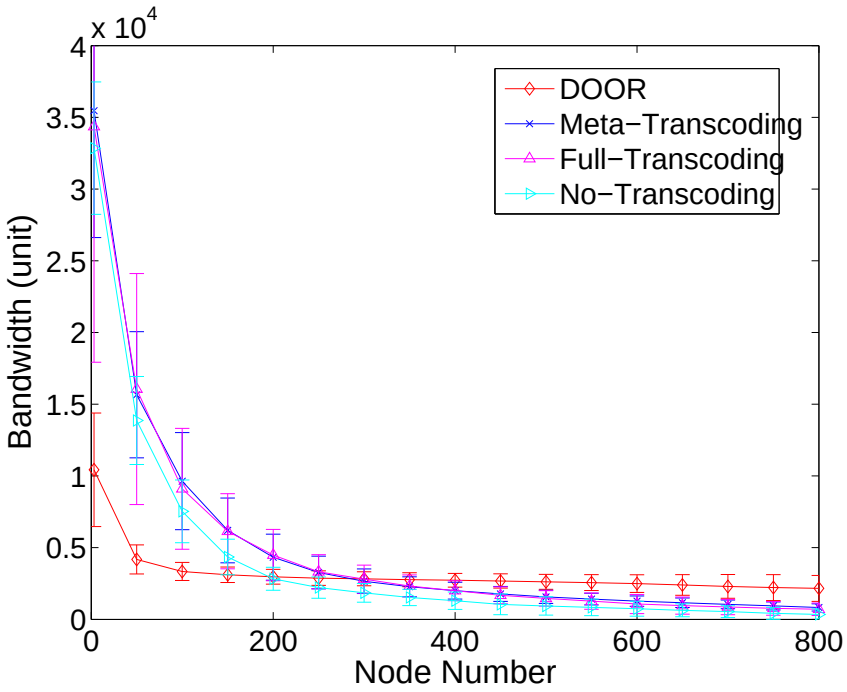

Figure 4.8: Data overlay rotation in frame rate transcoding

Figure 4.10 show the number of nodes with exhausted battery energy for bit rate transcoding and frame rate transcoding on the data overlay. As shown in the figure, when the uploading capacity threshold is set to 160 Mbtis, 320 Mbits, and 640 Mbits, compared to Meta-Transcoding, DOOR significantly reduces the number of battery-exhausted peers when the thresholds are 320 Mbits and 640 Mbits by balancing the bandwidth usage among the peers. When the threshold is 640 Mbits, at time 1000 second, the number of battery-exhausted nodes in DOOR is zero, while it is 140 in Meta-Transcoding (recall a total of 2000 nodes). However, when the threshold is 160 Mbits, the number of exhausted peers in DOOR is larger than that in Meta-Transcoding. The reason is that the available resources are evenly consumed, and cannot meet the bandwidth requirement of most peers.

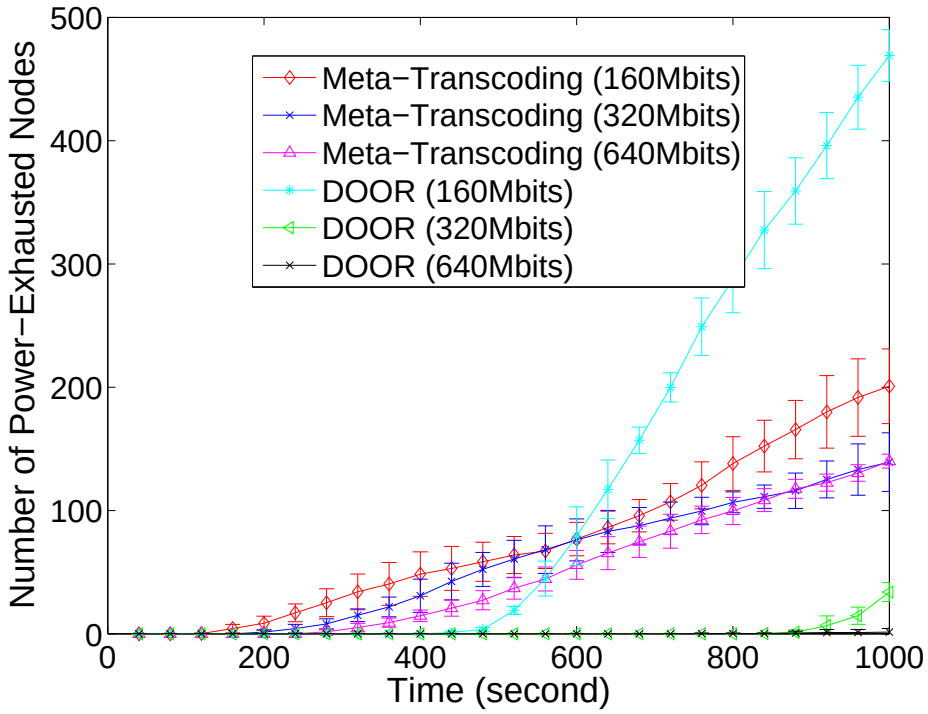

Figure 4.9: Battery-exhausted nodes on the data overlay in bit rate transcoding

4.7 Summary

The popularity of P2P/overlay streaming systems and increasing participation of mobile devices have brought up the heterogeneity and fairness problem. In order to serve heterogeneous peers and improve the robustness and fairness of the streaming systems, a dynamic rotation scheme in data and meta-transcoding overlays is proposed to balance the peer contribution of CPU and bandwidth resources. The proposed scheme is evaluated by simulations of a large scale system and the results show that the proposed scheme is effective at balancing CPU and bandwidth consumption of individual nodes over other schemes.

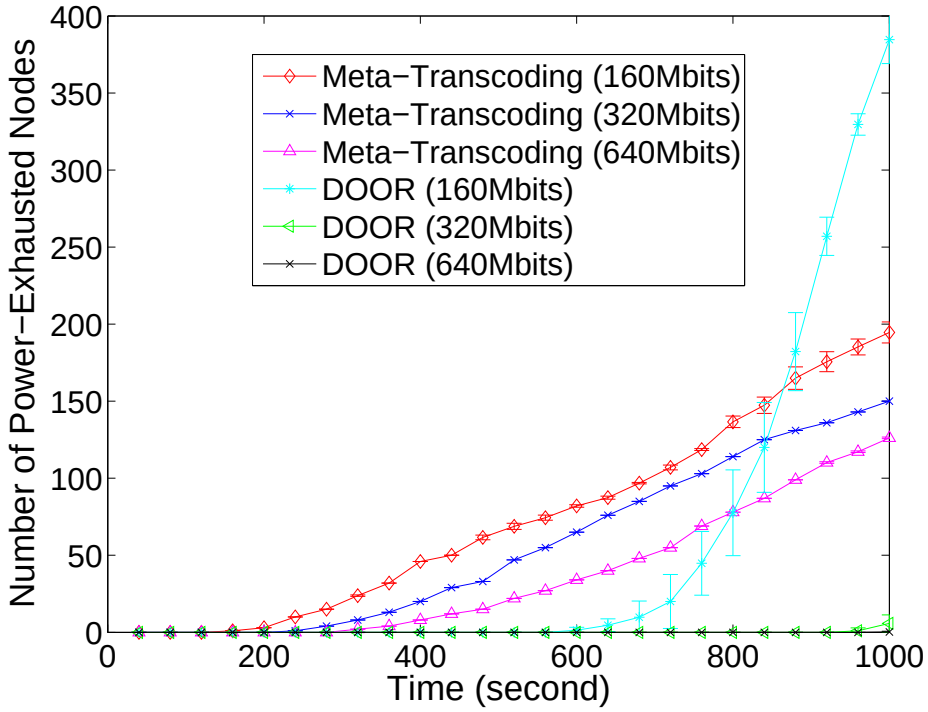

Figure 4.10: Battery-exhausted nodes on the data overlay in frame rate transcoding

Chapter 5: Towards Optimal Resource Utilization in Heterogeneous P2P Streaming System

5.1 Introduction

In previous chapters, we have examined how to efficiently address the MDH problem in both Internet on-demand and live streaming systems by effectively utilizing the limited available resources and balancing the usage of the CPU, storage, and bandwidth. All these efforts are to improve the clients' perceived QoS in a heterogeneous streaming environment via different resource management schemes. While the client perceived QoS is important, so is the system's objective to serve as many clients as possible with the available resources. In this chapter, we discuss how to optimize the resource utilization in a system to achieve both objectives simultaneously.

Along the deployment of various Internet on-demand and live streaming systems [5,7,9], lots of research has been conducted to improve the system performance, such as stability [69–72], fairness [59,62], peer perceived streaming quality [73–75], incentives [76–78]. However, in existing strategies, the utilization of the available resources has not been thoroughly examined and the resource allocation in current studies is mainly through the admission control policy. This problem is becoming more and more severe as Internet streaming is increasing popular and attracts various users using heterogeneous devices participating in the streaming service. Ideally, a streaming system should always aim to serve as many clients with as high quality as possible.

