TRAFFIC OPTIMIZATION WITH REDUNDANCY ELIMINATION ON MOBILE AND WIRELESS NETWORKS

by

Shan-Hsiang Shen

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2015

Date of final oral examination: 12/18/14

The dissertation is approved by the following members of the Final Oral Committee:

Srinivasa A. Akella, Associate Professor, Computer Science
Suman Banerjee, Associate Professor, Computer Science
Shan Lu, Associate Professor, Computer Science
Thomas Ristenpart, Assistant Professor, Computer Science
Xinyu Zhang, Assistant Professor, Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to express my millions of thanks to my advisor, Prof. Aditya Akella. He gave me a lot of flexibility to work on the research I wanted to do, and patiently understood my half-baked ideas and gave me right directions. I have learnt a lot from him especially about how to think about research problem carefully. He is always full of passion for research that encourages me to keep moving on. In addition, I also appreciate his financial support during my study that released my stress and focused on research. He has done what the best advisor can do, and if time were rewound, I would still choose him as my advisor.

I would also like to thank Prof. Suman Banerjee. I worked on some interesting projects with him. He let me know the importance of deploying our research as a working system to benefit our society. In addition, I would also like to thank Prof. Shan Lu, Prof. Thomas Ristenpart, and Prof. Xinyu Zhang who gave me lots of valuable suggestion and comments on my thesis.

I would also like to thank Dr. James Kempf, Dr. Neda Beheshti, and Ramesh Mishra who are my mentors at Ericsson Lab where I spent one summer. I learned from them how to do good implementation in industry. I would also like to thank Dr. Jia Wang, Dr. Zihui Ge, Dr. He Yan, and Dr. Ajay Mahimkar at AT&T lab. They mentored me on some projects, and gave me many suggestions on my research.

I am so lucky to get the opportunity to work with the following graduate students; we discuss research ideas and complete great works together: Ashok Anand, Theophuilus Benson, Aaron Gember-Jacobson, Wenfei Wu, Robert Grandl, Junaid Khalid, Raajay Vishwanathan, Keqiang He, Liang Wang, Xiaoyang Gao, Anand Krishnamurthy, Li-Hsiang Kuo, Vivek Shrivastava, Shravan Rayanchu, Sayandeep Sen, Yadi Ma, Tan Zhang, Jongwon Yoon, Ashish Patro, Peng Liu, and Joshua Hare.

I would especially like to thank my friends from Taiwan: Cara Chen, Phoebe Yip, Yaya Huang, Kevin Ying-Tien Lin, Hsiang-Kuo Tang, Elise Wu, Yu-Ting Ho, Jie Yang, David Wang, Randy Wang, Ping Chen, Sarah Lin, Jiun-Yi Tsai, Hsun-Yu Chan, Pi-Yu Kao, Huan-Yang Chen, Chang-Mei Liu, See-Yeun Chen, Ya-Fang Cheng, Chi-Wei Tsang, Elu Tu, David Chiu, Hoilai Tseung, James Shih, Chia-Chen Yang, Eric Fu, Ting Ong, Wesley Lee, Szu-Yi Chen, Claire Pi, Sara Chen, Chien-Ming Huang, Shih-Yi Ho, Jeremy Liu, Janice Hwang, Amy Chen, Mike Hsieh, Tien-Yi Tsai, Sen Yan, Vicky Chen, Ed Huang, Yi-Fan Su, Yueh-Hsuan Chiang, Po-Hung Wu, Yu-Chi Lai, Shengnan Wang, and Yi-Cheng Wang. Thank all of you for spending wonderful time with my, and make Madison as my second hometown.

Finally, I would like to thank my father, Lie-Hang Shen, my mother, Hui-Rui Li, and my younger sister, Joyce Shen. Without your support, I cannot complete any works and get the degree. My father shares his research experience with me. My mother and my sister encourage and motivate me to keep my passion to pursue this path. I would also like to thank my other family members.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Appendix
Figure Page

ABSTRACT

Mobile video services dominate half of the cellular traffic and 40 % of the WiFi traffic today [2, 3]. However, spectrum resources in mobile and wireless networks are quite restricted. The Cisco Visual Networking Index [4] shows that the average mobile network downstream speed in 2013 was 1387 kbps, whereas the standard HD (720p) videos on YouTube are encoded as 5000 kbps [5]. Thus, to provide higher goodput performance and better quality of experience (QoE) for mobile video services, it is essential to use resources more efficiently.

Redundancy is prevalent in today's networks, and leads to waste of network capacity and degradation of network performance. The redundancy can be at the byte level-i.e., repeating bytes in flows or it can be at the information level-i.e., inherent meaning of objects for users such as the content of videos. In wireless channels, 60% of traffic has been detected as repeating bytes [6], and among popular video queries, 27% [7, 8] of videos are redundant. Thus, eliminating these redundancies should improve network performance.

Some existing solutions which improve goodput by removing traffic redundancy at the byte or information levels. Byte level solutions like EndRE [9] and SmartRE [10] provide fine-grained redundancy elimination (RE) to eliminate repeating bytes as small as 32-64B, but as they do not leverage the broadcast nature of wireless radios, they lose the chance to cache inter-client content. RTS-id [11], ExOR [12], COPE [13], and Ditto [14] leverage the broadcast nature, but they do not provide fine granularity. Thus, they lose some opportunity to match bytes inside a single packet.

At the information level, some solutions rely on proxies to cache popular videos in order to reduce repeating video objects transmitted over the networks. These proxies mostly identify videos

by the URL address rather than the video content. However, a recent study [7] shows that many videos with different URLs share the same content, so the redundant videos may be cached; the wasted storage space ultimately leads to a lower cache hit rate and longer video start-up latency. In addition, the proxies cannot adapt video format and bit rate dynamically based on device and channel diversity. The unsuitable information downgrades QoE. Other solutions provide dynamic video adapting such as MPEG-DASH, but they cannot support linear bit rate adapting and bandwidth will be underutilized, if the available bandwidth falls between two standard video bitrates. SoftCast [15] can adapt the bit rate linearly, but it requires physical layer modification, which makes deployment difficult.

In this thesis, we study how to leverage the unique characteristics of mobile/wireless networks and services to reduce redundancy and improve efficiency. To this end, we propose REfactor and iProxy.

REfactor eliminates the byte level redundancy at fine granularity and fully leverages the broadcast nature of wireless radios. To realize the benefits of IP-layer content overhearing, various challenges must be overcome that arise from the probabilistic nature of wireless reception (which could lead to inconsistent state) and the limited resources of wireless devices. We overcome these challenges through careful data structure (self-addressing cache) and wireless redundancy elimination designs (model driven RE). We evaluate the effectiveness of our system by conducting experimentation on real traces. We find that our design is highly effective and can improve goodput by nearly 25%.

REfactor can remove redundancy at the byte level, but it cannot perceive the object related information at the application layer. Thus, we provide another solution, iProxy, to remove redundant videos in cache storage and adapt bitrates for clients to improve QoE. iProxy acts as a middlebox between video providers and clients. It elevates the traditional view of caching from ”data” to ”information”, which optimally will reduce redundant videos in caches. It uses information bound references as video identification to collapse multiple related cache entries into a single one, improving hit rate and lowering the storage costs. Furthermore, it incorporates a novel dynamic linear

rate adaptation scheme to ensure high stream quality with client and channel diversity. Our evaluation shows that iProxy can improve cache hit rate by 65%, reduce start-up latency by 13 second, and improve video quality (average bit rate) by 16% compared to MPEG DASH.

Chapter 1

Introduction

Mobile and video traffic dominates the global internet traffic. According to the Cisco Visual Networking Index (VNI) [4], global mobile data traffic increased by 81 percent in 2013; mobile video traffic composes half of the cellular traffic [3] and 42% of the WiFi traffic in campus networks [2]. Moreove, 40% of YouTube traffic is requested by mobile devices [16]. The standard HD (720P) is encoded as 5000 Kbps bit rate [5], but the average mobile network downstream speed in 2013 was only 1387 kbps [4] Thus, with the growing traffic volume, it is becoming increasingly important to propose solutions to improve goodput and Quality of Experience (QoE) in mobile and wireless networks, and the restricted wireless spectrum resources should be used more efficiently.

1.1 Mobile and Wireless Networks Today

It is more difficult to maintain high goodput and QoE in mobile and wireless networks than in wired networks. First, the bandwidth of wireless links is more restricted than wired networks; while average speed in wired networks was 16 Mbps in 2013 [17], the average mobile network downstream speed was less than 1.4 Mbps in the same year [4]. Second, data is broadcasted into open wireless channels, so interference from different devices causes random loss that costs extra time for retransmission. Third, the condition of wireless channels is unstable and varies with time and location [18]. These transmission issues lead to decrease in goodput and user experience, mainly due to lower throughput and higher latency.

Limited CPU power and storage space restrict optimization for mobile devices. For example, most smartphones are equipped with memory space of less than 2 GB (DRAM). Thus, if we want

to eliminate redundancy in network traffic, a mobile device cannot cache much content to match the redundancy. Furthermore, the computation power is also slower than in wired devices. For example, one of the fastest mobile processors, Qualcomm Snapdragon 800, is around 6 times slower than Intel Core i7-3920XM in terms of GeekBench, which is a cross-platform benchmark [19]. It leads to difficulty in achieving line speed process when a device is doing complex encoding or decoding tasks. With these resource limitations, the solutions for mobile and wireless devices should be lightweight.

1.2 Improving Network Capacity

Aside from traffic optimization, there are still some options to improve goodput and QoE. First, we can increase network capacity throughout networks, and add more infrastructure such as APs or cellular towers. But this option is expensive and is not always helpful. For example, adding more APs may cause higher interference, which degrades performance. Another option is that cellular carriers offload some traffic to WiFi from cellular networks, which provides higher bandwidth capacity. Unfortunately, the infrastructure is not always available for offloading. Finally, we can pre-fetch popular content when network load is low and save transmission time from the original content providers during peak hours. However, this option does not optimize the last hop, i.e. the wireless link. Thus, it still leads to bad performance caused by unstable channel quality.

1.3 Improving Network Efficiency

Other solutions use limited network capacity more efficiently by optimizing network traffic. In mobile and wireless networks, severely constrained throughput and unstable channel quality are some of the common issues that make it difficult to maintain good mobile services. For example, during video streaming, if the required data is delayed, video frames will be lost or delayed. This results in poor quality and degraded video QoE.

Goodput is defined as the application level throughput, i.e., the number of useful information bytes received by a particular destination; the useful information bytes exclude retransmitted bytes,

and redundant bytes for protection. At a higher level, clients will be more willing to use the services that provide better QoE. Various factors determine QoE; in this thesis, the focus is mainly on video performance, including start-up latency, video stall, and video quality, as mentioned in some of the previous studies [20, 21].

To this end, the amount of actual bits that need to be transmitted should be reduced, yet the full information should still be delivered to destinations, or recovered. Without information loss, clients can receive high quality services.

1.4 Redundancy in Networks

The resources should be used more efficiently to achieve higher goodput and better QoE. However, todays networks are rendered less efficient due to redundancy.

Redundancy consumes network capacity and is observed at two levels: byte and information levels. Redundancy at the byte level occurs when data repeats in traffic; it can be detected by comparing traffic content byte by byte. [6] detects 60% redundant traffic in wireless channels. Redundancy can also happen at the information level.

Different objects may provide the same information from a client's point of view, but the files associated with these objects may not be exactly the same at byte level. We convert 100 videos into five different formats, which are .mp4, .flv, .mov, .mpg, and .avi; then we put all of them into an existing byte level RE encoder; only 1.06% of redundant bytes are detected. However, they include 80% redundant information. The videos encoded in different formats will look the same to the video viewers, if they include the same information. The two videos become redundant, if both of them are stored in a cache. It leads to a wastage of storage that drops the performance of a proxy system, and increases video start-up latency.

Unnecessary information is also redundancy. For example, the screen resolution of mobile devices is usually lower; if we send a higher resolution video, the mobile devices cannot play it in this high resolution. Based on our evaluations, if we send a video with resolution higher than device screen resolution, the start-up latency increases as high as 14 s. In addition, clients cannot play the video with wrong formats provided by video servers. Channel quality varies more frequently in

mobile and wireless networks than wired networks, so the amount of information that can fit the channel condition should be considered carefully.

1.5 Redundancy Elimination in Mobile and Wireless Networks

The redundancy should be removed for better network efficiency. Redundancy elimination (RE) is of greater benefit in wireless environments than in wired environments. Bytes are broadcasted to the channel, so any nearby devices can easily overhear them. If we can leverage this feature, more useful content can be cached and more future redundant bytes can be potentially detected. Furthermore, wireless channels are not reliable, so we should use the spectrum more carefully for higher goodput. Finally, shorter packets provide lower packet loss rate. A packet can be successfully received only if every bit of it is correctly decoded. Therefore, when the bit error rate is fixed, it is significantly beneficial to reduce packet size by removing redundant bytes.

1.6 Existing RE technologies

There are several previously existing RE technologies working at the byte level or the information level.

Existing Byte Level Approaches

To remove the redundant bytes, many previous studies worked on redundancy elimination (RE) in the network layer. To achieve better granularity, a number of existing approaches either cut packets into chunks for matching (i.e., EndRE [9]) or match shorter content first and then extend the matching length (i.e., SmartRE [10]). With better granularity, these approaches provide more opportunities to maximize matched traffic content. However, they are not designed for mobile and wireless environments, and do not take into consideration overhearing features, so they fail to maximize the advantage of caching more useful content and detecting inter-mobile device redundancy.

Some wireless approaches leverage overhearing and broadcast features. For example, RTS-id [11] reduces unnecessary transmissions between nodes, ExOR [12] chooses better forwarding relays, and COPE [13] combines content by network coding. These approaches match redundancy at the packet-level. Other recent studies also involve inter-flow overhearing, which overhear and cache content for different destinations. For example, Ditto [14] leverages overhearing at the data chunk level (chunks are 8 to 32 KB long) in multi-hop mesh networks. Ditto can take advantage of caching more useful content from multiple different flows.

However, existing mobile and wireless approaches cannot leverage the full benefits of content overhearing. They overhear and match redundant content in coarse granularity. For instance, Ditto overhears content based on the granularity of 8 - 32 KB, which is as long as several packets, so it cannot work on shorter flows. Previous studies [22, 23] show that short flows and flows with dynamic content dominate a significant fraction of the Web and enterprise flows. Another issue is that Ditto uses pull-based transport, which is limited to request-response applications. Finally, Ditto is only available for mesh networks and is difficult to apply to other scenarios such as the infrastructure mode. ExOR and COPE work in a finer granularity at packet level, but they are still not fine-grained enough to match redundancy inside a packet.

Existing Information Level Approaches

With byte level RE, we can remove some redundancy, but it still difficult to remove redundancy at higher levels. Since the application layer is closer to users and can more easily determine the inherent information of the content, some studies have also paid attention to the application layer and try to reduce redundant information in networks. A video proxy is a good example (MiddleMan [24]). According to the historical access records, a video proxy can cache popular videos and reduce the number of requests to remote content providers. However, proxies are not currently efficient at identifying content. Since they identify videos only based on their URLs, if one video is associated with several different URLs, they will be identified as having different content (different information) and all of them will be cached. A recent study [7, 8] shows that

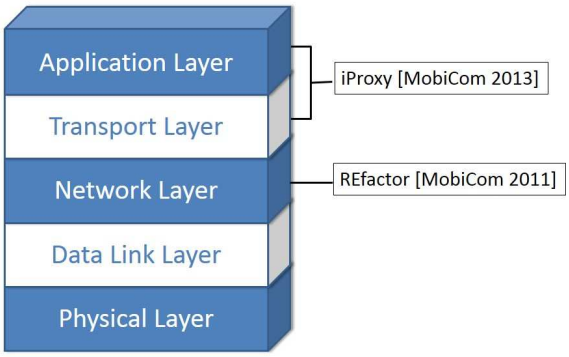

Figure 1.1 An overview of network stack

among 25 popular queries from YouTube, Google Video, and Yahoo! Video, 27% of the videos are redundant. As a result, cache space is wasted due to redundancy, and this wastage leads to lower cache hit rates.

In addition, it is also redundant to send more information than clients need. For example, clients cannot play videos with resolution higher than screen resolutions, or videos with wrong encoding formats. In addition, the information should fit network conditions. However, traditional proxies directly pass videos to end-users and do not take into account that wireless channels are unstable and user devices vary; end-users may experience lower quality (high stalling time and start up latency), because the redundant amount of information uses up network capacity.

Some existing solutions provide suitable information to clients. For example, MPEG DASH can adapt video bit rate according to the network and device condition. However, MPEG DASH only provides limited numbers of video versions, so if the available bandwidth is between two versions, the bandwidth will be underutilized. Another study called SoftCast implements a new linear video encoding scheme which provides linear bit rate adaption that fits any network conditions. However, SoftCast [15] needs to modify physical layers on both sender and receiver sides, which makes the deployment difficult.

To summarize, the existing content-aware approaches have some known issues. At the byte level, solutions are not fine-grained enough to detect redundant bytes in shorter flows or inside a

single packet. Other solutions consider fine-grained redundancy detection, but are not designed for mobile and wireless networks. At the information level, existing proxy solutions detect redundant information only by comparing the URLs associated with the content. It fails to detect some redundancy based on content information.

In this thesis, we propose novel RE technologies in different layers of a network stack, as shown in Figure 1.1, to further reduce redundancy by leveraging the unique characteristics of mobile/wireless networks. In the network layer, REfactor removes redundancy from traffic toward mobile devices at the byte level. REfactor focuses on traffic from APs to mobile devices. The iProxy system is located mainly in the application layer and also acquires information from the transport layer. It works as a novel video proxy between video providers and mobile devices, and improves the experience of video clients.

1.7 REfactor

REfactor focuses on the network layer RE and provides a fine-grained solution leveraging the wireless overhearing feature. It provides more opportunity to improve goodput and lower packet loss rate, and it results in better bandwidth usage and the improvement of service performance.

Challenges

To efficiently remove redundancy in mobile and wireless networks, the following are the challenges. Firstly, wireless overhearing is probabilistic, so a sender cannot know exactly which content a receiver has overheard. It means caches in senders and receivers are not guaranteed to fully synchronize. This affects the correctness and performance of RE. Estimation is needed to determine which redundant bytes are worth being removed. If this estimation is wrong, a recovery approach is desirable. Another challenge comes from the limitation in resources such as computation power and memory space in mobile devices. Thus, a lightweight solution is required for design and implementation.

REfactor Design

To overcome these challenges, we developed novel data structures. A simple approach is proposed to estimate reception probabilities used in a model-driven fashion; it calculates the expected benefit to remove a redundancy and the cost if the estimation is wrong. A recovery approach is designed to solve correctness issues. To provide a lightweight design for mobile devices, REfactor uses the concept of self-addressing packet chunks. A shorter and cheaper hash value is provided for each chunk; it is directly used to locate each chunk in cache memory. This way, no storage space is needed for an extra map pointing from hash values to cache locations (memory address). In addition, the hash is short and does not require any complicated calculation. Consequently, REfactor is easy to use.

Evaluation

We implemented REfactor in Click software router as modules and ran an encoder module on a desktop computer that sent encoded packets to a laptop computer running a decoder module through an AP. Our evaluation showed that REfactor can improve goodput in infrastructure by about 25%. REfactors goodput improvements are not just the result of removing repeated chunks from packets; its focus on packet chunks provides more opportunities for overhearing, and the smaller packets create a much lower probability of errors in the packets, resulting in 7-27% fewer packet losses. We found that, while blindly applying RE to all packets can result in a drop in goodput, model-driven RE is quite beneficial. The REfactor system can tolerate up to a 20% error in the estimation of the probability of overhearing. In addition, our self-addressing-chunks approach offers high-speed operation (e.g., up to 0.8 Gbps in software) while requiring modest-sized caches (64 - 256 MB) and ensuring effective duplicate removal (75% of optimal).

1.8 iProxy

With REfactor, we can remove most redundancy at the byte level. However, it only scans for byte by byte redundancy and cannot retrieve an overview of object information. Video content, for example, may be encoded into different resolutions or formats whose video files will be different, even though they include the same information. Redundancy is also caused by sending too much or unsuitable information.

Thus, we need an approach to detect and remove redundant information in a higher layer content. We provide a novel video proxy, called iProxy, with two key features: efficient caching and video adaption.

Caching on Content

In the application layer, to reduce network transmission delay and increase 4G core network capacity, iProxy acts as a middle box or proxy between clients and video providers, and it caches frequently accessed videos. To efficiently use storage in the proxy and provide smooth videos, iProxy presents a novel system for video streaming. Instead of using a URL, iProxy looks into the content and retrieves the information to identify the videos. Information-bound references (IBR) are borrowed to do this. IBR is the frequency domain data transferred from the original video data and is linear to the scene that the human eye sees; as such, similar IBR values imply that the videos also look similar. Thus, an IBR value is a better tool than a URL for identifying videos. When iProxy receives a video, it calculates the videos IBR value and compares the IBR value to others in the cache. If the IBR value matches, it recognizes that this video has already been cached in an iProxy, so that only one of the versions is kept in storage. In this way, limited storage space can be used much more efficiently, and more requests can be hit to satisfy more end users.

Dynamic Bit Rate Adaption

IBR can identify videos even with different resolutions, formats, or bitrates. However, if iProxy only caches one version of a video, it might not satisfy all end users, since they might use different devices supporting different versions of the video. To solve this issue, iProxy includes a dynamic video encoder that can convert videos to the formats that satisfy end-users requests. With a suitable video format, we can reduce video startup latency by reducing unnecessary video pre-processing time. In addition, iProxy can trace available network bandwidth to the end users and dynamically adjust the bitrate of video streaming. Thus, end users have a better experience, with lower stalling time and smoother videos. To enhance the performance of dynamic video encoding, iProxy caches the frequency domain data instead of the original data. When an end user requests a video with a particular format, iProxy can just encode the video with frequency domain data instead of re-encoding the video from its original format. Thus, when a new video comes in, iProxy first converts it to frequency domain data; this can be used not only to retrieve the IBR value, but also to be cached for future requests.

Evaluation

We implement the video adapter by modifying FFmpeg, which is a popular open-source video encoder. iProxy is used in a desktop computer and streams videos to clients through cellular networks (AT&T). At the client side, we use a commercial video player called VPlayer, which is installed in an Android phone. According to the evaluation, iProxy improves cache hit rate by 65% with the same storage space. In addition, iProxy reduces start-up latency by 13 seconds compared to fixed-rate video stream, and improves video quality (average bit rate) by 16% compared to MPEG DASH. Thus, iProxy can use storage space more efficiently and satisfy more requests than a traditional proxy. Moreover, iProxy improves the QoE of watching videos.

1.9 Outline

The rest of this thesis is organized as follows. In Chapter 2, we introduce the solution for byte level RE named REfactor, and our information level RE solution is presented in Chapter 3. Chapter

4 introduce some related works, and compare them with our solutions. We conclude and discuss future direction in Chapter 5.

Chapter 2

REfactor-ing Content Overhearing to Improve Wireless Performance

REfactor is designed to remove byte level redundancy and improve channel resource usage. The bandwidth in wireless environment is more limited than wired networks. Besides, interference make random error and retransmission more serious. Thus, reducing redundant traffic become more critical.

Wireless nodes can overhear each others' traffic with the same channel. We can use the overhear natural to benefit wireless transmission. In this work, we present a wireless RE called REfactor, which can cache more useful content and remove more redundancy content by overhearing others' traffic.

2.1 Background and Motivation

Traditional overhearing-based approaches to improve wireless network capacity and throughput have relied on packets being overheard in full. For example, RTS-id [11] adds a special ID field to an RTS packet to allow receivers to determine if they recently overheard a packet, thereby avoiding transmission of the packet. In contrast, some recent approaches have argued that shifting the focus from packets to *content* can result in substantial throughput and capacity improvements. Ditto [14] was the first system based on this notion of *content overhearing* (as opposed to the conventional packet overhearing ideas). Ditto functions on named data chunks that are independent of packets. Wireless mesh routers in Ditto cache directly received chunks and chunks reconstructed from overheard packets. When a client requests a particular chunk, Ditto attempts to serve the

request from a upstream wireless mesh router, avoiding the need to transfer a chunk all the way from the mesh gateway to the client.

Limitations of the State-of-the-art

In what follows, we argue that Ditto's approach does not leverage all redundancy opportunities, and its narrow focus limits its applicability to a variety of practical scenarios.

Limitations due to large chunks. Ditto names data chunks of size 8KB or larger. This leads to two problems: First, Ditto fails to identify finer-granularity content overlap across network flows. In fact, recent studies have shown that a major portion of redundancy in Internet traffic arises from overlapping chunks as small as 64B in size [22]. Second, many nodes may not overhear a large chunk in full and may fail to reconstruct it. Indeed, experiments using Ditto show that, on average, 75% of the potential locations for overhearing in a campus testbed could not completely overhear, and failed to reconstruct, almost 50% of chunks [14]. A recent RE framework for cellular networks operates on chunks as small as 8 Bytes, enabling redundancy removal within a client's traffic at fine granularities [6]. However, applying this system to other wireless scenarios (such as those in Section 2.1) misses out on overhearing opportunities to remove redundancy between clients.

Limitations due to pull-based transport. Ditto's reliance on named data chunks, each of which spans several packets, forces it to use an alternate transport protocol instead of using TCP end-to-end. Specifically, Ditto uses a pull-based transport protocol, DOT [25], where remote servers send chunk IDs to clients, who then request them one after the other; requests may be opportunistically served by a local cache. This leads to several problems:

First, it requires chunk identifiers to be known beforehand. This works for static content but not for dynamically generated content; clients are forced to use default transport designs for dynamic content, removing the opportunity for performance improvements. Second, applications with with short messages—e.g., gaming flows, twitter feeds, several request-response applications, short HTTP flows, etc.—may actually observe a degradation in performance in the average case (because the pull-based approach invariably adds additional RTTs). Lastly, no performance benefits are

offered on last hop wireless links. Chunks must be transferred in full across the last link from mesh router to client.

Another set of popular approaches for improving wireless capacity are those based on network coding [26, 13]. As the authors of Ditto mention, it may be possible to use opportunistic content-overhearing to augment coding and improve its overall effectiveness. However, given the mismatch in the granularities and transport models used in Ditto and prior coding approaches, it is unclear if the synergy between overhearing and coding can be exploited.

REfactor

REfactor shows that a careful re-factoring of content overhearing can address the problems above optimally and dramatically improve wireless capacity and performance. We argue for pushing content-awareness "lower down the stack" through the use of *IP-layer packet caches that perform redundancy elimination* (RE) [27]. Packet caches can be used to suppress byte strings that have appeared in earlier overheard packets both within and between clients. We refer to our approach as REfactor. The cleaner re-factoring in REfactor offers many benefits:

- IP layer RE can remove duplicates as small as 64 Bytes in an application-agnostic fashion, even from dynamically generated content. REfactor benefits applications with short flows—even those lasting a single packet—which are common in enterprise settings [23]. Thus REfactor leads to more effective overhearing-based designs.
- REfactor requires small IP-layer modifications and retains the conventional push based model of content dissemination that is prevalent today.
- Because REfactor leverages all possible opportunities for overhearing, it's performance is reasonably robust to errors in some aspects of the design (in particular, reception probability estimation, which is a notoriously hard problem). Thus, it is easy to use in practice.
- REfactor leads to smaller packets which consume less bandwidth and suffer lower loss rates. Operating on packets also allows REfactor to run at very high speeds: As we show in Section 2.3, our prototype offers 0.6-0.9Gbps.

- REfactor can be applied transparently in a variety of scenarios, including wireless infrastructure-based and peer-to-peer communications, and opportunistic routing in multi-hop mesh networks. In particular, REfactor is more directly aligned with packet-based coding approaches. Hence it provides practically viable opportunities to enhance network coding to further improve network capacity.

REfactor Overview

We start with a common scenario where REfactor can be used: an AP operating in infrastructure mode with some number of associated clients. The AP and clients can overhear and cache packets. When the AP receives a packet from the wired network, it scans the content for duplicate strings of bytes that appeared in earlier packets. The AP then calculates the expected benefit for the receiving client from performing RE on the packet, which depends on the AP's estimate of whether the receiving client is likely to have cached the relevant earlier packets, either from transmissions to the client or from overheard transmissions to some other clients. If the likely benefit is high, the AP "encodes" this packet, i.e., removes the duplicate bytes and inserts a shim instead. The shim contains a pointer to a memory location in the client and allows the client to reconstruct the original packet using its local cache. Thus, if the receiving client has the content pointed to by the shim, then it can decode the packet. However, if the content is not cached in the client, the client needs to request the missing content from the AP, incurring additional transmissions. This penalty is imposed when the AP's estimate of whether the client has the content is incorrect.

In Figure 2.1(a), we illustrate the benefit REfactor offers in this scenario. The transmission of the packet payload `abc` to *C1* is overheard by *C2*, and the chunk `ab` is added to all caches. The data `abd` is sent to *C2* via a packet with a shim header "`1`" plus the non-redundant data `d`. Because *C2* overheard and cached the chunk `ab`, it as able to reconstruct the full packet using the cache entry specific in the shim header. The reduction in the size of the second packet transmission improves overall network throughput.

