5-2016

TailoredRE: A Personalized Cloud-based Traffic Redundancy Elimination for Smartphones

Vivekgautham Soundararaj
Clemson University, vsounda@g.clemson.edu

TailoredRE: A Personalized Cloud-based Traffic Redundancy Elimination for Smartphones

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Vivekgautham Soundararaj
May 2016

Accepted by:
Dr. Haiying Shen, Committee Chair
Dr. Rong Ge
Dr. Walter Ligon

Abstract

The exceptional rise in usages of mobile devices such as smartphones and tablets has contributed to a massive increase in wireless network traffic both Cellular (3G/4G/LTE) and WiFi. The unprecedented growth in wireless network traffic not only strain the battery of the mobile devices but also bogs down the last-hop wireless access links. Interestingly, a significant part of this data traffic exhibits high level of redundancy in them due to repeated access of popular contents in the web. Hence, a good amount of research both in academia and in industries has studied, analyzed and designed diverse systems that attempt to eliminate redundancy in the network traffic. Several of the existing Traffic Redundancy Elimination (TRE) solutions either does not improve last-hop wireless access links or involves inefficient use of compute resources from resource-constrained mobile devices. In this research, we propose TailoredRE, a personalized cloud-based traffic redundancy elimination system. The main objective of TailoredRE is to tailor TRE mechanism such that TRE is performed against selected applications rather than application agnostically, thus improving efficiency by avoiding caching of unnecessary data chunks. In our system, we leverage the rich resources of the cloud to conduct TRE by offloading most of the operational cost from the smartphones or mobile devices to its clones (proxies) available in the cloud. We cluster the multiple individual user clones in the cloud based on the factors of connectedness among users such as usage of similar applications, common interests in specific web contents etc., to improve the efficiency of caching in the cloud. This thesis encompasses motivation, system design along with detailed analysis of the results obtained through simulation and real implementation of TailoredRE system.

Dedication

I would like to dedicate this thesis to my parents, who are an example that with hard work and dedication anything is achievable and without whom coming to the US and doing my Masters at Clemson University would still be a dream; my uncle, who always encouraged and motivated me; and all my wonderful friends who believed in me more than I did in myself, which inspired me to do more than I thought I could. Finally, I would like to dedicate this thesis to Dr. Haiying Shen, who helped me all throughout my Masters and gave me the opportunity for research. I am very thankful for her advising, time and support.

Acknowledgments

While writing this thesis I had the support and encouragement of all my professors, colleagues, friends and family. I would like to extend sincere thanks to all of them.

I would like to thank my advisor and committee chair Dr. Haiying Shen for her support and insights all throughout my Masters work. I would like to thank my committee members Dr. Walter Ligon and Dr. Rong Ge for their valuable advice and time for helping me significantly improve my thesis.

I am thankful to Dr. Lei Yu for his guidance, motivation and directions for this research. I would like to thank Shenghua He, who worked with me closely throughout this project. In addition, I would like to thank all my colleagues in my research group for their encouragement and advice.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we motivate and introduce the research. We present the background for Traffic Redundancy Elimination, the motivation for developing a new Traffic Redundancy Elimination system, and the objectives and contributions of our research.

Mobile devices such as smartphones, tablets etc., are becoming increasingly popular in the recent days. The proliferation of the mobile devices has led application developers to deploy a large number of mobile applications especially Internet-based multimedia applications and news application such as YouTube, Quora, Spotify, BBC, CNN etc. As user bases of these applications continue to grow day by day, web data traffic generated by these applications also experience an unprecedented growth. These devices primarily use Wi-Fi or Cellular Data Networks (3G/4G/LTE), commonly known as last hop wireless access links, to access the Web. Although the technology associated with such links are evolving periodically [33], they are not up to par to counter heavy mobile data traffic growth [12], bogging down the last hop wireless links drastically reducing their performance. Interestingly, several research works have analyzed the data traffic and shown that a significant amount of redundancy exists in the Web traffic [20] [21] [5] [35] [42] [24]. The redundancy stems primarily from common end-user activities such as repeatedly accessing, retrieving or distributing the same or similar contents over the Internet. For instance, when a mobile user during the time of the day loads multiple times the links belonging to a particular

website, several data objects such as HTML banners, multimedia like pictures, graphics, audio, video and Ad banners are fetched multiple times. These data objects usually exhibit self-similarity as they belong to a particular web domain. The repeated transfers of such redundant data objects represent a waste of network resources. In more practical terms, redundant traffic can be a serious problem for limited-bandwidth last-hop access links (e.g., wireless or cellular access networks), or even for high-bandwidth links operating at or near their capacity.

Redundant traffic can also be an issue economically if users are charged based on the traffic volumes of data sent and received. Besides cost, this redundant data communication over last-hop wireless links is actually a more significant source of power consumption on the mobile devices. To alleviate this problem traffic redundancy elimination research has attracted much attention in recent years from both academia and industries. There has been a consensus among the researchers that redundancy elimination offers significant benefits in practice. The overall benefit of data redundancy elimination is better network delivery performance in terms of higher network throughput and higher bandwidth savings leading to lower response time and higher network utilization. Traditional redundancy suppression techniques such as data compression [3] (e.g., GZip) can remove the redundant content within one object efficiently. Other techniques such as object caching include web proxy caches [46] [45] and peer-to-peer media caches [32], have also been deployed to serve the frequent and repeated requests from the cache instead of the original source. However, compression of data objects and application layer object-level caching cannot eliminate all the redundant contents alone [42]. This is because neither object compression nor object-level caching work well for contents that have been changed in only minor ways.

In order to perform effective redundancy elimination within as well as across data objects, such as same or similar documents, multimedia files, or web pages, protocol-independent redundancy elimination techniques operating on individual packets [42] [2] have been explored and investigated recently. These techniques use fingerprinting chunks, computation of hash values of the chunks and caching of redundant chunks of the data streams

and can be categorized into different types. In-network solutions, such as Cisco's [13] Wide Area Application Services (WAAS) and RiverBed's SteelHead [37] involve placing middle-boxes in network links. They cache all the data chunks from the flows that traverse the link and encodes whenever the chunks are already in the cache entries. Middle-boxes near to the source node will cache the payloads from the flows and encode the data chunks with tokens if it detects these chunks in future. The middle-box near to the destination nodes will reconstruct original data using its own cache and tokens. Other In-network TRE solutions [41, 4, 7] are proposed to reduce the traffic redundancy by creating packet caches at every router in a network and employing optimized redundancy aware routing algorithms to efficiently forward the data within the network. However, these solutions can not improve performance over last-hop links in mobile devices, since they do not conduct TRE at the last hop.

TRE techniques categorized into receiver-based TRE solutions, such as Celleration [51], AC [38], PACK [50], Refactor [39], need the end-clients to send feedback of their cache predictions or contents of cache itself to the sender so that the sender can conduct the TRE based on these feedbacks. These methods can be deployed in the mobile devices to reduce the redundancy and improve the throughput over the last hop. However, since most of the TRE operations (hash computing and feedback sending) are conducted at the receiver side, it will incur a large amount of computational overhead on the mobile devices. In addition, the sender conducts the TRE based on the prediction messages, which contains only parts of the chunk information in receiver's cache reducing the effectiveness of traffic redundancy elimination conducted. Hence, these methods impose a large amount of computation and up-link transmission overheads on mobile clients as they are constantly required to conduct hash computing and make predictions to notify the senders. As most of the mobile devices have limited compute and up-link bandwidth resources, these methods can provide limited benefits to the mobile users. In addition, as they do not employ a fully synchronized cache between the sender and the receiver, the sender can not be fully aware of the contents in receiver's cache, which reduce the redundancy hit ratio and the effective

bandwidth savings attainable by TRE. EndRE [2], a sender-based TRE approach, offloads the TRE computing operations from the clients to the application servers, which saves the limited computation and energy resources of the mobile devices. Additionally, by transmitting the hash information along with the chunks during the transmission, it can keep sender's cache between receiver's cache synchronized, so that the sender can be aware of all the chunks cached at the receiver, which will improve the redundancy hit probability, thus the bandwidth savings. Nevertheless, EndRE is agnostic to the applications and conducts TRE against the traffic from any application. Since memory resource in the mobile device is limited, the chunks from mobile applications with lower redundancy ratio will occupy the cache space, which will reduce redundancy hit ratio and the bandwidth savings caused by TRE.

All of the existing TRE design or solutions are not well apt for mobile computing environment for either or both of two important reasons.

- Application agnostic TRE solutions will result in reduce chunk hit ratio and eventually reduced bandwidth savings since "will be unused" chunks from less redundant application resides in limited memory resource for prolonged period of time.

- Compute, uplink transmission and energy cost imposed by the existing TRE solution thwarts the cost saved from benefits of performing TRE. Limited computation capacity, uplink bandwidth and battery of the mobile devices requires TRE to operation efficient.

In order to make the best of the limited cache resource at the mobile devices to improve the operational efficiency of TRE, we propose a novel TRE system named TailoredRE consisting of a pair of TailoredRE clone (in the cloud) and client (in the smartphone). With the advent of cloud computing, the cloud has been leveraged to extend the capacity of mobile devices and improve the energy efficiency. Considerable research works [11] [40] [16] [15] have been proposed to offload the application execution from mobile devices to the cloud. Their solutions improve the execution time and save the battery on

the mobile devices by running computation intensive tasks on the cloud servers via computation offloading. Following a similar stance, in TailoredRE, the clone acting as a proxy for the smartphone operates in the cloud and performs compute intensive TRE operations including chunk partitioning, hash computing, chunk caching and redundancy elimination. To address the inefficiencies in other TRE methods, the TailoredRE clone provides personalized TRE service to the smartphones based on the user's application usage behaviors. Firstly, TailoredRE clone keeps track of the application usage activities of the smartphone along with redundancy level associated with each individual application by analyzing data requests and responses pertaining to the applications. Based on the metadata collected, the clone calculates a TRE value for each individual application and makes a decision on which mobile application TRE will be conducted against, based on the highest TRE value. By conducting TRE against the application with the highest TRE value, TailoredRE can always avoid the cache pollution from the application with lower redundancy ratios, which will improve the cache hit ratio greatly, which in turn enables our system to provide higher efficiency than other methods.