It's a good point to first review the admission control policy. Currently, the admission control strategies in existing systems can be roughly classified into four categories: `Random`, `Preemption-based`, `Reputation-based`, and `Best-Fit` algorithms.

1. `Random` algorithms choose a number of parents that can provide aggregate bandwidth for the newly arriving peer. Based on scalability [79], these algorithms are classified into scalable random algorithm [60, 80] and non-scalable random algorithm [6, 65, 81]. A scalable random approach builds a sparse graph among peers, and the newly arriving peer walks through the graph to select parents randomly. On the other hand, a non-scalable approach builds a partial or full list of all peers, and the newly arriving peer selects its parents randomly from the list. For example, in Chainsaw [60], a newly arriving peer repeatedly connects to a randomly selected peer from a set of known hosts, until the number of its neighbors reaches a threshold. However, how the bandwidth is allocated to different neighbors is not discussed in detail. In CoolStreaming [6], the newly arriving peer contacts a bootstrapping peer to get a deputy peer, and then retrieves a list of candidate parents from the deputy peer. As to its resource allocation policy, the nodes exchange information about available packets, and simply send packet without considering their bandwidth constraints. Similarly, in PRIME [81], the newly arriving peer contacts a bootstrapping peer to get a subset of participating peers randomly, and requests them to serve as its parents. As to its resource allocation, each peer's out-degree should be proportional to its uploading bandwidth in order to maximize the bandwidth utilization. However, it is not clear how to achieve this.

2. `Preemption-based` algorithms [62, 69] first choose a number of nodes as the parent nodes for a joining peer. If these candidate parent nodes do not have enough bandwidth and the contribution of the joining peer is more than existing nodes in the system, these existing nodes in the system will be preempted by the newly arriving peer. Different from Dagster [62], End System Multicast (ESM) [69] only allows the joining peer to preempt a free-loader (a peer contributes no resources). One drawback of preemption algorithms is that the system is not stable as the positions of all peers in the system are changed constantly with the arrivals and departures of peers.

3. `Reputation-based` algorithms [82] provide different services to peers according to their contributions and reputations. Generally, the reputation of a peer is determined by its contribution to the system. For example, Yan et al. [82] proposed a rank-based optimal resource allocation and admission control algorithm for P2P systems. However, it only allocates resources of source nodes in a static fashion without considering the resources of the all the peers dynamically.

4. `Best Fit` algorithms, for example, used in P2Cast [83], aim to find several existing peers with the largest available bandwidth for a newly arriving peer. Upon arrival, each peer selects peers with the largest available resources to maximize its downloading throughput.

These admission control schemes allocate the available system resources in a very coarse manner. They only consider the individual peer interest and do not optimize the throughput of the P2P streaming systems with given available resources from the system perspective.

As mentioned before, although existing protocol designs for P2P streaming have provided a rich set of solutions to the system stability, scalability, fairness, and/or peer perceived streaming quality, prior research has paid little attention to the optimal resource utilization achievable with the given resources. Note that although incentive mechanisms (which have been extensively studied in the past a few years, see [76–78, 84–92]) may be helpful in improving the system throughput, in essence, incentive mechanisms are used to balance resource utilization among peers for fairness so that free-riders could be prohibited.

In this chapter, a general theoretical framework [93] is proposed to maximize the throughput of a P2P streaming system in heterogeneous environments with given resources, while maximizing the lowest quality perceived by all the peers simultaneously. A new admission control policy is proposed, for which each peer is represented by two entities, a *supplier* entity and a *consumer* entity. Accordingly, a bipartite graph is constructed based on these two sets of entities where a linear formulation returns a max-min resource allocation. Based on the framework, a distributed algorithm is proposed to achieve near-optimal performance

with acceptable control overhead. Through extensive simulations, we compare the design with several popular existing designs, and find that most of these designs have not fully utilized the available resources in the system, leaving spacious room for further improvement. On the other hand, the results show that our proposed algorithm can always outperform these existing ones with acceptable overhead.

The rest of this chapter is organized as follows. The system model and the resource allocation algorithm are discussed in Section 5.2, Section 5.3, respectively. The protocol design is discussed in Section 5.4. Experimental results are presented in Section 5.5. Finally, we conclude this chapter in Section 5.6.

5.2 Problem Modeling

In a P2P steaming system, a peer is expected to demand services from other peers and provide services to other peers simultaneously. In a heterogeneous environment, the best quality requested by a peer could be different from other peers, thus *Relative Quality* is defined as the ratio between its actual received bandwidth and its required bandwidth and is used to represent how satisfied a peer is with the streaming service received from other peers.

From the system's perspective, the problem considered here is how the system should efficiently allocate available resources so that the worst quality perceived by the peers is maximized. The objective is to seek an algorithm that allocates the available resources to maximize the minimum quality perceived by a peer given the amount of resources contributed by all peers. To characterize the problem, the following notations are used. Assume that there are n peers in the system. For each peer j, it is assumed that it declares its resource request r_j, and contribution c_j when it arrives. To simplify the work, we assume r_j and c_j represent downloading bandwidth and uploading bandwidth for peer j, respectively.

To allocate appropriate resources to each peer, each peer is represented with two non-overlapping entities: its bandwidth request of r_j and its bandwidth contribution c_j. r_j is regarded as the best quality peer j would like to have. r_j is adjusted to reduce or increase the

service quality this peer j may receive, based on the actual contribution the peer provides to the system. Therefore, $r_j^{'}$ and $c_j^{'}$ are used to represent actual bandwidth assigned to peer j and the actual bandwidth contributed by peer j.

5.3 Resource Allocation Algorithm

With the above notations, the resource allocation problem can be formulated as follows.

Without loss of generality, at time t, it is assumed that there are n peers in the system. A bipartite graph can be built. In the bipartite graph, the left column consists of n peers' estimated actual contribution $c_i^{'}$, and the right column consists of n peers' resource request $r_i^{'}$, i=1,2,...,n. The bipartite graph is shown in Figure 5.1. For each $r_j^{'}$ in the right column, an edge connecting $r_j^{'}$ and $c_i^{'}$ in the left column is built, for any $i \neq j$. Let the weight over the edge connect peer i and peer j be w_{ij}. w_{ij} denotes the bandwidth contributed by peer i to peer j. Based on all of the known values of $c_i^{'}$ and $r_j^{'}$, w_{ij} can be calculated. Since the system may not have sufficient resources to serve all of the peers' demand, one virtual peer S is introduced. The weight of the edge over S and $r_j^{'}$ is w_{sj}. w_{sj} denotes the extra virtual resource that should be assigned to j, in addition to what has been provided by other peers in the system. In general, this value should be 0 to guarantee that peer j really gets a fully satisfied quality $r_j^{'}$. Please note that introducing the virtual peer S is to guarantee that a solution can always be found to the resource allocation problem.

In order to make each peer j receive an acceptable quality of streaming service, we have

$\sum_{i=1}^{n} w_{ij} + w_{sj} \geq r_j^{'}, i \neq j, j = 1, 2, ..., n$

Since a peer i contributes no more than what it promises, we have

$\sum_{j=1}^{n} w_{ij} \leq c_i^{'}, j \neq i, i = 1, 2, ..., n$

Considering that a peer cannot serve itself, for each single pair peer i and peer j, $i \neq j$, that is

$w_{ij} \geq 0, w_{sj} \geq 0, i \neq j, i, j = 1, 2, ..., n$

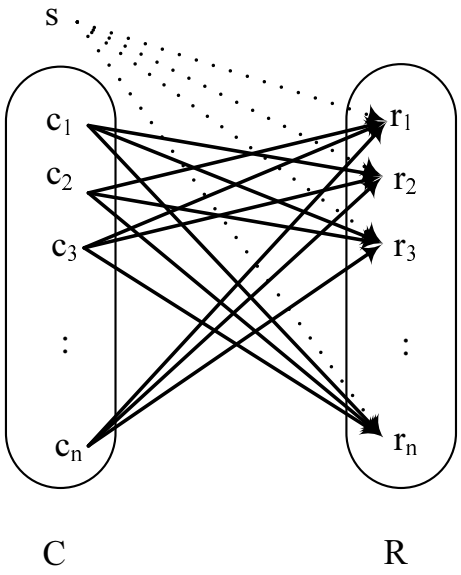

Figure 5.1: An example of bipartite graph

As S is the introduced virtual peer, S cannot contribute any real bandwidth resource to j. Thus if $w_{sj} > 0$, peer j will not get desired video quality.

As mentioned before, Relative Quality is used to represent how satisfactory the peer is. For each peer j, its relative quality is defined as $\frac{r_j^{'} - w_{sj}}{r_j^{'}} = 1 - \frac{w_{sj}}{r_j^{'}}$.

From above definition, it can be inferred that if the peer j wants to get fully satisfied quality, w_{sj} is equal to 0.