REfactor in Other Scenarios

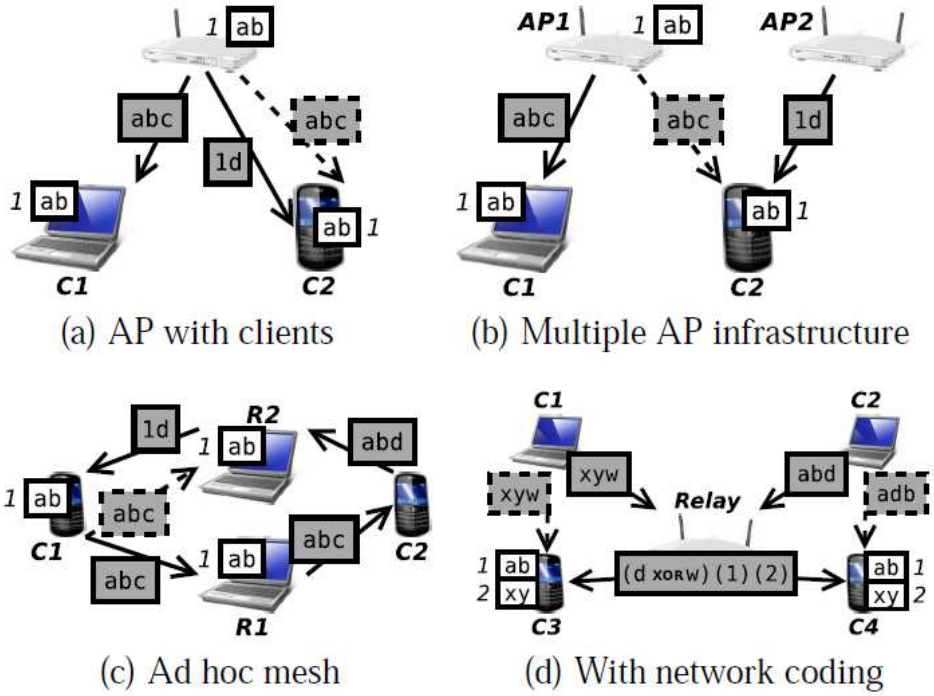

Figure 2.1 REfactor applied to diverse scenarios

REfactor also helps in other diverse scenarios.

Multiple AP infrastructure. We start with multiple APs operating in infrastructure mode. As shown in Figure 2.1(b), a client may be able to overhear transmissions from both it's associated AP and other nearby APs. An AP can remove redundancy based on any chunks a client may have overheard, regardless of which AP they were overheard from. *C2*, which is associated with *AP2*, overhears *AP1*'s transmission of `abc` to *C1*. *AP2* can therefore remove redundancy from its transmission of `abd` to *C2*.

Ad hoc meshes. REfactor can also be applied to transmissions between clients via a mesh or ad hoc network. Figure 2.1(c) shows the use of REfactor to achieve transmission reduction and, correspondingly, capacity improvement, in a small mesh network; this can be easily extrapolated to a larger mesh. Using normal forwarding, based on metrics such as ETX [28], 4 transmissions are required for two clients to send a packet to each other via a relay. By applying REfactor to the situation, we can reduce the size of the fourth transmission, resulting in $4 - \delta$ transmissions,

where δ is proportional to the amount of redundancy removed: *C1*'s transmission of `abc` to *R1* is overheard by *R2*, which caches `ab`. *C2* transmits `abd` to *R2*, followed by *R1* transmitting `abc` to *C2*. Lastly, *R2* removes the redundancy from `abd`, sending `1d` to *C1*, since it knows *C1*'s cache contains `ab`.

Opportunistic routing in multi-hop meshes. In a similar fashion, REfactor can also be applied to opportunistic routing schemes in mesh networks (not shown in Figure). In approaches such as ExOR [12], the transmitter orders relays on the basis of their packet overhearing probability, before sending a batch of packets. Using REfactor, ExOR can be modified in two ways: First, the effective batch size can be reduced by removing strings that are duplicated either within the batch, or across prior batches sent by the transmitter. Second, the ordering of relays could take into account whether or not a relay has portions of content in the batch already cached; a relay with high overhearing probability could be given a high priority for forwarding if it has a significant fraction of bytes in the batch cached as it could prove invaluable in speeding completion time of the batch.

Networking coding. Network coding systems, such as COPE [13] have traditionally relied on coding full packets without paying attention to packet contents. REfactor can be combined with network coding to leverage duplication in packet payloads to help coding improve network capacity even further. We present the combination REfactor + COPE in Figure 2.1(d). In this scenario, *C1* has a packet destined for *C4* and *C2* has a packet destined for *C3*, both of which must be sent via the relay. COPE imposes only 3 packet transmissions compared to 4 in the regular case, as *C3* can overhear *C1*'s transmission and *C4* can overhear *C2*'s transmission, providing a coding opportunity. REfactor + COPE leverages this overhearing even further by removing chunks known to exist in the destination client's caches: Assuming *C3* overheard an earlier transmission `abc` and *C4* overheard `xyz`, the relay can remove the redundancy (`ab` and `xy`) from the current packets and code the remainder of the current packets, `d` $\oplus$ `w`. The coded packet, plus small shims to "encode" the removed redundancy, is broadcast to *C3* and *C4* simultaneously. COPE, in contrast would broadcast the much larger `abd` $\oplus$ `xyw`. *C3* and *C4* are able to obtain their packets by reversing the network coding and filling in removed chunks from their caches. Thus, REfactor + COPE reduces

the number of transmissions to $3 - \delta$, where δ is the relative difference in the size of a full unencoded packet (e.g., `abd`) and the above coded packet along with the shims. Assuming chunks are all the same size, $\delta \lesssim \frac{2}{3}$ in the example above, resulting in nearly $\frac{2}{9}$ better capacity than COPE.

Design Challenges

Although REfactor can offer substantial advantages as the above examples show, a careful design is needed to realize the benefits in practice. First, since overhearing is probabilistic in nature and caches are fixed size (hence, old content is evicted over time), a sender may not have an accurate view of whether the intended receiver has a certain content chunk already cached. In turn, this could lead to incorrect encodings and the resulting retransmissions negate the benefit of duplicate suppression in REfactor. Enforcing explicit synchronization of caches—which is a candidate solution for this problem—can add excessive overhead. Second, wireless nodes may be processing and memory constrained (for example, the clients in the above scenario could be smartphones), so REfactor mechanisms should require minimal resources from them. Designing REfactor to maximally leverage IP-Layer content overhearing while accounting for the above issues is challenging.

2.2 REfactor Design

In this section, we describe the design of REfactor. For simplicity, we focus on the setting outlined in Section 2.1, namely, optimizing the downlink traffic performance of a wireless AP with a collection of clients. However, our basic building blocks, along with a few extensions described at the end of this section, apply to other scenarios as well.

REfactor applied to the single AP scenario involves the following steps: (i) When the AP receives a packet from the wired backend, we "chunk" it and compute a "fingerprint" per chunk. (ii) For each chunk, the corresponding fingerprint is used to refer to a content cache data structure that helps determine the probability of the intended receiver having cached the chunk. (iii) The AP computes the expected throughput benefit from removing chunks in the packet. If this exceeds a certain threshold, the AP removes the chunks and replaces them with the fingerprints instead. (iv)

If the AP observes a hash collision for a chunk, it does not encode the packet, and it invalidates the chunk stored in its content cache for the collided hash. (v) If a client is unable to decode a packet using a fingerprint supplied by the AP, it requests a chunk retransmission from the AP. (vi) The AP updates the cache-residence probabilities associated with each chunk of the packet. We describe each of these steps and the underlying design issues next.

Chunking

Prior works have considered several different approaches for removing packet-level redundancy, which trade-off memory usage, processing time, and redundancy opportunities. The earliest, by Santos and Wetherall [29], supports redundancy elimination (RE) at the full packet level. While simple, this approach severely limits RE opportunities, because lots of redundant bytes happen inside a packet. Support for partially redundant payloads is provided by two different classes of approaches: Max-Match and Chunk-Match [9].

In Max-Match, the encoder computes a rolling Rabin-Karp hash [30] for each 32 Bytes region of a packet and selects a subset of these, based on hash values, to serve as packet fingerprints. The fingerprints are stored in a hash table, with each fingerprint pointing to the corresponding packet, which is stored in a packet cache in FIFO fashion. Fingerprints computed for an incoming packet are checked against the fingerprint table; a matching packet, if found, is retrieved and compared byte-by-byte around the 32 Bytes match region to identify the region of maximum overlap. The overlap region is removed and replaced with a shim, which carries the memory offset of the packet in the downstream decoder's FIFO-ordered packet cache from which the missing bytes can be constructed. The downstream cache is maintained in a similar fashion.

Chunk-Match computes and selects Rabin hashes in a similar fashion, but the chosen 32 Bytes regions form the boundaries of the chunks into which the packet is divided. A SHA-1 hash, which forms a fingerprint, is computed for each chunk and inserted into a chunk hash table. Each unique chunk is cached in FIFO order. When an incoming packet has a chunk matching against the chunk hash table, the matched region is replaced with the chunk hash.

In both cases, the MAXP algorithm[1] has been found to be effective at selecting hashes offering a uniform distribution across a wide variety of packet payloads [27, 22]. We employ this in our design.

Chunk-Match's focus on chunks means it is less effective at identifying redundancy than Max-Match. As a result, packet-level RE systems have traditionally preferred Max-Match [9, 27, 31]. But we choose Chunk-Match in designing REfactor due to the following benefits:

1. *Effective memory usage:* A specific chunk only needs to be stored once, while Max-Match's packet-based approach requires storing full packet payloads, even if part of the payload already exists in another payload. [2]
2. *Better at accommodating overhearing:* As we argue below, Chunk-Match can be used to design simple techniques to handle wireless overhearing, without requiring complex operations at clients or APs or imposing too much overhead. In contrast, Max-Match requires clients to employ additional data structures and meta data to track overheard content, which imposes additional memory and computing overhead.
3. *More overhearing opportunities* Overheard packets that have duplicate bytes suppressed can be more effectively leveraged in Chunk-Match, because Chunk-Match can cache whatever chunks remain in those packets. In contrast, Max-Match would discard such packets because it needs full payloads.
4. *Ability to leverage partial packets:* Although not discussed in this paper, Chunk-Match can be effectively combined with partial packet recovery schemes [32] to further leverage partially overheard payloads. It is difficult to do so with Max-Match.

Chunk-Match still has key limitations due to which it cannot be applied directly in REfactor. First, the large size of the SHA-1 hash means the effectiveness of redundancy removal is limited. Also, the decoder (i.e., the client) has to compute and store SHA-1 hashes for cached chunks, which is expensive from both an energy and memory view-points. As an improvement, the encoder can only maintain the chunk hashes in a hash table, while the decoder maintains only the

[1] MAXP selects hashes that are the maximum over all hashes computed over a p-byte region.

[2] The optimizations to Max-Match suggested by EndRE [9] to address high resource costs are not feasible for REfactor.

chunks in a FIFO cache: the encoder looks up chunk hashes for a match and replaces each match with a memory address in the decoder's FIFO cache [9]. However, this approach is only suitable for point-to-point deterministic RE, e.g., across a wired link. In wireless, probabilistic overhearing causes the encoder to lose track of the decoder's chunk cache. One alternative, of having the AP compute chunk hashes and transmit them along with packets and maintaining a chunk hash table at the client impose high compute, network and memory overhead, as we show in Section 2.5.

Our Modifications

We modify and extend prior Chunk-Match designs in important ways to to address these drawbacks and to better tailor Chunk-Match to wireless overhearing.

Self-addressing chunks. In REfactor, we need to carefully manage the location of chunks within AP and client caches. Chunks must be cached such that the AP can provide a fingerprint in place of a redundant chunk that allows a client to locate a chunk within its cache, or identify a cache miss. As shown in Figure 2.2, *C2* only overhears the second packet transmitted to *C1*. A FIFO cache would be insufficient because the AP and *C1* would store the redundant chunk in the second cache slot, while *C2* would store the chunk in the first slot because it is the first chunk *C2* overheard. A proposed RE system for cellular networks suffers from a similar issue in the presence of packet loss [6].

The key innovation in our approach is that we select a slot in the cache (encoding or decoding) based on the content of a particular chunk(Figure 2.2). Thus, a chunk is *self-addressing*, i.e., the chunk itself identifies its location in the cache, and a removed chunk can be identified by the cache location. In particular, we use a compact n-bit hash (we use $n = 20$) of a chunk as the memory address where it is stored. The encoder simply sends the n-bit hash instead of the chunk to the downstream decoder.

This approach avoids the pitfalls of employing FIFO-based caches without relying on tightly synchronized caches; namely, an offset into the cache refers to the same chunk regardless of what other chunks a client may have overheard. It avoids the high computational cost of SHA-1 hashes

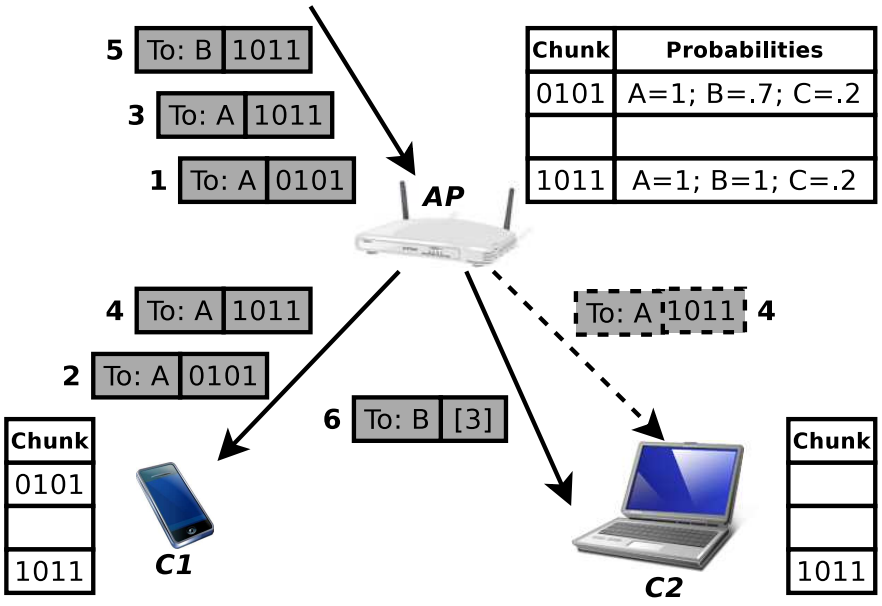

Figure 2.2 REfactor in practice. Solid lines indicate normal packet delivery and dashed lines indicate overheard packets. Transmissions are numbered in order from 1 to 6. AP and client cache contents after all transmissions are shown.

and the overhead of maintaining the corresponding metadata. Furthermore, it is easy to identify cache misses: a lookup at a fixed-offset given by a chunk-hash can be used to determine whether or not a chunk is located in the cache.

The size of the chunk-hash represents a trade-off between the amount of memory required and the potential for redundancy. An n-bit hash allows a cache to store 2^n chunks but requires $2^n * m$ bytes of memory, where m is the maximum chunk size (based on parameters of the MAXP algorithm). A larger hash allows more unique chunks to be stored but requires more memory in clients which may already be resource poor, e.g., smartphones. The size of the chunk-hash and the method used to compute it also impacts the likelihood of collisions. We evaluate the trade-offs in hash-size in Section 2.3 and discuss how to deal with collisions in Section 2.2.

Overhearing estimation. Our second modification accounts for the facts that different clients overhear packets with different probabilities and that overhearing probabilities change over time. In our example (Figure 2.2), it is possible that *C3* (not shown) may not overhear either transmission to *C1* because *C3* is farther away and can only receive transmissions at a lower rate. However, *C2*

is able to hear half of the transmissions to *C1*. If we remove the chunks that have not been overheard, it costs some extra transmission to recover the missing chunks. To maximize the removal opportunities and reduce the cost for missing chunk recovery we want to be able to estimate how likely it is a particular client has a particular chunk. We add a *reception probability vector* to each chunk entry at the AP to aid the decision of whether or not to remove redundancy. Section 2.2 discusses how this vector is computed/updated and how it is used to guide the decision of whether or not to remove redundancy.

Handling cache misses. Lastly, we need to account for the fact that the AP's estimation of the contents of a client's cache may not be completely correct. In our example, we remove the redundancy from the transmission to *C2*, but it is possible *C2*'s cache may not have contained the removed chunk. We address this *cache miss* by providing a content chunk request mechanism. Since the shim header contains all the necessary information to identify a specific block of redundant content, the client uses the shim header from the original packet in its request for missing data. The AP replies to the missing content request with the shim header and the content of the chunk, allowing the client to properly reconstruct the packet and pass the packet to the network stack for normal processing. As a result of this recovery mechanism, our Chunk-Match approach requires the AP to store the contents of chunks in its cache, contrary to what recent optimizations suggest [9].

Removing Redundancy

In prior RE systems, a redundant chunk is either always removed [31] or a removal decision is based on network-wide optimization [10]. Furthermore, these systems assume packet caches at the sender and receiver are tightly synchronized, so a chunk present in the sender's cache is guaranteed to be in the receiver's cache. This synchrony assumption does not hold in REfactor because of opportunistic overhearing. Potential differences in the contents of sender and receiver caches requires REfactor to make a removal decision based on estimates of the receiver's cache contents.

A naive approach that always removes redundancy imposes high cost: Every chunk missing from a client's cache requires two additional packet transmissions to receive the missing data from the AP. Figure 2.3 shows the expected transmission time savings from removing a single 64 Bytes chunk from a packet (details in Section 2.2). There is no expected benefit when the probability of the receiver's cache containing the chunk is $< 90\%$, due to the high cost of cache misses. Therefore, we want to minimize cache misses by making a wise decision on whether or not to remove a redundant chunk.

Reception Probability Vectors

Our insight is to include a *reception probability vector* with each chunk stored in the AP's cache. A vector, V, contains an entry for each client currently connected to the AP, indicating the likelihood of a client having the chunk in its cache.

We know a client's cache will be guaranteed to contain a chunk if it existed within packets successfully sent to that client in the past. Hence, for the destination client d, we set $V_d = 1$ after the packet containing the chunk has been ACKed. If the chunk was removed from a packet sent to the client, we set $V_d = 1$ either: (*a*) after a request for the chunk has been received (Section 2.2) and the reply to the client has been ACKed, indicating the client has now cached the chunk it was missing; or (*b*) a few seconds after the original packet was ACKed, indicating the client already had the chunk because no request for a missing chunk was received.

All other clients could only have received a chunk via overhearing. We show in Section 2.5 that a highly accurate reception probability estimate is not necessary to realize the benefits of REfactor and wireless resources are limited to do extra complex probing , so we take a low-overhead approach to estimating overhearing likelihood: reception probability is based on the rate r_d used to communicate with the destination client d and the rate r_i the AP uses to communicate with the overhearing client i. Measurements by Afanasyev et al. on an indoor 802.11g testbed [1] show that the chance of overhearing is relatively consistent for all 802.11b rates (1-11Mbps) and the five lowest 802.11g rates (6-24Mbps); noticeable differences in overhearing probability only exist for

Rate	Fraction of nodes who overhear
6-24Mbps	0.15
36Mbps	0.12
48Mbps	0.08
54Mbps	0.06

Table 2.1 Median fraction of nodes in the Jigsaw testbed who overhear transmissions at various 802.11g rates [1]

the three highest 802.11g rates (36, 48, and 54Mbps). The median fraction of nodes expected to overhear a transmission at a given rate is shown in Table 2.1.

Based on these findings, reception probabilities for a given wireless deployment can be calculated using a simple heuristic formula: If $r_i \geq r_d$, $V_i = 0.99$. A client which normally receives transmissions at a higher rate is very likely to receive transmissions at a lower rate, but we still want to be able to discern between clients which are guaranteed to have the chunk($V_d = 1$) and clients which are highly likely to overhear the chunk($V_i = 0.99$). If $r_i < r_d$, $V_i = \frac{e_d}{e_i}$ based on conditional probability, where e_j is the recipient fraction when sending at rate r_j, such as those shown above in Table 2.1.

More complex mechanisms, e.g., CHARM [33], may be able to provide better estimates of reception probability. In general, accurate estimation is hard and expensive, in part because overhearing probabilities can change at fine timescales [34]. However, as shown in Section 2.3, highly accurate predictions are unnecessary. In particular, we find that *directly using* the measurements in Table 2.1 may be good enough and estimating reception fractions for each deployment may not be needed. This is a highly desirable property of REfactor.

Reception probability vectors for chunks are updated every time the chunk is transmitted. For each client we store the maximum of an existing probability and the probability for the current transmission. When new clients join the network, reception probabilities are recorded for the clients for any chunks transmitted *after* they connect. When clients leave the network, reception

probabilities for the clients are not stored for any newly transmitted packet chunks, and probabilities for the clients are invalidated in existing vectors.

Deciding to Remove: Model-Driven RE

REfactor decides whether or not to remove a redundant chunk based on the reception probability and a simple model of expected benefits. Benefit is measured as the reduction in transmission time resulting from the removal of a redundant chunk. We refer to this approach as *model-driven* RE.

The transmission time for a packet is a combination of wireless header transmission time t_h and per-byte payload transmission time t_b, which depends on the data rate to the client in question. Our experiments show a typical value of $t_h = 290\mu s$ for an indoor setting with a client and AP separated by 2m, and $t_b = 0.885\mu s$ for a transmission rate of 11Mbps. For simplicity, we assume all packets are MTU (1500B) in size, making the total transmission time for a normal packet $t_h + 1500t_b$. Removing a k byte chunk of redundant content from a packet and replacing it with a h byte header makes the transmission time $t_h + (1500 - k + h)t_b$, a savings in air time of $(k - h)t_b$. If the load due to other nearby APs is ρ, then only $(1 - \rho)(k - h)t_b$ of the savings can be used toward improving the throughput of the current AP's own transmissions.

A removed chunk which does not exist in a client's cache requires two extra packet transmissions to obtain the missing chunk. The additional transmission time is $2t_h + (2h + k)t_b$, reducing the savings by this amount. Recall that V_d is the probability client d's cache contains the chunk. The expected benefit of removing a chunk in terms of free airtime (in μs) that could be used toward additional transmissions of the AP is:

$$Exp[B] = V_d(1 - \rho)(k - h)t_b - (1 - V_d)(2t_h + (2h + k)t_b)$$

This equation is a worst case estimate of expected benefit. In practice, the fixed header transmission time $2t_h$ associated with obtaining missing chunks only needs to be incurred once for each packet with ≥ 1 missing chunks. Applying the equation to multiple chunks in a packet will take into

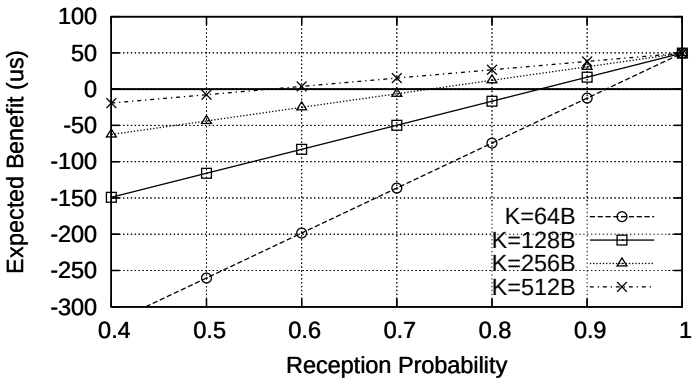

Figure 2.3 Expected benefit from removing a single 64B chunk from a packet with K total redundancy

account the fixed header transmission time for missing chunks multiple times. We adjust $Exp[B]$ by taking into account the total number of redundant bytes, $K = \sum k$, in a packet, setting the fixed header transmission time for obtaining missing chunks to $\frac{2k}{K}t_h$ for each k byte chunk. This change allows REfactor to be more optimistic in removing redundancy.

Figure 2.3 shows the expected benefit from removing a single 64 Bytes chunk from a packet with K total redundancies for varying reception probabilities, assuming $\rho = 0$. As the graph shows, expected benefits increase with reception probability. Furthermore, for a given reception probability, higher amounts of total redundancy K increase the expected benefit from removing a single k-byte (in this case 64 Bytes) chunk. Similar graphs can be plotted for other rates.

The AP uses the expected benefit model to encode redundant chunks if $Exp[B]$ exceeds some threshold.

Collisions

We say that a hash collision happens when an n-bit hash of a chunk for a new packet indexes to an already occupied cache slot. Collisions should be handled carefully as they impact transmission correctness. In REfactor, the AP checks the new and already cached chunks for collisions by performing a byte-by-byte comparison of their contents. If they do not match exactly, the cache

entry is marked as a collision. No chunks which hash to a collided entry are ever removed from a packet by the AP. All clients will also be able to detect the collision because they will never receive a packet with a collided chunk removed, so they will recognize the collision in their byte-by-byte comparison. This approach reduces the potential redundancy removal opportunities, but it ensures no client application will receive an incorrectly reconstructed packet.

To avoid the entire cache filling with collision entries, the wireless AP will periodically initiate a *cache flush*. A cache flush clears all entries from the AP's and client's caches using a three phase process: (1) the AP broadcasts a cache flush request to all clients, (2) the clients clear all the entries in their cache and send an ACK, (3) when the AP has received ACKs from most of the clients, it clears its cache. The AP does not cache and chunks from new packets while a cache flush is in progress. In the event a client does not acknowledge the flush request, due to lost packets or client disconnect, the AP will not encode any packets sent to the client until a retransmitted flush request has been acknowledge.

Whenever a client associates with the network, the client empties its local cache. An AP uses the association as a signal to clear old reception probability entries for the client.

Other Scenarios and Issues

REfactor uses the same caching and model-driven RE mechanisms to improve throughput in the scenarios presented in Section 2.1 but requires a few design extensions to fully function in these scenarios. Namely, the ability to estimate reception probabilities and communicate cache contents for *unassociated clients*, i.e. clients not directly communicating with an AP or via a specific relay.

Unassociated client reception. Clients may be able to overhear transmissions from other APs (as in Figure 2.1(b)) or nearby mesh nodes, or relays (as in Figure 2.1(c)). However, the relay has no way of knowing the client can overhear without explicit knowledge of the client's presence. Furthermore, the relay cannot estimate the reception probabilities for the client without knowing the rate the relay would use to communicate with the client.

We extend clients to notify a relay when they can overhear traffic from that relay. In infrastructure mode, a client can determine the AP it can overhear from (*AP1*) based on its beacons and send a message via its associated AP (*AP2*) to notify *AP1* its transmissions can be overheard by the client. *AP2* includes the rate it uses to communicate with the client, which provides an upper bound on the rate *AP1* would be able to use to communicate with the client. In a mesh network, a client can send a list of all relays it can overhear from to each of the relays in the list.

Overhearing notifications only need to be sent periodically. A relay will maintain reception probability vector entries for unassociated clients for all chunks sent after an overhearing notification is received.

Unassociated client caches. Knowing a client can overhear transmissions from another AP (*AP1*) is insufficient for the client's associated AP (*AP2*) to be able to leverage overhearing opportunities. Periodically, *AP2* must request cache information for the client from *AP1*. *AP1* sends a bit vector to indicate which cache slots the client likely overheard. *AP2* can update its reception probability vectors for these slots to account for chunks it may not have known the client overheard. As shown in Figure 2.1(b), *C2* overheard the chunk `ab`, allowing *AP2* to potentially remove the chunk from a future transmission to *C2*. *AP2* uses the bit vector from *AP1* to update its cache to reflect this.

In a mesh network, cache contents can be updated using the same mechanism, or a relay can update its cache based on its knowledge of the path taken by a packet. Consider the example scenario shown in Figure 2.1(c): if `abc` was received by *R2* from *R1* and *R2* knows *C1* can overhear *R1*, *R2* can add the chunk `ab` to its cache and indicate with high likelihood that *C1* overheard the chunk. In this manner, cache contents are communicated implicitly. A similar idea applies to the network coding approach shown in Figure 2.1(d).

2.3 Network Coding and Subcarriers

In this section, we propose an REfactor enhancement with network coding and subcarriers supports. The key issue of this idea is that we want to consider in wireless transmission are how to use subcarriers more efficiently and how to further reduce unnecessary transmission. In addition, this enhancement improves REfactor to reduce inter-client redundancy even if the content is not

overheard.

Subcarriers

Orthogonal frequency-division multiplexing (OFDM) divides channels into subcarriers with narrower bandwidth, which are used to carry data in parallel. Each sub-carrier works as a traditional modulation scheme. The key advantage of OFDM is that it can work better when the channel condition is severe.

As shown in [35], frequency diversity is significant in wireless channels. Thus, a subcarrier connecting to different receivers may have different signal quality; different bytes also have different features and importance for receivers. Thus, it is critical to interleave bytes with different requirements into suitable subcarriers.