In addition to the tailored RE design, our proposed TailoredRE system also considers sharing the cache resources between different individuals in order to utilize efficiently the cache resource in the cloud. TailoredRE system also considers to share the caches between users in order to conserve much more cache resources in the cloud. But there are two challenges to implement the cache sharing between clones in the cloud. The first challenge is to decide which clones can share the cache since if clone's localities are far, the communication cost between the clone and cache will be high. The second challenge is to maintain the synchronization between the clone's cache in the cloud and that in the individual mobile device.

To solve the first challenge, we proposed a two-step grouping algorithm to group the clones, firstly geographically and then based on user's interest similarity, into clusters. The clones within the same cluster will share the cache while the clones that belong to different clusters will not. To solve the second challenge, we designed a two-layer caching

mechanism, in which there are two kinds of caches, the logical cache in the upper layer and the physical cache in the lower layer. The chunk information, which occupies tiny cache resource, will be cached in the logical cache and used to keep synchronization between the clone and client while the raw data chunks will be cached in the physical cache, which can be shared by all the clones in the same cluster.

In summary, the contributions of our work are listed as follows:

- **Redundancy analysis and profiling.** Analyze redundancy ratios of real traffic data traces generated by mobile applications during daily use of smartphone and identify that there is indeed significant redundancy in the traffic and verify whether different applications have different redundancy ratios.

- **Personalized TRE.** Use the redundancy profile to tailor the RE mechanisms in our method and perform TRE against certain applications rather than agnostically against all data so that we avoid cache pollution and obtain higher cache hit ratio resulting in higher bandwidth savings.

- **Computing offload.** Leverage the rich computation resources of the cloud to offload the operational cost of performing the TRE from the mobile devices to their clones in the cloud. Clones in the cloud are fully aware of the cache contents of their respective smartphones.

- **Cache sharing.** We also group the multiple individual clones into clusters based on the similarities of their user's interests to avoid duplicate inter-user redundant chunks among clones in the same cluster, which improves the efficiency of cache management and conserve largely the cache consumption in the cloud.

- **Real trace experiments.** Developed our TailoredRE simulator and use the real traces from mobile applications to simulate the TRE redundancy hit ratio, bandwidth savings and throughput in our proposed TailoredRE system. Moreover, we developed a prototype of TailoredRE to show the benefits of our system in real time traffic.

In this chapter, we have presented need and motivation behind the redundancy elimination. In addition, we reviewed various existing TRE systems and their drawbacks. We briefly described our proposed system and highlighted the contributions our work. The rest of this thesis is organized as follows. Chapter 2 discusses related works. Chapter 3 describes in detail our system design, architecture and mechanisms involved in our system. Chapter 4 presents the analysis of experiments and results obtained through simulation and real implementation of TailoredRE system. In Chapter 5, we summarize and conclude the research.

Chapter 2

Related Works

The intent of this chapter is to acquaint the readers with recent developments conducted in the area of traffic redundancy elimination. We first introduce the studies that have been conducted in the recent past to survey and explore various aspects involved in redundancy elimination. Then, we introduce different categories of TRE solutions that have been proposed in the recent times. Then, we present a review of previous studies that have designed systems under each category of TRE solutions.

2.1 Redundancy Elimination: Inception and Evolution

Since the inception of World Wide Web, a sector of research areas in academia and Information technology industries has been devoted to focused on the problem of improving Web performance by reducing download times and bandwidth requirements. In 2000, Spring *et al.* [42] introduced the concept of Redundancy Elimination (RE) based on the insight is that dynamic and personalized web contents which comprise of most of the Web is not caught by traditional Web caching platforms such as Squid [43] but likely to derive from similar information [45] [36]. Instead, they suggested a protocol-independent technique for identifying redundant information that does not suffer from the disadvantages posed by web caching platform. The protocol-independent technique is an adaptation of algorithm first

proposed by Manber in [28] to find similarity between files in a large scale file system to reduce storage overhead. The algorithm also termed as Rabin Fingerprinting is subsequently applied by Broder [10] to detect similar Web documents. They utilized the redundancy suppression technique to analyze network traffic and found that there is a high level of redundancy in the traces collected. Their main focus was to show the potential for byte savings by exploiting this redundancy. However, they have also proposed an architectural description of a shared cache architecture which formed the basis of In-Network RE solutions (also known as "Middleboxes" or WAN Optimizers). We review In-Network RE solutions in detail in the next section.

With the growing popularity of Protocol-Independent Redundancy Elimination techniques and wide scale deployment of redundancy elimination middleboxes to improve the effective bandwidth of network access links of enterprises and data centers, link loads in small ISP networks [14] [19], a few years later, Anand *et al.* in [5] have conducted a large scale trace-driven study using several terabytes of packet payloads to investigate on the effectiveness and efficiency of packet-level protocol-independent. While middleboxes have been predominantly used in data center and ISP links to improve, their study found that 75% to 90% of bandwidth savings achieved by middlebox's bandwidth savings is due to redundant byte-strings from within a particular end-user traffic implying that pushing redundancy elimination capability to the end hosts, i.e. an end-to-end redundancy elimination solution, could obtain most of the middleboxs bandwidth savings. This finding eventually motivated a number of researchers to pursue the design of End-to-End Traffic Redundancy elimination system. They also explored a wide array of redundancy detection algorithms and compared them in terms of their effectiveness and efficiency. Another key findings included in this study claims that relatively new redundancy elimination algorithms such as MODP, MAXP and Winnowing outperform the widely used Rabin fingerprint-based algorithm by 5%-10% on most traces and by as much as 35% in some traces. From their trace analysis, they also observe that redundant chunk matches in traces follow a Zipf-like distribution implying that increasing the chunk cache size would offer only diminishing returns. They also claim from

their spatial and temporal analysis that most matches (70%) are small in size (less than 150 bytes) while only about 10% of matches are full packet matches and most matches are from recent packets in the cache respectively.

Halepovic *et al.* in [24] studied many predominant factors that influence the practical effectiveness of protocol-independent redundancy elimination (i.e., the byte savings achieved on a network link). Based on their study, they claim that factors such as data chunk size, the chunk selection algorithm, the sampling period, and the cache replacement policy. Based on a careful investigation of the impacts of chunk size, sampling period, and cache management on the TRE benefits, they proposed an array of new techniques such as include a size-based bypass, chunk overlap, savings-based cache management, and content-aware chunk selection, intended to enhance the current protocol-independent TRE techniques. We evaluate these techniques on full-payload packet traces from university and enterprise environments and demonstrate their effectiveness.

Recently, Zhang *et al.* [48] have conducted a detailed survey on different possible techniques involved in major processing mechanisms of Traffic Redundancy Elimination which includes fingerprinting, cache management and chunk matching mechanisms. They have also elaborated on currently deployed redundancy elimination systems and different ways to improve their performances. So far, we have seen works that have conducted a survey on different aspects of redundancy elimination systems. We review different approaches employed to perform redundancy elimination in upcoming sections.

2.2 In-Network RE Solutions

In this section, we review in detail about some of the widely deployed commercial WAN Optimizers and their drawbacks. We also present a few Network-wide Redundancy aware routing solution proposed as an extension to overcome the drawbacks caused by traditional WAN Optimizers. Finally, we also look at their shortcomings.

Several commercial In-Network TRE solutions [13] [37] involve placing a "WAN

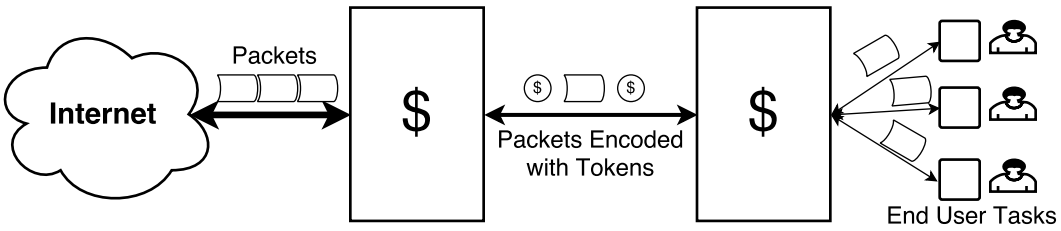


Figure 2.1: Shared Cache Architecture

Optimization" middle-boxes across bandwidth-constrained links. As we have mentioned in the last section, the technique of WAN optimization was first pioneered by Spring *et al.* in [42]. They envisioned that protocol-independent redundancy suppression technique can be used to improve the throughput of the back-haul WAN links with the architecture shown in Figure 2.1. In this architecture, caches enabled with redundancy suppression algorithm are placed at both ends of a bandwidth-constrained link. Each cache converts repeated strings into tokens and passes these encoded, smaller packets to the other end of the channel, where the original packet is reconstructed. Tokens are passed in place of replicated bytes between the two cache locations on opposite sides of a bandwidth-constrained link. User machines continue to communicate using un-encoded packets. The motivation behind this architecture is trading memory and computation for bandwidth savings.

Redundancy Elimination Middleboxes also referred to as WAN Optimizers typically works similar to the above mentioned architecture by caching all data payload or chunks from the flows that traverse the link. By maintaining synchronized cache across links, it removes redundancy by encoding the flow whenever duplicate chunks entries are detected in the cache. WAN Optimizers could take the form of either hardware appliance or virtualized software. Cisco's [13] Wide Area Application Services (WAAS) which implements a combination of data redundancy elimination, data deduplication and protocol acceleration to achieve bandwidth gains. RiverBed's SteelHead [37] is a WAN Optimizer which employs data reduction and TCP acceleration techniques to optimize WAN links. These WAN optimizers or Middle-boxes can be cost-effectively deployed only in in-network WAN links and not feasible for performance improvement of the last-hop wireless links. Middle-

box, implemented in commercial WAN accelerators can eliminate redundancy over a single in-network link and only remove intra source-destination pair redundant traffic and cannot remove inter source-destination pairs redundancy.