Thus, the objective of the algorithm is

$\max(\min\ (1 - \frac{w_{sj}}{r_j^{'}}))$, $\forall j = 1, 2, ..., n$

Equivalently, a linear program $\mathcal{LP}$ can be built,

$\max\ v$

subject to:

$\sum_{i=1}^{n} w_{ij} + w_{sj} \geq r_j^{'}, i \neq j, j = 1, 2, ..., n$

$\sum_{j=1}^{n} w_{ij} \leq c_i^{'}, i \neq j, i = 1, 2, ..., n$

$w_{ij} \geq 0, w_{sj} \geq 0, i \neq j, i, j = 1, 2, ..., n$

$v = min(v_j)$

$w_{sj} \times (r_j^{'})^{-1} \geq v, j = 1, 2, ..., n$

$c_j^{'} \geq 0, r_j^{'} \geq 0, j = 1, 2, ..., n$

If there exists a feasible region when $v = v_1$ and there is not a feasible region when $v = v_2$, the optimal value falls in the range $[v_1,\ v_2]$. Using binary search, a value v such that the solution of $\mathcal{LP}$ returns the optimal solution of maximizing the minimum quality of peers in a P2P system can be found. Note that if a peer j is made to receive a quality no worse than q_j', w_{Sj} just needs to be set to 0 in the above $\mathcal{LP}$. However, in this way, $\mathcal{LP}$ may not have a feasible solution. We will discuss this in the protocol design.

In the worst-case scenario, the above $\mathcal{LP}$ should be invoked upon each peer's arrival or departure. For example, Table 5.1 shows the computation time to solve $\mathcal{LP}$ with certain peer numbers using `Mosek` [94] (a third-party Linear Programming solver package). The experiment is run on a machine of Pentium D CPU 2.8 GHz with 1 GB memory and the algorithm for linear programming is the *interior-point method*. When the number of peers is increased to a relative large value, the computation overhead is increased significantly.

Table 5.1: $\mathcal{LP}$ Execution time (Seconds)

# of peers	50	100	200	400	600	800
Time	0.11	0.39	2.66	28.18	124.43	379.40

As mentioned before, the overhead due to solving $\mathcal{LP}$ involves each peer's arrival and departure. Frequently updating the connections among peers but not improving their qualities is unacceptable. Hence, a practical system needs a less time-consuming, more stable solution, which does not call for a linear programming over all peers in the system but results with near optimal resource utilization. Moreover, to reduce the cost of centralized management and to increase the scalability of the P2P streaming system, a distributed system is preferred, or at least a hybrid system without lowering the resource utilization is desired. There are two ways to reduce the cost of running a linear program: (1) reduce the

number of variables, or (2) reduce the number of *rounds* or *phases* when running the algorithms for convergence. In the chapter, a mixed approach is applied to reduce the overhead of solving $\mathcal{LP}$.

Conceptually, a few *representative entities* are employed and each of them represents a *cluster* of demands/supplies with a size of the cluster proportional to $\sqrt{n}$. Here, each peer has two entities: *resource supply* and *resource demand*. Each entity of a peer (but not the peer itself) appears in the clusters, but a peer may have its supply entity and demand entity in two different clusters. Based on this data structure, resource utilization calculation is run in a parallel manner to reduce the number of variables involved in the linear program $\mathcal{LP}$. The approach developed by Luby and Nisan [95] is used to run $\mathcal{LP}$ on a parallel machine using $O(\sqrt{n})$ processors where $c \cdot \sqrt{n}$ $(1/2 \leq c \leq 2)$ is the number of entities in each cluster. The algorithm reaches a running time of $\log(\sqrt{n})$ with the performance bounded by at least $\sqrt{n}$ of the optimal one. Details of the algorithm are as follows.

Definition 1. Balanced cluster. *Assume the system has k clusters with sizes of $x_1, x_2, \ldots, x_k$, these clusters are* balanced *if*

$$\frac{\sqrt{n}}{2} \leq k \leq 2 \cdot \sqrt{n} \text{ and } \max_{\forall i \neq j} \frac{x_i}{x_j} \leq 2. \tag{5.1}$$

Assume at time t, the system consists of n peers. Also, a set of clusters are constructed such that the number of clusters could be up to $c \cdot \sqrt{n}$, where c is a constant. At any time, these clusters are kept balanced. There are two layers of clusters: *uploading layer* of resource supply entities and *downloading layer* of resource demand entities. A peer's resource supplier entity has its uploading "neighbors" in the downloading cluster and a peer's resource demand entity has its uploading neighbors in the uploading cluster. Constituting such clusters involves two actions: *merging* and *splitting*.

1. **Merging.** When a peer leaves a cluster and that departure causes the number of peers in this cluster to be $o(\sqrt{n})$ (i.e., violate the second inequality in Eq. 5.1), the

cluster will merge with another one with the smallest size to be into a larger cluster with a size proportional to $\sqrt{n}$. Assume the two smallest clusters with sizes x_i and x_j and the largest cluster is x_l. Then it is assumed the leaving peer is from the cluster with size x_i, the new cluster has a size of $x_i + x_j$ and it satisfies Eq. 5.1 (see below). That means whenever a peer leaves, at most one merging operation is needed. The assignment in this new (bigger cluster) will be adjusted using $\mathcal{LP}$.

$$\begin{aligned} x_i + x_j &\leq \min\{x_j + x_j,\ x_l/2 + x_l\} \\ &= \min\{2 \cdot x_j,\ 1.5 \cdot x_l\} \\ x_i + x_j &\geq x_j + 1 \geq x_l/2. \end{aligned}$$

2. **Splitting.** When a peer arrives, the relation between the number of existing clusters k and $n+1$ is examined (assume before this arrival, there are n peers).

   (a) If $\sqrt{n+1}/2 \leq k \leq 2 \cdot \sqrt{n+1}$, the newly arriving peer distributes its demand entity into the cluster with the most remaining supply and distribute its supply entity into the cluster with the most resource demand, respectively. For each new arriving peer, the splitting action is taken once.

   (b) If $k \leq \sqrt{n+1}/2$, the largest cluster is split into two smaller ones: sort all demand and supply in ascending order of their values and after aligning them, split the cluster into two by halves.

   Assume the largest cluster (in terms of size) has a size of x_l before this new arrival. Then assume it is separated into two smaller clusters with sizes of x_l^1 and x_l^2, then these clusters are balanced again.

$$x_l^1 + x_l^2 = x_l \text{ and } \min\{x_l^1,\ x_l^2\} \geq \frac{x_l}{2} \geq \frac{x_i}{2}, \forall i.$$

$$k \geq \frac{\sqrt{n}}{2} \text{ implies } k+1 \geq \frac{\sqrt{n+1}}{2}.$$

Upon a new peer i's arrival, this peer has two disjunct entities: supply entity c_i' and demand entity q_i'. After contacting with each cluster and getting the information from that single cluster, then the peer puts its supply entity into the cluster deficient of the most resource and puts its demand entity into the cluster with the most remaining resource (these two clusters are different). Since there are $c\sqrt{n}$ clusters for downloading and uploading, the maximum number of entities per cluster in the P2P system is up to $2 \cdot n/(c \cdot \sqrt{n} - 1)$ ($\leq 2 \cdot (\sqrt{n}/c + 3), \forall n \geq 4$) under the constraint of Eq. 5.1. The computation overhead of running a linear program is reduced to find out the best quality for the peers. Upon each arrival, at most $c \cdot \sqrt{n}$ peers are affected and the $\mathcal{LP}$ runs over at most two clusters with the size of $(2/c) \cdot \sqrt{n} + d$, where $d \leq 3$ and d is constant. Upon each departure, a merging procedure may be involved. The distributed version of $\mathcal{LP}$ reduces the computation overhead significantly. The remaining part of the work is to prove that the resource utilization is not much affected negatively.

Theorem 1. *Let $w = w_1 + w_2$. If x is a r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.*

Theorem 2. *The above algorithm results a solution in which at most $\sqrt{n}$ peers get the optimal quality or all peers get quality at least $1/\sqrt{n}$ of the optimal one.*

Proof. Based on Theorem 1, the case when only one peer arrives or one peer leaves is considered. Assume at time t, the clusters of uploading streams (supplies) are $C_1,\ C_2,\ \ldots,\ C_i$ and the clusters of downloading streams (demands) are $L_1,\ L_2,\ \ldots,\ L_j$, where $i,\ j = c \cdot \sqrt{n}$.

1. Assume a peer arrives. Please note that this peer gets the maximum remaining resource available for the clusters, and thus, its quality is at least $1/\sqrt{n}$ of the optimal one. (The upper bound of the quality is to collect all remaining resource from all $c \cdot \sqrt{n}$ clusters.) If the splitting procedure happens, at least $\sqrt{n}$ demand entities get their

optimal quality, assuming the entities in the newly generated clusters are deficient of resources from the supply entities in the original cluster.

2. Assume a peer leaves. If the *merging* procedure has not occurred, it is claimed that a peer, if its quality is not satisfied, will connect the remaining entities in the merged clusters to find more neighbors. Otherwise, with more entities, resource distribution will be more balanced (this is the convex property of $\mathcal{LP}$).