Consider a simple scenario. There is an AP that transmits packets to two clients: $ClientA$ and $ClientB$. The AP has multiple strategies to interleave bytes through different subcarriers. According to their channel quality, we can classify subcarriers into four categories:

- The subcarriers connecting to both of $ClientA$ and $ClientB$ have higher quality
- The subcarriers connecting to $ClientA$ have higher quality
- The subcarriers connecting to $ClientB$ have higher quality
- The subcarriers connecting to both of $ClientA$ and $ClientB$ have lower quality

A ideal strategy for the AP is to use the subcarriers belonging to the first category to transmit bytes desired by both clients. The subcarriers in the second and third categories are only suitable for $ClientA$ and $ClientB$, respectively. The fourth category subcarriers can only use to carry less important bytes or shorter content, which is affected by a bit error rate less than longer content.

One of the issues to apply network coding and subcarriers to REFactor is how to encode packets that can be applied to the strategy mentioned above.

Packet Encode

Good packet encoding can reduce traffic on a channel and improve its goodput (defined as valid bytes excluding retransmitted bytes divided by transmission period). REfactor removes content in the chunk level from packets if this content appears in previous packet for a destination or is overheard by this destination. Shorter packets reduce the amount of traffic on channel and the packet error rate, while the bit error is fixed.

To further enhance REfactor to benefit the content that is not overheard we apply network coding. By combining several chunks, we can further reduce traffic that improves goodput.

Packet Processing

For each packet, we need to determine how to encode it. We can reduce traffic by RE and/or network coding and we need to figure out which parts of packets redundant and which part can be combined with other packets by network coding. Briefly, when an AP receives a new incoming packet, the AP processes this packet as below:

- Cuts packets into chunks.
- Matches each chunk with an associated entry in the cache.
- If this chunk is in the destination cache, it replaces it with a shim.
- If this chunk is not in the destination cache, but in another mobile devices cache, we mark this chunk as a candidate for network coding.
- Looking for a candidate packet in queue that can be combined with the new incoming packet.
- If a candidate packet cannot be found, the new incoming packet is inserted into a queue to wait for future combinations.

Notice that each chunk can either be replaced by a shim or marked as a network coding candidate, but not both. Furthermore, RE is given a higher priority than network coding, because RE can reduce a little bit more bytes and require less computation power to decode.

In this section, we discuss how we process packets before inserting them into output queues in more detail.

Chunking

We still use Chunk-Match as original REfactor to chunk packets. Each chunk only needs to be stored once instead of caching whole packets as in Max-Match in which cached packets may have some content overlap. Thus, memory can be used more efficiently in Chunk-Match case. In addition, Chunk-Match doesn't need a complex data structure; we can easily locate chunks in cache by self-addressing which uses the hash value of the chunk as an index pointing to a location in a cache. Min-Chunk can cache chunks even if a packet is only partially received.

As mention in EndRE, there are several ways to chunk a packet. To decide the boundary of chunks, we apply MAXP algorithm. We compute Rabin hashes for each byte with in 32 bytes regions and select a chunk with the largest hash value as a boundary. The MAXP algorithm has been found to be effective at selecting an uniform distribution hash.

Chunk Categories

After chunking a packet, we mark each chunk as one of the following categories.

- **RE chunks:** chunks have been already cached by their destination and can be replaced by shims.
- **Network coding chunks:** chunks are not in their destination's caches, but in other clients' caches. These chunks can potentially be combined with other chunks.
- **Remaining chunks:** chunks are not in any cache.

A chunk in the first category can be inside inter-client traffic or overheard from intra-client traffic. Its destination has already had this chunk, so it can be removed as redundancy. The second category of chunks has appeared in intra-client traffic, but was not overheard by its destination. One or more of other clients has this chunk in its cache (because it was in intra-client traffic). The third category is a chunk that has either never appeared in a system or has been replaced by another chunk, so no one has this chunk and we cannot do any encoding on it.

Chunk Classification

Each chunk should belong to one of three categories. To determine the category, we need to figure out the chunk is in the cache of which mobile devices. If a chunk was periodically sent to a mobile device, it must be in that mobile device's cache, because IEEE 802.11 retransmits missing packets that can guarantee that the packets are successfully received. To determine if the chunk has been overheard by other mobile devices, we rely on a model-driven RE that uses relative Wi-Fi transmission rates to establish the expected benefit of removing a chunk by a shim. Each cache entry maintains a list of reception marks corresponding to each mobile device. If the chunk was sent to a mobile device or if the expected benefit is larger than a threshold (highly possible that this chunk is overheard), the associated reception mark is set as true; otherwise it is set as false.

To locate a chunk in a cache, we compute the shorter and computationally cheaper hash value (20-bit) of this chunk as original REfactor. This hash value is used as an index pointing to a cache entry. We dont need a separate table which consumes extra memory storage to map hash values to the locations in the cache and we still can locate where each chunk should be in the cache. This approach is called self-addressing. After locating a cache entry, we can figure out which mobile devices have this chunk and its category by checking the reception marks.

Chunk Pairing Algorithm for Network Coding

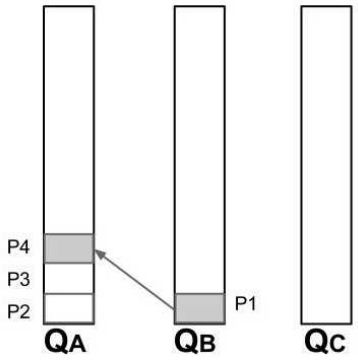

Figure 2.4 An AP maintain a queue for each client

It is straightforward to select and remove RE chunks by REfactor. However, we need to carefully select the network coding chunks. All chunks combined together should satisfy the following criteria:

- Each receiver of this encoded chunk should have all original chunks other than the chunk it needs. For example, $ChunkA$, $ChunkB$, and $ChunkC$ are encoded, and $Client1$ should have $ChunkB$ and $ChunkC$ to decode $ChunkA$.
- Packets for the same flows should be transmitted in order.

Algorithm

We maintain a queue for each client as shown in Figure 2.4. Each packet put in a queue is tagged a profile that includes a matching list. Each element in the matching list shows the maximal number of chunks in this packet that can be potentially combined with the chunks in other clients. This matching list is used to calculate the benefit to combine packets by network coding. To build up this matching list, we scan all network coding chunks (that cannot be replaced by shims) to count the number of chunks that are in the cache of other mobile devices.

In Figure 2.4, $P1$ is the next out-going packet, and we want to select another packet to combine with it. Because packets belonging to a flow should be sent in order, if $P4$ is picked up, an AP should send out $P3$ and $P2$ first and then combine $P1$ and $P4$. In this way, $P3$ and $P2$ lose

their chance to be encoded. Thus, we need to calculate the overall benefit to select a packet as in following equation:

$$Benefit = (Benefit\_gain) - (Benefit\_loss) =$$

$$min(\#ofChunksinP_{sent}, \#of\_Chunks\_in\_P_{pick})$$

$$-\sum_{i<I} \#of\_Chunks\_in\_P_i$$

where P_{sent} is a packet that is ready for transmission at the head of a queue, and P_{pick} is a candidate packet that is picked up to be combined with P_{sent}. $\#ofChunksinP_{sent}$ is the number of chunks that have been cached by P_{pick}'s destination and $\#ofChunksinP_{pick}$ is the number of chunks that have been cached by P_{sent}'s destination. In addition, I is the number of packets ahead of P_{pick} in a queue. The packet ahead of P_{pick} will loss its encoding opportunity. Thus, if we choose P_{pick}, this loss opportunity is a cost, so we need to subtract this cost. If the overall benefit is larger than zero, we can pick up the packet P_{pick} to combine.

Symbol Interleaving and Categories

As mention in [35], Wi-Fi frequency diversity is significant. In this section, we discuss how to take RE and network coding into account to interleave symbols more efficiently.

Consider the scenario with one AP and two clients again. As , Wi-Fi subcarriers can be categorized into four groups that are: high quality to two clients, high quality to one of them, high quality to the other, and low quality to both of them.

As subcarriers, symbols can be also divided into four groups according to the requirement of the chunks to which they belong.

- **Network coded symbols:** symbols within encoded chunks that are the combination of multiple flat chunks. Because multiple destinations need these encoded chunks to decode them

to flat chucks they require, these symbols should be transferred by the subcarriers that have good channel quality to all destinations

- **Shim symbols:** symbols that belong to shims. Shims replace redundancy and include hash values used to restore them to the original content in the destinations. The length of shims is quite short, so they encounter much less loss rate than longer content, if we assume a fixed bit error rate. Thus, it is safer to put these symbols into the subcarriers that suffer worse channel quality to destinations than any other symbols. The advantage is that we can increase channel utilization by using all kinds of subcarriers.

- **Other symbols:** symbols belong to flat chunks that are neither combinations of other chunks nor shims. Thus, only a single destination needs each of these chunks, and we can use the subcarriers that are only good for each single destination.

By applying the above strategy, if enough traffic is available to fill up channels, all subcarriers can be fully utilized during transmission.

Discussion

We enhance REfactor, and in this section, we will discuss the case that we improve from original REfactor.

Inter-client Redundancies: REfactor can remove inter-client redundancies only if these redundancies have been overheard, otherwise receiver cannot recover packets and need to re-request missing content that cause overhead. However, if we combine content for different clients by network coding, more inter-client content can be removed especially when overhearing possibility is low.

Overhearing Possibility: in optimization case, we only use the subcarriers with good quality connecting to a destination only or few other clients. Thus, most of clients other than the destination cannot overhear content. In this case, the performance of original REfactor is limited by lower

overhearing possibility. To use subcarrier more efficiently, this REfactor extension applies smart subcarrier interleaving and network coding.

2.4 Implementation

Our REfactor prototype is implemented as a pair of Click [36] modules. The *encoder* module is used at the AP to cache chunks, identify and remove redundancy and respond to requests for missing chunks. The *decoder* module is used at clients to cache chunks, reconstruct packets and request missing chunks. Each module is about 400 lines of code. Both use kernel-level Click to enable REfactor to work at the max 802.11g transmission rate of 54Mbps.

We chose to implement REfactor in Click because of the ease of deployment and flexibility this approach provides. Clients can easily run our Click decoder module to obtain the benefits of REfactor without operating system or application modifications. Furthermore, the encoder module can easily be deployed on an upstream network middlebox to serve multiple wireless APs. This avoids the need to modify AP firmware, which is often proprietary, and does not constrain REfactor due to the limited memory and processing power in many APs [37, 38].

2.5 Evaluation

We conduct an evaluation of the various benefits of REfactor. Our default settings is an AP operating in infrastructure mode with two associated clients. We also show the benefit of using REfactor in some of the other scenarios discussed in Section 2.1. Our evaluation utilizes traffic from real-world traces containing realistic packet chunk redundancy patterns. We focus on the following sets of issues: *(i)* How does our scheme, which uses self-addressing chunks, compare against SHA hash based alternatives (Section 2.2) in terms of speed and effective RE? *(ii)* What is the trade-off between cache-size and collision likelihood imposed by the self addressing chunks approach? *(iii)* What is the overall benefit of REfactor under various realistic redundancy patterns and varying levels of overhearing? What aspects of REfactor's design contribute most to its benefits? Can and should REfactor's operation be adapted to observed traffic patterns? *(iv)* How does

Min chunk size	REfactor	SHA hash based scheme
32	640 Mbps	64 Mbps
64	910 Mbps	118 Mbps
128	1203 Mbps	152 Mbps

Table 2.2 Comparison of encoding throughput for REfactor and a SHA hash scheme for different minimum chunk sizes.

REfactor perform in an actual infra-structure-based wireless setup? *(v)* How does REfactor help in the other scenarios in Section 2.1?

Speed and Redundancy Removal

We evaluate the effectiveness of self-addressing chunks compared to a scheme where the encoder (AP) transmits SHA hashes to the client in encoded packet shims (Section 2.2).

Speed. We benchmark the encoding speed on a desktop with a 2.4 GHz CPU and 8GB DRAM, mimicking a middlebox (co-located with the AP) which can perform encoding on behalf of the AP. Table 2.2 compares the encoding speeds for REfactor and a SHA hash based scheme. With a 1GB chunk cache and a minimum chunk length of 64B, our unoptimized Click module can encode at the rate of 910Mbps. This rate is sufficient for an AP to serve 30 clients each at the rate of 18Mbps. In contrast, SHA hash based encoding is 8$\times$ slower (118Mbps); SHA1 hash computation is a major performance bottleneck in this scheme. Our lightweight decoding operations impose low overheads on clients, as well. We used a low-end laptop with 1.66 GHz CPU and 2GB DRAM to measure the decoding throughput. The measured decoding throughput is 160 Mbps (for chunk size $\geq$32B). In contrast, the decoder throughput for a SHA hash based scheme is only 50 Mbps, even with 128B chunks, because clients have to compute SHA1 hashes for each cached chunk.

Redundancy removal. We compare the effectiveness of redundancy removal for REfactor and the SHA hash scheme in Table 2.3. We use a real trace with high overall redundancy (45%) and a 1GB chunk cache for both schemes. With small chunks (32-64B), REfactor (0.31) detects 75% of the

Min chunk size	REfactor		SHA hash based scheme		
	Redundancy detected	Effective RE	Redundancy detected	Effective RE	Effect. RE w/ hash shipping
32	0.31	0.27	0.41	0.22	0.03
64	0.28	0.26	0.38	0.29	0.19
128	0.23	0.22	0.31	0.27	0.22

Table 2.3 Comparison of effective redundancy removal for REfactor and a SHA hash scheme

redundancy detected by the SHA hashing scheme (0.41), the gap being due to collisions. However, the shim overhead of the SHA hashing scheme is quite high as a shim must carry a 20B hash. As a result, REfactor's redundancy removal (0.27) is 25% better than the SHA hash scheme (0.22). Using larger chunks (64-128B), the effectiveness of the SHA hash scheme improves, despite a decrease in detected redundancy, since the relative shim cost drops. However, decoding overhead on clients is still high.

One way to overcome SHA hash computation overhead at the decoder is for the encoder to ship SHA hashes for *every* chunk contained in a packet, as opposed to just sending SHA hashes for encoded regions. Unfortunately, the additional overhead of shipping SHA hashes reduces the effectiveness of RE by 25% (Table 2.3, last column). Using larger chunks (256B), the shipping cost and encoding overhead would go down, but the detected redundancy itself significantly drops to 0.2 (not shown).

To summarize, REfactor's design, in particular, the use of self-addressing chunks, gives the right trade-off in terms of speed, overhead and effectiveness of RE.

Caching

We now study the effectiveness of our self-addressing chunk storage and provide guidelines on how to configure caches. In particular, we vary the hash-size n from 14-bits to 22-bits in size, resulting in 1MB to 256MB sized caches, and we compute how much redundancy we are able to

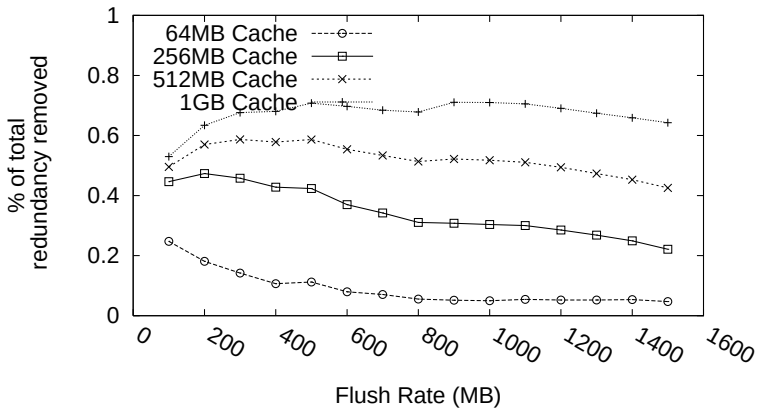

Figure 2.5 Impact of cache flush rate and cache size

remove from network traffic relative to an ideal infinite cache, which identifies 50% of bytes as redundant for the specific trace we study. In all cases the average chunk size is 64B. We show our results in Figure 2.5.

As expected, larger caches identify greater amounts of redundancy overall: e.g., a 512MB cache can identify nearly 60% of the overall ideal redundancy, whereas a 64MB cache can only identify up to 25%. In practice, caches can be provisioned on the basis of the average client's constraints: in an environment with laptops, using 256-512MB for caches is reasonable. When handhelds are employed, 128MB caches may be used.

We find there appears to be a "sweet spot" for flushing rates for most large cache sizes. For example, flushing a 512MB cache after every 500MB of traffic allows 10% more redundancy to be removed than flushing every 100 MB, and 16% more compared to flushing every 1.5GB. This is because a rapid rate of flushing (e.g., every 100MB) controls collisions better but reduces opportunities for removing redundancy; on the other hand, too slow a rate of flushing (e.g., every 1.5GB) does not eliminate collided entries fast enough. In general, flushing after every 200-300MB of traffic works well.

Goodput

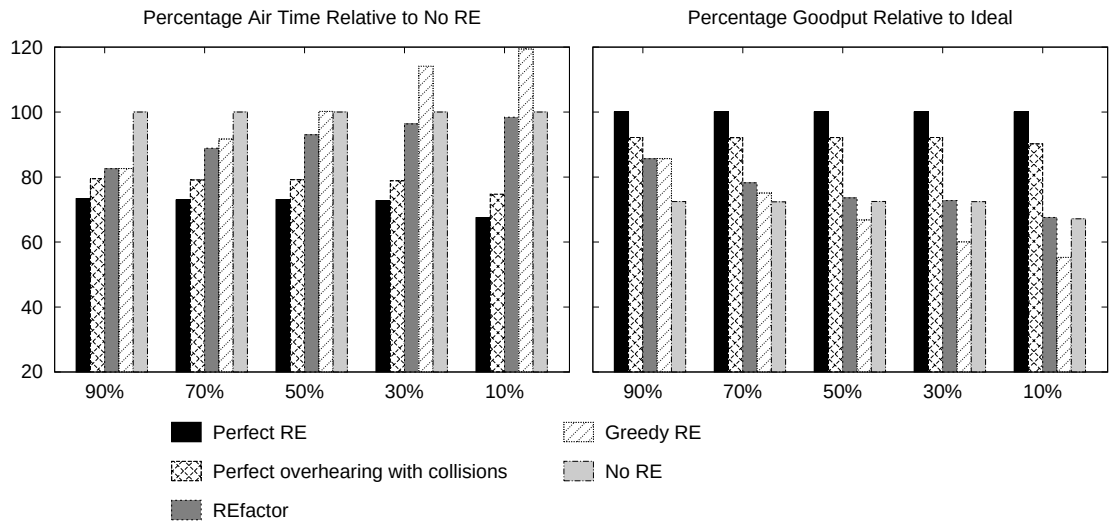

Figure 2.6 Total air time across both clients (a; left) and *C2*'s goodput (b; right) for a trace with high overlap.

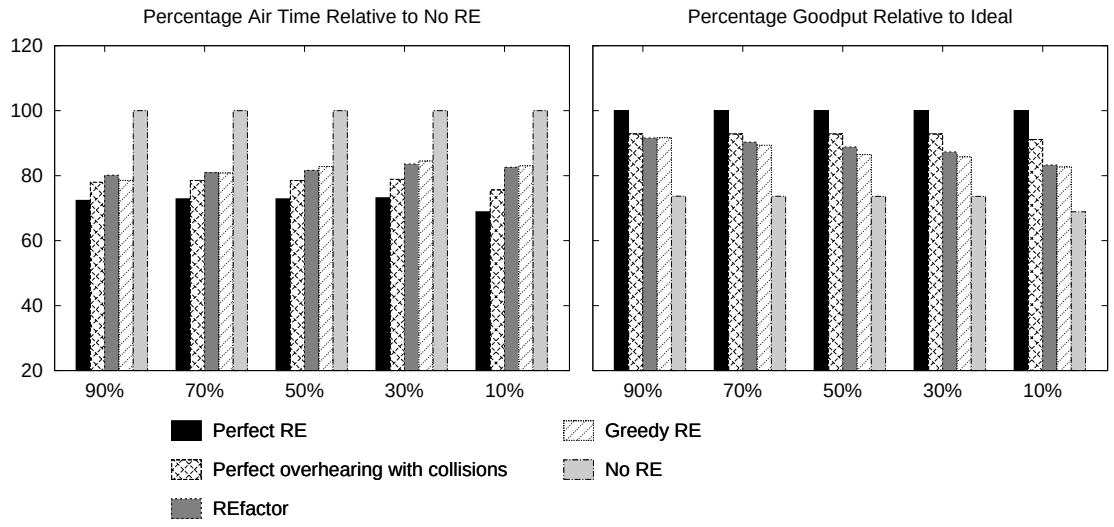

Figure 2.7 Total air time across both clients (a; left) and *C2*'s goodput (b; right) for a trace with low overlap.

We now evaluate the improvements REfactor provides. We use a simple emulated two-client setup with a single AP operating in infrastructure mode, as described in Section 2.1. In our setup, *C1* is located close to the AP, with perfect overhearing and a fixed high transmission rate (54Mbps),

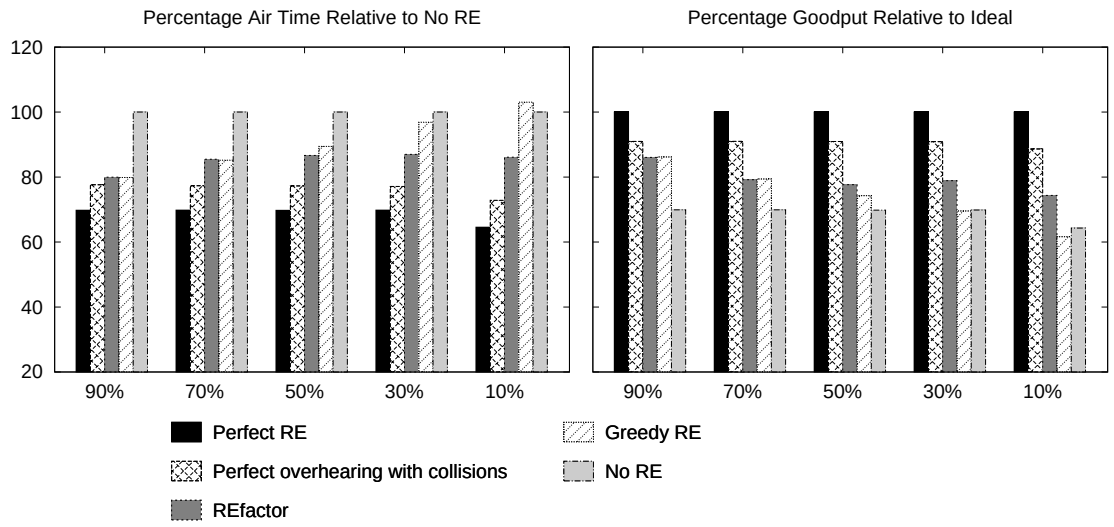

Figure 2.8 Total air time across both clients (a; left) and *C2*'s goodput (b; right) for a trace with medium overlap.

while we vary *C2*'s location relative to the AP. Depending on *C2*'s location, its ability to overhear transmissions to *C1* varies.

We experiment with five different overhearing scenarios, using overhearing probabilities at *C2* of 90%, 70%, 50%, 30% and 10%. An overhearing probability of 90%, for example, means the bit error rate is such that a *full length packet* (1500B) is overheard with 90% chance; a smaller packet that REfactor creates could, obviously, be overheard at a higher probability. We assume that *C1* overhears all transmissions to *C2* and the packets for *C2* can be received correctly by *C2* in the possibility of 90%.

We use a simple Click [36] configuration to emulate overhearing, where we super-impose packet reception probabilities at various packet sizes and transmission rates. These are derived from real-world measurements we collected in a relatively noise-free environment. For the five overhearing scenarios above, we fix the transmission rate to *C2* at 54Mbps, 36Mbps, 24Mbps, 11Mbps and 1Mbps, respectively.

We deploy our Click modules as Figure 2.9. The encoder module encodes incoming packets according to our Model-driven RE algorithm. The decoder recovers encoded packets and asks for missing chunks. The random drop module drops packets according to bit error rate (BER) and

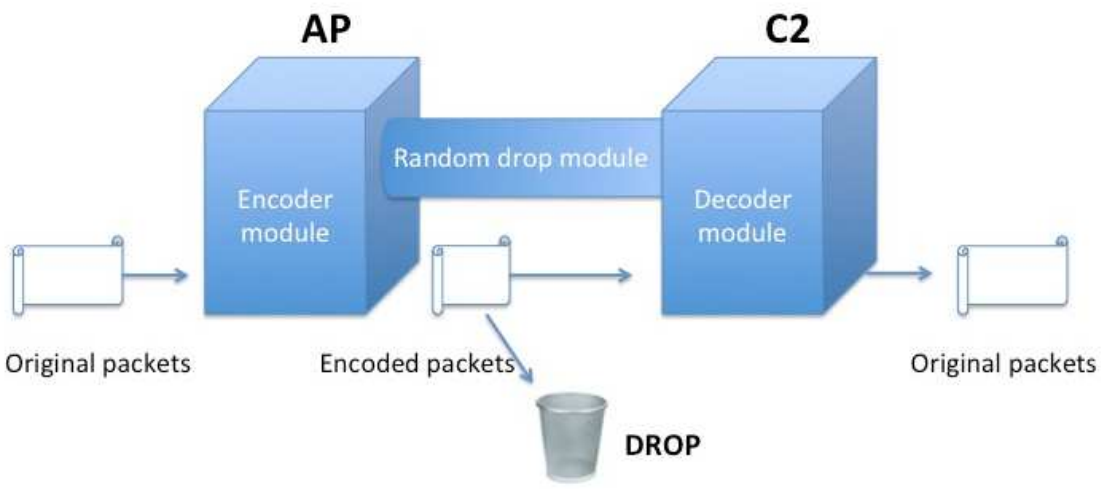

Figure 2.9 Click modules for goodput evaluation.

Overhearing %age	BER
90	8.8×10^{-6}
70	29.7×10^{-6}
50	57.8×10^{-6}
30	100.3×10^{-6}
10	191.9×10^{-6}

Table 2.4 The mapping from overhearing possibility to BER.

packet length. This module can simulate which packet can be overheard by $C2$. We calculate overhearing probability as following equation:

$$\textit{Overhearing probability} = (1 - BER)^{1400 \times 8}$$

In all five cases, we set up different BERs to let full length packet overhearing probability achieve the desired percentage. Thus, we can map the overhearing probability to BER as Table 2.4.

When receiving a packet, the random drop module determines its destination. If the destination is *C2*, we use the BER 8.8×10^{-6} (the lowest lost rate) to calculate the packet lose rate, which is

also determined by packet length (longer packets suffer higher loss rate). If the destination is *C1*, we pick up a BER according to what overhearing possibility we want to evaluate and Table 2.4 to calculate final overhearing possibility (shorter packets tend to be overheard in higher possibility). Finally, the random drop module decides to drop the packet or not by running a random number depending on packet loss rate (if its destination is *C2*) or overhearing possibility (if its destination is *C1*).

The traffic transmitted to each client in our experiments is based on a real-world trace we gathered on an outbound link from a university web server. We use traffic to destinations in the real trace to construct a traffic mix for the two clients in our simulation. We pick three sets of traces that offer high (49%), medium (23%), and low (4%) inter-client redundancy, i.e., bytes shared across clients, where redundancy is measured using the Max-Match approach with a large cache; the intra-client redundancy is 1%, 24% and 46%, respectively. Note that the overall redundancy is roughly similar (≈50%) across the three traces.

We do not claim the traces we study reflect the actual redundancy we expect to see in traffic sent to, and shared between, wireless clients; quantifying the redundancy is not a goal of our work and this issue has been explored in prior studies [9, 22, 2]. Rather, our goal is to use the traces to recreate a variety of realistic redundancy patterns, i.e., granularities of redundancy, and spacing of redundant bytes in time and across hosts, to study how they impact the benefits of REfactor given its design choices, and also to understand the conditions in which REfactor offers the most benefits.

We use three metrics: (1) *Total air time* defined as the total amount of time spent in transmission to either client (including missing chunk requests and responses). This is an indication of the medium's utilization. (2) *Goodput of a client* defined as the total number of bytes transferred to the client divided by the total time spent by the client in receiving them, which includes time spent in retransmissions due to packet losses or missing chunk transmissions due to cache misses. (3) *Packet loss rate* at the client.

High Inter-client Redundancy

We first study the performance where a lot of content is shared *across* clients and little overlap exists *within* a client's traffic. The results in Figure 2.6 show the overall air time (a) and *C2*'s goodput (b).

The bar "perfect RE" corresponds to the case where we assume no collisions occur and all clients can overhear all traffic. In "perfect overhearing with collisions", we assume cache sizes are limited to 512MB, resulting in collisions, but nodes can overhear all traffic. The bar "greedy RE" reflects a RE approach which always encodes packets as opposed to REfactor's use of model-driven RE. Finally, "no RE" represents a situation where no content overhearing or redundancy elimination is applied.