To overcome this problem Anand *et al.* proposed Network-wide RE solutions [4] [7]. The contribution of their solution include a routing algorithm to eliminate both Intra and inter source-destination redundancies by routing potentially duplicate packets onto common links. The proposed redundancy aware routing algorithm in [4] essentially solves a linear optimization problem to compute a route with minimum cost for a given physical network topology and traffic redundancy profiles. The linear problem is subjected to link capacity and flow conservation constraints. Following this Song *et al.* proposed a greedy heuristic algorithm for redundancy aware routing with limited resources in [41]. The proposed algorithm iteratively reroutes the traffic for each source-destination pair in order to minimize the network resource cost until the number of overlay nodes reaches its maximum limit, or the traffic for all the source-destination pairs has already been rerouted, or the network cost cannot be reduced further. SmartRE [7], proposed by Anand *et al.* as a sequel to [4] aims at coordinating the network-wide redundancy elimination split caching responsibilities across multiple routers in accordance to the available resources to improve the effective resource utilization of the routers. Network-wide RE solutions aim to improve the overall bandwidth utilization within a network at the cost of memory and computational requirements at each router in the network. ICN-RE [34], proposed by Perino *et al.* aims to bridge Information-Centric Networking (ICN) and Redundancy Elimination (RE). ICN, a novel form of networking centered around information or content, where the network elements, such as routers, are aware of the data they transfer requiring complex routing algorithm and ubiquitous caching. A Web document or file in ICN-RE consists of a set of chunks, and each chunk associated a fingerprint derived by hashing. Each router caches the chunk along the path and caches placed at routers in the network are indexed using the fingerprints and lookups are performed via exact match. Finally, a server informs clients, upon requesting a document, about the fingerprints in the requested file using a manifest file that contains the

fingerprints of the two chunks composing the requested content. If ICN router happens to previously cached chunks of the requested file, it serves the clients rather than forwarding the client to the original content server. It should be noted that ICN-RE is designed for a content-centric network, not yet available today.

The major drawback of the In-Network solutions as a whole is that although they significantly improve the bandwidth utilization of backhaul networks, due to non end-to-end nature of TRE performed, it fails to improve last-hop access networks.

2.3 Receiver Based RE Solutions

In order to provide significant performance improvement on last-hop access networks, it is necessary for RE technique to exhibit end-to-end nature resulting RE to be pushed into network stacks of sender and receiver. Depending on where most of the functionalities in RE lies, we can classify such end-to-end techniques into either sender based and receiver based. In this section we present existing received based RE solutions (also known as a prediction-based RE solutions) along with significant challenges posed by these techniques to the current mobile computing environment.

To overcome the shortcomings of In-Network TRE solutions, several receiver based (prediction based) TRE solutions such as [50] [38] [51] [39] have been proposed. They primarily involve receiver to cache. Most of the receiver based solutions that have been proposed primarily aim at improving the performance of the last-hop wireless links. In this section, we examine the techniques associated with these solutions.

Zohar *et. al* proposed Celleration [51], a TRE system designed for the new generation of data-intensive cellular networks. Celleration activates three mechanisms at the gateway, namely, flow coding, ad-hoc learning of mobile devices, and flow reduction. A Celleration-enabled gateway, located at the cellular network Internet entry point, extracts similarities in repetitive chunk flows across mobile devices with the flow coding mechanism, predicts individual mobile users future data with ad-hoc learning mechanism, and enables

bandwidth savings for mobile end users with the flow reduction mechanism. In the flow coding mechanism, the gateway enabled with the flow coding mechanism continuously parses the crossing flow to a sequence of variable sized, content-based chunks, which will be signed by using SHA-1. The chunks signature sequences will be stored in the gateways local cross-user signature store. It recognizes a crossing flow by its chunks signatures, which will also be used to look up the potential future data in the cross-user signature store. Once a crossing flow has been recognized, the ad hoc learning between the gateway and the mobile device. In this phase, the mobile end device will respond with a list of time-limited approvals by checking whether the corresponding data of the predicted chunks signatures are in its local cache or not. If the ad-hoc learning indicates that some predicted data chunks already exist in the cache of the mobile device, the flow reduction mechanism of Celleration, will be activated by refraining the gateway from forwarding the approved chunks to the mobile device as they arrive at the gateway. The shortcoming of Celleration lies in ad-hoc learning phase where the computation and up-link transmission overhead incurred when the mobile device computes and sends a list of time limited approvals. In addition to the inefficiency, the prediction failures that could occur in the gateway might result in retransmission overhead.

Zohar *et. al* also proposed Prediction-Based Cloud Bandwidth and Cost Reduction System (PACK) [50] which aims at eliminating the traffic redundancy for cloud computing customers. PACK offloads the computation cost of TRE from the cloud servers to the end-clients. The receiver in PACK upon receiving the data stream parses it into chunks, computes the signature and caches them along with their associated meta-data in the chunk-chain store if no matches in the signature are found. If a matching signature is found in the cache, the receiver retrieves the corresponding chunk chain and sends PRED message containing the range of the predicted data, the hint (e.g., the last byte in the predicted data) and the signature of the chunk. Upon receiving, the sender verifies the hint and signature of the PRED message and if matched the corresponding range of the data is replace with PRED-ACK removing the redundancy. As PACK offloads the expensive TRE operation end-clients, it results in starvation of resources in already resource-constrained

mobile devices. Moreover, as the sender if not fully synchronized with receiver, failure in predictions can cause additional overhead due to transfers of PRED and PRED-ACK messages resulting in reduced bandwidth savings. It is also to be noted that PACK performs redundancy elimination is limited to TCP traffic only.

Sanadhya *et. al* proposed Asymmetric Caching (AC) [38] where they consider some baseline assumptions that sender is never completely aware of the contents in the receiver's cache. Their design imposes the receiver to send the contents of the cache to the sender according to reactive strategy. In reactive strategy, feedback is sent upstream only when data traffic is flowing to the destination. In this process, the receiver tries to match arriving flow (byte stream) with past data in the cache and feedback that is likely to be most useful with respect to the arriving flow is selected. The receiver performs the feedback selection by partitioning the arriving flows into flowlets. Flowlet, a contiguous subset of a byte stream, is computed using technique described in [8] which involves segmenting a piecewise stationary time series into a separate time series which are individually stationary . If any of the currently arriving flowlet is matched with one of the past flowlets in cache. Along with regular cache, the sender maintains the feedback cache to maintain the feedbacks from the receiver. Upon receiving the data, the sender computes hashes and looks them up in its regular cache. If there is no hit, the hash is added to the regular cache, but the same hash is then looked up in the feedback cache. If either of the cache lookups results in a hit, the hash is sent to the receiver. Otherwise, the original data segment is sent as-is. It is worthwhile to note that Asymmetric Caching (AC) consumes resources of already resource constrained mobile devices in order to perform compute expensive flowlet extraction and bandwidth expensive upstream transmission of the feedbacks.

A special array of receiver based solution such as Ditto [17], RTS-id [1], REfactor [39] involves leveraging wireless overhearing to perform redundancy elimination. These techniques operate on the fact that wireless radios can opportunistically overhear packet transmissions on the network. Therefore they are also termed as Opportunistic caching.

In RTS-id [1], the receivers in a wireless network cache the overheard packets, and

the sender adds a special ID to the 802.11 RTS packet so that the receiver can check if the data packet to be transmitted is in its cache. RTS-id operates by augmenting the standard 802.11 RTS/CTS process with a packet ID check, so that if the receiver of an RTS message has already received the packet in question, it can inform the sender and bypass the data transmission entirely. RTS-id relies on overhearing packets (both headers and payloads) in full. Since the probability of overhearing a complete packet by the receiver is minimal the performance improvement achieved using this approach would be very minimal. Many content-centric approaches such as Ditto [17] and REfactor [39] have been proposed as an alternative.

Ditto [17] proposed by Dogar *et. al*, is the first content overhearing system. Ditto involves content-based naming and caching of overheard data to eliminate redundant transmissions. In Ditto, content overhearing is implemented at the granularity of 8-32KB chunks and employs an unconventional pull-based transport protocol called DOT to implement content-based naming. This prevents Ditto from being applicable to short flows and flows with dynamic content, which makes up a significant fraction of Web traffic flows [5] and are typical of request-response applications. Additionally, Ditto caches content only at mesh nodes, not wireless clients, providing no benefits over the last hop wireless links.

REfactor [39] proposed by Shen *et. al* addresses the drawbacks in Ditto [17] and RTS-id [1]. REfactor considers an Access Point (AP) operating in infrastructure mode with some number of associated clients where they overhear and cache packets. AP before forwarding to the incoming packets to the client it looks up for duplicate strings of bytes that appeared in earlier packets in its cache. Based on the reception probability vector associated with each client, AP then calculates the expected benefit for the receiving client from performing RE on the packet. If the likely benefit is high, the AP eliminates the duplicate bytes from the packet and inserts a shim instead. The shim contains a pointer to a memory location in the client so that it can decode the packet. The problem with REfactor arises if the content is not cached in the client, the client needs to request the missing content, incurring additional transmissions. This penalty is imposed when the APs

estimate of whether the client has the content is incorrect.

One of the crucial problems associated with the receiver based (prediction based) TRE is that they demand computational or up-link bandwidth resources from the receiver to perform TRE. In addition to that, in the receiver based methods, the sender and receiver caches are not fully synchronized and the sender has to rely on the receiver's predictions, thus affecting the maximum bandwidth savings obtainable.

2.4 Sender Based RE Solutions

In this section, we present a detailed review on sender-based RE solutions along with its shortcomings.