The new algorithm can be regarded as a sparse matrix. The number of non-zero variables in each row and each column is bounded by $c \cdot \sqrt{n}$ and $\sqrt{n}/c$, respectively. The illustration of the clusters for uploading streams and downloading streams is shown in Figure 5.2. In this figure, the server icon represents the uploading streams (supply entities) and the monitor icon represents the downloading streams (demand entities). Please note that the supply entity and the demand entity of a certain peer may exist in different clusters. For example, the supply entities and the demand entities of peer i, peer j, peer k are in different clusters. □

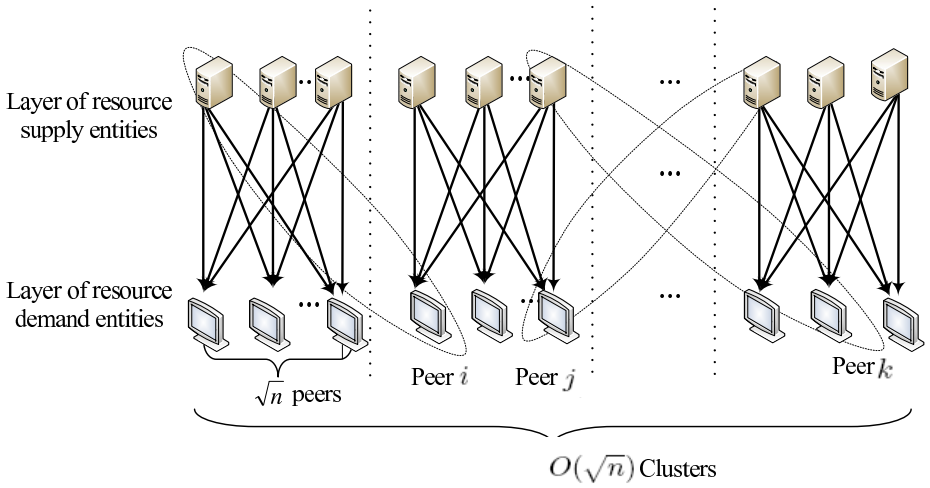

Figure 5.2: A P2P streaming system with two layers of entities

5.4 Protocol Design

Based on what have been discussed in Section 5.3, a set of protocols are designed. In the protocol design, mechanisms are also introduced to avoid peer cheating by associating a reputation system with the peers.

5.4.1 Protocol Design

The protocol consists of three parts, including *Peer Arrivals*, *Peer Goodness Adjustment*, and *Peer Departure*. In this section, the details of these parts are presented as follows.

Peer Arrival

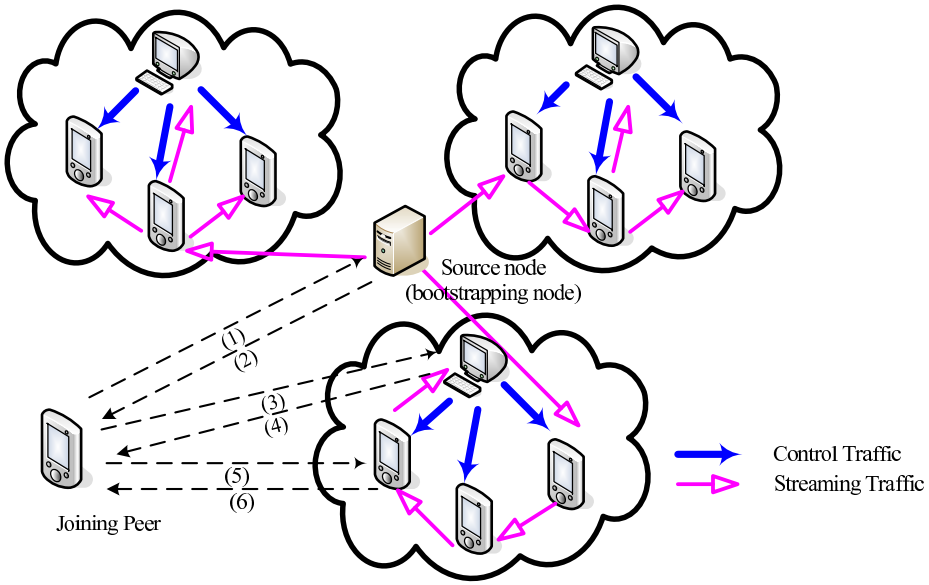

Figure 5.3: Peer Arrival Process

When a peer j arrives, it sends a joining request to the bootstrapping peer. The bootstrapping peer examines the list of the peers with unused resources and sends back such connection information to the joining peer. When a peer j receives the response from the bootstrapping peer, it specifies an inquiry, in a format of $(c'_j,\ q'_j)$, and delivers to the set of nodes capable of accepting new peers. Among all the possible acceptable quality levels,

q'_j is the best desirable quality for the peer, while c'_j is the declared resource supply from this peer. Here, the modified $\mathcal{LP}$ is run to identify the resource allocation with tolerable overhead of computation. **C** and **Q** are used to denote the sets of estimated supplies and estimated demands for the peers existing in the system, respectively. Let p be a new-coming peer. c'_p and q'_p are initialized to c_p and q_p, respectively. The *Allocate_System_Resource* works as follows:

Algorithm 1 `Allocate_System_Resource`($\mathbf{C}$, $\mathbf{Q}$, p)

1: Consider $\mathcal{LP}$ with all peers in the group and this new coming one p.
2: **for** a peer j is with goodness value `good` **do**
3: $w_{Sj} = 0$.
4: **end for**
5: Solve $\mathcal{LP}$.
6: **if** $\mathcal{LP}$ does not have a solution with $v \geq 0$ **then**
7: Flush all peers' goodness values to be `bad`.
8: **end if**
9: Accept p and update/notify peers the flow change among peers based on the value w_{ij}.

The corresponding joining procedure is shown in Figure 5.4.1.

Peer Goodness Adjustment

The routine is used by the centralized system to keep track of peer's behavior in terms of whether they make enough supply as what they have promised when joining the system. As aforementioned, each peer has a goodness value, which is `bad` or `good`, depending on whether the peer has committed the supply it declared or not. Therefore, if a peer makes supplies the same as what it promised, it has a higher priority to be served with the desirable quality. Otherwise, the quality that this peer will receive depends on how much resource available in the system (in a best-effort manner).

To correctly monitor the behavior of different peers in the group, each peer needs to record who has been uploaded to it in the past as well as the amount of supply made by these peers, and whether the service quality it receives is satisfying or not. For this purpose, each peer maintains a list of peers that have uploaded to it with the corresponding service amount of traffic. Such information is collected passively or actively when it is time to perform

dynamic resource allocation. Based on this principle, the *Peer Goodness Adjustment* is implemented with the following *Adjust_Peer_Goodness* for updating the goodness of peer j and its up-loader peer k.

Algorithm 2 `Adjust_Peer_Goodness`$(j,\ k)$

1: Set a counter for each peer in the system in the recent l time.
2: Set a threshold for each peer's goodness: if in the last l time, a peer j is reported badly in uploading for more than s times, its goodness value is set to `bad`.
3: **if** j reports its poor quality due to k and k's counter $\geq s$ in the last l time **then**
4: Set k's goodness `bad`.
5: **end if**
6: **if** A peer i's goodness is `bad` and its record is clean in the last l time **then**
7: Set i's goodness `good`.
8: **end if**

Peer Departure

After a peer leaves, the system invokes a new resource allocation, which reflects the possible change of resource allocation caused by the departure of the peer. At a peer's departure, it sends a departing message to its neighbors and the bootstrapping peer for the possible merging operation. The bootstrapping peer also updates the goodness values of those neighbors of the leaving peer, if necessary.

5.5 Performance Evaluation

In this section, the strength and weakness of the hybrid resource allocation scheme is evaluated via measuring and comparing its performance with the optimal allocation scheme and other four typical protocols that employ representative admission control strategies.

A simulator is implemented based on a linear programming solver provided by a third-party linear programming package (Mosek 5.0) [94]. In the simulation, a live streaming lasting 3600 seconds is delivered to all the peers. During these 3600 seconds, 1000 peers dynamically join and leave the system. These peers join the system following a Poisson distribution with the mean arrival rate as 1 request per three seconds and the maximum arrival interval is set to 15 seconds. It is assumed that a peer does not leave the system

until it receives full-quality video of 500 seconds.

When a peer arrives, it declares its promised supply to the system and the demand for streaming quality. The resource supply and demand are termed in bandwidth. The peer quality demands are shown in Table 5.2, represented in the heterogeneous bandwidth consumption. That distribution is based on the research results reported in [96, 97].