The overall airtime is plotted relative to the total airtime under "no RE", showing the lowering in utilization due to various schemes. The goodput is plotted relative to *C2*'s goodput under "perfect RE" showing how close to ideal the improvement in goodput gets. We make these observations:

1. REfactor offers substantial improvements relative to "no RE", with airtime (Figure 2.6a) being 20% lower in the highest quality link case (90% overhearing) and 7% lower in the lowest quality link case (10% overhearing). REfactor's airtime is between 6% and 30% worse than "perfect RE," the difference arising due to the need to account for collisions, and the need to account for and recover from cache misses. *C2*'s goodput (Figure 2.6b) is 24% and 4% better than "no RE" in the highest quality and lowest quality link cases, respectively.
2. In the poor quality link case, e.g., 10% overhearing, REfactor may overhear as few as 10% of the packets compared to "perfect overhearing with collisions"; thus, one may expect that it should only be roughly 10% as effective in improving goodput (Figure 2.6b) as "perfect overhearing with collision". Instead, we see that REfactor is 4% more effective than "no RE", whereas "perfect overhearing with collision" is 34% more effective. Thus, REfactor does not seem to lose as much performance as we might expect under low quality links. The reason for this is that encoded REfactor packets are smaller, and hence they experience lower

packet loss rates compared to "no RE"; see Table 2.5 which shows that REfactor imposes 7-27% fewer drops than "no RE". Fewer losses helps improve goodput. More importantly, the packets that are not lost also carry valuable unique bytes that contribute to removing redundancy from future packets. This effect is likely to be much more pronounced in situations where there is a much greater amount of content shared across clients, e.g., in flash crowds.

3. Comparing REfactor against "greedy RE", the gap is small at high quality links, but increases significantly when link quality falls below 50%. At 50% overhearing, our approach improves goodput by over 13%, whereas "greedy RE" results in a 6% *drop* (Figure 2.6b). Thus, model-driven RE in REfactor plays a crucial role in ensuring robust performance, especially under poor overhearing and high inter-client redundancy.
4. Consider the performance for the link with 70% overhearing probability, where "greedy RE" still offers non-trivial goodput benefits (5%; Figure 2.6b) compared to "no RE". Comparing this with the results for the 90% link, we can conclude that if REfactor used 90% as the reception probability estimate for the link in its model driven RE, but the actual probability was 70%, then REfactor's overall performance would still be better than "no RE". This shows REfactor's overhearing probability estimation is robust to a certain degree of error especially when link quality is reasonable. However, at poorer link qualities (50% or below) mistakes can prove costly. Thus, it helps to be conservative with encoding, i.e., use a high threshold for expected benefit in model driven RE, when link quality is poor and when inter-client redundancy is high.

Low Inter-client Redundancy

Next, we look at the traffic mix with very high redundancy within client traffic (i.e. high intra-client redundancy) and low redundancy between clients (i.e. low inter-client redundancy). We plot total airtime and relative goodput achieved by *C2*, both in relative terms as before, in Figure 2.7. We note the following:

1. Compared to the high inter-client redundancy case above (Figure 2.6a), REfactor offers better airtime goodput improvements relative to "no RE": e.g., at 30% overhearing, REfactor is

Overhearing %age	Loss % (% better than "no RE")
90	6.2 (27)
70	6.7 (21)
50	7.1 (16)
30	7.6 (11)
10	7.9 (7)

Table 2.5 Loss % with REfactor. The loss rates due to "no RE" and "perfect RE" are 8.5% and 5.3%, respectively. We show % lowering of loss rate relative to "no RE" in brackets.

22% better than "no RE" in the low inter-client redundancy case (Figure 2.7a), whereas in the high inter-client redundancy case (Figure 2.6a), it is 8% better than "no RE". Because REfactor does not have to deal with overhearing, it is able to derive substantial benefits most of which are due to IP-layer RE itself.

2. The variation in overhearing rate has a slight effect on the benefits of REfactor, with benefit dropping as overhearing becomes poor. While most of REfactor benefits with this trace are from intra-user redundancy, the trace does have a small amount of inter-client redundancy (4%); at low overhearing probability, model-driven RE in REfactor would conservatively decide against encoding most, if not all, inter-client redundant packets, resulting in a drop in goodput.
3. The performance of "greedy RE", which encodes all packets, is slightly better than REfactor. Whatever bytes are saved in this fashion contribute to high goodput and the small number of cache misses that result (for the inter-client traffic) can be easily recovered through retransmissions. Thus, when intra-client redundancy is predominant—the redundancy pattern can be determined by profiling traffic on the fly—it is best to turn off model-driven RE and encode all data.

Medium Inter-client Redundancy

C2's Distance from AP	No RE Goodput	REfactor Goodput	Percentage Improvement
3m	4.0Mbps	3.4Mbps	20%
6m	3.0Mbps	2.6Mbps	14%
10m	1.3Mbps	1.2Mbps	6%

Table 2.6 Performance improvement provided by REfactor in a real infrastructure-based wireless setup.

In Figure 2.8 we show a situation where redundancy is roughly equally inter- and intra-client. Comparing with Figures 2.6 and 2.7, the performance offered by REfactor is intermediate compared to the prior two cases, as expected. We also note that the performance of "greedy RE" is almost comparable to that of "no RE" at 30% and 10% overhearing. Thus, with less redundancy, the impact of incorrect estimation of link overhearing is even less pronounced: more specifically, if REfactor used 90% or 70% as the overhearing probability estimate for a link that current has 50% overhearing (or even lower), the performance of REfactor may still be noticeably better than not using RE or overhearing. This also means that model-driven RE can be somewhat more aggressive, i.e., use a lax threshold for expected benefit, under this kind of traffic pattern.

Overall Benefits: Testbed Results

While the results we have discussed so far are derived from an emulated infrastructure-based scenario, we also measure REfactor's performance using an actual wireless AP and two clients. We use the high inter-client redundancy trace, and we vary *C2*'s distance from the AP from 3 to 10 meters to explore a range of overhearing probabilities. Table 2.6 compares the goodput without RE and using REfactor.

Similar to the results in Figure 2.6b (where REfactor's goodput improvement over "no RE" ranges from 24% to 4%), the benefits from REfactor in a real wireless setup range from 20% to 6%. This confirms that our emulated setup provides a reasonable representation of REfacotr's performance improvements in practice.

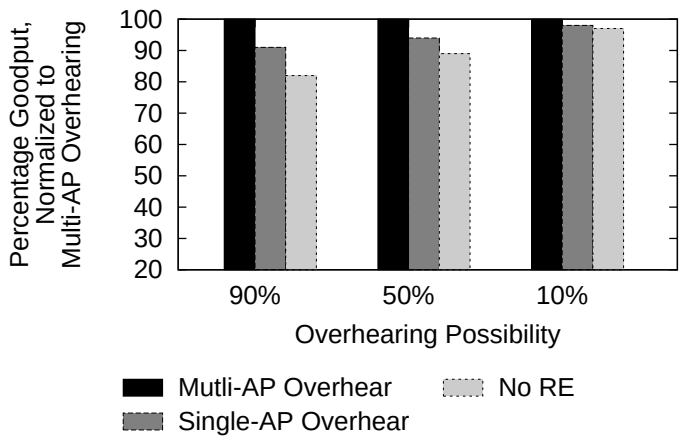

Figure 2.10 Goodput improvements in a multi-AP scenario

Multi-AP Improvements

We extend our setup to two APs operating in infrastructure mode and three clients: *C1* and *C2* associated with *AP1* and *C3* with *AP2*. Additionally, *C2* is able to overhear transmissions from both APs. Figure 2.10 compares the relative goodput for *C2* without RE, when overhearing only from its associated AP, and when overhearing from multiple APs. We observe that *C2* realizes up to 10% more benefit from REfactor when taking advantage of traffic overheard from other APs. As expected, returns diminish as overhearing probabilities decrease.

Our multi-AP simulation assumes both APs can overhear each other's transmissions, avoiding collisions. However, collisions may occur in the case of hidden terminals. For example, if *AP1* transmits to *C1* at the same time *AP2* transmits to *C3*, the two destinations will receive their respective packets, but the packets will collide at *C2*, who will be unable to overhear either packet. Such collisions prohibit *C2* from receiving maximum benefits from REfactor. However, REfactor may be able to reduce the likelihood of collisions due to decreases in the size of *C1* and *C3*'s packets.

REfactor + Network Coding

We implemented a simplified version of COPE [13] within our Click prototype and experimented with the scenario in Figure 2.1d. Our simplified version of COPE XORs packets, but we

Overhearing %age	Overall Improvement (% better than COPE)	Relay to C3/C4 Improvement (% better than COPE)
90	14	38
70	11	26
50	10	21
30	6	13
10	3	7

Table 2.7 Air time savings %age with REfactor + COPE.

ignore confounding factors like pseudo-broadcast, retransmissions, and reception reports. When the *relay* is scheduled to send packets, it determines if it has packets for *C3* and *C4* that can be coded, removes redundancy from them and sends a coded packet along with a shim. For simplicity, we assume the relay has perfect knowledge of what *C3* and *C4* overheard for both COPE and our approach; in practice, the relay has to rely on feedback from clients [13].

The air time savings of REfactor + COPE, compared to just COPE, are shown in Table 2.7 for different overhearing percentages between *C1–C3* and *C2–C4* (assuming both links have the same overhearing probabilities). With 90% overhearing, REfactor provides 14% air time savings. This savings is purely from reduced transmission sizes from the *relay* to *C3*/*C4*: the savings for *relay–C3/C4* transmissions is 38% with 90% overhearing. Compared to air time savings with "perfect overhearing with collisions" in the single AP case, this is almost twice as much, a result of the transmission reductions realized via network coding. At lower overhearing (e.g., 10%), the overall relative benefit of using REfactor reduces (to 3%) because there is less content overheard.

REfactor + NEWS

We evaluation the performance of NEWS by simulation. In lower layers and application layer, we rely on NS-3 [39], which supports WiFi channel simulations such as delay and random loss. We implement IP layer algorithms in Click [40], which cuts packets into chunks, caches the chunks,

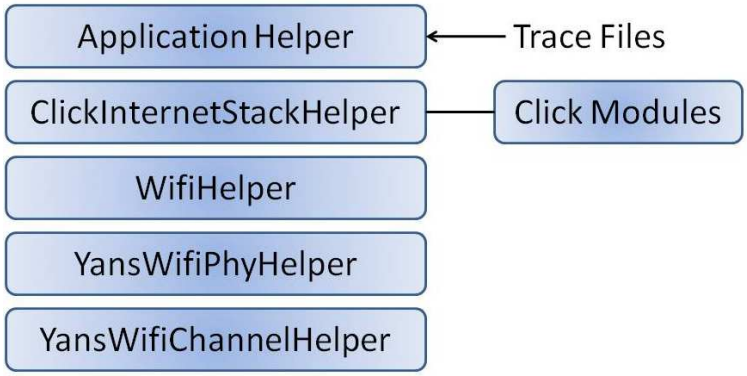

Figure 2.11 The architecture of NS-3, which connects to Click modules

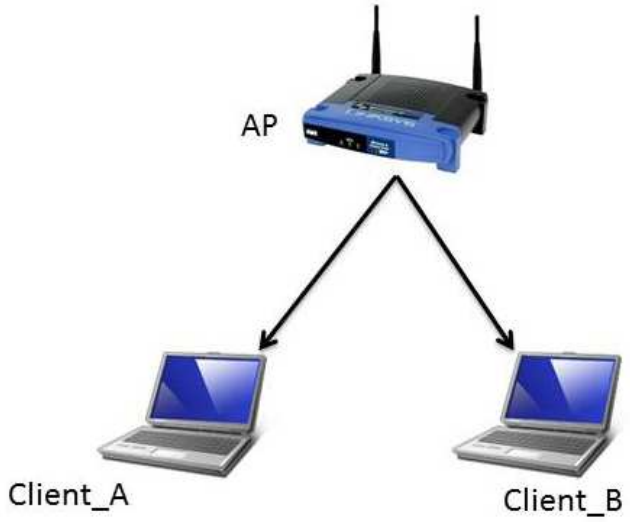

Figure 2.12 The scenario we use in our experiment

and removes or combines the chunks to reduce traffic. Figure 2.11 shows the architecture we uses. Click modules can be embedded into NS-3 as a layer. Packets coming from application helper are sent to the input of a Click module. After processed, these packets are outputted to lower NS-3 layers.

We set up a simple scenario that includes one AP and two clients as shown in Figure 2.12. The AP send traffic to both of clients. There are three different kinds for subcarriers. $ClientB$ encounter better overall channel quality connecting to the AP than $ClientA$. $ClinetA$ can overhear

	REfactor	NEWS
The percentage of bytes saving	9%	1.2%
Goodput	773 kbps	708 kbps

Table 2.8 The result for high inter-client redundancy.

	REfactor	NEWS
The percentage of bytes saving	18%	17%
Goodput	1000 kbps	980 kbps

Table 2.9 The result for low inter-client redundancy.

the traffic for $ClientB$ in low possibility. Part of subcarriers have good channel quality (30dB) connecting to both clients. We use these subcarriers to carry network coded chunks. Because the subcarriers with good channel quality connecting to $ClientA$ are too narrow, so we send the chunks only for $ClientA$ using a lower bit rate. We observe the goodput(bytes received by application layer divided by time) received by $ClientA$, and compare NEWS with REfactor.

As REfactor, the traffic transmitted to clients is based on real world-trace files. We use traffic to destinations in the real trace to construct a traffic mix for the two clients. We pick two sets of traces that offer high (49%) and low (4%) inter-client redundancy. In addition, the intra-client redundancy is 1% and 46% respectively.

Table 2.8 shows experiment results for high inter-client redundancy, while Table 2.9 provides experiment results for low inter-client redundancy. Because $ClientA$ cannot overhear most traffic for $ClientB$, we remove much fewer redundancies in high inter-client redundancy case. REfactor can only remove 1.2% in this case; however, NEWS can apply network coding on some bytes, so it can remove 9% bytes, which is more than 6 times more than REfactor. The number of packets doesn't change much, so goodput improvement is not as critical as byte saving, but NEWS is still more than 9% better than REfactor.

In low inter-client traffic case, the performance different between NEWS and REfactor are less. Both or them can remove intra-client redundancy well.

REfactor	NEWS
708 Mbps	678 Mbps

Table 2.10 Comparison of encoding throughput for REfactor and NEWS.

To understand the overhead of network coding, we compare the encoding throughput of REfactor and NEWS module in Click. We run our experiment in a laptop with Intel i5-3210M CPU and 4 GB RAM. Because NEWS need to scan packet chunks for network coding, its encoding throughput (678 Mbps) is slightly lower than REfactor (708 Mbps).

In summary, NEWS removes more inter-client redundancy than REfactor does, which increase goodput on wireless links. Although overhead is added, the encoding throughput is still acceptable for nowaday wireless traffic.

Chapter 3

An Information-Aware QoE-Centric Mobile Video Cache

In this work, we discuss a how to optimize transmission in application viewpoint. We focus on video service, because it becomes a dominate service now. The key metrics to evaluate is watching experience. To improve video watching experience, we provide a middle box called iProxy.

In this section, we describe the motivation first in Section 3.1, the design in Section 3.2, and finally give some evaluation in Section 3.4.

3.1 Motivation and Background

Video Caching

In [41], they mention that 3G/4G networks trace all traffic centrally for device location tracking and billing issues. The gateway becomes a bottleneck with high traffic concentration, when the number of base stations grows. Thus, caching in a gateway reduces the load in 3G/4G core networks (CNs). [42] also says that the key to improve 4G networks is to increase capacity and to handle more users, but not speed.

However, a convention proxy only uses URLs to identify objects, thus it may cache the same content with different URLs which provides redundancy in cache. Instead, iProxy identifies objects by looking into the content of objects. It results better video caching performance.

Video Watching Experience

Video watching experience determines the client engagement; with better watching experience, clients will be more willing to enjoy the services. if more clients use video services and spend more time on watching videos, both video providers and network operators can get more revenue. Thus, video watching experience becomes an important factor, when providing video services.

Multiple key factors determine user engagement. For example, if a client waits for long time before receiving a video service, s/he may loss toleration and quit the service before video playing. However, some other factors such as user interests are not easy to be predicted by video providers (may need some history records).

In [20, 21], they define user engagement as follows:

- Watching time of each video view. Higher user engagement implies that a client is willing to spend more time on one video view.
- The number of video watch for each viewer. If a client satisfies a video service, s/he will watch more videos, because s/he expects the same good quality for other videos provided by the same video service.

Their works figure out how quality metrics are relative to user engagement and build a model for it. The quality metrics they consider are:

- **Join Time:** How long clients need to wait before videos play. When receiving videos, players at client side spend some time to pre-process videos and fill up their buffers. Higher bit rate videos take longer time. Thus, if we send unnecessary higher bit rate videos to clients, longer processing time increases the join time. For example, smartphones can show only lower resolution videos, so they prefer lower resolution videos (lower bit rate).
- **Buffering Rate:** The percentage of time is spend on re-filling a buffer. When network condition is not good enough, videos stall to wait for incoming data. To reduce the buffering rate, we need stable bandwidth. However, wireless environment provides various channel quality, so video bit rate should be adapted to fit current available bandwidth.

- **Rate of Buffering Event:** The number of buffering events during video playing. The buffering events interrupt video watching, and affect watching experience. It is caused by unstable channel quality. To reduce the events, we can adapt video bit rate or increase buffer size at client side.

- **Average Bit Rate:** Higher video bit rate implies better video quality. Clients usually feel better when watching higher quality videos. However, higher bit rate also means higher bandwidth requirements, so when channel quality is unstable, a higher video bit rate causes more and longer buffering events. Therefore, a bit rate is a trade-off and should be set carefully.

- **Rendering Quality:** A rendering rate is the number of video frames per second. It determines how smooth videos are playing. Usually, if the rending rate is lower than 30 fps, video viewers may affect watching experience. When channel quality is terrible or computational power at client side is not enough, some frames may be received later than their deadline and video players drop these frames.

Based on user engagement and quality metrics presented above, we propose the design requirements of iProxy as the next section.

Design Requirements

The goal of iProxy is to use cache storage efficiently and provides good quality of experience (QoE) that is reflected in engagement, i.e., fraction of the total video time that a user watches, and abandonment, i.e., whether a user quits a video before it starts.

Efficient caching: Cache space is restricted and the performance is limited. If we can reduce redundant content in cache, high cache hit rate, which lowers video start-up delay, can be achieved.

A conventional proxy identifies videos by URLs. Thus, for each requested URL, a proxy needs to store a copy of its content. However, the content associating with different URLs may be actually the same. For example, the same video may be uploaded to youtube.com and dailymotion.com.

People use different URLs to request the different instants of this video from these two video providers through a conventional proxy. The proxy identifies the videos as different content. Cache storage space is wasted, if both of the copies are stored.

Furthermore, a video can have various versions such as different resolutions, bit rates, and file formats, but the content of the different versions is actually the same (they include the same information). In this case, if we can store only one of the versions, storage can be further saved to satisfy more requests.

Good QoE: It is nontrivial to correctly define what good QoE is, because different users may care about different criteria, but in my thesis, I focus on start-up latency, video stall time, and video quality (bit rate) as mentioned in Section 3.1. clients are more willing to use the video service with good QoE and decrease the chance that clients leave before the end of videos.

- **Lower start-up delay:** It is the the waiting time after an user requests a video. This delay is mainly affected by cache hit rate and video pre-processing time. If the request is hit in cache, a proxy do not need to redirect this request to a video provider such as Youtube, which takes extra round-trip time between a proxy and a video provider. In addition, if a video is unnecessarily too high in bit rate, an user device takes long time to pre-process it before the video plays. Thus, increasing cache hit rate and providing suitable videos for clients are our goals.

- **Video stall:** If available bandwidth is not enough or video bit rates are too high, some frames are loss or retransmitted, which cause video stall. Video players stall the videos to wait for following data. Thus, videos are interrupted and clients may not be comfortable with it.

- **Video quality:** If we reduce the bit rate of videos to prevent traffic from using out bandwidth, some detail in videos will miss. This affects users' experience, because they may expect better video quality. In addition, because bit rate is lower than available bandwidth, remaining bandwidth is wastage.

- **User device:** It determines what kinds of videos can be decoded and played. If we give an end user incompatible videos, the end user may be unable to play it or need to degrade

video quality to play it. For example, if video resolution is higher then screen resolution, the user device needs to reduce the resolution to fit screen size. Higher resolution videos usually come with higher bit rate. Thus, it wastes bandwidth for transmission and also results higher start-up latency caused by longer pre-processing time.

Other requirements: our another goal is that our system should be deployed more easily. To make our system work, we only need to add a middle box as a proxy in a network without modifying existing end user devices and video providers.

Information-Bound Referencing

The previous section mentions that an URL is not a good identification for video content. Thus, we introduce information-bound referencing (IBR) from [43]

IBR retrieves information from multimedia data as identification. This identification isn't tied to a specific protocol, host, file name, and multimedia format such as resolution and bit rate, but is bounded only to the information it presents.

IBR transforms multimedia information into frequency domain. For each frame of video or a picture, its frequency domain data is linearly relative to how it looks like by human eyes. Thus, frequency domain data is a good choice to identify content.

In this section, we discuss how to calculate IBR value for multimedia content and how to apply it to iProxy.

IBR for a single frame

We begin by the IBR calculation of a single frame. [43] presents three options to retrieve fingerprinting for an image, which are leveraging spatial structure, using color distribution, and frequency domain analysis.

Frequency domain analysis, which uses Hamming distance to distinguish images is the better choice for the following reasons. First of all, the Hamming distance between transmission images

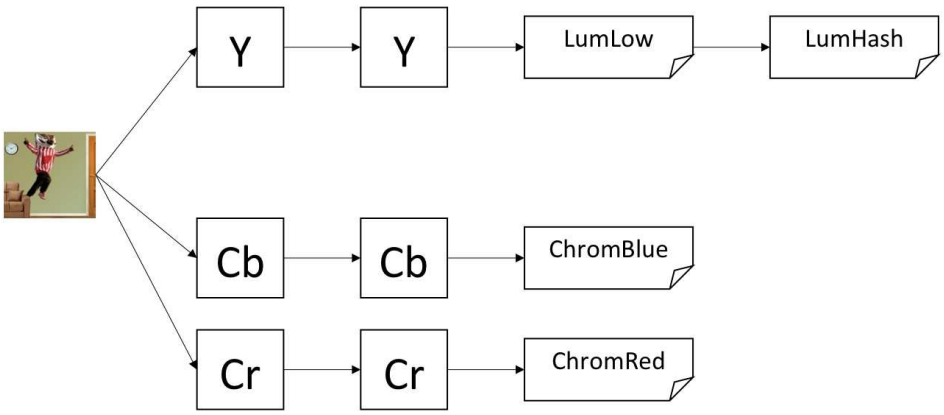

Figure 3.1 Retrieve IBR from an image.

is low enough to recognize them that include the same or very similar information. Second, the Hamming distance between different images is high enough to distinct images. [43] shows that frequency domain analysis provided by Discrete Cosine transforms(DCT) works well with the threshold Hamming distance of 11.

To calculate IBR value for an image, we use the 64-bit fingerprint as $Hash64$ which is provided by [44] called $ImgIBR$. As shown in Figure 3.1, We represent an image as three components: $luma$ (Y), $chromablue$ (Cb), and $chromared$ (Cr). YCbCr representation can concentrate energy in the first few low frequency coefficients in Cb and Cr, so it is easier for us to sample important information.

We do DCT on YCbCr to get FY, FCb, and FCr that are frequency domain data. Then, We couple [LumLow, LumHash, LumHigh, ChromBlue, ChromRed) as a fingerprint for an image ($ImgIBR$). LumLow and LumHash provide an overview of the fingerprint, and others capture

information in more detail.

IBR for a Video

In [43], videos are cut into chunks and we only process frames at chunk boundaries. The way to chunk videos is using scene detection [45]. Scene detection scans the similarity of frames and groups frames with similar content together.Thus, frames in each scene look similar, so their $ImgIBR$s are also similar. We call the frames at chunk boundaries $keyframes$ where the images change significantly named scene changes. To identify the scene changes, we need to pick up an good image feature that can be easily computed and is consistent. According to experiments, [46] decide to use the variation in the amplitude of the zero-th frequency of the Y-component.

Chunking algorithm: to measure the similarity, [46] defines a distance $DistSeq$ and a $ChunkThresh$. The distance between frame i and i + 1 is $DistSeq(i) = \frac{|A_{i+1}-A_i|}{min(A_{i+1},A_i)}$. If the distance is less than $ChunkThresh$, we say these two frames are similar and group them into the same chunk.

Smaller chunks are more sensitive to identify content variants. However, cutting a video into smaller chunks means that the number of chunks increases. Thus, it also increases IBR calculation time and matching time. [46] suggests that $ChunkThresh$ should set to 0.5.

For each chunk, only the first and the last $ImgIBR$s are stored. All $ImgIBR$ pairs of chunks are concatenated to form the IBR value of a video, $VideoIBR$. 424-byte audio IBR using an existing audio fingerprinting algorithm [47] is included.

IBR lookup

After IBR calculation, we need to know how to look up and match IBR, then we can identify videos. As mentioned in 3.1, the same content may provide slightly different IBRs caused by different encoding methods. Thus, to lookup matching IBRs, fuzzy matches are used.

To do the fuzzy matches, a suitable data structure is desired. Locality-Sensitive Hashing(LSH) [48, 49] has been used to solve multimedia search problems.

[46] use bit sampling to implement Locality-Sensitive Hashing(LSH) [48, 49] which has been used to solve multimedia search problems. Assuming l is the number of hash functions and k is the number of bits per hash function. We can get a k-bit string from each hash function which is the concatenation of $h_1^i(Hash64)$,... ,$h_k^i(Hash64)$ where $h^i, i = 1...l$, is a hash function. We note this k-bit string as $g_i(Hash64)$. Because there are l hash functions, we can map each $Hash64$ to l buckets that are $g_1(Hash64)$,... ,$g_l(Hash64)$. Given a new query for $Hash64'$, we retrieve the entries in buckets $g_1(Hash64')$,... ,$g_l(Hash64')$ and match these against $Hash64'$. [46] suggests l = 20 and k = 20 which is reasonable.

When we try to match a given video chunk IBR, we first look for some candidate IBRs in LSH data structure for the $Hash64$ of the first fame of the video chunk IBR. Then, we try to match the other fields directly.

IBR Lookup Performance

The analysis of IBRs over a large collection of videos in the wild shows that, using conservative match thresholds, IBRs can match related variants with zero false positive rate. We found zero false positives in our own trace-based analysis (Section 3.4). However, false negatives are non-negligible (5%) meaning that not all hits will be identified. We believe that evolution in multimedia fingerprinting schemes will result in even more robust IBRs with far lower false negative and zero false positive rates for matching. In [50], they shows that IBR can defense the integrity attacks of inset (bogus content is embedded), quantization (lower the video quality), and resize (rescale videos), but IBR cannot determine the subtitle deference. However, many modern mp4 videos separate video content from subtitles.

Channel Diversity

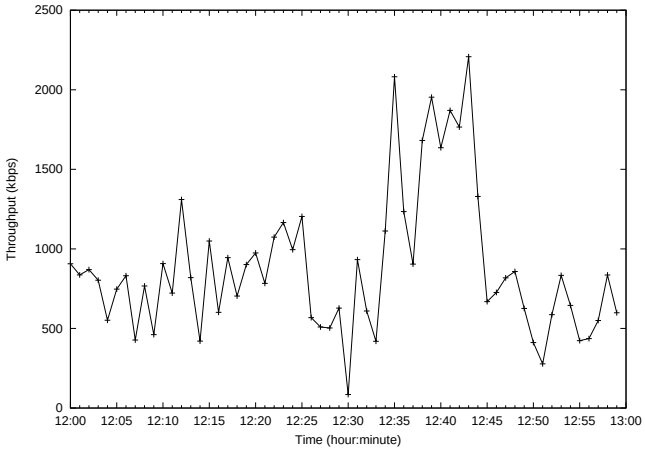

Figure 3.2 Throughput diversity with in two hours period

For higher QoE, we should estimate and efficiently use available bandwidth on channel. Video streaming requires stable bandwidth instead of higher average bandwidth. However, wireless networks are not stable enough to stream videos. Channel quality is varying with different time and different locations.

WiScape [18] measures the throughput, jitter, and loss rate of 3G networks in many spots around Madison, WI, and it also measures how moving speed affects performance. It presents that a cell phone experiences various channel quality in different locations. Even in the same location, standard deviation of throughput may be high.

To further understand channel diversity, we measure throughput variation during the period of one hour. We run a server in a desktop. The server sends TCP dummy data to a cell phone via 3G network as fast as possible. We measure the throughput observed by the cell phone as showed in Figure 3.2. The measurement starts at 12:00 and ends at 13:00.

Throughput lies in the range between 85 kbps and 2208 kbps. We can observe that there is one burst which are over 1500 kbps , and just after this burst the throughput drops to less than 250 kbps. It means channel quality is highly various.