A special class of sender-based RE solution called preliminary negotiation approach is suggested in the early LBFS work [30] as well as in Wanax [25]. The LBFS protocol is built on top of NFS version 3 and uses four special Remote Procedure Calls(RPCs) such as GETFP, MKTMPFILE, CONDWRITE, and COMMITTMP. While reading client requests description of chunks in file ¡*hash, size*¿ pairs and missing chunks (chunks not in its cache) are read using normal NFS READ RPC. Write is buffered locally until close of file and are performed using atomic updates where data to new temporary file and commited contents of temporary file to file being written. WANAX is a TRE system tailored for the developing world where WAN bandwidth is scarce. In WANAX, three mechanisms are activated, namely, multi-resolution chunking (MRC), intelligent load shedding (ILS) and a mesh peering protocol. Multi-resolution chunking (MRC) is enable to achieve high compression rate and high storage performance with small memory pressure. Intelligent load shedding (ILS) is activated to maximize effective bandwidth by adjusting storage and WAN bandwidth usage. Mesh peering protocol is enabled to reduce latency by fetching content from relatively higher-speed local peers instead of over slow WAN links when the content is available from other local peers, respectively. WANAX incurs a three-way handshake latency for non-cached data. The major drawback of these solutions is the need of extra

memory buffers and the added transmission delay.

EndRE [2], a sender-based TRE solution proposed by Aggarwal *et al.*, offloads the TRE computation overhead from the clients to the application servers, which saves the limited computation and energy resources of the mobile devices. Additionally, by transmitting the hash information along with the chunks during the transmission, it can keep the cache between the sender and receiver synchronized. The synchronization makes the sender be aware of all the chunks cached at the receiver and improve the redundancy hit probability, which improves the TRE efficiency and thus resulting in much higher bandwidth savings. Nevertheless, EndRE is agnostic to the applications, which means that it conducts TRE against the traffic from any application. Since memory resource in the smartphone is limited, the chunks from applications with lower redundancy ratio will occupy the memory space, which reserves less space to the applications with high redundancy ratios. As a result, the redundancy hit ratio during TRE will degrade and so will the bandwidth savings.

2.5 Summary

In this chapter, we have discussed the development and evolution of various types of Redundancy Elimination systems and have reviewed many studies conducted for the improvement of the TRE systems. There has been an unique contribution by every TRE system that has been proposed in solving a particular aspect of Redundancy Elimination problem. But all of the existing methods have not adapted to the wireless or mobile computing environment in addition to not taking into account the user preferences towards the web contents. TailoredRE, unlike other existing methods, leverages rich computation resources of the cloud to design efficient TRE for smartphones by offloading computation from smartphones to its clones in the cloud. It also takes into consideration user's predilection towards certain applications and tailors the TRE mechanism to perform TRE against those applications rather than agnostically efficiently utilizing the limited cache resource to achieve higher bandwidth savings. By employing clustered clones, we avoid caching of

duplicate inter-user redundant chunks by the clones, thus minimizing storage overhead in the cloud.

Chapter 3

System Design

The intent of this chapter is to explicate the system design of TailoredRE. First, in order to help readers get a clear perspective of various aspects of any TRE system, we look at the functional overview of a generic TRE system. Then we briefly introduce the components of TailoredRE and describe high-level functionality present in each of these components. We then proceed to the architectural details of TailoredRE is introduced to explain the methods employed by TailoredRE. Finally, each component in TailoredRE and their interactions are described in detail.

3.1 Overview of Generic RE

The main idea of protocol-independent redundancy elimination is to encode the outgoing packets by replacing identified redundant data chunks with fixed-size meta-data. As all the protocol-independent RE techniques operate at the packet level or on the packet payloads, we can also refer these techniques as packet level RE techniques. At the receiver or end-hosts, the packets are reconstructed by replacing the encoded content from the chunk cache by using the index information carried in the encoded packets. The packet-level RE techniques rely on deploying a fingerprint table and a cache at each node in a network path or in the case of end-to-end techniques at end hosts.

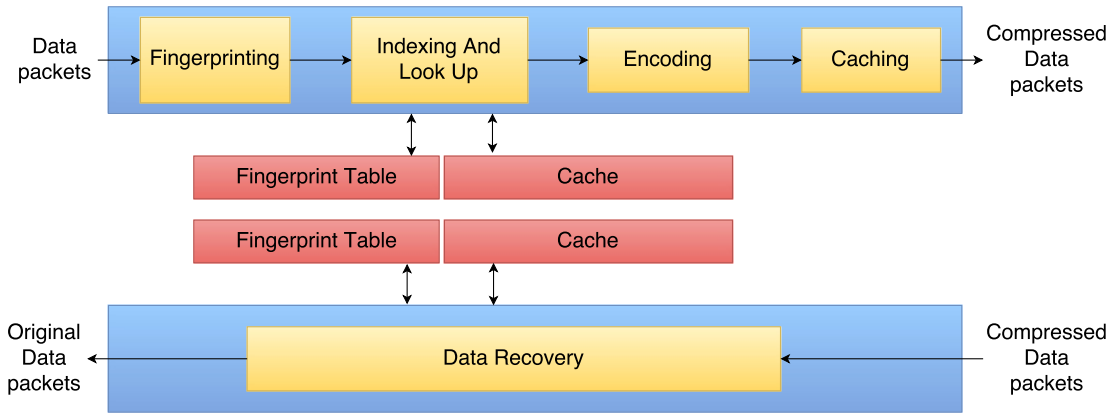


Figure 3.1: Processing blocks in a TRE system

A typical protocol-independent packet-level TRE implementation is shown in Figure 3.1. For every incoming packet in a particular direction, the TRE algorithm first computes a set of fingerprints by applying a hash function to each chunk of the packet, which is a sub-string of the packets payload. Limited by the size of the fingerprint or hash table, only a subset of these fingerprints or hash values is selected as its representative fingerprints in some way. The various algorithms employed in the process of selection of representative fingerprints along with their trade-offs are discussed in forthcoming sections. Both the representative fingerprints and pointers pointing to the locations of the chunks in the packet cache used to calculate its corresponding fingerprints are stored in the fingerprint table. Each representative fingerprint is then checked against the fingerprint table to find a matched data chunk in the packet cache. If such a matched chunk in the packet cache is found, the original data chunk in the packet is encoded with a metadata, which consists of sufficient information to reconstruct the encoded data chunk at the receiver side, such as a fingerprint. In practice, the size of such meta-data is much smaller than that of the original data chunk. In this example, two chunks are identified as redundant data chunks and the packet is encoded by replacing the original redundant data chunks with the corresponding meta-data. When the end-host receives the encoded packet, it will reconstruct the original packet following the information carried in the metadata by using the fingerprint table and packet cache at the receiver side.

The major processing stages involved in redundancy elimination include fingerprinting, indexing and lookup, storing data, and data encoding and decoding as shown in 3.1. Fingerprinting, also called chunk selection, facilitates the identification of redundant chunks within and across the packets. For every incoming packet in a particular direction, it calculates a set of fingerprints for each packet by applying a hash function to each chunk of the packet and selects a subset of these fingerprints as the representative fingerprints. Each representative fingerprint is then checked against the fingerprint table in the processing stage of indexing and lookup. If one fingerprint already exists in the fingerprint table, a redundant chunk is then identified and its corresponding position of the matched region in the packet cache is also located by using its location information stored in the fingerprint table. Hence, the lookup procedure in this stage involves two parts: fingerprint lookup in the fingerprint table and redundant chunk lookup in the packet store.

If one or multiple redundant chunks have been identified in an arriving packet, the packet will go through an encoding procedure by replacing every identified redundant chunk by its corresponding fingerprint description, which consists of the fingerprint as well as the byte range for the matched region in the packet cache. Finally, the new packet is inserted into the packet store and its representative fingerprints are also indexed and stored in the fingerprint table together with the location information of the chunks used to calculate these representative fingerprints in the packet cache. Data decoding performs the reverse operations of data encoding and tries to reconstruct the original packet from the compressed packet by retrieving the chunks from the packet cache by using the information carried in meta-data. The TRE decoder uses the fingerprint value stored in the meta-data to check against the fingerprint table. If such a fingerprint value is found in the fingerprint table, the data chunk will be fetched from the packet cache by using the pointer information stored in the fingerprint table and the count of the redundant bytes before and after the chunk used to calculate the fingerprint. Then, the original packet is reconstructed by replacing the meta-data with these fetched data chunks from the packet cache.

As described above, several mechanisms are activated in the implementation of

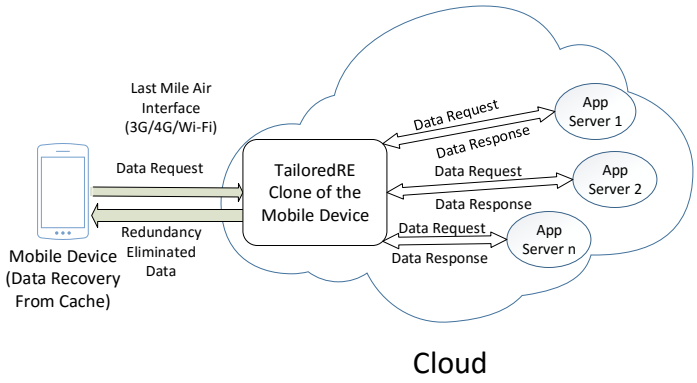


Figure 3.2: TailoredRE System

protocol-independent RE. In TailoredRE system, the core mechanisms involved resembles that of the above. We look at TailoredRE system in detail in forthcoming sections.

3.2 System Overview of TailoredRE

The objective of this research is to design an efficient TRE system to achieve higher network throughput and eventually higher bandwidth savings that could be achieved by using existing TRE solutions. Thus, we developed a system, TailoredRE, a personalized cloud-based redundancy elimination system, that takes into account critical aspects of Traffic Redundancy elimination in smartphones. The overall view of the TailoredRE system is depicted in Figure 3.2. Here we see that TailoredRE system executing in the cloud parses, identifies and eliminates redundant data traffic flowing from the application servers to the mobile device, meanwhile acting as a proxy agent for the mobile device forwarding data requests and responses pertaining to the mobile device. A brief description of the core TailoredRE's functional advancements which make it headway against existing solutions is provided in this section.

Our proposed TailoredRE system consists of following four critical functional components: *Cloud clone based TRE, Application-adaptive RE, MAXP-chunk partitioning algorithm* and *Cache sharing among clones*.