Table 5.2: Bandwidth distribution

Upload (Kbps)	Percentage	Download (Kbps)	Percentage
[0, 200)	35.94%	[0, 360)	25.47%
[200, 360)	27.96%	[360, 600)	64.78%
[360, 600)	24.86%	[600, 1000)	8.01%
[600, 1000)	5.14%	[1000, 2000)	1.67%
[1000, 1500)	4.76%	[2000, 5000)	0.07%
[1500, 10000)	1.34%	-	-

As some peers may not keep their initial promises along their sessions in the system [98, 99], the 1000 nodes fall into two categories: *truthful peers* who keep their promise, and *un-truthful peers* who do not keep their promise. In the experiments, the ratio of truthful and un-truthful nodes is set to 1 : 1. When a peer does not keep the promise, it selects a percentage between 40% and 80% randomly and contributes that percentage of its promised bandwidth to the system. To bootstrap the system, a bootstrapping (source) peer is assigned to provide $1M$ bps bandwidth resources.

The performance of the design is evaluated by comparing with systems using

`Random-selection` [6], `Best-fit` [83], `Preemption-based` [62], and `Reputation-based` [82] admission algorithms as discussed before, and these four algorithms are abbreviated as `Random`, `Best-fit`, `Preemption`, and `Reputation`, respectively. `Optimal` is used to represent the optimal solution and `Hybrid` is used to represent our proposed hybrid protocol using several clusters of entities.

5.5.1 System Throughput

The system throughput is defined as the number of peers that the system can support with available resources. If the relative quality of the peer is less than 0.33, the peer is not counted as peers supported by the system. Figure 5.4 shows the accumulated throughput of all algorithms during the experiments. It is shown that `Preemption` achieves the lowest throughput among all these systems. Precisely, 32.8%, 37.4%, 89.3%, and 91.8% of clients can be served by `Preemption`, `Random`, `Best-fit`, and `Reputation`, respectively. Both `Hybrid` and `Optimal` can support all clients in the system.

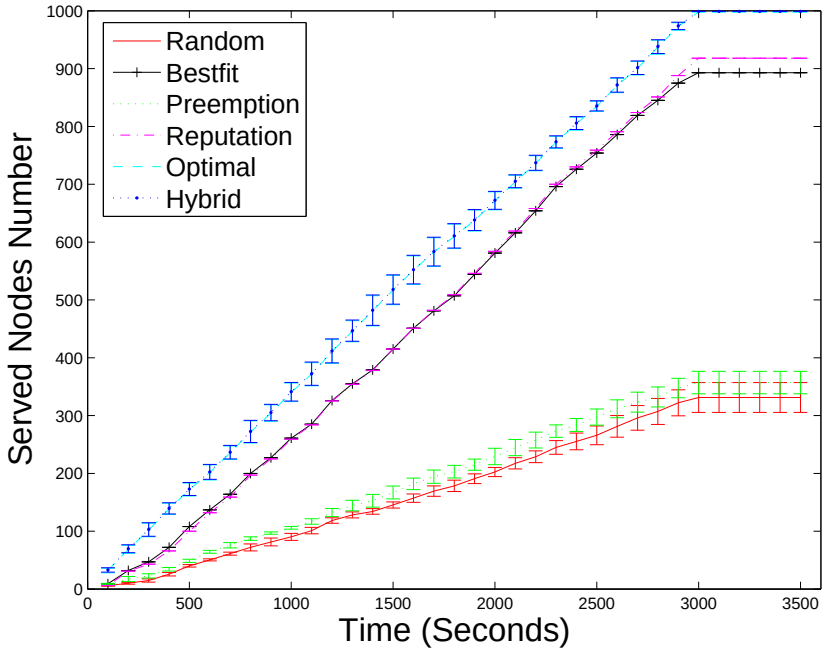

Figure 5.4: System Throughput (Cumulative)

5.5.2 Relative Quality

While system throughput reflects the system overall throughput, it does not reflect individual peer's perceived streaming quality. The average relative quality perceived by all peers in different systems are also studied. Figure 5.5 shows the average relative quality perceived by all peers in different systems. As defined before, the relative quality is the ratio of the

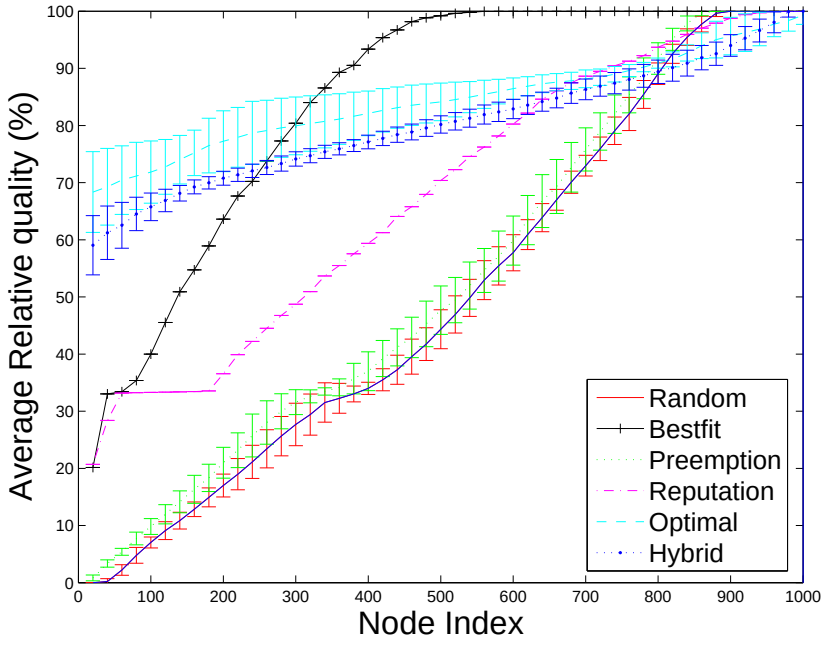

Figure 5.5: Average Quality of All Nodes using Different Algorithms

resource actually received and resource demanded by each peer. We list the average relative quality of each peer. If one peer is not accepted by the system, its relative quality is set to 0. The x-axis is the node number. The y-axis is the average relative quality which is calculated by averaging the relative quality for each client during its stay.

As expected, the average relative quality of the `Hybrid` scheme is less than that of the `Optimal` scheme. It can be observed that the average relative quality of the `Hybrid` scheme is still better than the other four schemes. The average relative quality of the peers in `Optimal` is always above 83.9% and the relative quality of the peers in `Hybrid` is 79.7%. But these two schemes are much better than the other four schemes.

5.5.3 Resource Contribution and Utilization

The previous evaluation results show that in general, peers in the `Hybrid` scheme can achieve better quality than in other systems in terms of streaming quality they receive as well as the overall system throughput. According to the design, `Hybrid` achieves a near-optimal resource allocation and utilization. In the following, how efficiently resources are used in

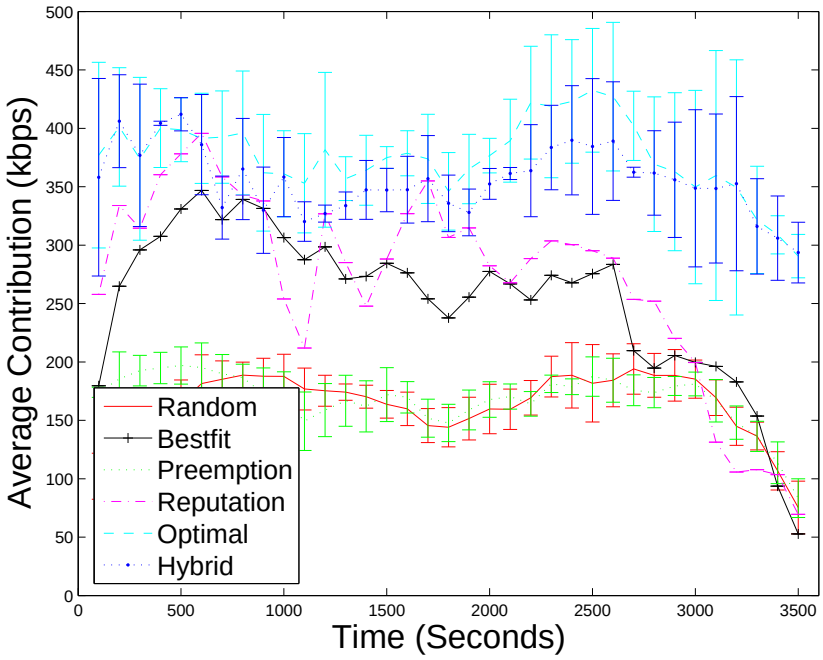

Figure 5.6: Average Quality of All Peers using Different Algorithms

different schemes are evaluated.

Figure 5.6 demonstrates the average node contribution along the system running time. The y-axis is the average bandwidth contribution of all joined peers. The figure shows that the average supply in the `Hybrid` approach is similar to that in `Optimal` and they are better than that in other systems and the average values of peer supply in `Optimal`, `Hybrid`, `Reputation`, `Best-fit`, `Preemption`, and `Random` are 373.7$k$bps, 353.6$k$bps, 271.7$k$bps, 250.7$k$bps, 168.1$k$bps, and 159.2$k$bps, respectively. This partially explains why `Hybrid` and `Optimal` can serve more peers with higher relative qualities than the other four systems.