We can conclude that during a video playback, available bandwidth keeps changing. To achieve our goal of reducing video stalls and keeping video quality, video bit rate should be adjusted dynamically with channel diversity.

Client Diversity

Client may use different kinds of devices such as laptops, tablets, and smart phones to access videos. The screen sizes and supported resolutions are different. Thus, videos with different resolutions may be requested.

Besides hardware, software in client devices may be also inconsistent. Operation systems can be iOS or Android. The video decoders each device have installed are different. To dynamically support the diversity, we need to first detect what software each device is running and then we can provide suitable video formats.

Thus, the issue is how we figure out the hardware and software of client devices. In addition, after we retrieve this information, how do we stream videos with suitable formats.

Cache Replacement Policy

In any cache systems, replacement policy is an important issue and is critical for cache performance. A main metric for cache performance is hit rate (HR). HR is the number of requests that match previous cached requests divided by the number of total requests. Hit requested won't be forwarded to data providers. Thus, higher HR provides lower video start up delay.

The key goal of cache replacement policy is to keep more useful data that will be requested in future as possible. [51, 52] describes some typical cache replacement policies including first-in-first-out(FIFO) policy, the random replacement policy, the last recently used(LRU), the least frequently used(LFU) policy, the last frequently used-aging policy, and greedy dual size(GDS).

FIFO: This is a low overhead policy and it doesn't need to record any extra information for cache entries. Incoming data is inserted into the tail of a queue and the oldest data is removed if

cache is full. This policy doesn't consider the importance of data, so some important data may be removed that can decrease HR.

Random replacement: This is also a low overhead policy. When cache is out of space, we randomly pick up one or more entries to remove. Thus, it may also remove some important data.

LRU: This policy keeps record when each entry is referenced. The main motivation of this policy is that an entry which is referenced recently will have more chance to be referenced again in near future. We can implement this policy by a queue. A referenced entry is moved to the tail of the queue. Thus, when cache is out of space, the entry in the head of the queue is removed, because it will have not be referenced for the longest period. This policy consumes some overhead to move entries.

LFU: This policy maintains a counter for each entry. This counter represents the number of times this entry is referenced. LFU will choose the entries with lowest count to remove when cache is full. It assumes the entry with higher used frequency will be referenced in future in higher possibility. However, we need to spend some space to store the counter and some time to update the counters.

LFU-aging: It is a modified version of LFU. LFU has a cache pollution problem. Some entries may be very popular in a particular period, so they can gain high count. However, after this period, they are less or never used. Because their counters have been already high, they have less chance to be removed. Thus, these entries occupy cache space as pollution. LFU-aging is developed to solve this problem. It reduces counts periodically ,so after awhile, the pollution will be removed.

GDS: This policy takes multiple factors into account. To calculate the importance of each entry, GDS defines a score which is the ratio of cost and object size ($\frac{cost}{size}$) for each entry. The cost is composed of reference count and other costs to ask data from remote provider including network bandwidth and delay. The size is the data size. When cache is full, GDS pick up the entry with the lowest score. GDS considers how expensive to remove each entry in more detail, and also take data size into account.

We will describe modified cache replacement policy for iProxy in Section 3.2.

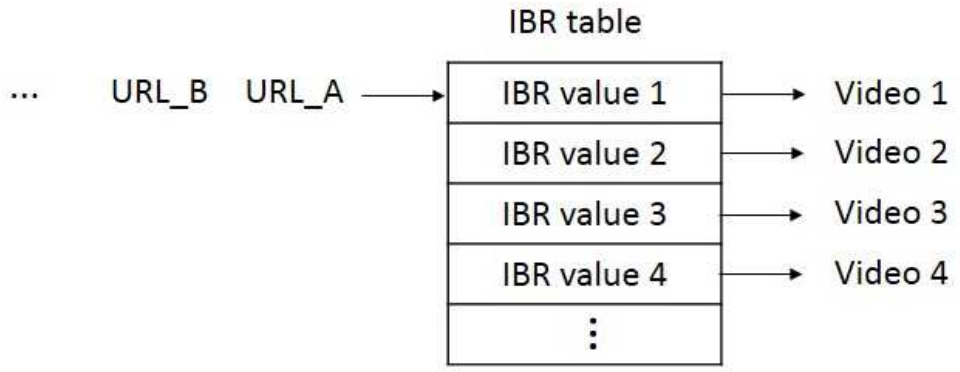

Figure 3.3 In the IBR table, multiple URLs map to one IBR value, which corresponds to exactly one video.

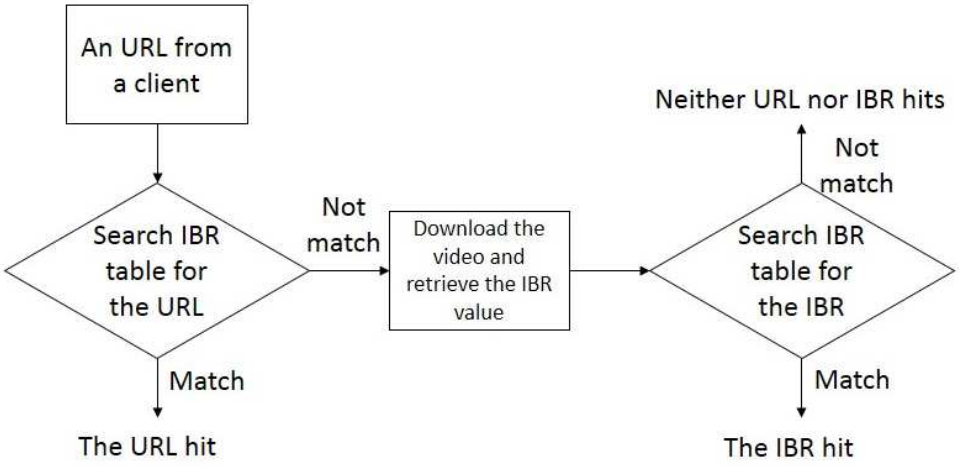

Figure 3.4 The flow of cache matching.

In this section, we discuss the overview of iProxy first, and then look into its key components. The key design requirements are efficient caching and good QoE.

3.2 Efficient Caching in iProxy

iProxy is located between cellular network users and video providers. No modifications are necessary at either end user devices or video providers; so, for example, end users can still use conventional schemes such as search engines and HTTP requests to peruse and retrieve video content. The users only need to set proxy in their browsers.

As mentioned in Section 3.1, we use IBR to identify video content. If videos have similar IBR value, we can say that they are the same content. Thus, URLs corresponding to the videos represent the same content (the IBR value). We need a data structure to store this information.

Therefore, iProxy stores an *IBR table* that maps URLs to IBR values. As shown in Figure 3.3 Each IBR value corresponds to exactly *one* video file. Multiple URLs may map to the same IBR value because these URLs represents videos with the same information.

After receiving a request for a URL, iProxy checks the IBR table first as Figure 3.4 to figure out if the same content has been already stored in the cache. There are three cases that arise:

The URL hits: The URL requested by an end user can be found in the IBR table, which means that this exact URL was requested previously. In this case, iProxy can send the cached video to the end user.

The IBR (eventually) hits: When the requested URL cannot hit any entry in the IBR table, iProxy forwards the request to the corresponding content provider. The provider transmits the requested video, and iProxy transforms it into frequency domain data. The transmission and transformation can be processed in parallel to reduce overhead. iProxy then derives its IBR from this frequency domain data as described in Section 3.1, and we try to match this IBR with other IBR in the IBR table. If a match is found, the same content was requested earlier, but from a different source and/or in a different format. In this case, iProxy compares the quality of the new downloaded video against the in-cache video corresponding to the matched IBR. iProxy keeps the higher quality version and drops the other. Finally, the IBR table is updated by inserting the new URL into the existing entry corresponding to the matched IBR. Using this approach ensures that with a limited cache size, iProxy can cover more and more URLs over time than a conventional proxy, which means its cache can potentially satisfy more requests.

Neither URL nor IBR hits: After video transmission from the provider and frequency domain transformation, the computed IBR may not hit any entry in the IBR table. In this case, a new entry created for the IBR, the requested URL added to the entry, and the frequency domain data for the video is cached. The new entry and the corresponding video need to replace one or more IBR entries and corresponding videos in the cache (discussed in Section 3.2).

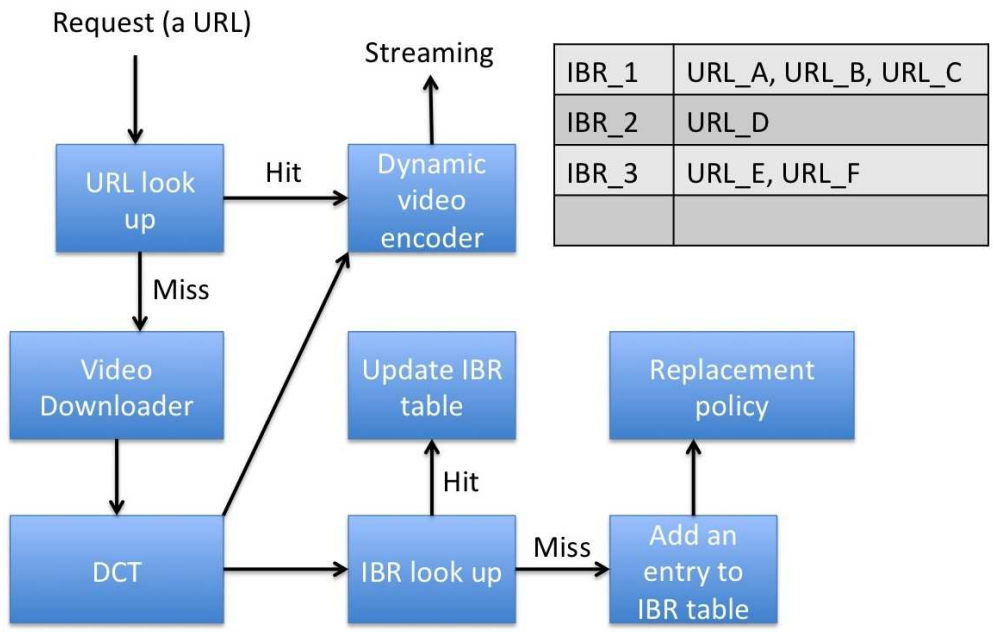

Figure 3.5 The Process of Video Matching.

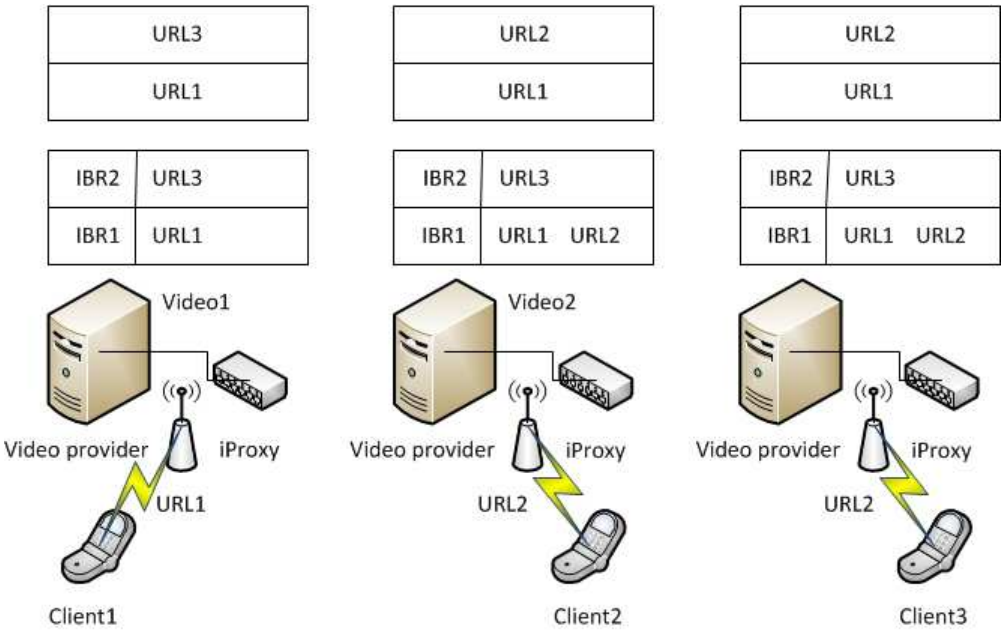

Figure 3.6 A Scenario of iProxy.

Figure 3.5 shows whole process to match video when a request comes in. As mentioned earlier, iProxy is just a middle box between clients and video providers. Clients as usual use URL to

request videos, and iProxy looks the URL up in the IBR table. If the URL is found, we can say that iProxy has already cached this video, and directly streams the video to the client.

However, if the URL is not found, iProxy downloads the video from an original video provider according to the requested URL. The downloaded video is transferred to frequency domain data and an IBR value is retrieved from the data. Then, iProxy does another IBR table lookup. In parallel, iProxy streams the video to client at this time point. If a similar IBR value is found in the IBR table, a video with the same information has been already in cache, so we only keep one of the copies and update the IBR table (add the URL to the corresponding entry). If no video in cache has the same information, iProxy inserts a new entry for this new cached video.

Example: Figure 3.6 compares iProxy to a conventional proxy. In this scenario, there are three smartphones called $Client1$, $Client2$, and $Client3$ which connect to a cellular tower; iProxy is behind the cellular tower. We assume there are several video providers, but Figure 3.6 only presents one of them. Assuming there are two entries in a cache and in beginning one of the entries is occupied by an URL and its content. The upper table is the cache table of a convention proxy and the lower table is the IBR table of iProxy. Both of them use FIFO as their replacement policy.

In Figure 3.6(a), $Client1$ requests a video with $URL1$. $URL1$ is not in IBR table, which maps an URL to a particular IBR value, so iProxy forwards the request to a video provider. After receiving a video named $Video1$, iProxy calculate its IBR value as $IBR1$ which does not exist in IBR table. Thus, the mapping between $URL1$ and $IBR1$ is inserted, and $Video1$ is cached. Then, the video is forwarded to $Client1$ via dynamic video encoding. A convention proxy works similar that insert $URL1$ into its table.

The $URL2$ request from $Client2$ is not in IBR table either in Figure 3.6(b). Thus, iProxy fetches $Video2$ from a video provider and calculate its IBR value, $IBR1$, which is close enough to the IBR value of $Video1$. Thus, we can say that $Video1$ and $Video2$ include the same or similar information. iProxy decides to keep $Video1$ and drop $Video2$. Finally, $URL2$ is inserted into IBR table to show that $URL2$ also maps to $IBR1$. In this case, iProxy does not replace any entry and all of $URL1$, $URL2$, and $URL3$ stay in the table. However, a convention proxy replace $URL3$ by

$URL2$ in this case and waste storage to cache the content for both $URL1$ and $URL2$, because they refer to the same video.

Later, any requests for either $URL1$, $URL2$, or $URL3$ will be hit in iProxy, but $URL3$ cannot be hit in a convention proxy. For example, $Client3$ request a video with $URL2$ and $URL3$. Both of $URL2$ and $URL3$ are in IBR table and maps to $IBR1$ and $IBR3$, respectively, so $Video1$ and $Video3$ will be forwarded to $Client3$ via dynamic video encoder. However, a convention proxy can only hit $URL1$ and needs to request $Video3$ for $URL3$ from a video provider that spends extra communication time and initial delay.

Replacement Policy

Section 3.1 describes typical cache replacement policies. However, iProxy presents some different features that the importance of each entry also depends on the number of URLs associating to this entry and how these URLs are referred. Thus, we modify some of typical cache replacement policy to fit into IBR features.

There are three criteria for the importance of each entry. The first two is the same as regular proxy and the third one is iProxy specific.

- Keep high frequent referred entries.
- Keep most recently referred entries.
- Keep more URLs covered.

To capture these three criteria, we develop a new replacement policy called "LFU-based IBR-score". We give each entry a score which is defined as

$$score_i = \frac{\sum_{URLs}(\frac{C_{hit}}{C_{stay}})}{VideoSize}$$

$score_i$ is the score of $entry_i$ and $\frac{C^u_{hit}}{C^u_{stay}}$ is the sub-score for individual URL u which refers to the IBR value that $entry_i$ represents. C^u_{hit} is the "hit count", i.e., the number of hits for URL u

and C^u_{stay} is the "stay count", i.e., how long URL u has been in cache in terms of the number of total requests arriving at the cache. Thus, $\sum_{u \in URLs}(\frac{C^u_{hit}}{C^u_{stay}})$ captures the contributions of different URLs u to the overall "interest" in the information represented at $entry_i$. We divide the frequency by video size to increase cache hit rate, because larger videos occupy more storage that is a cost. It results that shorter videos tend to get higher scores.

Compared to LFU, LFU-based IBR-score does not have the "cache pollution" problem because it considers how long an entry stays in cache(C^u_{stay}).

GDS is shown that it is more effective than LRU and LFU, but it is not designed for IBR-based cache. LFU-based IBR-score (the number of access per URL for an IBR table entry) outperforms GDS (the number of access per IBR table entry), because it captures more detail information that means that different encoding/formats and different content providers (associate with different URLs) may provide different access pattern as Figure 3.7. Thus, LFU-based IBR-score shows better prediction for future access. Larger objects occupy more storage space, so as GDS our scheme treats video size as a cost.

We take a scenario shown in Figure 3.8 as an example. There are two candidate entries to replace in an IBR table. One of them includes two URLs that are URL_1 and URL_2, and the other has another two URLs that are URL_3 and URL_4. There are nine access slots, and the URLs are accessed in different slots. We assume both entries associate with the videos with the same size, and we only consider access frequency in this scenario.

We compare LFU-based IBR-score and GDS to see how they choose an entry to replace. Both of them calculate access frequency as score and pick the entry with lower score. GDS considers whole entry and doesn't care about each individual URL. Thus, it scores $Entry_1$ as $\frac{4}{9}$ and $Entry_2$ as $\frac{4}{7}$, so it will choose $Entry_1$ to replace. URL_1 affects the score of whole entry a lot, because it was accessed long time ago. However, URL_2 belonging to $Entry_1$ is accessed more frequently than others recently, so URL_2 should be kept in cache that implies we should replace $Entry_2$ instead of $Entry_1$.

However, LFU-based IBR-score calculates the access frequency of each URLs separately, so it can reduce the effect of URL_1. In this scenario, LFU-based IBR-score scores $Entry_1$ as $\frac{10}{9}$ and

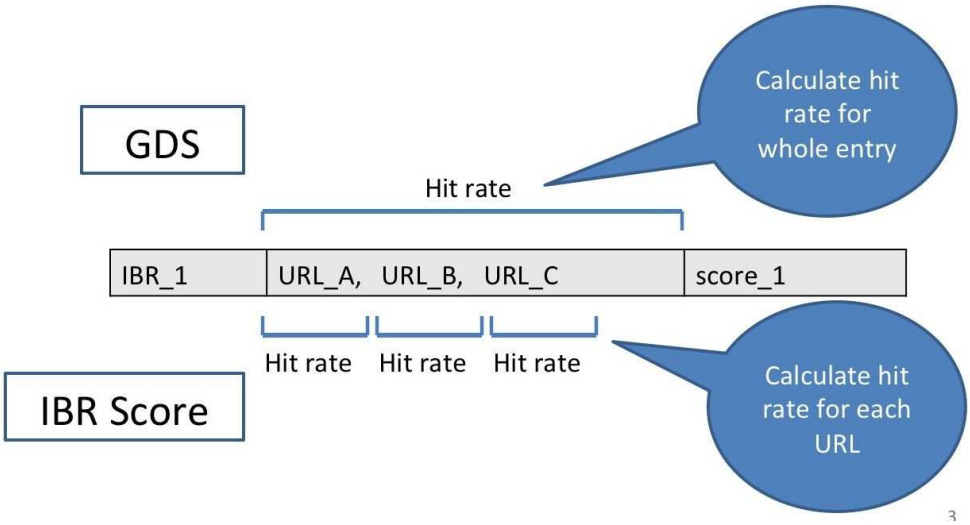

Figure 3.7 IBR score V.S. GDS.

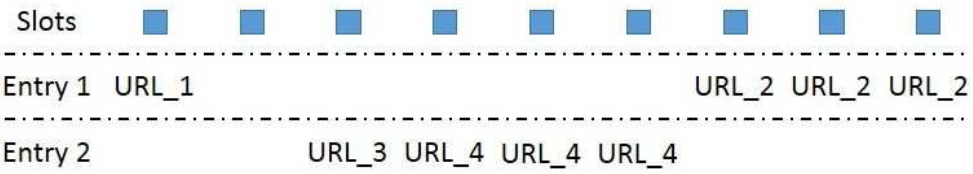

Figure 3.8 An comparison of IBR score and GDS.

$Entry_2$ as $\frac{9}{14}$. Therefore, LFU-based IBR-score selects $Entry_2$ with lower scores to replace. In this way, URL_2 with high recent access frequency stays in the cache.

We also design a modification to LRU called "LRU-based IBR-score" and it is computed as $score_i = \frac{\sum_{URLs}(\frac{1}{C_{last}})}{VideoSize}$. C_{last} means how long the last hit for a URL has been in the cache in terms of the number of total requests to the cache. $\frac{1}{C_{last}}$ is the sub-score for an individual URL. Higher $\frac{1}{C_{last}}$ means this URL is hit more recently. As the earlier scheme, this simultaneously prioritizes most recently referred entries and those that cover more URLs.

Cache Deployment

Figure 3.9 shows a 3G cellular network architecture. There are some potential candidates to locate iProxy, which are radio network controller (RNC), serving GPRS support node (SGSN),

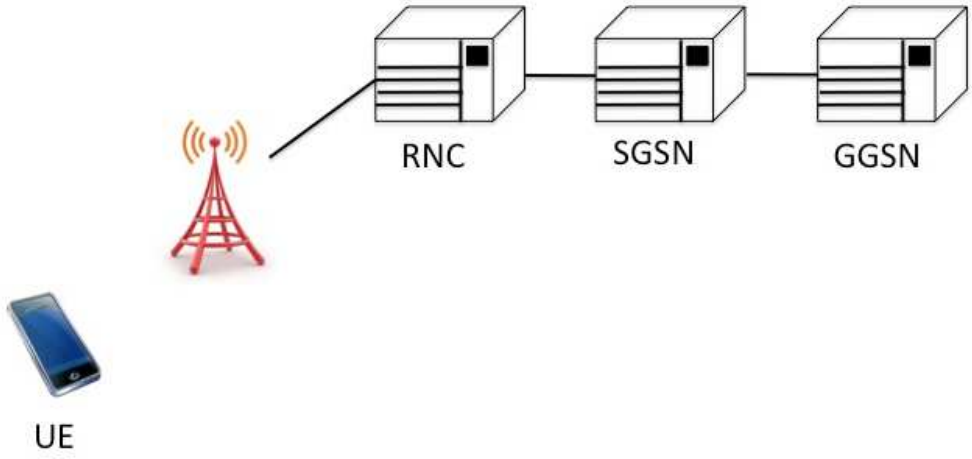

Figure 3.9 iProxy Cellular network architecture.

and gateway GPRS support node (GGSN). RNC is closer to user equipment(UE), so the latency between RNC and UE is lower. However, RNC can serve less users, because it is in the lower layer of the architecture. The number of served users affects the amount useful data a proxy can cache that determines hit rate. Thus, locating at RNC may provide lower hit rate. GGSN is far away from UE, so it can provide higher hit rate with higher latency from it to UE. Thus, the location is a trade-off between latency and cache hit rate.

Cache hit rate is affected by served population. However, the population each cellular network component covers is varying and depends on its service area. For example, the cellular network components in urban may serve more population than the ones in rural. In Section 3.4 we evaluate the trade-off between hit rate and latency.

Many cellular providers already employ transcoding proxies [53] that transform web content to a suitable format for mobile device users to save bandwidth and computation resources. For example, the transcoding proxies reduce image resolution for smartphone users. However, they dont dynamically adapt video according to channel quality. Thus, iProxy can work with them in parallel. iProxy focus on video content, and transcoding proxies work on other content.

3.3 Optimizing QoE

As mentioned in Section 3.1, iProxy should provide better QoE for users. There are three aspects: ensuring high bit rate, lower buffering time, and lower start-up delay. Good QoE can maximize user engagement and minimize abandonment. To this end, iProxy provides dynamic video encoding which can dynamically change video bit rate during streaming and video format according to user devices. Our goal is to stream videos with high bit rate without using out bandwidth.

Bit Rate and Buffering: Handing Channel Diversity

3G Network condition such as channel quality, traffic congestion, signal interference may vary in different location and different time [54, 18]. Available bandwidth is highly varying and the dynamic range can change between Kbps to several Mbps within 10s. To provide better QoE, iProxy needs to ensure higher bit rate without using out bandwidth.

Current video streaming either sets up bit rate only when start up or reacts to network condition change slowly and discretely. In some cases, clients can decide video quality before or during playback. For example, Youtube provides different video resolutions from 240p, 360p to 1080p for some videos. It can also detect available bandwidth and automatically choose appropriate resolution to stream. However, they cannot change the bit rate smoothly and the change is discretely that causes performance cliff problem.

Besides, because of client diversity, end users need and can play different versions of encoded videos such as particular video resolution and formats. Dynamic video encoding can encode video according to client requirements, instead of storing every possible versions in cache. Storage can be used more efficiently.

iProxy provides a novel approach to dynamically encode videos to suitable bit rates and formats. This approach do not need to cooperate with CDN servers, and do not require modification at both content provider side and client side.

Before describing our framework, we give some background on MPEG 4-based video encoding whom our dynamic encoder is based on.

Mpeg 4-based Video Encoding Figure 3.11 presents a regular Mpeg 4 video encoding process. The whole process can be divided into discrete cosine transform(DCT), scale, quantization, motion estimation, and entropy coding.

DCT [55, 56] computes frequency domain data for video content. More important data is located in low frequency components and data including more detail is located in high frequency components. DCT is widely used in lossy compression multimedia content such as MP3 and JPEG.

Quantization determines the bit rate of video stream. The quantization process is following DCT and compresses a range of continuous values into one discrete value. Usually we give more bits to represent low frequency components, which are more important, and less bits to represent high frequency components, which are less important. By different quantization strategies, we can determine the bit rate.

Motion estimation can further reduce video size. Before compressing a video frame, we cut a video frame into multiple blocks and match these blocks with other blocks in the same, previous, or following frames. If some blocks are matched, we only keep their reference pointers (also called motion vectors) instead of blocks themselves. Frames can be categorized into I-frames, P-frames, and B-frames. I-frames do not refer to any other frames, P-frames refer to earlier frames (usually I-frames), and B-frames refer to earlier or following frames as shown in Figure 3.10.

Finally, we code all data including the output from quantization and motion estimation using entropy coding. Entropy coding gathers statistic about how offer each symbol appears. High frequency symbols are given shorter codes for representation. Thus, in average the length of code is reduced which decrease the size of encoded videos.

In this paper, we provide a dynamic video encoding which can encode video dynamically into different bit rate or format according to what clients need.

Dynamic video encoding: a linear bit rate adapter: To provide high QoE in cellular networks with channel diversity, iProxy proposes a flexible video bit rate that is closer to available bandwidth and suitable video formats to match clients' requirements.

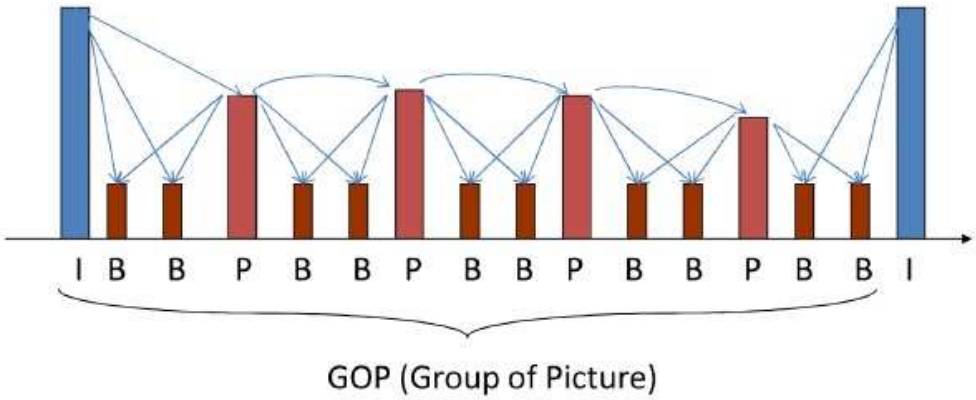

Figure 3.10 Video frames refer to other frames to reduce video size.