Cloud clone based TRE: As we have seen in Chapter 2, a plethora of existing TRE

solutions foist resources of already resource-constrained mobile devices. TailoredRE system, on the other hand, leverages the rich computation resources available the cloud to offload the operational cost of performing the TRE from the mobile devices to their clones in the cloud. In the cloud, the TailoredRE clone, acting as a proxy for a mobile device, performs major TRE operations such as redundancy detection in data streams, hash computation, caching and encoding of data streams. Clones and mobile devices maintain a synchronized caches between them.

Application-adaptive RE: Different from other TRE systems, TailoredRE clone provides personalized TRE to its mobile device users based on the profile information obtained through Redundancy Profiling. Redundancy profiling involves analysis of redundancy metrics such as the ratio of redundant bytes to total bytes in byte streams of traffic belonging to a wide array of applications a mobile user uses on his mobile phones. Through this analysis, we select an application against which redundancy is conducted, rather than performing redundancy application agnostically. The detailed review of analysis and selection process will be described in forthcoming section. The application adaptiveness of Redundancy Elimination constitutes the personalization feature of our system.

MAXP Chunk Partitioning algorithm: There exists a plenty of chunking or fingerprinting algorithms to compute a set of representative fingerprints in a data stream. These algorithms pose a significant trade-offs between each other. TailoredRE system adopts MAXP algorithm to find the boundaries by looking for the local MAXima in a sampling period of P, such that the traffic data are divided into chunks by these boundaries. The basis behind the algorithm selection is detailed in forthcoming section.

Cache sharing among clones: Mobile traffic data naturally exhibit Inter-User Redundancy since multiple users access same popular contents in the web during a given period of time [48] [26]. In order take advantage of this phenomenon and utilize the cache resource efficiently in the cloud, we design a cache sharing mechanism in which clones are clustered based on the profile information of the mobile users. Then, the clones in the same cluster will share their caches. The detailed review of the sharing mechanism will be

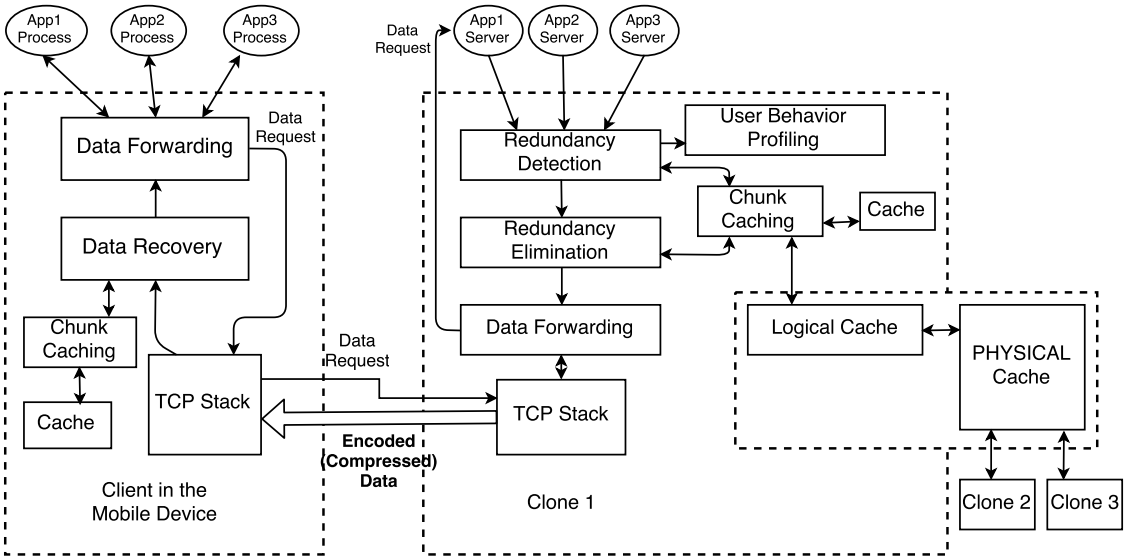


Figure 3.3: Architecture of TailoredRE.

described in forthcoming section.

3.3 Cloud-Clone based TRE

In this section, we list the functional components of Cloud-Clone and explicate all the functionalities that reside in these components. We also discuss in detail how these functional components interact with each other and with the clone to perform redundancy elimination. Figure 3.3 shows the functional architecture of TailoredRE system.

As shown in Figure 3.3, the architecture of TailoredRE system includes the following components: Data Forwarding (DF), Redundancy Detection (RD), User Behavior Profiling (UBP), Redundancy Elimination (RE), Chunks Caching (CC) and Data Recovery (DR). The cloud-based Clone in TailoredRE system includes Data Forwarding (DF), Redundancy Detection (RD), User Behavior Profiling (UBP), Redundancy Elimination (RE) and Chunks Caching (CC), while TailoredRE client includes Data Forwarding (DF), Chunks Caching (CC) and Data Recovery (DR).

Clone in the TailoredRE, acting as a proxy for the mobile device, is the gateway

for all mobile device's data access to the Internet. The clone conducts TRE in the data response supposedly forwarded to the mobile device. Upon receiving the data response packets from the remote server, the clone strips the packet and segregates the application raw data, then deliver it to Redundancy Detection (RD) component as an input. The RD component will partition the data into chunks according to MAXP algorithm which we will explicate later in this chapter, compute their hashes and check whether the chunks exist in the cache. If a chunk exists in the cache, the Redundancy Elimination (RE) component encodes the data stream with its hash value, otherwise, the data stream is encoded with newly found hash value along with the chunk. The Chunks Caching (CC) component is responsible for caching the chunks with the Least Recently Used (LRU) strategy.

At the client side, the Data Recovery (DR) component decodes the incoming data stream sent by the clone. If a hash value decoded in the data stream is found in its cache, it retrieves and recovers the corresponding chunk. For other parts of the data stream which has been found along with their chunk values, the client's Chunks Caching (CC) component adds them to its cache. The recovered data will be delivered to the higher modules in the application process for further processing.

3.3.1 Data Forwarding (DF)

Data Forwarding (DF) component is responsible for performing basic forwarding function in both the TailoredRE clone and client. The clone performs Data Forwarding (DF) like a proxy for its mobile device, and it forwards all the requests from the mobile applications to the application or web servers in the cloud and the response data from the servers back to the smartphones. Upon receiving the data response from the application server, it will perform TRE operations and send out the encoded (compressed) data to the mobile device. At the client side, its Data Forwarding (DF) component will forward the recovered data to the corresponding higher modules in the application process.

In TailoredRE system, Data Forwarding (DF) component is responsible for the simple data transmission work, and let other components handle much more complicated

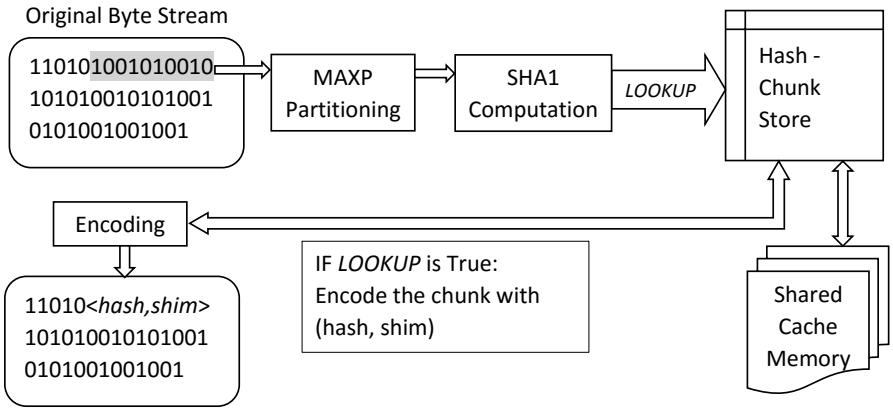


Figure 3.4: Redundancy Detection and Encoding Process

data processing task.

3.3.2 Redundancy Detection (RD)

Redundancy detection component in the clone of TailoredRE performs most vital function of detecting redundancy in the data streams. Redundancy detection involves partitioning the traffic data into chunks, computation of hashes for the chunks, and encoding the data stream containing redundant chunks with their hashes. An overview of redundancy detection is provided in Figure 3.4 and can be described in detail as follows.

3.3.2.1 Chunk partitioning

As we have seen in Data Forwarding section, the clone receives the data response packets from the application server pertaining to the mobile device's ongoing data request. After receiving these packets the clone strips off the packets' header information and partitions them into chunks with variable size by using MAXP algorithm, which is used to find the boundaries of the chunks. We will describe the details of MAXP algorithm in later section.

3.3.2.2 Hash computing

After the data is partitioned into chunks, hash computing module of the Redundancy Detection (RD) component will compute the hash value of a chunk with SHA-1 hash function [18]. SHA-1 is a cryptographic hash algorithm where SHA stands for Secure Hash Algorithm. In our design, we use SHA1 algorithm as the hash function whose result is expressed as a 160 bit hex number. After the data stream has been parsed by this module, the output produced will contain a list of chunks with their corresponding hash values.

3.3.2.3 Chunk Lookup

After the Hash computation, the data chunks After we check the hash rather than the chunk data directly since the chunk size is much larger and needs much more time to compare. But sometimes, hash collision, that is the two different chunks have the same hash, may happen, which will result in the error data recovery at the client in the mobile device. In order to avoid this kind of collision and guarantee the reliability of TailoredRE system, we compare the chunk after the hash matches. If both the hashes and chunks match, we can make sure that this chunk is redundant and give the checking results to the Redundancy Elimination (RE) component. Since the cache cloud is fully synchronized with that in the mobile device, this chunk definitely will be found in the cache in the mobile device, thus we can replace it with hash during data transmission. Since chunk size is comparatively larger than the hash size, the bandwidth will be saved when we transmit the hash rather than the chunk itself.