5.5.4 Peer Duration

In consideration of social welfare of the system, such as system throughput, resource utilization, and supply, `Optimal` is the best and `Hybrid` can achieve near-optimal performance. They perform much better than the other four systems. Now we examine the peer duration. Peer duration is one main design objective for distributed P2P systems [100]. A peer

duration is the flow time it stays in the system for streaming data. The less duration of the peer, the better the relative quality.

Figure 5.7 shows the results of peer duration in the system. To zoom in on the difference, we take the duration in log scale (base $e \approx 2.718$). To evaluate the flow time of peers, the peers' durations are sorted. In the experiment, if the peer cannot get the streaming data completely within 3600 seconds, its duration is set to 3600.

The result demonstrates that `Optimal` has less duration than `Hybrid`. Furthermore, `Hybrid` has less duration than `Reputation`, `Random`, `Best-fit`, and `Preemption`. As less duration for each peer means better relative quality for that peer, `Hybrid` scheme provides better relative quality to all clients than other four systems.

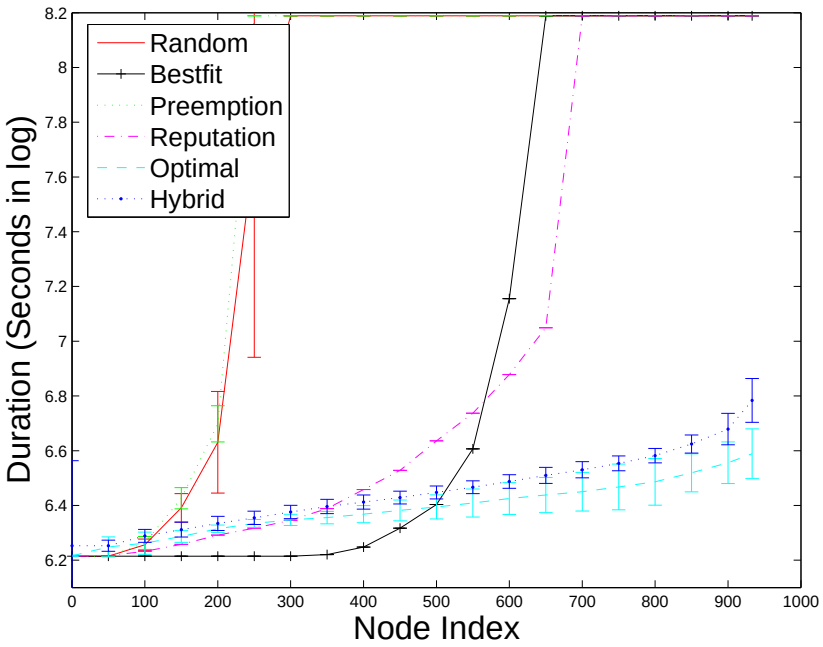

Figure 5.7: Comparison of Node Duration

5.5.5 Overhead

Though `Hybrid` and `Optimal` schemes have shown their efficiency in utilizing the system resources and minimizing service-time for peers, it comes with some cost of management, which is measured as an overhead. In this section, the control overhead due to frequent

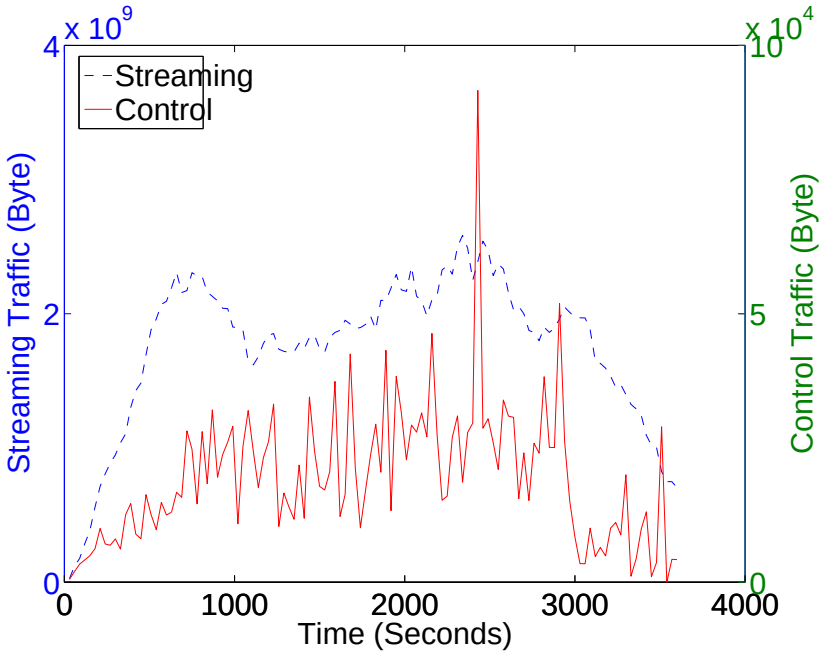

Figure 5.8: Comparison of Streaming Traffic and Control Traffic

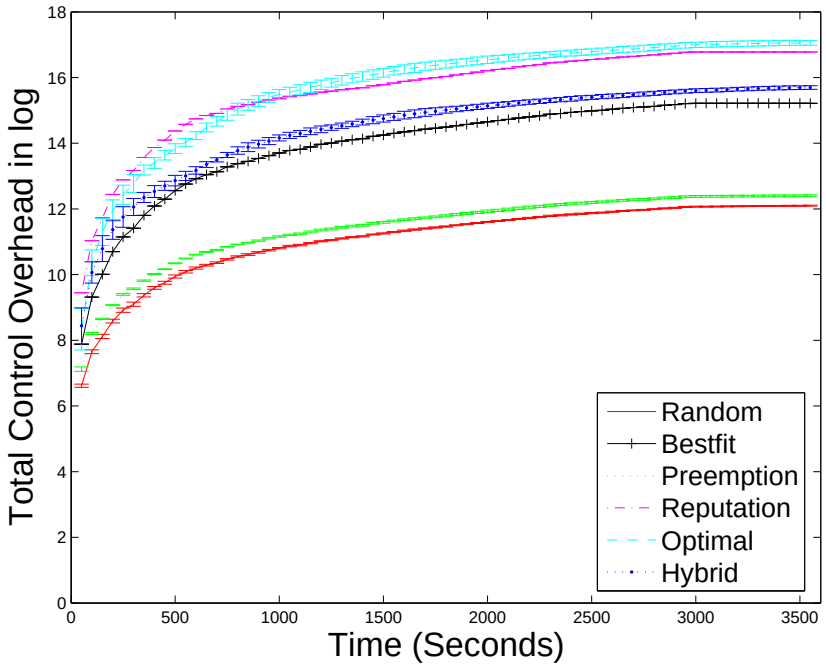

Figure 5.9: Control Overhead in Different Systems (Cumulative)

bandwidth re-allocation in terms of bytes and microseconds is measured.

Figure 5.8 shows the total control traffic amount in `Hybrid` scheme. The *left* y-axis is the total streaming traffic in bytes and the *right* y-axis is the control traffic in bytes. As shown in Figure 5.8, compared with the total streaming traffic, the control overhead is negligible for `Hybrid`. In our simulation, the control traffic only accounts for 0.005% of the streaming traffic.

We also compare the control overhead in `Optimal` and `Hybrid` with that of other four systems. Figure 5.9 shows the total control overhead in all systems. As expected, compared with total control overhead in `Reputation`, `Best-fit`, `Random`, and `Preemption`, the total control overhead in `Optimal` is the largest; while the total control overhead in `Reputation` algorithm is significantly larger than the other four systems. `Hybrid` has much less control overhead compared with the `Optimal` scheme. The result reflects the tradeoff between the performance and the control overhead. It is observed that the control overhead of `Hybrid` is reduced to a great extent compared with the optimal scheme, but `Hybrid` has near-optimal resource utilization compared with `Optimal`. By choosing `Hybrid` instead of `Optimal`, the scalability of the system is improved significantly. Furthermore, considering the ratio of the control overhead to the total streaming traffic, that overhead of `Hybrid` is still acceptable.

The time overhead in terms of microseconds is shown in Table 5.3. Although `Hybrid` has higher overhead than `Best-fit`, `Random`, and `Preemption`, it is better than `Reputation` and its overhead is no more than 4% of that of `Optimal`.

Table 5.3: Comparison of Overhead (in microseconds)

	Running time per peer (%)
Best-fit	138113 (3.0)
Random	127244 (2.8)
Preemption	127990 (2.8)
Reputation	211136 (4.6)
Optimal	4563609 (100)
Hybrid	179705 (3.9)

Our experimental results indisputably show that the distributed resource allocation

scheme can always achieve near-optimal performance in terms of system throughput, peer perceived quality and resource utilization compared with the optimal resource allocation.