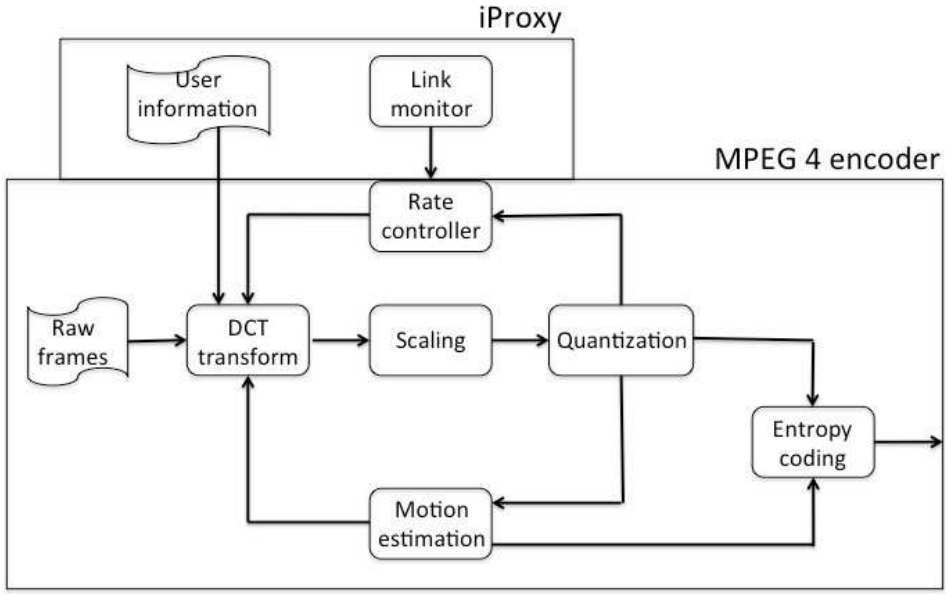

Figure 3.11 iProxy the flow of mpeg4 encoding.

Many studies use layered video encoding for the channel and client diversity issues such as MPEG DASH. MPEG DASH pre-encodes a video into several versions with different bit rates and works with a http server. Each video version is cut into several chunks. Clients measure the

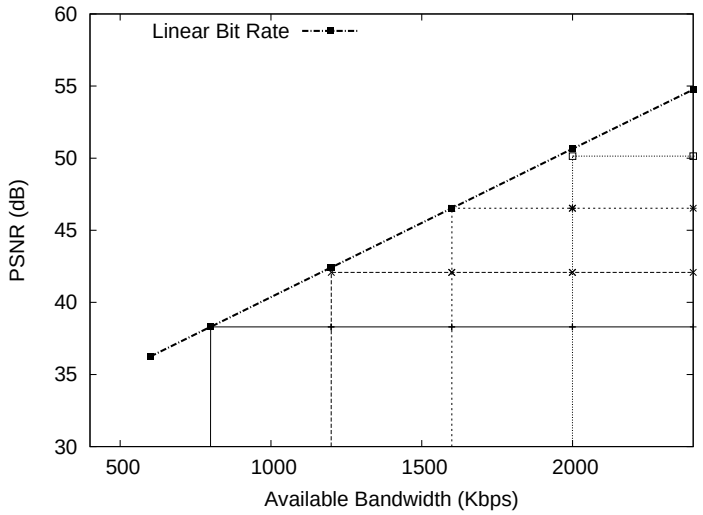

Figure 3.12 An Example of Performance Cliff Problem.

performance of a current chunk and select the next chunk from different versions. Thus, clients can dynamically change video bit rates.

However, MPEG DASH encounters Performance cliff [15] problem that causes a waste of network resources. For example, there are only six versions of videos which provide 38 dB, 42 dB, 46 dB, and 50 dB in peak signal-to-noise ratio(PSNR) as shown in Figure 3.12, but require channel quality as high as 5 dB, 9 dB, 13 dB, and 17 dB in SNR. we can only serve clients one of them in any particular time. Therefore, when channel quality is 7dB, network can accept a video with 40dB quality. However, we can only transmit a video with lower 38 dB quality; sending 42dB video causes buffering because bandwidth is not enough. Both of the cases result in poor QoE. In addition, it is hard to predetermine a set of versions that will work for all possible mobile clients and MPEG DASH needs $k\times$ storage space than in iProxy to store k possible versions.

To fully use channel resources, iProxy provides a linear bit rate adapter that dynamically encodes videos. Thus, the bit rate of encoded videos can fit any amount of available bandwidth as the strictly rising line in Figure 3.12. According to channel condition, it can dynamically encode video with suitable bit rate.

Host: gdata.youtube.com\r\n

Connection: Keep-Alive\r\n

User-Agent: com.google.android.youtube/2.3.4(Linux; U; Android 2.2.1; zh_TW; HTC Desire Build/FRG83D) g

Table 3.1 http header.

At a high level, the linear bit rate adapter works as follows: The original Mpeg 4 encoder uses a rate controller. The rate controller takes a particular bit rate and feedback from a quantization module as inputs. Then, it calculates and adjusts appropriate parameters for the quantization module to output video stream with this bit rate. iProxy adds two modules to the origial MPEG 4 encoder that are a link monitor module and an user information module as Figure 3.11. The link monitor module keeps monitoring available bandwidth during streaming and user information module collects the information of user devices. These two modules provide parameters for MPEG 4 encoder to determine a right bit rate and format.

To increase the performance of dynamic video encoding, we cache frequency domain data and motion estimation vectors for each video instead of caching encoded videos. The advantage is we can skip DCT when dynamic video encoding.

When receiving a video from a video provider, iProxy transforms the video into frequency domain data and retrieves its motion vectors. As showed in Section 3.1, frequency domain data is also needed during IBR value calculation. Thus, caching frequency domain data and IBR value calculation can be done simultaneously.

Different from other mobile video schemes, iProxy don't need any modification at client and video provider sides. In addition, it don't rely on physical layer support. Consequently, it is easier to deploy in existing infrastructures.

Baseline bit rate using in-context information: A baseline bit rate should be chosen before dynamic bit rate adapting. The baseline bit rate may be picked up or reset each time we start a streaming or there is a significant change in the mobile device context. For example, mobile devices may change associated cell towers or cell sectors during streaming. The change can result

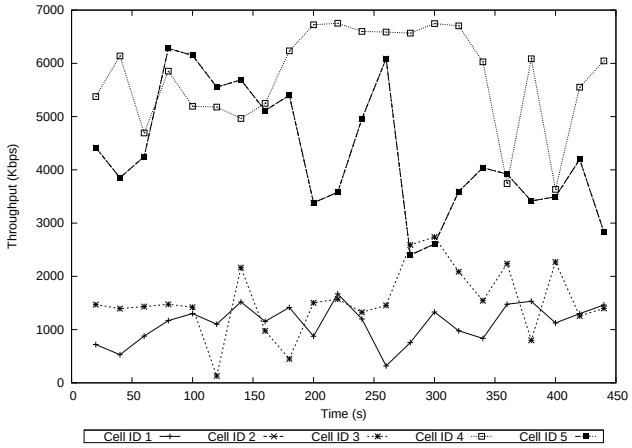

Figure 3.13 Throughput is varied with different CellID/location.

significant available bandwidth change, and we can adapt the bit rate quickly based on the in-context information. We build a history record about this in-context information. When iProxy start to stream a video, the history record can be applied to determine the baseline bit rate.

[54] shows how in-context information affects network performance. We also measure the throughput of 3G networks with five different cell sectors/locations as shown in Figure 3.13. Distance between any two locations are about 2 miles on average. We can observe that average throughput in different locations can be as large as 4000 Kbps. Thus, we can set a baseline bit rate according to the cellID that clients communicate with.

TCP information feedback After picking up a baseline bit rate, we must adapt the bit rate to fit achievable throughput, which is varying during streaming. For this, we rely on per-packet TCP feedback; iProxy collects TCP information from network stack such as the congestion window (CWND) and RTT. This information is a good indicator that shows how congestion a link is. We can estimate available bandwidth as the following equation.

$$Available\_bw = \frac{CWND \times segment\_size}{RTT}$$

Finally, iProxy uses an exponentially-weighted moving average (EWMA) of $Available\_bw$ to compute a suitable bit rate. EWMA can avoid rapid bit rate shift. We set the alpha to 0.9, which

makes bit rate change more smoothly. This bit rate is sent to the rate controller module in Figure 3.11. iProxy also monitors the changes of cell towers clients communicate to. Baseline bit rate will be reset, if the changes are detected. If the expected bit rate do not exactly match real network bandwidth, we rely on buffer at client side to smooth the videos. Compared to conventional video service, our solution uses the buffer more efficiently and reduces the chance to use out the buffer. Cellular providers may not want video streaming use out bandwidth, so they can set a max bandwidth for each streaming according to their QoS strategies.

Access control: Video providers may prohibit clients from accessing high resolution video, if the clients pay less. To realize this access control, we add an access control layer above iProxy that can set a max resolution for each URL in IBR table or client request. This max resolution is one of the input of dynamic video encoder and is an upper bound when the encoder adapts bit rate.

Low Start-up Time: Accommodating Client Diversity

Because of client diversity, clients may use varying devices with different operating systems, screen resolution, and decoders. The video formats clients can deal with may be inconsistent. Thus, to stream compatible videos, client specific information is required by dynamic video encoding.

Operating systems: One of our goals is to intelligently provides end users videos with suitable formats to improve video quality and compatibility. The major operating systems running in mobile devices include Android, iOS, and windows mobile version. The videos each operating system can handle are not totally the same. For example, iOS cannot support flash relative format such as .flv files.

Most of video requests use Hypertext Transfer Protocol (http). A http header provide some useful information including an operating system version as Table 3.1.

In this example, the field User-Agent shows client's operating system version, which is Android 2.2.1 and its device model, which is HTC Desire.

Screen resolution: The common screen resolutions of smartphones vary from 240x320 to 720x1280 as shown in [57] which is the maximal resolution a smartphone can show. Thus, it

is unreasonable to send a smartphone videos with resolution larger than the smartphone can play, because larger resolution means higher bit rate that consumes more bandwidth and smartphones take longer time to pre-process high bit rate videos before the video plays. The smartphone needs to shrink video resolution to fit its screen resolution. It wastes network bandwidth and increases start-up latency.

As showed earlier, we can retrieve device model from packet headers and the screen resolution of any particular model is fixed. This is the upper bound of resolution iProxy should provide to end users. It means we may reduce the resolution if the screen resolution cannot support high resolution videos.

Decoder: iProxy should deliver compatible video formats to end users that means we should ascertain what end users can decode. The videos end users can decode are decided by the decoder end users install.

Operating systems in-built support some video decoders. For example, as shown in [58], Android 2.3.3 supports 3GPP(.3gp), MPEG-4(.mp4), MPEG-TS(.ts), and webM(.webm). Different operating systems may support different video formats. Thus, the operating systems end users use give us an important hint what video formats they can decode.

However, end users may install other extra decoders. For instance, end users can install flash player to play .flv videos. This information can be provided by application markets such as Google Play or 3G providers.

Device power In [59], smartphones consume more power to decode higher resolution videos. The smartphone HTC Touch HD consumes two times more power to decode DivX video in VGA resolution than to decode DivX videos in CIF resolution. Decoding videos in CIF also consume more power than decoding videos in QCIF. Moreover, video formats affect power consuming, too. For example, videos in H.264 format save more power than videos in DivX.

Therefore, iProxy should deliver videos in suitable formats and resolution depending on device power level, especially when power is low. However, retrieving power level without modifying end users is difficult. It is helpful to let end users insert power level into http request, so iProxy can decide how to encode videos according to this power level.

When the power level drops under a threshold, dynamic video encoder in iProxy encodes videos in lower resolution which may be lower than the screen resolution of end user's devices. It is a reasonable strategy to keep mobile devices longer when video playback.

Applicability to Prefetching

While iProxy operates as a cache, it is also applicable to prefetching which is becoming popular in the mobile video context. In particular, the cellular provider can prefetch high quality versions of videos that users are likely to access and store the corresponding frequency domain representations in iProxy, just as above. For each entry created in this manner, the provider can also obtain a list of URLs where alternate versions of the video are stored; this could be done by crawling the Web and identifying the duplicates of the stored entries using the IBR matching algorithms described in Section 3.1. Subsequently, iProxy operates just as described above both in terms of how storage is managed and how content is streamed.

Scalability

In Section 3.4, we show that the time needed to dynamic-encoded video is 42 times shorter than the video's length, so one single machine can support multiple clients at the same time. To support even more clients simultaneously, we can leverage parallelism, as shown in Figure 3.14. Here, a single cache tries to match requested videos and download unmatched videos. In addition, there are multiple nodes that work on dynamic video encoding. Encoding workload can be spread among these nodes. All requests are sent to the cache first and, after obtaining the requested videos, each video is redirected to one of the dynamic video encoding nodes. The dynamic encoding nodes finally transmit videos to clients with a specific video format. Cellular network providers can add dynamic video encoding nodes based on expected total workload.

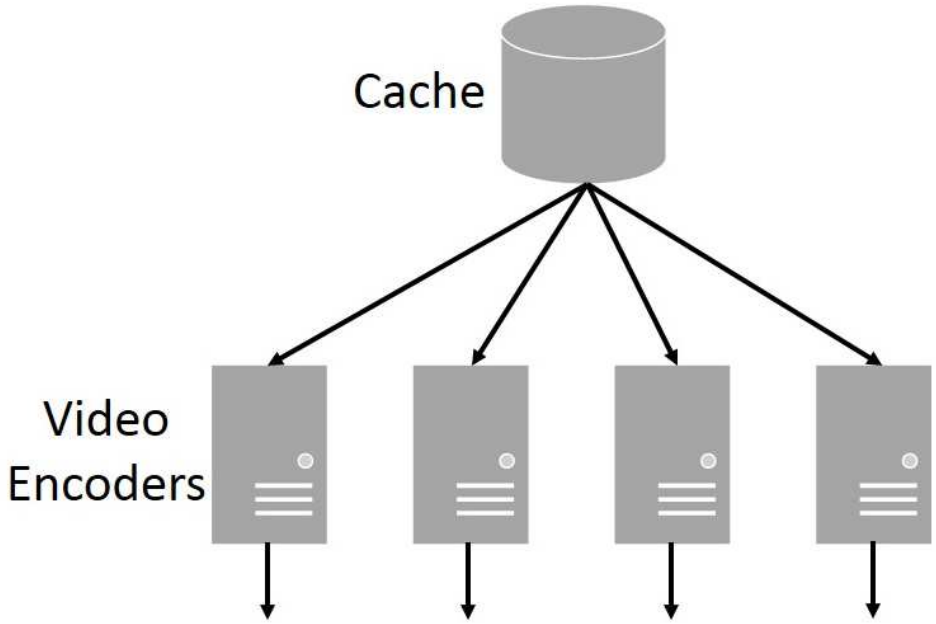

Figure 3.14 Multiple dynamic video encoding nodes can be added to improve scalability.

3.4 Evaluation

Our evaluation addresses the following issues: (a) **Caching**: How do different replacement policies compare and how important is info-awareness? How much cache storage is needed? (b) **Handling diversity:** To what extent does selecting the right resolution and encoding help improve start up times? (c) **Dynamic encoding**: How well does dynamic encoding adapt to changing conditions? Does it lead to improvements in quality metrics such as buffering rate and bit rate that impact end-user QoE?

We first describe our prototype implementation.

Implementation

At the client side, we use *unmodified* Android smartphones, running VPlayer [60]. VPlayer can send a request with a URL to iProxy through a 3G connection.

We implemented iProxy in a 4-core desktop with Intel(R) Core(TM)2 Quad 2.66GHz CPU Q6700 and 8GB RAM. Our prototype proxy, written in C, can fetch videos from video providers, calculate IBRs, compare IBRs, cache raw video data, match videos, and dynamically encode videos. We modified the pHash library [61] to calculate IBRs.

For each request, our proxy iProxy checks its IBR table; we use a locality-sensitive hashing (LSH) [62] based index to aid fast lookups at scale. If the URL is not in the table, iProxy downloads the video from an original website and redirects it to the client. After the video is downloaded, we derive its IBR, and compare this IBR to others in cache (using LSH to retrieve candidates, and then obtaining nearest matches). If we find a matched IBR, we delete the copy with lower quality and update the IBR table by adding the new URL.

The video is then sent to a modified FFmpeg [63] module that supports our dynamic video encoding. It interprets requests from clients to determine client-side constraints. FFmpeg uses FFserver [64] to stream the video to VPlayer in the client's device. During streaming, the FFserver module retrieves dynamic CWND and RTT by reading the relevant network stack variables and calculates suitable video bit rates to use. It then sends this information to FFmpeg which adapts bit rate.

Traces

Our analysis of iProxy relies on two sets of real traffic traces. Our first trace is the Web video data set [65] we used the motivating statistics in Section 3.1. This data set had a total of 10,000 videos corresponding to the search results for the top 25 queries at three popular video content providers. This amounts to about 400 URLs per query: note that the videos corresponding to some of the URLs corresponding to a query may point to the same underlying information, whereas others may correspond to "related" videos that have entirely different information altogether.

In addition, we leverage packet traces collected over the University of Wisconsin's Wireless Network. The traces span three days, from April 26, 2010 (Monday) to April 28, 2010 (Wednesday). We prune the traces to only capture the subset of devices that are known to be smartphones

or tablets, using the techniques identified in [2]; the techniques have known false negatives (but minimal false positives), meaning that we don't capture all handheld devices. Furthermore, we focus mainly on the trace subset containing the HTTP protocol. For each packet, we collect the first 128B, which includes the HTTP header. The total size of the videos is about 300GB.

We identify video traffic based on the "content type" field in the HTTP header, similar to [2]. We derive URLs from the HTTP requests for videos, and download the corresponding video files in their entirety. The total size of the videos is about 40 GB. 58% of these videos are duplicated requests according to their URLs (their URLs are them same as another videos in the data set). In addition, 21.5% videos are unique videos that means 78.5% videos includes the same content with other videos.

Caching: Hit Rates, Storage, Deployment

Replacement Policies: To evaluate the performance of our IBR specific cache replacement policies, we can compare it with other traditional cache replacement policies, including FIFO, Least Recently Used (LRU), Least Frequently Used (LFU), and Greedy Dual Size-Frequency (GDS) [66]. Unless otherwise specified, we assume that the underlying cache is IBR-based. We use hit rate (HR) as the main metric to compare the policies. We use different cache size to do the evaluation that are 20 GBytes, 15 GBytes, 12 GBytes, 8GBytes, and 4GBytes. We also compare iProxy with conventional proxy with GDS policy marked as "No IBR".

Results from our analysis of the university data set are shown in Figure 3.15. If we use a cache as large as 20 GB, iProxy with any replacement strategy can catch almost all duplicate URLs. However, conventional proxy (marked as No IBR in Figure 3.15) cannot (1.8% less than iProxy), because they don't use cache space efficiently. When cache size gets smaller, the performance difference of replacement policies becomes more obvious. If we just use FIFO, the hit rate is even worse than a conventional proxy, when cache size is less than 12 GB. The conventional proxy use GDS policy that works much better than FIFO, so iProxy with FIFO loss its benefit.

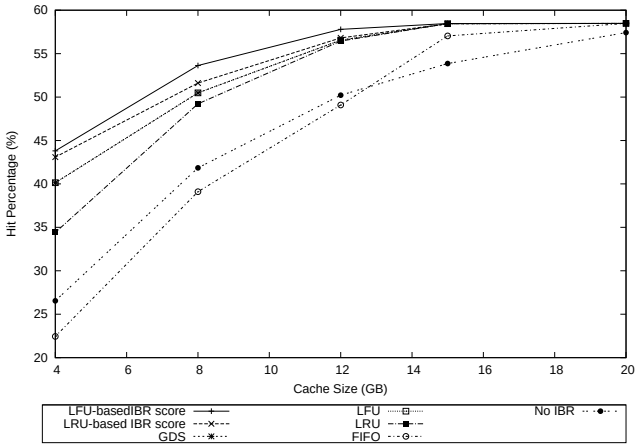

Figure 3.15 Hit rates for various cache policies in iProxy.

Other replacement policies can get benefit from IBR-based iProxy. When cache size is larger than 12 GB, they achieve similar performance as 20 GB cache size. Even when cache size is as small as 8 GB, iProxy still can catch 92 % of duplicate requests. Among these policies, our newly designed LFU-based IBR score works better than others, especially when cache size is small. LFU-based IBR score is 7 % better than GDS and 28 % better than conventional proxy with 8 GB cache size. When the cache size shirks to 4 GB, LFU-based IBR score is 10 % better than GDS and 65 % better than conventional proxy.

In summary, iProxy can improve cache hit rate as much as 65 % when cache size is small. In addition, with our newly designed policies (LFU-based IBR score and LRU-based IBR score) can extend the benefit, because they are aware of the importance of information as captured by the hit rate across multiple related URLs. This shows that we need to use an info-aware replacement policies to realize the benefits of iProxy.

The rest of the paper assumes the LFU-based IBR-score policy is employed in iProxy with 12GB cache.

A closer look at hit rate: To obtain better insights into the results above, we now examine: (1) hit rate over time and (2) the coverage of URLs. We compare iProxy with a conventional proxy that uses URLs to identify videos and the GDS replacement policy which prior works have identified to

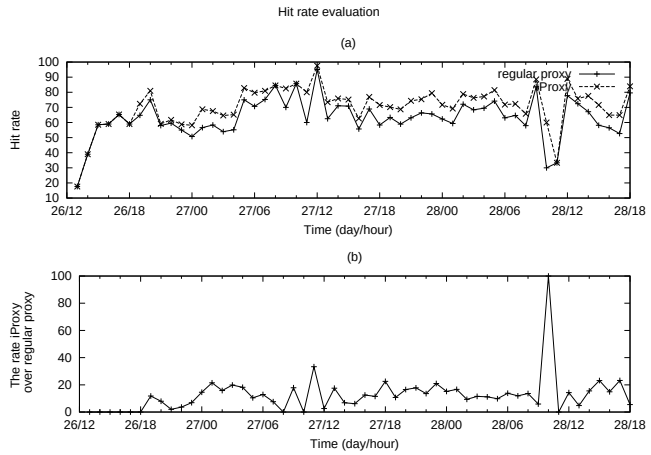

Figure 3.16 Hit rate evolution. Y-axis is time: 28/06 refers to 6AM on April 28th.

be optimal for conventional caches. Figure 3.16 shows the hit rates of the two caches. The y-axis represents time in day/hour format.

We write a proxy emulator by C++ and feed the same university trace file as Section 3.1 into it and count how many requests can be hit by iProxy and a convention proxy. Proxy setup is also the same as Section 3.1.

Before 18:00 April 26, the cache is not full, so we cannot see any benefit from using iProxy. After that iProxy can perform better than a conventional proxy. The improvement ranges from 0-20%. In addition, the highest benefit we can see is 100% HR improvement which happens at 10:00 April 28.

Larger URL coverage benefits hit rate improvement. To show this fact, in Figure 3.17, we compare URL coverage between the conventional proxy and iProxy. The URL coverage of iProxy increases dramatically compared to conventional proxy just after the cache has warmed up, because until this time most entries only cover a single URL in iProxy. Soon, many of them are replaced by new entries with multiple URLs. Over time, some entries with multiple URLs may be replaced, so we cannot see significant URL coverage increases, but the coverage continues to be better than a conventional proxy.

Storage Requirement: Recall that iProxy stores the frequency domain (DCT) data for the highest quality video each IBR entry. What storage cost does this impose? As exemplified in

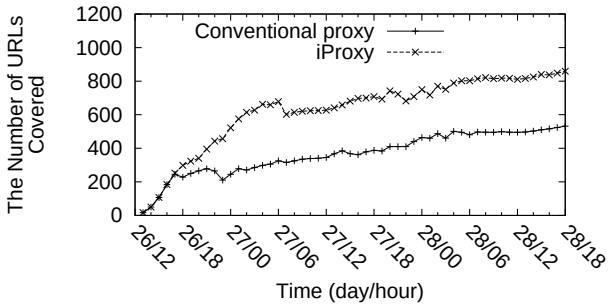

Figure 3.17 URL coverage evaluation

Table 3.2, DCT data is larger than individual video files (we show only four different bit rates for a single example video), but: (1) the difference is not significant compared to any single video file (2.1X in the worst case) and (2) the DCT data size is 2.4X smaller than the sum of the sizes of the individual video files, meaning that iProxy outperforms the naive strategy of storing multiple versions.

Deployment Evaluation: As mentioned in Section 3.2, the performance of iProxy depends on where we deploy it and what is the served population. If we put iProxy closer to clients, the latency is lower and performance better. However, less served population means lesser overlap in accesses which affects hit rate. We study this question using an emulation based on our traces.

To understand this, we first measure the throughput between a machine at our university and the nearest YouTube server, which determines the cost of a cache miss and the relative improvement from a hit. We calculate mean and variance, and model the cost as a normal distribution. In addition, we use the data in [67] which shows the delay to RNCs, to SGSNs, and to GGSNs from UEs in a tier-1 cellular network.

We identified a total of 1400 unique users in our traces. We divide them into smaller groups and associate each group with a node in the same level of the cellular hierarchy. We compute and report the average throughput across all the groups. The results are shown in Table 3.3. In this table, the entry corresponding to row “500” and column “SGSN,” means that we divide the 1400 users randomly into three groups of 500, 500, and 400, and “assign” them to 3 SGSNs. We emulate requests of UEs in each group, identify cache hits/misses and compute the effective

Video bit rate	Size
885 kbps	3744 KB
1063 kbps	4588 KB
1277 kbps	5403 KB
1385 kbps	5862 KB
DCT data	8154 KB

Table 3.2 For a single example video, we show the size of the raw data stored with an iProxy cache entry, vs. that of different formats of the video file.

throughput observed by the clients. Each miss goes to the YouTube server, whereas a hit is served from cache.

Our goal is not to show that one location is clearly better than the other, but rather to show the trade-off between population density, iProxy location and performance, helping the cellular providers make the appropriate choices. In general, our results show that when the population density is low (e.g., 300 users per RNC, 800 or more at SGSN, and 1400 or more at GGSN), aggregating a few number of iProxy's at the SGSN and GGSN is reasonable. When the population density is high (e.g., 1400 users at an RNC), deploying at the RNCs is a good choice.

QoE

The setup we use to test iProxy's ability to support good QoE is shown in Figure 3.18. We stream videos from a iProxy proxy located on our campus to mobile phones also located close by. However, the transfers themselves traverse the Internet and the cellular backbone before reaching the end device. In practice the proxy would be deployed a lot closer to clients; thus, our experiments below show a lower bound on the effectiveness of iProxy in ensuring QoE.

Start Up Time

We setup an experiment to show how iProxy improves start up latency. Here, we assume a set-up where users click on embedded links to videos, e.g., links in emails, blogs, etc.

Population	RNC	SGSN	GGSN
1400	1373.55 kbps	1303.9 kbps	1271.93 kbps
800	1282.18 kbps	1259.99 kbps	1176.55 kbps
500	1244.51 kbps	1194.67 kbps	1182.06 kbps
300	1190.22 kbps	1165.02 kbps	1123.62 kbps

Table 3.3 Throughput vs. location and population.

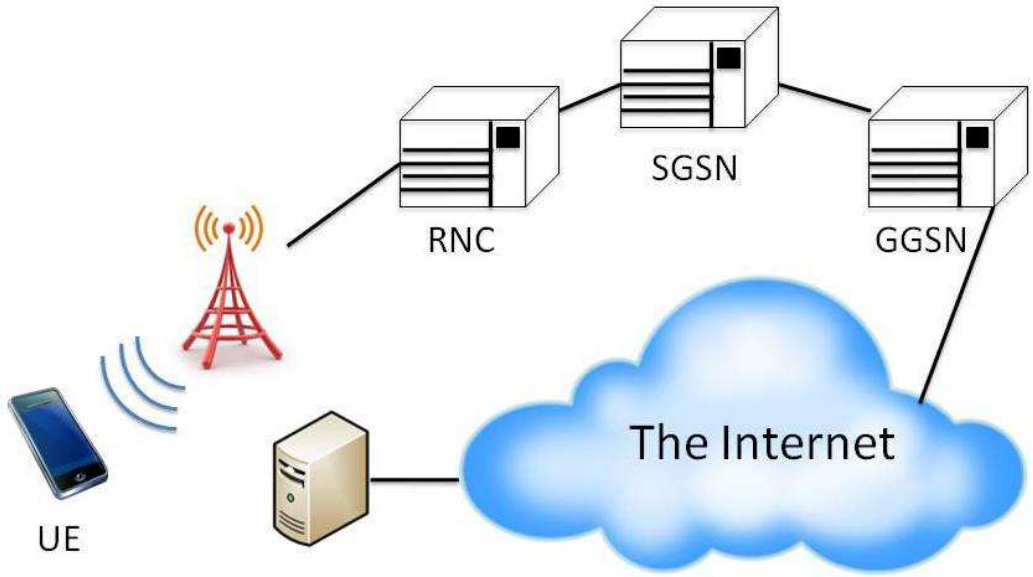

Figure 3.18 Experimental scenario for video performance.

In Table 3.4, we stream three different videos to an Android smart phone (Samsung GALAXY SII) with a 480×800 screen. The first video is in VGA (640×480 resolution, .flv) from Dailymotion, the second one is in XGA (1024×768 resolution, .flv) from YouTube, and the third one is in 360×288 (.asf) resolution from Yahoo! Video. The first column shows the improvement in video start up latency in seconds in the case where both iProxy and a conventional proxy observe a cache hit for the video request. The second column shows the improvement when only iProxy observes a cache hit.