3.3.3 User Behavior Profiling (UBP)

User Behavior Profiling (UBP) is a component of clone in TailoredRE that collects and keeps track of the statistics of meta-data pertaining to the data responses that is being received and forwarded to the mobile devices. The duty of UBP component is to capture the application identity the data response has been forwarded to and data request is sent from, the size of forwarded data and the cumulative sum of the size of the chunks

that hit the cache periodically. The primary purpose of this collection is to analyze and profile the usage of applications by every individual user. The analysis results of the UBP component include the redundancy ratio and activity of each application, the TRE factor value, which indicates the expected benefits obtained by performing redundancy elimination against the data responses of a particular application is found. In overall, the primer of this component is to deduce the application usage behavior of the users of the mobile device, we have termed this component as User Behavior Profiling component. Based on the results from the User Behavior Profiling component, TailoredRE system can adapt its Redundancy Detection and Redundancy Elimination functions to be conducted against the data responses of the particular application which maximize the expected bandwidth savings attainable thus making it application-adaptive RE. Later on, in this chapter, we will also see that this module also provides significant information regarding clustering of clones. Both of them will be described later in our paper.

3.3.4 Redundancy Elimination (RE)

While the data response traffic of an application traverse through the clone in TailoredRE, the clone initially partitions them into chunks, and then computes the hashes of these chunks as the fingerprints. Subsequently, it will check whether the chunks exist in the cache. If a chunk exists, it will replace the chunk with its hash in the outgoing data stream, shown in 3.4, and update the chunk's offset in the clone's cache at the same time, otherwise, it will maintain the chunk in the data stream, and store the chunk and its hash into the clone's cache. Since in our system, the compute expensive TRE operations notably hash computation operation is offloaded to the clone in the cloud to conserve the computing resource in the mobile device, the clone will send the hash along with its chunk. Upon receiving the receiving the packets from the clone, the client retrieves the chunks and their corresponding hashes directly. We can also observe in redundancy elimination process that the hashes should be sent no matter whether the chunk exists in the cache or not in order to successfully recover the data in the receiver or the client, while the chunks need not be

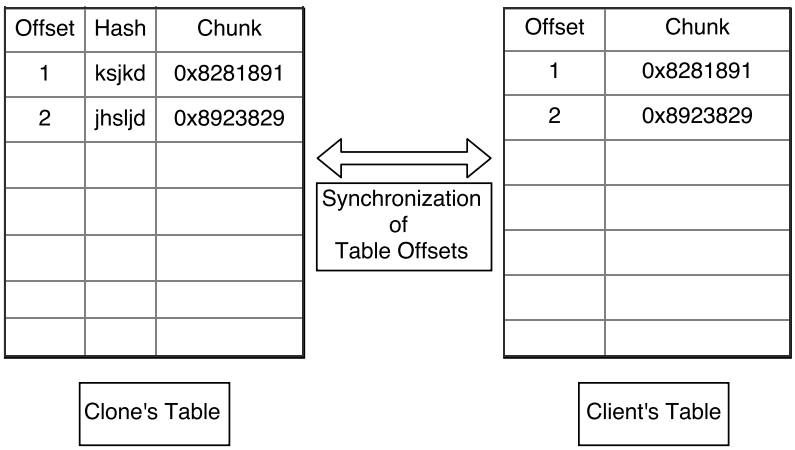


Figure 3.5: Table Structure Utilized for RE at Client and Clone.

sent if it exists in the cache. The one of the trade-offs in the redundancy elimination system primarily exists between the bandwidth overhead due to hash transmission and bandwidth savings obtainable by the redundant chunk elimination, also termed as chunk deduplication. In the next segment, we look at how we eliminate this unnecessary transmission overhead of hash values.

In order to reduce the overheads of hash transmission and improve the bandwidth savings during TailoredRE operation, we adopt a light-weight hash transmission mechanism. The data structures in both the clone's cache and client's cache are maintained in the form of tuple table. In the clone's table, the tuple includes the chunk's offset, chunk's hash and the chunk itself, while in the client's table, the tuple includes just the chunk's offset and the chunk, as shown in Figure 3.5. Instead of transmitting hash values, TailoredRE chooses to transmit the offset of the hash-chunk pair in the tuple table. When the clone sends the offset to the client during the transmission, the client will find the chunk based on the offset and operate the chunk insertion, movement and remove as what the clone has done, so that the clone's cache and the client's cache can be kept synchronized. 4-byte offset can indicate 2^{32} chunks, which is large enough to indicate all the chunks in the cache in our system. But the hash of the chunk, computed with the SHA1 algorithm, is 20 bytes. Thus, we can see that, by transmitting the chunk's offset instead of the chunk's hash, the transmission

overhead is reduced by about 4 times.

3.3.4.1 Chunks Caching (CC)

Both the clone and client have the CC components, which are responsible for the cache management for the clone and client. The clone's cache and client's cache should be fully synchronized so that if the clone find a redundant chunk in its cache, it can be certain that the chunk definitely in the client's cache.

In order to maintain the full synchronization between the clone's cache and the client's, the CC component in the clone and the CC component in the client have to behave synchronously. The cache is maintained as a queue data structure. In our TailoredRE system, both the CC component of the clone and that of the client use Least Recently Used (LRU) [24] caching policy to manage the chunk caching. That is, when a new chunk needs to be cached, if the chunk exists in the cache, CC component will move the chunk from its previous position to the front of the queue. If the chunk does not exist in the cache and the cache is fully occupied, then CC component will remove the chunk at back of the queue, and insert the chunk into the front of the queue at the same time. By operating like this, the chunk in the cache that has not been used for relatively long time will be removed when a new chunk needs the cache space to insert, but the chunk that is frequently used will always stay the relative front position in the cache, which will make the cache resource utilized efficiently and benefit for the redundancy hit during the TRE operation.

3.3.4.2 Data Reconstruction (DR)

DR component, in the TailoredRE client, is responsible for recovering the compressed data into the raw data.

The client keeps receiving the data packets sent by the clone and deliver the packets to the DR component upon receiving the packet. DR component will strip the header of the packet and obtain the application data in the packet. Subsequently, it split the data by the shim between each chunk. If the chunk unit just contains the chunk information

(offset), DR will find the chunk in the cache based on its offset in the cache and replace it with the chunk, and the CC component will update this chunk to a new offset since this chunk is used. If the chunk unit contains both the chunk and its offset, which means the chunk is a new chunk, CC component will cache it conforming LRU.

After the data is reconstructed, it will be forwarded by DF component in the client to the destination application process according to the port number in the packet header.

3.4 Application-adaptive RE

Application-adaptive RE is one of the most important features of TailoredRE system. Different from other TRE methods, TailoredRE clone provides personalized TRE, done by RE component, for its smartphone user adaptive to the profiling of user's applications, done by UBP component. That is, before conducting TRE, the clone will firstly analyze the redundancy ratios of the traffic from the user's preferred mobile applications and the activities the user shows towards to them. Subsequently, the clone will make a decision on which mobile application the clone conducts TRE against based on the profiling of user behavior. The selection method will be described in detail in architecture section. UBP component is crucial to TailoredRE clone, it is also one of the distinguishing factors in our TailoredRE system.

3.4.1 Profiling of user's applications

There are two important metrics based on which UBP component will analyze and profile for the user of a mobile device. The first metric is the redundancy ratio distribution over different mobile applications, and the second one is the data response size (*i.e.*, cumulative sizes of data response payloads over a period of time) distribution over the various applications.

UBP records applications the user k frequently uses, and maintain an application set $U_k = \{A_1, A_2, ..., A_n\}$, where A_i is a two-dimension metric (r_i, d_i) to record redundancy

ratio, r_i, and data transmission volume, d_i for mobile application i. UBP simply adopts the accumulatively history record when that application has been chosen as the target application (*i.e.*, the application against which the TRE would be conducted against), in order to avoid extra computing and memory resource. For example, if the last time, TailoredRE clone conducts TRE against application m, and its cumulative redundancy ratio r_m can be easily collected by calculating the ratio of the total hit cache and the total transmission data during a certain time. Next time, when the UBP find that application m is not being used by the user, TailoredRE switches to the new target application, but the r_m will continue recording its redundancy ratio for profiling.

When application i has been chosen as the target by TailoredRE clone to conduct TRE against, UBP will simply update its redundancy ratio metric, r_i, by computing $\frac{V_{hit}}{V_{total}}$, where V_{total} is the total data transmission volume and V_{hit} is the total bytes of the chunks that hit its cache during a certain time window T. For the application that is not chosen as the target, its r_i will not be updated until it is chosen as the target again. For the system setup period, the target is selected randomly, since there is no history profiling for all the applications.

The data transmission volume for each application can also be easily collected by the clone. Since the data volume during a certain time can reflect the activity of the user, we can use the relative data volume as the activity factor, denoted as a_i, where $a_i = \frac{d_i}{\sum d_i}$, to represent the degree of activity of mobile application i the user shows towards to. For example, if facebook, youtube and Quora data transmission volumes are respectively 1GB, 2GB and 3GB during a certain time, the activity factors of them are respectively 0.17, 0.33 and 0.5.

With these two metrics, for application i, UBP can calculate a TRE value factor, denoted as v_i, computed by $v_i = r_i \cdot a_i$. The value factor reflects the benefits when TailoredRE conducts TRE against the application i. It is reasonable that the application with higher redundancy ratio and higher activity will have higher value factor, which should be chosen as the target application with higher probability.

3.4.2 Adaptiveness in TailoredRE

Other modules in the module through which the data responses traverse, such as Redundancy Elimination, Redundancy Detection component takes inputs from the user behavior profiling (UBP) component, and chooses the application with the highest TRE value factor as the target application to conduct TRE against. Consequently, the cache can only be reserved for the target application *i.e.*, the chunks from the target application will only be cached in the cache and rest of the applications data responses will be forwarded directly the mobile device. If the any other application's value factor increases , TailoredRE switches to perform TRE against this application and flushes the previous cache.

Since the redundancy ratio of the traffic generated by the same application sometimes fluctuates over the time, as shown in Figure 4.2, frequent decision changes will cause the frequent cache flush, which will degrade the TailoredRE system's redundancy hit ratio. To avoid this scenario, TailoredRE system will set the decision update period for a larger-scale duration, for example, one hour. Additionally, if UBP component finds that a target application has not sent data for a long time, it will activate the RE component to update the decision.

With the application-adaptive RE design, RE component in our TailoredRE system, can always conduct TRE against the application with highest TRE value factor maximizing the redundancy hit ratio and total attainable bandwidth savings.