5.6 Summary

While the popularity of P2P/overlay streaming systems have attracted many users and researchers, the resource allocation and utilization problem in existing P2P systems has not been well studied. In this chapter, a theoretical framework was proposed to maximize the resource utilization in heterogeneous streaming systems, while the minimum quality perceived by peers in the system is also maximized simultaneously. By comparing with the existing coarse admission control policies, we propose a distributed resource allocation approach and show that the proposed hybrid design can achieve near-optimal performance with acceptable control overhead.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This dissertation investigates efficient resource management schemes to address the MDH problem in live and on-demand streaming systems. The contributions of the dissertation are as follows. (1) To address the MDH problem in on-demand streaming systems, a model is constructed to study the tradeoff between the CPU and the storage consumption for various meta-caching schemes. Accordingly, a scheme is proposed to adaptively use the meta-caching scheme to reduce the CPU consumption according to the clients access pattern and available resources. (2) In live streaming systems, the idle peer computation resource is utilized to enable effective online transcoding in overlay streaming systems based on an additional transcoding overlay. (3) To address the fairness problem in streaming systems, a dynamic bi-overlay rotation scheme is proposed to balance the peer contribution of the CPU and bandwidth resource in both data overlay and metadata overlay, thus enhancing the robustness of the system. (4) To optimize the throughput of streaming systems and maximize the streaming quality received by peers with given resources, a general theoretical framework is built to maximize the system resource utilization in streaming systems based on P2P/overlay. Based on this framework, a distributed resource allocation approach is proposed to achieve near-optimal performance with acceptable overhead.

6.2 Future Work

The future work is to make heterogeneous mobile devices participate in P2P based live and on-demand streaming systems conveniently. The results of the dissertation have shown

that efficient resource management is the appropriate approach to address the MDH problem under heterogeneous environments. In the future, we plan to further investigate the following problems.

A. On-demand and Live Streaming Systems with Quality Monitoring

Previous research efforts of streaming systems often focus on how to improve the efficiency, scalability, and more functionalities of streaming applications. Little attention has been paid to monitor peers perceived quality in real time. In the real on-demand and live streaming system, if individual peer quality can be actively monitored, and corresponding actions such as assigning new parents or extra resources can be taken in time, the users' watching experience will be improved effectively. In the future, a real-time quality monitoring and diagnosis mechanism will be studied and designed to improve the video quality perceived on the client side.

B. P2P-assisted Video On Demand (VoD) Streaming System

Recently more and more P2P live streaming systems such as PPLive [5] and PPStream [7] begin to provide P2P-assisted Video On Demand (VoD) streaming services to their customers. For the customers, VoD streaming makes it possible for them to watch the partial video that is interesting to them. But for the streaming service providers, VoD streaming will incur more resource demand and more control overhead in order to achieve the same level of QoS compared with the P2P live streaming services. Especially, with the increasing number of mobile devices joining the P2P-assisted VoD systems, the situation is more severe. Thus, in the heterogeneous environment, how to efficiently manage resources in the client and the servers in order to serve heterogeneous mobile devices well and satisfy their quality requirement needs to be further investigated.

Bibliography

[1] "New wireless standard," http://www.ieee802.org/11/.

[2] "3G standard," http://www.itwire.com.au/content/view/5383/127/.

[3] "Comscore," http://www.comscore.com/.

[4] "Youtube Users," http://www.comscore.com/press/release.asp?press=2324.

[5] "PPLive," http://www.pplive.com.

[6] X. Zhang, J. Liu, B. Li, and T. Yum, "Coolstreaming/donet: A data-driven overlay network for efficient live media streaming," in *Proceedings of IEEE INFOCOM*, Miami, FL, USA, March 2005.

[7] "PPStream," http://www.ppstream.com/.

[8] "UUsee," http://www.uusee.com.

[9] "TVants," http://www.tvants.com/.

[10] X. Liu, H. Jin, Y. Liu, L. Ni, and D. Deng, "AnySee: Peer-to-peer live streaming," in *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[11] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM SIGMETRICS*, June 2000.

[12] "Mysee," http://www.mysee.com.

[13] "CDMA2000," http://www.umtsworld.com/technology/cdma2000.htm.

[14] "CDMA2000 Users," http://www.cdg.org/technology/3g.asp.

[15] "CDMA2k Users," http://www.reuters.com/article/pressRelease/idUS19317+03-Jun-2009+GNW20090603.

[16] "HSPDA," http://www.3gnewsroom.com/3g_news/dec_04/news_5314.shtml.

[17] "UMTS," http://www.umtsworld.com/technology/overview.htm.

[18] "EVDO," http://www.evdoinfo.com/content/view/37/61/.

[19] "Verizon.com," http://www.verizon.com/.

[20] "V-CAST," http://products.vzw.com/index.aspx?id=video.

[21] "AT&T.com," http://www.att.com/.

[22] "Cingular Video," http://www.wireless.att.com/learn/messaging-internet/media-entertainment/video.jsp.

[23] "Nielsen mobile," http://www.nielsenmobile.com/.

[24] "Mobile users," http://www.nielsenmobile.com/documents/CriticalMass.pdf.

[25] "Wireless access," http://wapreview.com/blog/?p=152.

[26] R. Mohan, J. Smith, and C. Li, "Adapting multimedia internet content for universal access," in *IEEE Transactions on Multimedia*, vol. 1 (1), March 1999.

[27] Z. Liu, Y. Shen, S. Panwar, K. Ross, and Y. Wang, "P2p video live streaming with mdc: Providing incentives for redistribution," in *Proceedings of IEEE International Conf. on Multimedia and Expo (ICME)*, Beijing, China, July 2007.

[28] V. Goyal, "Multiple description coding: Compression meets the network," in *IEEE Signal Processing Magazine*, vol. 18, September 2001.

[29] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," in *Proceedings of IEEE ICNP*, Atlanta, GA, USA, November 2003.

[30] Z. Liu, Y. Shen, S. Panwar, K. Ross, and Y. Wang, "Using layered video to provide incentives in p2p live streaming," in *Proceedings of ACM P2P-TV*, Kyoto, Japan, August 2007.

[31] "Apple trailer," http://www.apple.com/trailers/.

[32] Y. Wang, A. Reibman, and S. Lin, "Multiple description codign for video delivery," in *Proceedings of IEEE*, vol. 93, January 2005.

[33] M. Ghanbari, "Two-layer coding of video signals for vbr networks," in *IEEE Journals on Selected Areas in Communications*, vol. Vol 7, no. 5, June 1989.

[34] M. Nakamura and K. Sawada, "Scalable coding schemes based on dct and mc prediction," in *Proceedings of IEEE ICIP*, Washington, DC, USA, October 1995.

[35] D. Liu, E. Setton, B. Shen, and S. Chen, "Pat: Peer-assisted transcoding for overlay streaming to heterogeneous devices," in *Proceedings of ACM NOSSDAV*, Urbana-Champaign, IL, USA, June 2007.

[36] J. Xin, C. Lin, and M. Sun, "Digital video transcoding," in *Proceedings of IEEE*, vol. 93(1), January 2005, pp. 84–97.

[37] R. Rejaie and J. Kangasharju, "Mocha: A quality adaptive multimedia proxy cache for internet streaming," in *Proceedings of ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, June 2001.

[38] P. D. Cuetos, D. Saparilla, and K. W. Ross, "Adaptive streaming of stored video in a tcp-friendly context: Multiple versions or multiple layers?" in *Proceedings of Packet Video Workshop*, Kyongju, Korea, April 2001.

[39] F. Hartanto, J. Kangasharju, M. Reisslein, and K. W. Ross, "Caching video objects: layers vs versions?" in *IEEE International. Conf. on Multimedia and Expo*, Lausanne, Switzerland, August 2002.

[40] T. Kim and M. H. Ammar, "A comparison of layering and stream replication video multicast schemes," in *Proceedings of ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, USA, June 2001.

[41] E. Amir, S. McCanne, and H. Zhang, "An application level video gateway," in *Proceedings of ACM Multimedia*, San Francisco, CA, USA, November 1995.

[42] S. Acharya and B. C. Smith, "Middleman: A video caching proxy server," in *Proceedings of ACM NOSSDAV*, Chapel Hill, NC, USA, May 2000.

[43] E. A. Brewer, R. H. Katz, Y. Chawathe, S. D. Gribble, T. Hodes, G. Nguyen, M. Stemm, T. Henderson, E. Amir, H. Balakrishnan, A. Fox, V. N. Padmanabhan, and S. Seshan, "A network architecture for heterogeneous mobile computing," in *IEEE Personal Communications*, vol. 5(5), October 1998, pp. 8–24.

[44] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas, "Dynamic adaption in an image transcoding proxy for mobile web browsing," in *IEEE Personal Communications*, vol. 5(6), December 1998, pp. 8–17.

[45] C. K. Hess, D. Raila, R. H. Campbell, and D. Mickunas, "Design and performance of mpeg video streaming to palmtop computers," in *Proceedings of ACM/SPIE Conference on Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, January 2000.

[46] T. Warabino, S. Ota, D. Morikawa, M. Ohashi, H. Nakamura, H. Iwashita, and F. Watanabe, "Video transcoding proxy for 3g wireless mobile internet access," in *IEEE Communication Magazine*, vol. 38(10), October 2000, pp. 66–71.