	Daily-motion	YouTube (HD)	Yahoo Video (.asf)
Both hit	0s	13s	∞
iProxy hit	2s	14s	∞

Table 3.4 Improvement in video start up latency using iProxy and a conventional proxy. ∞ means the client cannot play the original video format, but can play re-encoded video from iProxy.

The video from Dailymotion is in a suitable format and resolution for client's device, so when both two proxies see a cache hit, iProxy does not offer any improvement. However, when a conventional proxy does not see a cache hit, it takes two extra seconds to retrieve the video from Dailymotion, delay start-up correspondingly.

The resolution of the second video from YouTube is much higher than the smartphone can play, causing the phone to spend time filling its cache and pre-process the video. Furthermore, because of the resolution mismatch, the XGA video appears pixelated on the smartphone. iProxy lowers start up latency by 13s; it also converts the video into a more suitable resolution (VGA).

The smartphone simply cannot open and play the third (.asf) video file. iProxy converts the video into a suitable format (.mpg), so the smartphone can still play it.

Bit Rate and Buffering

Speed: To support user video playback without lag, the time needed to dynamically encode a video from raw data should be small, otherwise users may face stalls waiting for encoded video data. To study this, we use modified ffmpeg to encode video into different bit rates to observe encoding speed compared to video length in seconds. We choose a 590s video and encode it with bit rates from 200 kbps to 1000 kbps. As shown in Table 3.5, encoding times are similar no matter the bit rate, and are 42 times shorter than the video length. This provides evidence that dynamic video encoding can work on-line, matching real-time video playback requirements. This is further evidenced by our experiments below.

at a fixed rate picked based on initial measurements, a common mode of operation even for high-end video content servers today. We also compare two alternative solutions: (1) we choose a

Video length	586.98 sec
200 kbps	13.34 sec
400 kbps	13.94 sec
600 kbps	14.03 sec
800 kbps	14.36 sec
1000 kbps	14.54 sec

Table 3.5 Video length and encoding time

Bandwidth	Fixed-rate	Startup	Dynamic
400 kbps	15fps/35s	24fps/9s	30fps/0s
500 kbps	19fps/21s	29fps/1s	30fps/0s
600 kbps	21fps/15s	26fps/5s	30fps/0s
700 kbps	26fps/5s	27fps/4s	30fps/0s

Table 3.6 We measure average frame rate and total stall time (frame rate/stall time) during streaming in three cases: (a) fixed bit rate, (b) choosing bit rate at startup time, and (c) dynamic bit rate adapting.

bit rate at startup time based on in-context information and stick to this bit rate during streaming, and (2) we dynamically adapt bit rate during streaming by collecting TCP feedback. We compare three video streaming schemes that are fit rate, choose bit rate at startup time, and dynamic bit rate adapting. The test video is encoded as 800 Mbps in bit rate, and 30 fps.

Table 3.6 shows that fixed-rate mode is dramatically affected by available bandwidth. It keeps bit rate fixed at 700 kbps, so when available bandwidth drops, frames of the video cannot arrive at the smartphone on time which forces frame rate to drop. It can keep 26 frames per second when bandwidth is higher than 700 kbps. However, the frame rate drops a lot at lower bandwidth, and it can reach only 15 frames per second with bandwidth at 400 kbps.

If we choose a bit rate at startup time based on in-context information, its bandwidth requirement is closer to available bandwidth, so the video streaming can keep higher frame rate and lower stall time than fixed-rate mode. However, we still can observe some frame rate drop and stall time.

iProxy can change bit rate dynamically according to available bandwidth; thus, it still keeps high frame rate when bandwidth drops. It can achieve more than 30 frames per second even when bandwidth is as low as 400 kbps.

We also measure total buffering time, a key indicator of user engagement Table 3.6 shows that iProxy can provide video with almost no buffering time, but fixed-rate video still suffers some buffering even with bandwidth as high as 700 kbps. Thus, iProxy can provide stable video playback even if bandwidth is low.

To measure the efficacy of our linear bit rate adapter in improving QoE, we experiment with scenarios where we estimate how well iProxy functions with rapid changes in network conditions. In each scenario we first send a video around 50 seconds long with a starting bit rate of 800 kbps to a smartphone over 3G. Every few seconds, a bandwidth shaper kicks in at the desktop where iProxy is running to change the available bandwidth according to a pre-defined pattern. Note that the available bandwidth is also affected by channel diversity; this is not in our control. We measure the average bit rate received by the smartphone and the extent of buffering, both of which impact engagement.

We compare our linear adapter against a version of iProxy that uses the state-of-the-art MPEG DASH scheme. Note that this scheme does not use cellular phone context. Also, it employs k different versions of the video (Section 3.3). While using large values of k enables greater adaptation, it also uses more storage. To strike a balance, we select $k = 4$, which uses nearly 2X more storage than iProxy with the linear adapter.

We tried out several different scenarios, each with a different way in which bandwidth gets shaped. We shows results for two randomly picked scenarios below.

Note that FFmpeg reports video bit rate of encoded video every two seconds.

Figure 3.19(a)shows the change in video bit rate in Scenario 1. In the beginning, the available bandwidth out of the desktop is 1000 kbps and video bit rate used by our linear adapter (Figure a) varies between 700 kbps to 1400 kbps. The variation is caused by uncontrolled background traffic. Twenty seconds later, when the bandwidth is shaped to 500 kbps, our linear adapter can detect the change and reduce the bit rate to around 400Kbps to avoid frame loss; our linear adapter keeps bit rate at 400 kbps. Note that, despite the use of EWMA, our linear adapter adjusts bit rate almost immediately after TCP detects packet loss. However, because of the use of a buffer in clients, these bursts in bit rate do not cause any perceptible impact to the user. When available bandwidth

drops further to 300 kbps, it is detected by our linear adapter based on the smaller average CWND, causing the linear adapter to continue to decrease the bit rate. Because TCP CWND can reach available bandwidth more quickly when bandwidth is low, the bit rate used by the linear adapter can more closely match the available bandwidth. On the whole, the average bit rate used by our linear adapter is 490Kbps.

Figure 3.19(b) shows the bit rate used by MPEG DASH. While DASH can also adapt, we see that its bit rate is often significantly lower than the available bandwidth, due to the discrete choices available (e.g., between 40s and 60s). The average bit rate of DASH is about 430Kbps; our linear adapter's bit rate was 16% higher on average. Between 40s and 60s, our linear adapter's bit rate was twice as high as DASH (200Kbps vs 100Kbps).

Figures 3.20(a) and (b) show the results for a second scenario where the shaper causes oscillations in the bandwidth. For the linear adapter, we observe that the bit rate gradually falls between 0 and 20s, perhaps because of poor channel conditions which cause the true available bandwidth to be much lower than the 1000Kbps limit set by our traffic shaper. Our scheme starts by using an statically picked initial rate of 1000Kbps; however, if the intial rate were set based on an accurate available bandwidth measure such as that inferrable from the latency spread observed for TCP ACKs, we would have picked a better initial bit rate. In general, relying on direct available bandwidth estimates in this fashion could help our bit rate adapter pick better rates even during the course of streaming, compared to our current scheme of using CWND/RTT; we plan to extend our adapter to work in this fashion in the future.

At around 20s, the bit rate settles to around 400Kbps. After 40s, the traffic shaper sets the bandwidth to 500Kbps, and our linear adapter immediately adapts its bit rate to around 450Kbps. In contrast, MPEG DASH generally uses lower bit rate than our linear adapter (410Kbps, on average), especially in the 20-60s time period where it is nearly 40% lower.

In both cases, our scheme suffered from *no* buffering events. In contrast, MPEG DASH saw up to 1s of buffering. While this may seem insignificant, prior studies [68] show that this lowers engagement to levels that start to matter to content providers.

	iProxy	MPEG DASH
Scenario 1	552.45 Kbps	458.17 Kbps
Scenario 2	565.97 Kbps	451.53 Kbps

Table 3.7 We repeat the experiment 100 times for each scenario, and iProxy provides higher average bit rate in both scenario.

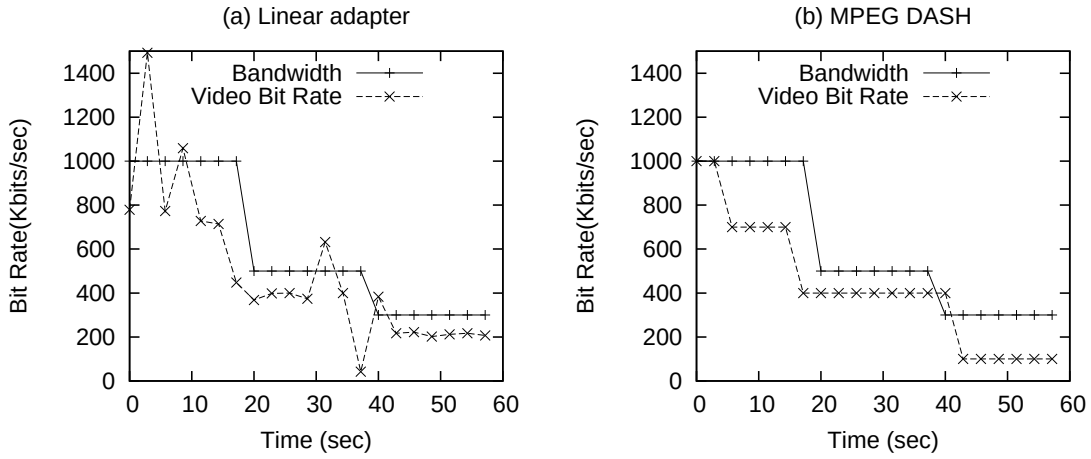

Figure 3.19 Scenario 1: the available bandwidth set by the shaper falls over time. The left figure is for iProxy using the linear adapter, the right is using MPEG DASH.

We repeat the experiment 100 times for each experiments for each scenario, and calculate average video bit rate for each instance. Then, Table 3.7 shows the average video bit rate of all instances for the two scenarios. We observe that iProxy works better than MPEG DASH in both scenarios, and the improvement is 21 % and 25 %, respectively. Therefore, iProxy provides videos with higher quality in terms of video bit rate.

Sensitivity Because iProxy estimates available bandwidth using simple passive measurements, there is a possibility of error especially when the available bandwidth drops suddenly. To understand how this impacts QoE, we use the same setup as above, but shape bandwidth such that it drops suddenly from 2000 kbps to 400 kbps roughly 11s into the video (total length of 40s). We observe that the video stream suffers from around 1s of buffering in all. While this is not conclusive, it indicates that iProxy's adaptation scheme is reasonably robust against sudden variations.

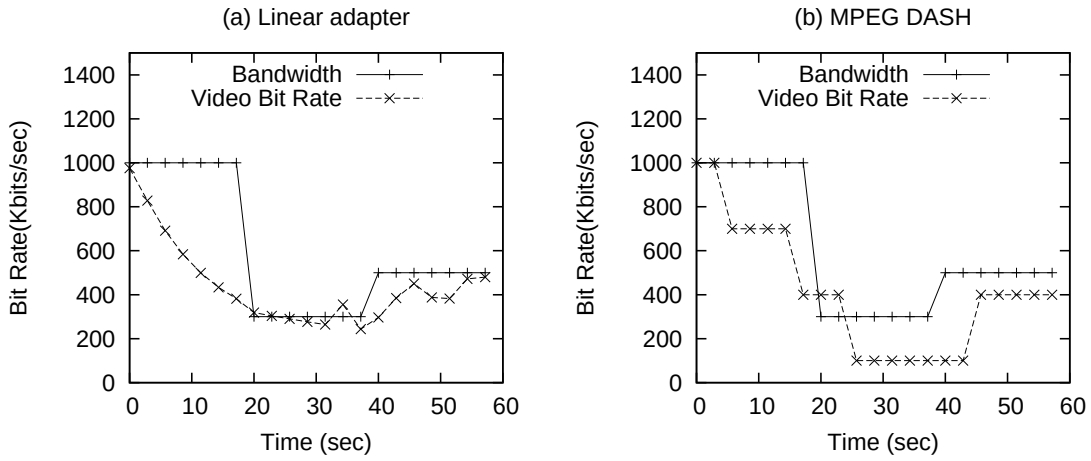

Figure 3.20 Scenario 2: the available bandwidth set by the shaper oscillates. The left figure is for iProxy using the linear adapter, the right is using MPEG DASH.

For completeness, we also study a more tradition quality metric, namely Peak signal-to-noise ratio (PSNR). PSNR is calculated by comparing the difference in quality between raw videos and compressed videos. Higher PSNR means higher quality. There are two aspects that affect PSNR: (a) *frame loss:* if some frames are dropped, a lot of information in the video will be lost, and (b) *the bit rate:* lower bit rate means we lose more information. We use an emulation-based study that also models cache deployment alternatives.

For the test video we used above, the video is compressed with 30.9dB in PSNR and 2.53Mbps in bit rate; thus, these are the maximal PSNR and bit rate we can achieve with iProxy's dynamic encoding module. We emulate three scenarios representing situations where proxies are located at the RNC, SGSN, or GGSN. We assume network bandwidth is 1.5Mbps. The average delay from the RNC to a mobile device is 90ms, from SGSN to a mobile device is 101ms, and from GGSN to a mobile device is 114ms. We measure the PSNR of the received video at the mobile device. We assume that the video player in the mobile device can buffer 10 seconds worth of video and drop any frames that pass the assigned deadline.

Figure 3.21 shows PSNR when different approaches are used to send the test video for each of the three deployment options. The first bar shows the original compressed video. The second

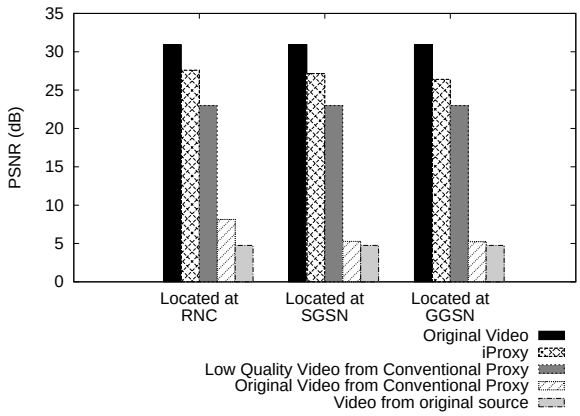

Figure 3.21 Video quality evaluation

bar is the PSNR of a video sent by iProxy. iProxy decreases video bit rate when available network capacity is lower. Thus, it suffers no frame loss and its PSNR is only slightly lower than the original video. In the RNC case, its PSNR is 11% lower, and in the GGSN case it is 15% lower, showing that iProxy's performance suffers with network delay.

The third bar in Figure 3.21 shows the case where we keep the video bit rate as low as possible instead of adapting bit rate; this options can also prevent frame loss. However, lower video bit rate means lower quality video and lower PSNR; and even though there is available network capacity it is left unused. Since this approach suffers no frame loss and uses fixed bit rate, the PSNR is the same in the three scenarios but it is inferior to iProxy in all three cases, Furthermore, compared to iProxy, this schemes leave 20% bandwidth unused in the RNC case, 18% bandwidth unused in the SGSN case, and 15% bandwidth unused in GGSN case.

The fourth bar shows a conventional proxy sending high quality video directly to mobile device. It results very high frame loss rate, and consequently has poor PSNR. In addition, the video quality is affected much more drastically by network delay compared to iProxy. For example, PSNR drops 55%, if we move the proxy from RNC to GGSN.

The last bar shows video quality when we stream the video directly from the video source. Without any help from proxy systems, the quality of video streaming is not acceptable with very low PSNR (i.e. 4.75dB). High delay and frame loss on the long path between video sources and

mobile devices hurt PSNR a lot.

Summary

In sum, our evaluation shows the following:

- iProxy can improve cache hit rates by 20% to up to 1.6X for the two traces we studied, compared to conventional proxies using state-of-the-art cache replacement schemes. Using information-aware schemes is crucial. iProxy uses 2.4X less storage than a naive cache design.
- iProxy improves start up delays by 2-14s compared to a proxy that is not intelligent in selecting the right video version to serve to a client when a URL is requested. In some cases, iProxy can play video that a conventional design simply cannot play.
- iProxy's linear adapter improves bit rate by 16% compared to state-of-the-art MPEG DASH scheme. Our adaption scheme gracefully changes bit rate in response to changing network conditions. Buffering is minimized, if not virtually eliminated.

Chapter 4

Related Works

There are many researches working on optimizing traffic and improve service performance. In this section, we discuss some previous works.

4.1 Finer-grained RE

To optimize transmission, many studies remove unnecessary content from traffic called redundancy elimination (RE). The basic idea is to scan the payloads of packets and compare them with previous traffic. Matched content is replaced by small shims in packets that reduces the size of the packets. Finally the payloads is cached for future use.

EndRE: it deploy RE in end systems as a service [9]. The advantage is that it can work with end-to-end encrypted, but it cannot benefit inter-path traffic that reduces the change to remove redundancy. EndRE discusses how to match payloads in packets in detail. It cuts payloads into smaller chunks and tries to match these chunks with the chunks in previous traffic. Three chunking schemes including MODP, MAXP, FIXED, SAMPLEBYTE are presented in EndRE and their performance in terms of bandwidth saving is evaluated. Thus, EndRE gives the overview of how to do chunk match in an RE system.

SmartRE: Instead of deploying RE in end systems, SmartRE [10] applies RE in routers. Redundancy is removed by upstream routers (encoders) and recovered by downstream routers (decoders). SmartRE further optimizes this approach by spreading decoding responsibility into multiple hops. It also takes intra-path traffic into account which means SmartRE can remove redundancy appearing between different paths to capture more chance to shorten packets. SmartRE relies on

a central controller to arrange decoding responsibility according to available resources (CPU and memory) in routers and traffic features.

Asymmetric Caching: Because of the roaming of mobile devices, the upstream cache for RE may be different at different time. It results that the upstream cache is likely to be aware of only part of a downstream cache in a mobile device and limits the amount of bytes we can reduce. To solve this problem, [69] deploys larger cache in mobile devices that can carry more content that go through different upstream caches. They shows that their solution can improve RE performance by 100%. However, the resources in mobile devices are more limited. It is not practical to use larger cache in mobile devices.

However, both of EndRE and SmartRE are designed for wired networks. They cannot take mobile and wireless features such as overhearing nature into account. Although asymmetric cache is designed for mobile devices, it do not consider the overhearing nature, ether. Thus, they lose some opportunity to cache more useful content in channels.

4.2 Solutions Relying on Wireless Overhearing Nature

Wireless transmission broadcast bytes into channel, so nearby devices have opportunity to overhear these bytes. This overhearing nature can be used to optimize transmission.

RTS-ID [11]: Before transmitting a packet to a receiver, a sender piggybacks its packet ID into an RTS message. RTS is sent before packet transmission. The receiver can check if this packet is already in its cache. If it is, the receiver send back a CTS-ACK and the sender won't send this packet. Thus, RTS-ID can save some transmission, if some packets have been already cached.

MORE [28]: It applies network coding into mesh networks. A sender broadcasts packets into channels. Some nodes that are far away from the sender still have chance to overhear some packets. Instead of forwarding all packets, a relay combines packets by network coding, and the receiver can decode the encoded packets by the packets it has overheard. MORE also reduces some transmission, if a receive can overhear some traffic.

ExOR [12]: It is also a solution for mesh networks. A sender broadcasts a packet into a channel. Multiple relays may overhear and receive this packet. The system will choose one of

them that is closer to the final destination to forward the packet. It can reduce the number of hops to reach the final destination. ExOR uses estimated transmission (ETX) to estimate the distance.

MIXIT [70]: As ExOR, a sender broadcasts packets, and the relays who receive the packets combine them by network coding and forward them, even if the received packets are partially correct. Relays use SoftPHY to recognize which chunks are received correctly and provide hints.Then, receivers can decode packets based on encoded packets with different network coding combination from relays.

DITTO [14]: It cuts objects into multiple chunks. When a client requests an object, Data-Oriented Transfer (DOT) service provides it what chunks are needed. Then, requested chunks are forwarded between relays, and can be overheard and cached by nearby relays. If a requested chunk has been already in the cache of a relay, we can save the transmission from a sender to this relay. DITTO leverages wireless overhearing nature and benefits inter-flow traffic.

MicroCast [71]: They propose a video streaming system that uses multiple interface to download videos. A video is cut into several segments, and the segments are send to different clients who request this video through cellular links. Then, the clients request missing segments from other clients by using WiFi. Clients also overhear the requests between other clients. Their goal is to reduce the load of cellular networks and avoid unnecessary transmission by overhearing on WiFi. They match the overheard data as large as several packets.

These approaches use wireless overhearing nature well, but there are two major drawbacks. First, they are not finer-grained enough to match redundancy content. RTS-ID, MORE, ExOR, and MIXIT match content in packet level; DITTO and MicroCast match and cache data as large as 8-32 KB. The second drawback is that they excepting RTS-ID can only apply to mesh networks and cannot benefit other scenarios such as infrastructure mode.

4.3 Video proxy

With growing mobile multimedia traffic, we want to improve video watch experience in wireless environment. Previous studies put one or more middle boxes as proxies to reduce the traffic from remote video providers.

MiddleMan [24] gives an idea that putting videos closer to clients can reduce start-up delay and the chance that video quality is disrupted by network condition. To increase scalability, MiddleMan deploys local proxy in each client and some standalone proxies. Each proxy responds to cache different videos, so storage can be use efficiently. When a client requests a video, it asks its local proxy first, and then ask others. To coordinate proxies and locate videos, MiddleMan uses a coordinator.

MiddleMan reduces start-up time and gives more smooth video playback. However, our work called iProxy can further reduce redundancy cache by applying IBR to identify videos which is evaluated to provide better cache hit rate. In addition, MiddleMan needs to modify client side, because they communicate with a coordinator to figure out what to cache. It makes deployment difficult.

As a general proxy, MiddleMan identifies videos by URLs. Thus, videos with the same content will be recognized as different videos and is put into cache. In addition, MiddleMan cannot adapt videos according to client diversity and channel diversity, so QoE cannot be guaranteed.

4.4 New Video Encoding Schemes

Some existing approaches improve video streaming by changing video encoding schemes to protect important video frames.

SoftCast [15]: The authors argue that whole video encoding process from DCT to 16-QAM encoding should be linear. SoftCast replaces non-linear entropy coding by scaling up important DCT components to provide error protection in wireless networks. Wireless channel quality varies with different time and location and different clients encouter different channel quality; SoftCast broadcasts videos to the clients, and the clients can decode videos even if some bytes are lost.

FlexCast [72]: It encodes a video by grouping equally important bits of a video using distortion grouping. Then, they modify Raptor codes [73] to do rateless video coding to protect more important groups. Thus, they reduce the chance to loss important bits, and keep video quality even in unstable wireless channels.

The new video encoding schemes above can successfully protect videos from unstable channels and provide good video quality. However, they need to modify physical layer that introduce the difficulty of deployment on existing infrastructures.

4.5 Video Adaption

To achieve better QoE in unstable wireless channels, many studies adapt video bit rate to avoid using out available bandwidth.

Scalable Video Coding (SVC): It considers network conditions to send suitable versions of videos encoded using H.264/SVC [74] in a P2P VOD system. Their work can use network resources efficiently by choosing the appropriate bit rate.

Smooth Streaming: It is similar to MPEG DASH (they can be thought of as vendor-specific realizations of DASH). It pre-encodes different versions of videos, and according to the feedback from a client, Smooth Streaming sends a suitable version to the client.

SVC and Smooth Streaming can change bit rate dynamically, but they cannot provide linear bit rate adaption. Instead, they only choose one of pre-encoded versions when available bandwidth change is detected. The non-linear bit rate adaption cause performance cliff problem as mentioned in Section 3.3.

Vantrix [75]: It transcodes and adapts videos based on user devices and available bandwidth in real time. It results in efficient bandwidth usage and better user experience. Vantrix also provides a smart caching to store popular videos and save the bandwidth in Internet backhaul. However, Vantrix still uses URLs to identify videos and may cache redundant videos. In addition, it transcodes videos from original format instead of from frequency domain data; it leads to consume more computational power to do DCT process for every streaming.

4.6 Video Traffic Optimization

Some other studies optimize traffic specifically for video traffic.

MuVi [76]: It keeps monitoring channel qualify from channel state information (CSI) that is reported by mobile devices. Based on the channel quality, they determine what modulation and coding scheme (MCS) is robust. Higher MCS provides more bandwidth, but results in higher bit error rate (BER). In addition, B-frames are the last important data in video stream, so MuVi uses higher MCS to transmit them and then we can complete B-frame transmission quickly. I-frames and and P-frames use most robust MCS that protects them from random loss.

AVIS [77]: It proposes a framework to schedule traffic for adaptive bit rate videos such as MPEG DASH. MPEG DASH can change video bit rate based on channel quality. However, the bandwidth estimation may fail because of co-existing flows. It results in unfair resource allocation among video flows. Thus, AVIS solves this problem by modeling a traffic schedule algorithm on wireless links. They can achieve fairness, and high bandwidth utilization. By enforcing the bandwidth allocation, MPEG DASH adapts to a suitable bit rate.

RSVP Related: In [78, 79], they provide a framework to deal with QoS problem in wireless networks. When a mobile device handovers to another AP, it is difficult to keep QoS guarantee by using Resource Reservation Protocol (RSVP) [80], because of the delay of mobile IP [81] registration. [78] puts a QoS proxy behind each access point (AP), the proxy in neighbor APs pre-reserves resources for a mobile device. When this mobile device handovers to another AP, the resources can be used quickly to reduce the delay caused by handover.

Scheduling and admission control: [82] proposes a framework for transmission scheduling and admission control. They introduce two types of devices: $helper$ and $user$. $helper$ contains video files cut into multiple chunks and encoded as different versions as MPEG DASH. $helper$ determines which client can download the chunk it requests based on rate limit. Moreover, $user$ decides which version of chunks to download according to video performance. The final goal of their work is to maximize sum transmission rate and network utilization. They implement this framework on the top of Android smartphones [83].

Whitespace: To get more available bandwidth, [84] uses the bandwidth of whitespace that is the unused frequency for a broadcast service. They provide video streaming service on buses. The buses are installed two kinds of interfaces. Cellular cards are used for uplink to send video requests

to servers. Then, servers stream videos back to the buses through whitespace channel. Thus, they efficiently use the channel to achieve higher performance.

Chapter 5

Conclusion and Future Works

In this thesis, we propose a system to optimize network traffic by looking into the content of traffic at byte and information levels; we also improve goodput and quality of experience, especially for mobile video traffic, which comprises more than 50% of total global traffic [85]. Mobile and wireless environments suffer higher random loss and broadcast data in the air, which is quite different from wired environments. In addition, the requirements of multimedia traffic are highly unique. For example, multimedia traffic prefers stable bandwidth rather than high average bandwidth. We have focused on four aspects in this thesis: (a) redundancy in traffic, (b) improving video viewing experience, (c) channel features in wireless environments, and (d) using resources efficiently.

Summarized below are the contributions of this thesis.

5.1 Contributions

Removing Redundant Content: We provide an RE solution designed for mobile and wireless environments that removes redundancy at the chunk level, which overs finer granularity than at the packet level; shorter packets result in essentially lower packet error rates, which in turn reduces mac-layer re-transmission. This solution eliminates more potential redundancy and further improve goodput, which provides better channel efficiency. Our novel video proxy system proposes an intelligent way to avoid the need to cache redundant videos so that significantly fewer duplicate videos will be stored; IBR is applied as a signature to identify videos.

Quality of Experience (QoE): We propose a video proxy system that improves QoE. A better cache design increases the cache hit rate, so clients don't need to wait for a long period to receive requested videos. It also results in successful reduction of video start-up latency and releases the load of 3G/4G core networks. In our system, we take channel diversity and client diversity into account to provide the most appropriate videos for clients, instead of merely forwarding the original video's formats; clients can enjoy smooth videos with less video re-buffering events and shorter start-up latency. In addition, REfactor removes redundant bytes in wireless links, so higher goodput results in higher network capacity, which can be used to provide higher quality videos.

Better Resource Usage: In iProxy, cache storage usage is optimized. Better video identification helps the cache to store more useful content. At the client side, resources such as CPU power and memory space are quite limited in mobile devices. REfactor provides a lightweight caching scheme for RE that uses less resources. In addition, iProxy retrieves user device information that determines available resources in mobile devices. Thus, our system can adapt to traffic according to this information and avoid sending unsuitably high bit rate video, which wastes CPU power in mobile devices.

Optimization for Mobile and Wireless: To provide high quality videos and fully employ available bandwidth, a linear bit rate adapter has been applied to iProxy. By dynamically changing the video bit rate, the adapter can make better use of the available bandwidth.Thus, wireless channels can be used more efficiently. In addition, wireless traffic is broadcast to channels, so all devices connecting to the channel can overhear the traffic. REfactor leverages this feature to cache more useful data for RE.