3.5 MAXP-chunk partitioning algorithm

This section elaborates on the core algorithm module in the TailoredRE system - the MAXP chunk Partitioning algorithm. There are numerous content dependent chunking algorithms that have been studied and implemented in the field of redundancy suppression and elimination. They not only pertain to network traffic redundancy elimination system but also can be applied to large file and storage systems. This algorithm determines the chunk boundaries using content instead of offset, so localized changes in the data stream

only affect chunks that are near the changes, which enables efficient and robust duplicate content identification across different data objects. A number of content-based chunking algorithms have been proposed, including Rabin fingerprinting used in [42, 17], MAXP [6], SAMPLEBYTE [2], DYNABYTE [23] and XOR-based rolling hash [50]. TailoredRE uses MAXP [6] to define chunk boundaries because MAXP provides uniformly distributed chunk boundaries across the payload and imposes a lower bound on chunk length and low computational overhead.

MAXP, a content dependent chunking algorithms, has been described elaborately by Bjørner *et al.* in [9]. It is also used by Manber in [28] to find similar files in large file system. MAXP is intended to improve computation efficiency of MODP and WINN. Anand *et al.* in [5] have done a trace-driven performance study of MAXP and MODP algorithms. In this study, they measure the bandwidth savings obtained by both the algorithm across different sampling periods for a given window size of 32 bytes. They have shown that due to the uniform selection of fingerprints in MAXP and clustered selection of fingerprints in MODP, MAXP outperforms MODP by 35%.

The algorithm of MAXP is depicted in Algorithm 1. To ensure that markers/fingerprints are selected uniformly in each block, in MAXP, the local-maxima is computed over the data bytes in the byte stream directly as digits to select the markers. Once the marker bytes are chosen, we compute fingerprints corresponding to that chunk. This results in reduced computational cost rather than computing a series of fingerprints before finding maxima in winnowing. This significantly lowers the computational overhead making it efficient and effective. Thus, we select MAXP as our redundancy detection algorithm.

MAXP selects a position *i.e.*, index in a byte array as chunk boundary if its byte value *i.e.*, byte directly as digit is the maximum (or minimum) over all of the byte values found over the p-byte region centered at that position. The packet payload is divided into chunks by these boundaries, as shown in Figure 3.6. The expected chunk size is p and all chunks must have length at least $\lfloor p/2 \rfloor$ except the last one at the end of payload [31]. We ignore the last chunk if its size is less than $\lfloor p/2 \rfloor$ such as the chunk following Chunk2 in

Algorithm 1 MAXP chunk partitioning algorithm

Input: Stream (bytes)

Output: Hash list

Initialisation :

1: w = 128 *//Minimal chunk size*

2: p = 256 *//Sampling interval*

3: len = length(Byte Stream) *//Stream Length*

//Marker Identification and Hash computation Over the Byte Stream

4: previousMarker = 0

5: nextMarker = 0

6: index = 0

7: **while** (index < len) **do**

8: index = previousMarker

9: nextMarker = maxima(i,i+p,data)

10: chunk = data prevMarker:nextMarker

11: hashList.add(SHA1(chunk))

12: previousMarker = nextMarker

13: **end while**

14: **return** *hashList*

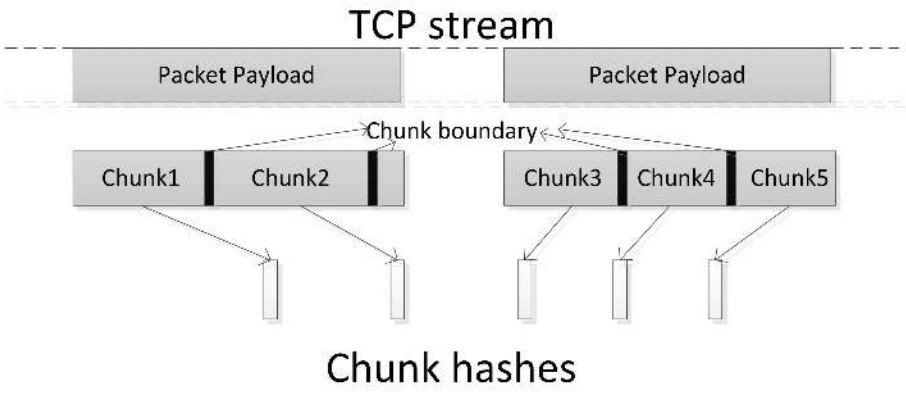


Figure 3.6: MAXP chunk partitioning.

Figure 3.6. p should be significantly larger than the sum size of a chunk hash and shim in TailoredRE because the effective bandwidth saving resulted from a successful chunk prediction is essentially chunk size minus the sum of the sizes of hash and shim. The determination of chunk size should consider the trade-off between prediction overhead and bandwidth savings. A larger chunk size reduces the number of chunks cached in the cache, which will increase the lookup speed while a smaller chunk size can increase the detection efficiency of redundant bytes. In TailoredRE system, p is set to 256 bytes, accordingly, the minimum chunk size is 128 bytes.

3.6 Cache sharing among the clones

Several previous studies [51] [49] [27] [22] have shown that redundancy exhibited in network traffic is of two types intra-user temporal redundancy and inter-user redundancy. Inter-User redundancy occurs when a peer group of users frequently accesses the same web-sites or applications due to their similar interests. In [26], Keralapur *et al.* have conducted a real network traffic trace-driven study of similarity between the users to formulate the user behavior profiling problem as a "co-clustering" problem.

In order to reduce the total cache consumption in the cloud, TailoredRE system is designed to share the cache resource between clones. However, there are two challenges we need to consider: the first challenge is *how to group the clones in order to make the cache*

sharing efficient and the second challenge is *how to share cache in the cloud, while keeping the cache in the cloud and that in the smartphone synchronized*. To solve the first challenge, we propose two-step clone grouping algorithm and designed a two-layer caching strategy to solve the second.

3.6.1 Two-step clone grouping algorithm

Our proposed algorithm includes geographically grouping and user interest-based grouping.

Firstly, TailoredRE roughly groups the clones based on their localities in the cloud, since the cache sharing between the clones with long distance will cause remote communication thus incur large bandwidth overheads in the network. For example, if clone A and clone B locate in the same virtual machine (VM) or physical machine (PM), and they share the memory resource in the same VM or PM, the communication bandwidth overhead between the shared memory and each clone will be small. However, if clone A and clone B are in two different machine clusters, if they share the memory resource, the communication bandwidth overhead will be high since they cost not only the private bandwidth (within PM), but also the public bandwidth (inter PM). Considering this, our TailoredRE system initially groups the clones by the locality, specifically, the clones which locate in the same PM will be grouped into a section.

Subsequently, TailoredRE system will group the clones in the same section into clusters based on their belonged user's application preference. Denote the user i's application set as $U_i = \{app_1, app_2, ..., app_k\}$, where app_k is the application ID, defined by the clone based on the application information in the traffic. U_i can be easily obtained in UBP, since UBP component keeps profiling user's behaviors. We assume that two users with the same application set will have the same interest, and there will be highly similarity in their data transmission. However, sometimes it is not easy to find two users who have complete overlap of their application preference, and the users have the lots of common applications also have the same interest with high probability and their inter-user redundancy would

also be high. For example, if user A and user B will have similar interest, they will watch the same objects online, such as the same figures, the same videos, thus, there will be high inter-user redundancy in their traffic. Considering this, we design the user interest based clone grouping algorithm as follows: Firstly, the grouping controller randomly picks up a user, and compare its application set with other users and count the number of common applications between this user and other users. Secondly, rank the user based on their count, and group the top N users' clones into one cluster. Subsequently, repeat these two steps for the left users in this section until all the clones are clustered. But some special case should be considered. If a certain user's the highest count of common applications with other users is less than 2, which means that this user's interest is not similar with other users, we will not group this user.

By geographically grouping and user-interest grouping, the clones in the cloud are grouped into clusters. The grouping operation generally happens when there is a batch of new users join in the section. There is a threshold for the grouping update, which means that if the increasing number is larger than this threshold, the grouping operation will be executed.

After the clones are grouped, the clones within the same cluster will share the cache, the clones inter-cluster will not. By this way, the cache resource can be used efficiently.

3.6.2 Two-layer caching mechanism

In our TailoredRE system, cache in the clone should be synchronized with that in the client for the purpose of improving the redundancy hit ratio, however, sharing cache with other clones will break this kind of synchronization. For example, assume clone A's cache and client A's cache both have the chunk 1 and clone B's cache and client B's cache both have chunk 2. When clone A and clone B share the cache with each other, clone A's cache will have chunk 1 and chunk 2, but client A just have chunk 1, thus the clone A's cache and client's cache are no more synchronized.

In order to solve this problem, we design a two-layer caching mechanism, in which

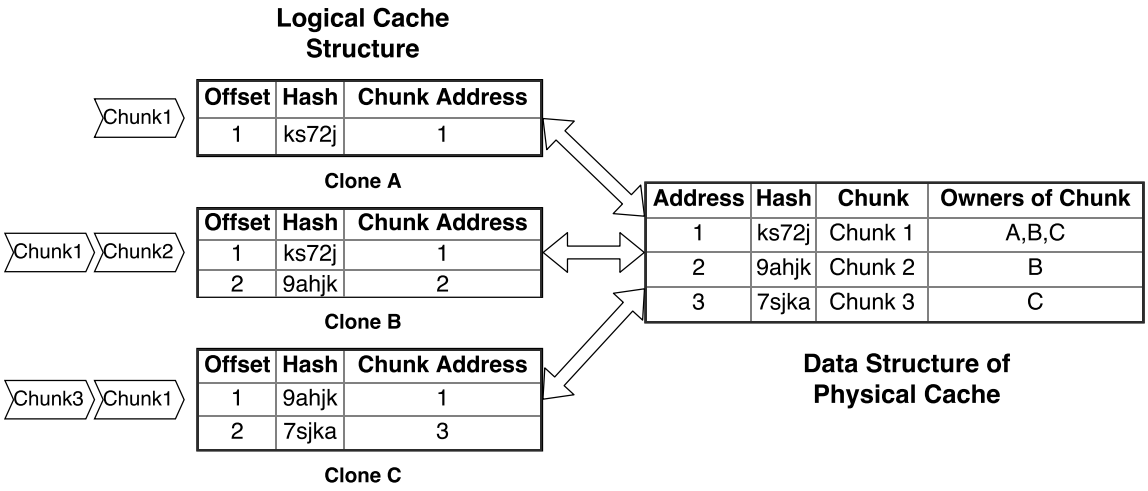


Figure 3.7: Operation of Clustered Clones

there are two kinds of caches as shown in 3.7: logical cache for individual clone and physical cache for all the clones in one cluster. The chunk information for each clone, including chunk hashes and chunk address in the physical cache, will be stored in the individual logical cache and the real chunk data will be stored in the shared physical cache. The logical cache is used to keep synchronized with the cache in the client while the physical cache will store the chunks to keep the correctness during the TRE. The logical cache is deployed in the CC component mentioned above and we just need to replace the regular cache with the logical cache. For other components, the logical cache has no difference with the regular cache. But there will be communication between the logical cache and the physical cache.