[47] X. Tang, F. Zhang, and S. T. Chanson, "Streaming media caching algorithms for transcoding proxies," in *Proceedings of the 31st International Conference on Parallel Processing (ICPP)*, Vancouver, Canada, August 2002.

[48] B. Shen, S. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," in *IEEE Transactions on Multimedia*, vol. 6, April 2004, pp. 375–386.

[49] N. Sarhan and C. Das, "Caching and scheduling in nad-based multimedia servers," *IEEE Transactions on Parallel and Distributed Systems*, no. 10, 2004.

[50] B. Shen, "Meta-caching and meta-transcoding for server side service proxy," in *Proc. of IEEE International Conf. on Multimedia and Expo (ICME)*, vol. vol. I, Baltimore, MD, USA, July 2003.

[51] D. Liu, S. Chen, and B. Shen, "Amtrac: Adaptive meta-caching for transcoding," in *Proceedings of the ACM NOSSDAV*, Newport, RI, USA, May 2006.

[52] ——, "Modeling and optimization of meta-caching assisted transcoding," in *IEEE Transactions on Multimedia*, vol. 10, December 2008.

[53] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, March 1999.

[54] D. Dolgikh and A. Sukhov, "Parameters of cache systems based on a zipf-like distribution," *Computer Networks*, vol. 37, June 2001.

[55] L. Cherkasova and M. Gupta, "Characterizing locality, evolution, and life span of accesses in enterprise media server workloads," in *Proceedings of ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Miami, FL, May 2002.

[56] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[57] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "Disc: Dynamic interleaved segment caching for interactive streaming," in *Proceedings of the 25th International Conference on Distributed Computing Systems*, Columbus, OH, USA, June 2005.

[58] T. Small, B. Liang, and B. Li, "Scaling laws and tradeoffs in peer-to-peer live multimedia streaming," in *Proceedings of ACM Multimedia*, Santa Barbara, CA, USA, October 2006.

[59] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative environment," in *Proceedings of 2nd IPTPS*, Berkeley, CA, USA, February 2003.

[60] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *Proceedings of 4th International Workshop Peer-to-Peer Systems*, Ithaca, NY, USA, February 2005.

[61] I. Kouvelas, V. Hardman, and J. Crowcroft, "Network adaptive continuous-media applications through self organised transcoding." in *Proceedings of ACM NOSSDAV*, Cambridge, UK, July 1998.

[62] W. Ooi, "Dagster: Contributor-aware end-host multicast for media streaming in heterogeneous environment," in *Proceedings of ACM/SPIE Conference on Multimedia Computing and Networking*, San Jose, CA, USA, January 2005.

[63] "The network simulator," http://www.isi.edu/nsnam/ns.

[64] M. Hefeeda, A. Habib, B. Boyan, D. Xu, and B. Bhargava, "PROMISE: peer-to-peer media streaming using CollectCast," in *Proceedings of ACM MultiMedia*, Berkeley, CA, USA, November 2003.

[65] D. Tran, K. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 2003.

[66] D. Liu, S. Chen, and B. Shen, "Dynamic bi-overlay rotation for streaming with heterogeneous devices," in *Proceedings of 15th Annual Multimedia Computing and Networking*, San Jose, CA, USA, January 2008.

[67] H. Jiang and S. Jin, "Nsync: Network synchronization for peer-to-peer streaming overlay construction," in *Proceedings of ACM NOSSDAV*, New port, RI, USA, May 2006.

[68] E. Setton, J. Noh, and B. Girod, "Congestion-distortion optimized peer-to-peer video streaming," in *Proceedings of IEEE ICIP*, Portland, OR, USA, October 2006.

[69] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *Proceedings of ACM SIGCOMM*, San Diego, CA, USA, August 2001.

[70] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proceedings of the 12th ACM Workshop on Network and Operating System Support for Digital Audio and Video*, Miami, FL, USA, May 2002.

[71] F. Wang, Y. Xiong, and J. Liu, "mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast," in *Proceedings of IEEE ICDCS*, Toronto, Ontario, Canada, June 2007.

[72] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *Proceedings of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[73] T. Bonald, L. Massoulie, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: Optimal performance trade-offs," in *Proceedings of ACM SIGMETRICS*, Annapolis, MD, USA, June 2008.

[74] M. Zhang, C. Chen, Y. Xiong, Q. Zhang, and S. Yang, "Optimizing the throughput of data-driven based streaming in heterogeneous overlay network," in *Proceedings of ACM Multimedia Modeling*, 2007.

[75] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proceedings of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[76] S. Zhong, J. Chen, and Y. Yang, "Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 2003.

[77] Z. Zhang, S. Chen, and M. Yoon, "March: A distributed incentive scheme for peer-to-peer networks," in *Proceedings of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[78] Y. Zhang, W. Lou, and Y. Fang, "Sip: A secure incentive protocol against selfishness in mobile ad hoc networks," in *Proceedings of IEEE WCNC*, Atlanta, GA, USA, 2003.

[79] V. Vishnumurthy and P. Francis, "A comparison of structured and unstructured p2p approaches to heterogeneous random peer selection," in *Proceedings of IEEE INFOCOM*, Barcelona, Spain, May 2006.

[80] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proceedings of ACM SOSP*, Bolton Landing, NY, USA, October 2003.

[81] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *Proceedings of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[82] Y. Yan, A. El-Atawy, and E. Al-Shaer, "Ranking-based optimal resource allocation in peer-to-peer networks," in *Proceedings of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[83] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-peer patching scheme for vod service," in *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, Budapest, Hungary, May 2003.

[84] A. Habib and J. Chuang, "Incentive mechanism for peer-to-peer media streaming," in *Proceedings of IEEE IWQoS*, Montreal, Canada, June 2004.

[85] D. Figueiredo, J. Shapiro, and D. Towsley, "Incentives to promote availability in peer-to-peer anonymity systems," in *Proceedings of IEEE ICNP*, Boston, MA, USA, November 2005.

[86] M. Sribatsa, L. Xiong, and L. Liu, "Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks," in *Proceedings of ACM World Wide Web*, Chiba, Japan, May 2005.

[87] L. Buttyan and J. Hubaux, "Enforcing service availability in mobile ad-hoc wans," in *Proceedings of ACM Mobihoc*, Boston, MA, USA, August 2000.

[88] ——, "Stimulating cooperation in self-organizing mobile ad hoc networks," in *ACM Journal for Mobile Networks (MONET)*, vol. 8, October 2003.

[89] T. Ma, S. C. Lee, J. C. Lui, and D. K. Yau, "A game theoretic approach to provide incentive and service differentiation in p2p networks," in *Proceedings of ACM SIGMETRICS/PERFORMANCE*, New York, NY, USA, June 2004.

[90] N. Salem, L. Buttyan, J. Hubaux, and M. Jakobsson, "A charging and rewarding scheme for packet forwarding in multi-hop cellular networks," in *Proceedings of ACM Mobihoc*, Annapolis, MD, USA, June 2003.

[91] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proceedings of ACM Electronic Commerce*, New York, NY, USA, May 2004.

[92] M. Adler, R. Kumar, K. Ross, D. Rubenstein, T. Suel, and D. Yao, "Optimal peer selection for p2p downloading and streaming," in *Proceedings of IEEE INFOCOM*, Hong Kong, China, March 2004.

[93] D. Liu, F. Li, and S. Chen, "Towards optimal resource utilization in heterogeneous p2p streaming system," in *Proceedings of 29th International Conference of Distributed Computing Systems (ICDCS)*, Montreal, Quebec, Canada, June 2009.

[94] "MOSEK5.0," http://www.mosek.com.

[95] M. Luby and N. Nisan, "A parallel approximation algorithm for positive linear programming," in *Proceedings of ACM STOC*, May 1993, pp. 448–457.

[96] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," in *Proceedings of ACM SIGCOMM*, August 2008, pp. 375–388.

[97] A. Bharambe, S. Rao, V. Padmanabhan, S. Seshan, and H. Zhang, "The impact of heterogeneous bandwidth constraints on dht-based multicast protocols," in *Proceedings of IPTPS*, February 2005, pp. 121–127.

[98] E. Adar and B. Huberman, "Free riding on gnutella," in *First Monday*, vol. 5, October 2000.

[99] B. Cohen, "Incentive build robustness in bittorrent," in *Proceedings of Workshop on Economics of P2P Systems*, Berkeley, CA, USA, June 2003.

[100] C. Courcoubetis and R. Weber, "Incentive for large peer-to-peer systems," in *IEEE Journal on Selected Areas in Communications*, vol. 24, September 2006, pp. 1034–1050.

Curriculum Vitae

Dongyu Liu received his Bachelor of Engineering in Computer Science from China University of Geosciences, China in 1997. He received his Master of Engineering in Computer Science from China University of Geosciences, China in 2000 and received his Master of Science in Computational Science of George Mason University in 2003, respectively.