Robustness: In wireless channels, REfactor proposes a model-driven RE to determine which redundant chunks can be removed. However, even if the model-driven RE removes redundant chunks incorrectly, receivers can still re-request the missing chunks from an AP, so our mobile RE solution has no correctness issues. In iProxy, the linear bit rate adapter works fast enough to satisfy video requests according to our evaluation. Moreover, the scalability can be further improved by deploying additional distributed dynamic video encoders that share the same video cache.

5.2 Future Research

Content-aware communication gives networks the opportunity to optimize traffic. I provide several solutions to eliminate redundancy at both byte and information levels in mobile and wireless networks. However, some aspects still need work. In this section, I will discuss these aspects.

Cache Policy in RE: Different traffic may encounter different redundancy patterns, so some entries are more useful than others for future traffic. To improve the efficiency of a cache, we need to design a cache replacement policy based on these redundancy patterns. Some observations concerning redundancy patterns are shown in [22] :

- **Protocols:** The percentage of redundancy is different in different protocols. SMTP has the highest percentage, which is 71%, while the lowest is RTSP, which is 2%.
- **Inbound or outbound traffic:** [22] shows that there is a slight negative correlation between the fraction redundancy and the volume of traffic for university inbound traffic, but a slight positive correlation for university outbound traffic.
- **Enterprise or university:** The range of traffic savings in enterprise traffic is 17%$\sim$61%, and the range in university traffic is 9%$\sim$15%. In addition, different enterprise sites show different percentages.

This policy ranks the contribution of each entry based on the historical record and chooses the lowest one to be replaced when the cache is full. According to the observations above, the replacement policy should consider some traffic properties. For example, traffic from enterprises has a higher ranking than traffic from universities. The content from popular servers tends to be repeatedly requested, so it is worthwhile to keep this content in the cache longer. We determine what pattern each traffic may have by looking into some fields of packet headers such as the IP address and port number. A good cache replacement policy can increase the hit rate in proxies and improve traffic savings in redundancy elimination systems.

RE in Sensor Networks: Sensor networks are used in wide areas. For example, CarSpeak [86] introduced a autonomous car system that uses sensor networks to exchange surrounding information among cars. Resources such as energy is quite limited in sensors. Thus, it will be critical to eliminate some unnecessary transmissions. To design an RE system in sensor networks, the following criteria should be considered:

- Computational power, memory size, and energy are limited in sensors, so we need a cheaper algorithm to realize RE.
- The sensor may be moving, as in CarSpeak, and the transmission between sensors represents an opportunity. Thus, transmission needs to be carefully scheduled.
- Information exchanged among sensors is usually geographically related. The path each sensor takes should be considered.

When we deploy several sensors in space, their sensing range will have some overlap. Different sensors may have the same information. Thus, a sensor don't need to send this information to others that already have the same information. To this end, we need an algorithm to detect a sensor's path of movement and estimate what information each sensor may sense. With this knowledge, a sensor can reduce redundant information exchanges, which in turn saves channel bandwidth and power.

QoS for Video Service over Cellular Networks: To provide high quality of experience for video service, QoS is an important issue. A better bandwidth guarantee proposes more stable bandwidth, which reduces the video buffering rate and increases the capacity for high bit rate videos. In addition, multiple mobile devices may access services simultaneously, and devices with the same priority should receive consistent resources. We seek to fulfill two requirements:

- **Fairness:** The resources allocated for each flow should be proportional to the flow's priority. Network policies and traffic types determine priority. For example, network managers want to reserve resources for video traffic to improve services.

- **Utilization:** In addition to fairness, we want to achieve higher overall network utilization. Thus, we can provide better services for a greater number of requests.

There are some key elements that should be considered when we design our QoS scheme.

- **Sub-channels:** Different clients receive various channel quality by connecting to different sub-channels as shown in [87]. How to assign flows to different sub-channels is critical.

- **Multiple Cellular Towers:** A single client may be able to hear a signal from different cellular towers. Relying on an OpenFlow central controller, we can arrange clients to different cellular towers to achieve higher overall network utilization.

- **Video Frames:** The importance of video frames is unique. In [76], different frames are referenced by different numbers of other frames, as shown in Figure 5.1. The frames referenced by more frames are more important. If we lose an important frame, more frames cannot be decoded. It is observed that the importance of the I-frame in different GOP locations are also different. [76] proposes that we should use a lower rate module and coding scheme (MCS) to protect important frames.

- **Bandwidth Limiter:** To provide stable bandwidth and enforce the resource location assigned by a resource allocator, We need an efficient way to limit the resources each flow can use. In traditional traffic filters, switches inform traffic senders to reduce their sending rate by dropping their packets. However, this approach causes CWND to be dropped frequently based on TCP protocol. This in turn results in the sending rate dropping unexpectedly and affects throughput.

To achieve better QoS support, three major components are required as shown in Figure 5.2: a monitor, an allocator, and an enforcer.

A monitor is an OpenFlow enabled switch. When a new flow that it has never seen before, it forwards the flow abstract to an OpenFlow controller. This controller determines the type and priority of the flow according to flow features and administration policies.

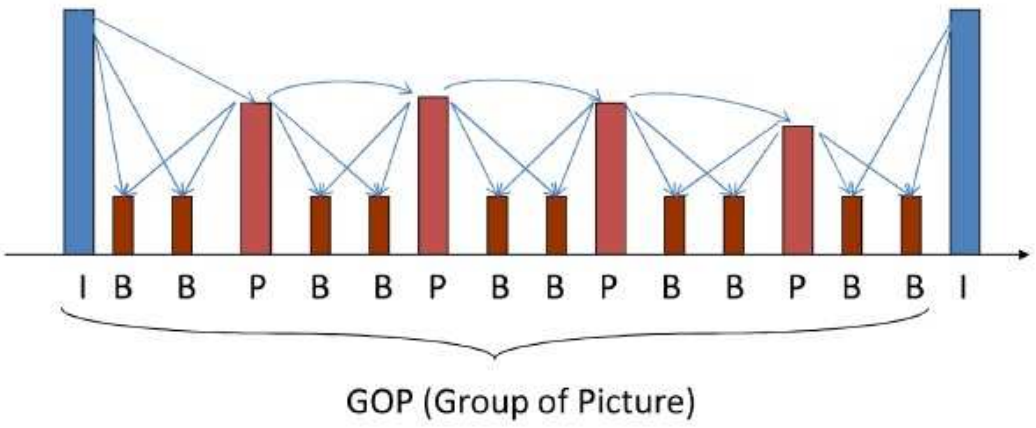

Figure 5.1 The references among video frames.

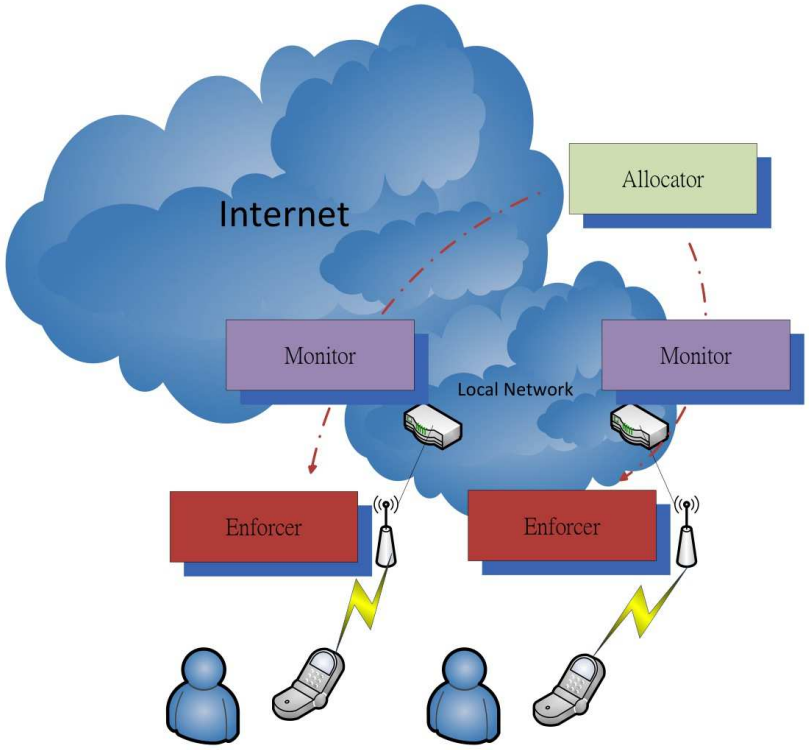

Figure 5.2 An Overview of the QoS System.

In addition, the monitor also gathers network resource information, such as channel quality and the load of each cellular tower. This is necessary information for the controller to allocate resources.

The goal of an allocator is to determine network resource allocation, according to the flow and resource information provided by monitors. An allocator schedules flows to sub-channels and time slots according to flow priority and channel quality. For video traffic, we also consider the importance of each video frame and assign lower rate MCS to important frames to reduce transmission errors.

After getting resource allocation, an enforcer applies the resource locations to flows by limiting the bandwidth they can use. We use OpenFlow enabled nodes to modify the ”window size” fields in TCP headers, so the sender will limit its sending rate according to this window size instead of dropping its CWND.

Advanced Video Bit Rate Control: To provide better quality of experience (QoE) [21], it is critical to have a good video bit rate adaptation scheme, which should control the rate at a level closer to available bandwidth. In this way, the video can achieve higher quality with a lower volume of video buffering events.

iProxy provides a scheme to determine the video bit rate by tracing CWND size and RTT. This design uses historical records to predict the following available bandwidth and determine a proximate bandwidth. To provide a more precise estimation, this work will develop a novel bit rate adaptation scheme.

Before designing the scheme, we should observe some key features of video streaming. Current videos are encoded in a variable bit rate (VBR), so the size varies from frame to frame. This means the video bit received rate is not equal to the video frame received rate. In addition, a video is composed of a series of frames, and each frame corresponds to a deadline, which is when the frame should be correctly decoded. If some frames cannot be decoded before their deadline, the video player will hold the video to wait for the coming frames. A video buffer can reduce the buffering events, but the buffer size is not unlimited, and the events still happen when the buffer is out of content. Thus, it is more important to maintain a minimal frame-received rate than a bit-received rate.

The basic idea of this work is to compare the frame-received rate with the frame-display rate of a video. If the frame-received rate is higher, we dont fully utilize available bandwidth, and the

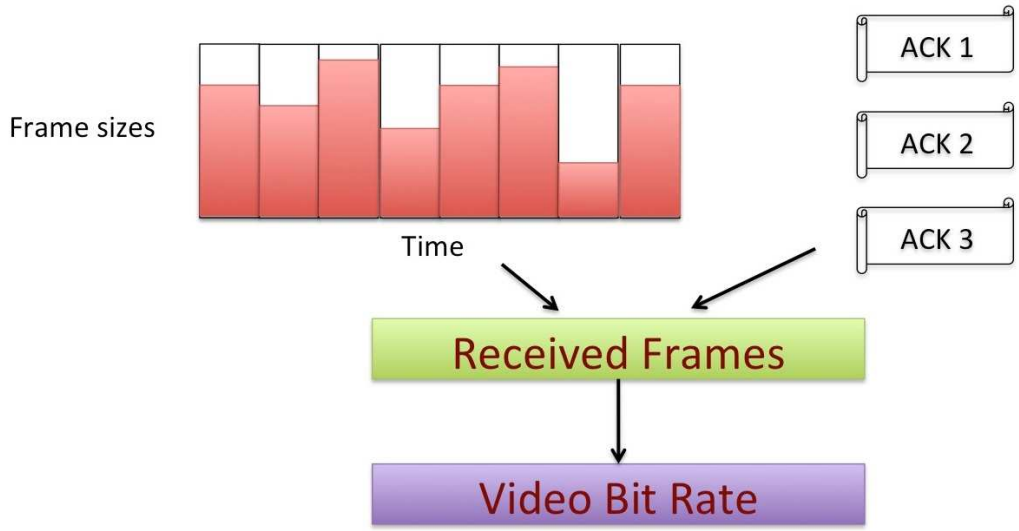

Figure 5.3 Frame-received Rate Estimation.

video buffer is filled up. It implies that we can send a video in higher quality (higher bit rate). On the contrary, if the frame-received rate is lower, we should lower the video quality. In this way, the buffer at the client side can be kept at a reasonable level. Thus, we wont use up the buffer, which causes buffering events and we will retain video quality at the same time.

In iProxy, a video is dynamically encoded during streaming; we retrieve the size of all frames from the video encoder and keep monitoring the TCP ACK from a video receiver as shown in Figure 5.3. We then calculate the last packet of each frame and the time stamp of the TCP ACK corresponding to the last packet. By gathering the time stamp for all video frames, we can determine the frame-received rate. Moreover, the video frame-display rate is determined when we receive the video. The frame-display rate can be 30 fps in general or 60 fps in higher quality.

To make the system workable, we face the following challenges:

- **An Algorithm to Adjust Video Bit Rate:** With the frame-received rate and frame-display rate, how do we adjust the video bit rate? We should design an algorithm to guarantee the maximal video quality without using up the video buffer. In addition, this algorithm should be fast enough to react to channel conditions.

- **The Latency between iProxy and Clients:** We should evaluate how the latency affects the performance of our algorithm to adjust video bit rate. The latency makes our system difficult to cache the channel quality change. Our algorithm needs to keep the video buffer level higher than some thresholds to prevent the video buffer from being used up. Thus, we need to determine the threshold carefully.

With this advanced video bit rate control, iProxy can adapt the video bit rate in a more reasonable way. This results in a lower buffering event with higher video quality, which improves QoE.

LIST OF REFERENCES

[1] M. Afanasyev and A. C. Snoeren, "The importance of being overheard: throughput gains in wireless mesh networks," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, p. 384396. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644940

[2] A. Gember, A. Anand, and A. Akella, "A comparative study of handheld and non-handheld traffic in campus wi-fi networks," in *Proceedings of the 12th international conference on Passive and active measurement*, ser. PAM'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 173183. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987510.1987528

[3] J. Erman, A. Gerber, K. K. Ramadrishnan, S. Sen, and O. Spatscheck, "Over the top video: The gorilla in cellular networks," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 127–136. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068829

[4] "Cisco visual networking index: Global mobile data traffic forecast update, 20132018," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html, 2014. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

[5] "Youtube advanced encoding settings," https://support.google.com/youtube/answer/1722171?hl=en, 2014. [Online]. Available: https://support.google.com/youtube/answer/1722171?hl=en

[6] C. Lumezanu, K. Guo, N. Spring, and B. Bhattacharjee, "The effect of packet loss on redundancy elimination in cellular wireless networks," in *Proceedings of the 10th annual conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, p. 294300. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879179

[7] X. Wu, A. G. Hauptmann, and C.-W. Ngo, "Practical elimination of near-duplicates from web video search," in *Proceedings of the 15th international conference on Multimedia*, ser. MULTIMEDIA '07. New York, NY, USA: ACM, 2007, pp. 218–227. [Online]. Available: http://doi.acm.org/10.1145/1291233.1291280

[8] "Near-duplicate web video dataset." http://vireo.cs.cityu.edu.hk/webvideo/, 2013. [Online]. Available: http://vireo.cs.cityu.edu.hk/webvideo/

[9] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: an end-system redundancy elimination service for enterprises," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, p. 2828. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855739

[10] A. Anand, V. Sekar, and A. Akella, "SmartRE: an architecture for coordinated network-wide redundancy elimination," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, p. 8798, Aug. 2009. [Online]. Available: http://doi.acm.org/10.1145/1594977.1592580

[11] M. Afanasyev, D. G. Andersen, and A. C. Snoeren, "Efficiency through eavesdropping: link-layer packet caching," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, p. 105118. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387589.1387597

[12] S. Biswas and R. Morris, "ExOR: opportunistic multi-hop routing for wireless networks," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, p. 133144, Aug. 2005. [Online]. Available: http://doi.acm.org/10.1145/1090191.1080108

[13] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Mdard, and J. Crowcroft, "XORs in the air: practical wireless network coding," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, p. 497510, Jun. 2008. [Online]. Available: http://dx.doi.org/10.1109/TNET.2008.923722

[14] F. R. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. G. Andersen, "Ditto: a system for opportunistic caching in multi-hop wireless networks," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, ser. MobiCom '08. New York, NY, USA: ACM, 2008, p. 279290. [Online]. Available: http://doi.acm.org/10.1145/1409944.1409977

[15] S. Jakubczak and D. Katabi, "A cross-layer design for scalable mobile video," ser. MobiCom '11. New York, NY, USA: ACM, 2011, p. 289300. [Online]. Available: http://doi.acm.org/10.1145/2030613.2030646

[16] "Youtube statistics," http://www.youtube.com/yt/press/statistics.html, 2014. [Online]. Available: http://www.youtube.com/yt/press/statistics.html

[17] "Cisco visual networking index: Forecast and methodology, 20132018," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html, 2014. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

[18] S. Sen, J. Yoon, J. Hare, J. Ormont, and S. Banerjee, "Can they hear me now?: a case for a client-assisted approach to monitoring wide-area wireless networks," ser. IMC '11. New York, NY, USA: ACM, 2011, p. 99116. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068827

[19] "Qualcomm snapdragon 800 vs intel core i7," http://versus.com/en/qualcomm-snapdragon-800-vs-intel-intel-core-i7-3920xm-extreme-edition, 2013. [Online]. Available: http://versus.com/en/qualcomm-snapdragon-800-vs-intel-intel-core-i7-3920xm-extreme-edition

[20] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the impact of video quality on user engagement," in *Proceedings of the ACM SIGCOMM 2011 conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 362–373. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018478

[21] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a predictive model of quality of experience for internet video," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 339–350. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486025

[22] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '09. New York, NY, USA: ACM, 2009, p. 3748. [Online]. Available: http://doi.acm.org/10.1145/1555349.1555355

[23] S. Guha, J. Chandrashekar, N. Taft, and K. Papagiannaki, "How healthy are today's enterprise networks?" in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, ser. IMC '08. New York, NY, USA: ACM, 2008, p. 145150. [Online]. Available: http://doi.acm.org/10.1145/1452520.1452538

[24] S. Acharya and B. Smith, *MiddleMan: A Video Caching Proxy Server*, 2000.

[25] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil, "An architecture for internet data transfer," in *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, ser. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, p. 1919. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267680.1267699

[26] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '07. New York, NY, USA: ACM, 2007, p. 169180. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282400

[27] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, p. 219230, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402984

[28] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wirel. Netw.*, vol. 11, no. 4, p. 419434, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1007/s11276-005-1766-z

[29] J. Santos and D. Wetherall, "Increasing effective link bandwidth by suppressing replicated data," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '98. Berkeley, CA, USA: USENIX Association, 1998, p. 1818. [Online]. Available: http://dl.acm.org/citation.cfm?id=1268256.1268274

[30] M. Rabin, "Fingerprinting by random polynomials," no. TR-15-81, p. 1518, 1981.

[31] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '00. New York, NY, USA: ACM, 2000, p. 8795. [Online]. Available: http://doi.acm.org/10.1145/347059.347408

[32] K. Jamieson and H. Balakrishnan, "PPR: partial packet recovery for wireless networks," in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '07. New York, NY, USA: ACM, 2007, p. 409420. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282426

[33] G. Judd, X. Wang, and P. Steenkiste, "Efficient channel-aware rate adaptation in dynamic environments," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*, ser. MobiSys '08. New York, NY, USA: ACM, 2008, p. 118131. [Online]. Available: http://doi.acm.org/10.1145/1378600.1378615

[34] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '04. New York, NY, USA: ACM, 2004, p. 121132. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015482

[35] A. Bhartia, Y.-C. Chen, S. Rallapalli, and L. Qiu, "Harnessing frequency diversity in wi-fi networks," in *Proceedings of the 17th annual international conference on Mobile computing and networking*, ser. MobiCom '11. New York, NY, USA: ACM, 2011, p. 253264. [Online]. Available: http://doi.acm.org/10.1145/2030613.2030642

[36] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, p. 263297, Aug. 2000. [Online]. Available: http://doi.acm.org/10.1145/354871.354874

[37] *Cisco Enterprise Wireless Products*, 2012. [Online]. Available: http://cisco.com/en/US/prod/wireless/wireless_for_enterprise.html

[38] *Linksys WRT54G Series*, 2012. [Online]. Available: http://en.wikipedia.org/wiki/Linksys_WRT54G_series

[39] *ns-3 network simulator*, 2012. [Online]. Available: http://www.nsnam.org/

[40] *Click [Click]*, 2012. [Online]. Available: http://read.cs.ucla.edu/click/

[41] S. Woo, E. Jeong, S. Park, J. Lee, S. Ihm, and K. Park, "Comparison of caching strategies in modern cellular backhaul networks," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 319–332. [Online]. Available: http://doi.acm.org/10.1145/2462456.2464442

[42] "The state of 4G: its all about congestion, not speed," http://ffmpeg.org/, 2013. [Online]. Available: http://arstechnica.com/tech-policy/2010/03/faster-mobile-broadband-driven-by-congestion-not-speed/

[43] A. Anand, A. Akella, V. Sekar, and S. Seshan, "A case for information-bound referencing," ser. Hotnets '10. New York, NY, USA: ACM, 2010, p. 4:14:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868451

[44] B. Coskun and B. Sankur, "Robust video hash extraction." IEEE, Apr. 2004, p. 292 295.

[45] J. S. Boreczky and L. A. Rowe, "Comparison of video shot boundary detection techniques," vol. 2670, p. 170179, 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.2179

[46] *InfoNames: An Information-Based Naming Scheme for Multimedia Content\textbar Whitepapers \textbar TechRepublic*, Mar. 2012. [Online]. Available: http://www.techrepublic.com/whitepapers/infonames-an-information-based-naming-scheme-for-multimedia-content/2883611

[47] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A review of audio fingerprinting," *J. VLSI Signal Process. Syst.*, vol. 41, no. 3, p. 271284, Nov. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1107802.1107829

[48] A. Andoni, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *IN FOCS06*, vol. 2006, p. 459468, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.8372

[49] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, p. 518529. [Online]. Available: http://dl.acm.org/citation.cfm?id=645925.671516

[50] A. Kumar, A. Balachandran, V. Sekar, A. Akella, and S. Seshan, "Infonames: An information-based naming scheme for multimedia content," UW-Madison, Technical Report TR1677, Jul. 2010.

[51] T. R. G. Nair and P. Jayarekha, "A rank based replacement policy for multimedia server cache using zipf-like law," *arXiv:1003.4062*, Mar. 2010, journal of Computing, Volume 2, Issue 3, March 2010, https://sites.google.com/site/journalofcomputing/. [Online]. Available: http://arxiv.org/abs/1003.4062

[52] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, p. 158170, Apr. 2000.

[53] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul, "An active transcoding proxy to support mobile web access," in *In Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 118–123.

[54] A. Gember, A. Akella, J. Pang, A. Varshavsky, and R. Caceres, "Obtaining in-context measurements of cellular network performance," in *Proceedings of the 2012 ACM conference on Internet measurement conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, p. 287300. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398807

[55] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. C-23, no. 1, p. 9093, Jan. 1974.

[56] "How i came up with the discrete cosine transform," Mar. 2012. [Online]. Available: http://www.scribd.com/doc/60936785/How-I-Came-Up-With-the-Discrete-Cosine-Transform

[57] *smartphone_sreen_resolution*, Feb. 2012. [Online]. Available: http://en.wikipedia.org/wiki/Smartphone

[58] *android_supported_format*, Feb. 2012. [Online]. Available: http://developer.android.com/guide/appendix/media-formats.html

[59] D. Sostaric, D. Vinko, and S. Rimac-Drlje, "Power consumption of video decoding on mobile devices." IEEE, Sep. 2010, p. 8184.

[60] "VPlayer," https://vplayer.net/, 2013. [Online]. Available: https://vplayer.net/

[61] *pHash.org: Home of pHash, the open source perceptual hash library*, Mar. 2012. [Online]. Available: http://phash.org/

[62] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," ser. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, p. 518529. [Online]. Available: http://dl.acm.org/citation.cfm?id=645925.671516

[63] "FFmpeg," http://ffmpeg.org/, 2013. [Online]. Available: http://ffmpeg.org/

[64] "Video streaming server for ffmpeg," http://ffmpeg.org/ffserver.html, 2013. [Online]. Available: http://ffmpeg.org/ffserver.html

[65] "Near-duplicate web video dataset," http://vireo.cs.cityu.edu.hk/webvideo/, 2013. [Online]. Available: http://vireo.cs.cityu.edu.hk/webvideo/

[66] P. Cao and S. Irani, "Cost-aware www proxy caching algorithms," in *USITS*, 1997.

[67] C. Serrano, B. Garriga, J. Velasco, J. Urbano, S. Tenorio, and M. Sierra, "Latency in Broad-Band mobile networks," in *VTC*, 2009.

[68] S. Krishnan and R. Sitaraman, "Video Stream Quality Impacts Viewer Behavior: Inferring Causality using Quasi-Experimental Designs," in *IMC*, 2012.

[69] S. Sanadhya, R. Sivakumar, K.-H. Kim, P. Congdon, S. Lakshmanan, and J. P. Singh, "Asymmetric caching: Improved network deduplication for mobile devices," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, ser. Mobicom '12. New York, NY, USA: ACM, 2012, pp. 161–172. [Online]. Available: http://doi.acm.org/10.1145/2348543.2348565

[70] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard, "Symbol-level network coding for wireless mesh networks," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, p. 401412. [Online]. Available: http://doi.acm.org/10.1145/1402958.1403004

[71] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, and A. Markopoulou, "Microcast: Cooperative video streaming on smartphones," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12. New York, NY, USA: ACM, 2012, pp. 57–70. [Online]. Available: http://doi.acm.org/10.1145/2307636.2307643

[72] S. Aditya and S. Katti, "FlexCast: graceful wireless video streaming," in *Proceedings of the 17th annual international conference on Mobile computing and networking*, ser. MobiCom '11. New York, NY, USA: ACM, 2011, p. 277288. [Online]. Available: http://doi.acm.org/10.1145/2030613.2030645

[73] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, p. 25512567, Jun. 2006.

[74] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, p. 11031120, Sep. 2007.

[75] "Vantrix website," http://www.vantrix.com/index.html, 2013. [Online]. Available: http://www.vantrix.com/index.html

[76] J. Yoon, H. Zhang, S. Banerjee, and S. Rangarajan, "Muvi: a multicast video delivery scheme for 4g cellular networks," in *Proceedings of the 18th annual international conference on Mobile computing and networking*, ser. Mobicom '12. New York, NY, USA: ACM, 2012, pp. 209–220. [Online]. Available: http://doi.acm.org/10.1145/2348543.2348571

[77] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang, "A scheduling framework for adaptive video delivery over cellular networks," in *Proceedings of the 19th annual international conference on Mobile computing & networking*, ser. MobiCom '13. New York, NY, USA: ACM, 2013, pp. 389–400. [Online]. Available: http://doi.acm.org/10.1145/2500423.2500433

[78] W.-T. Chen and L.-C. Huang, "Rsvp mobility support: a signaling protocol for integrated services internet with mobile hosts," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2000, pp. 1283–1292 vol.3.

[79] S.-C. Lo, G. Lee, W.-T. Chen, and J.-C. Liu, "Architecture for mobility and qos support in all-ip wireless networks," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 4, pp. 691–705, 2004.

[80] L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation protocol (RSVP) – version 1 functional specification." [Online]. Available: http://tools.ietf.org/html/draft-ietf-rsvp-spec-13

[81] C. E. P. <charliep@iprg.nokia.com>, "Mobile IPv4 Challenge/Response extensions." [Online]. Available: http://tools.ietf.org/html/rfc3012

[82] D. Bethanabhotla, G. Caire, and M. Neely, "Joint transmission scheduling and congestion control for adaptive streaming in wireless device-to-device networks," in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*, 2012, pp. 1179–1183.

[83] J. Kim, F. Meng, P. Chen, H. E. Egilmez, D. Bethanabhotla, A. F. Molisch, M. J. Neely, G. Caire, and A. Ortega, "Adaptive video streaming for device-to-device mobile platforms," in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, ser. MobiCom '13. New York, NY, USA: ACM, 2013, pp. 127–130. [Online]. Available: http://doi.acm.org/10.1145/2500423.2505292

[84] T. Zhang, S. Sen, and S. Banerjee, "Video streaming using whitespace spectrum for vehicular applications," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 471–472. [Online]. Available: http://doi.acm.org/10.1145/2462456.2465731

[85] "Cisco visual networking index: Global mobile data traffic forecast update, 20122017 [visual networking index (VNI)]," http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html, 2013. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

[86] S. Kumar, L. Shi, N. Ahmed, S. Gil, D. Katabi, and D. Rus, "Carspeak: a content-centric network for autonomous driving," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 259–270. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342403

[87] A. Bhartia, Y.-C. Chen, S. Rallapalli, and L. Qiu, "Harnessing frequency diversity in wi-fi networks," in *Proceedings of the 17th annual international conference on Mobile computing and networking*, ser. MobiCom '11. New York, NY, USA: ACM, 2011, pp. 253–264. [Online]. Available: http://doi.acm.org/10.1145/2030613.2030642