In the data structure of physical cache, for each chunk, there is a marker list or owner list of chunks denoting which clones the chunk belongs. For example, for chunk1, its marker list stores the clone IDs of clone A, clone B, and clone C, which means that this chunk belongs to clone A, clone B and clone C. Each operation in the logical cache, such as chunk insertion and chunk remove, will incur the chunk update in the physical cache. The operations in each clone's logical cache and simultaneous operation in the cluster's physical cache can be shown as below:

- **Chunk movement:** After chunk partitioning and chunk hashing, clone's RD component will check whether the chunk exists in the cache. CC component will firstly

check chunk hash in the clone's logical cache, and the logical cache will check whether the chunk really the same with this chunk (to avoid the hash collision) in the cluster's physical cache. After a clone's RD component finds that the chunk's hash exists in its logical cache and the chunk exit in the cluster's physical cache, it will move that chunk to the front of the clone's logical cache. In this case, the cluster's physical cache needs to do nothing about the chunk update.

- **Chunk insertion:** After a clone's RD component finds that a chunk is a new chunk, and if the clone's logical cache is not full, this chunk will be inserted into the cache. Firstly, the hash will be inserted into the clone's logical cache, and the logical cache will check whether the chunk is already in the physical cache inserted by other clones. If the chunk exists already, the physical cache will put this clone's ID into the chunk's marker list or owner list, otherwise, the physical cache will insert this chunk into its cache table.

- **Chunk deletion:** After a clone's RD component finds that a chunk is a new chunk, and if the clone's logical cache is full, then the last one in the clone's logical cache will be removed. Firstly, the logical cache will find the chunk in the cluster's physical cache according to its address and check its marker list. If there are the markers from other clones in the list, the physical cache will just remove the clone ID of the clone logical cache belongs to in the marker list, otherwise, the physical cache will remove the chunk.

The logical cache provides the CC component with the interface, such as lookup (Chunk), insert (Chunk), delete(Chunk), the same with cache we mentioned above. Thus, for the CC component and other components in the clone, they are the same. But inside these two kinds of caches, they are different. In the logical cache, it just stores the chunk information, such as the chunk hash and its address in the physical cache. However, the in the regular cache, it will store both the hashes and chunk.

3.7 Summary

Overall, this chapter provided a detailed description of the important components of TailoredRE and how all the components together constitute to provide efficient personalized cloud based redundancy elimination service. We have also looked into some of the distinguishing factors of TailoredRE that makes it advantageous than the existing TRE systems. The next chapter starts by discussing the real world trace analysis of the network traffic captured from various widely used smartphone applications. Then, we define the metrics to evaluate the TailoredRE system and analyze the results we obtained through the simulation of TailoredRE system. After that we delineate the details constituting the real implementation of TailoredRE system followed by analysis of the results obtained from real experimentation.

Chapter 4

Experiments

This chapter starts by presenting the trace analysis and simulation results of TailoredRE performed on the real data traffic traces collected over a period of approximately two months beginning January, 2016. We look at some of the implementation details of the Simulation along with metrics that we utilize to measure the performance. Then we dive into trace collection and analysis methods along with results we have observed from analyzing the traces. Then we describe our simulation methodology and analysis of results obtained through simulation. We then pivot to discuss the implementation details of real world prototype of TailoredRE system. We describe the details involved in all the necessary modules in our prototype to provide readers a perspective of how the TailoredRE system operates in real time. Finally we analyze the results obtained through the experimentation of our real world prototype.

4.1 Simulation and Metrics

To explore the effectiveness and efficiency of our TailoredRE system, we developed a simulation program consisting of a pair of clone (sender) and client (receiver) developed in Java. Also, a simulation platform, including three java classes: Cloud.java, Cluster.java and Clone.java, were developed to simulate the performance of the our proposed caching

sharing mechanism in the cloud. All of our simulation experiments are real trace-driven, and traces we use are shown in Table 4.1 and 4.2.

We measured the following metrics in our experiment:

- **Redundancy hit ratio:** It denotes the ratio of the total bytes of chunks that hit the cache to the total data volume and computed by $\frac{V_{hit}}{V_{total}}$, where V_{total} is the total data volume and V_{hit} is the total bytes of the chunks that hit the cache.
- **Bandwidth saving ratio:** It is calculated by $\frac{V_{hit}-V_{overhead}}{V_{total}}$, where $V_{overhead}$ is the data volume of chunk information in content transmission and V_{total} is the size of the total transmitted content data.
- **Normalized throughput:** It is computed by $\frac{r_{RE}}{r_{No-RE}}$ in order to show the final throughput improvement caused by both RE and network overhead reduction.
- **Average Cache Saving:** It denotes the reduction of the cache resource consumption caused by cache sharing between the clones in the same cluster.

4.2 Trace Analysis

This section is devoted to discussing the results we obtained from the large scale data trace analysis.

4.2.1 Trace Collection

We conduct our statistical data analysis on payloads of packets from real wireless traces. To perform our data collection we set-up a Wi-Fi hotspot or AP(Access Point) in a Intel Core i7 laptop PC powered by Windows 10 operating system using mHotSpot [29], a HotSpot management application. The laptop PC has 802.11ac Wi-Fi card for Wireless Internet connection using which we created Wi-Fi hotspot. We then let the mobile device connect to this hotspot, and then use the device to browse web contents provided by various applications ranging from multimedia to news applications. We then capture all the traffic associated with the HotSpot connection through Wireshark [44], a packet sniffing tool,

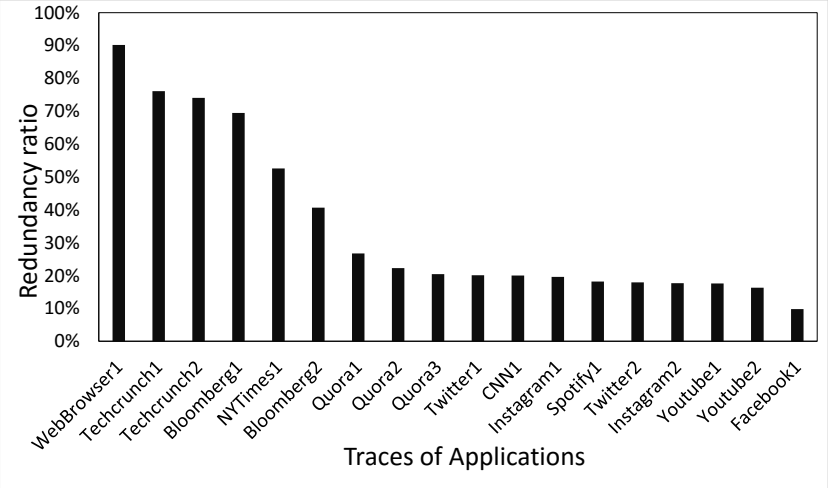


Figure 4.1: Trace Analysis

running on the laptop PC. We have collected network traffic generated the single user for a period of two months. Then, we segregate packet traces of various applications based on the their IP addresses using Wireshark's filtering tool to ensure that traces contain traffic from a single application.

Then, we perform preprocessing of the traffic traces by using python script to strip the packet headers including physical, MAC, IP and transport layer headers of each packet. We perform this preprocessing step to isolate the application data payloads from the packet headers since our TailoredRE is performing on application data above the TCP or UDP layer. We collected about 68GB data traces in total, including 5 large-scale data traces: Web Browser (10GB), Quora (8.45GB), Facebook (7.10GB), Instagram(10.4GB) and Spotify (12GB), listed in Table 4.1, and other 22 short data traces shown in Table 4.2.

4.2.2 Trace Analysis

We have run our simulator to analyze the redundancy ratios of all the application data traces. The chunk size is set to [128, 394] bytes. We set the cache size as large as 10MB, which is large enough to detect almost all the redundant chunks in the data traces. After the chunk partitioning, hash computing and caching operations, we detect the total data size of the chunks that hit the cache and calculate the total hit ratio as the trace's

Traces	Data size (GB)	Redundancy ratio (GB)	Activity
Web Browser	10.1	81.3%	0.4
Quora	8.7	20.6%	0.2
Youtube	7.3	16.7%	0.2
Instagram	10.4	18.8%	0.1
Spotify	12	11.4%	0.1

Table 4.1: Traces in the simulation experiment

redundancy ratio. The redundancy ratios for different data traces are shown in Figure 4.1.

From Figure 4.1, we can see that different applications indeed have different redundancy ratios. Some application data, such as Web Browser trace and Quora trace, have higher redundancy ratios, while others, like facebook and Bloomberg, have lower redundancy ratios. However, since the smart phone always have limited memory resource, the applications with lower redundancy ratio will occupy the cache resource but contribute less hit ratio and less bandwidth saving. In next section, we will show how our proposed TailoredRE system improves the redundancy hit ratio and improve the RE efficiency.

4.3 Analysis of TailoredRE efficiency

In this section, we describe our simulation methodology and then compare our simulation to the performance of some of the existing TRE techniques in terms of the metrics we discussed in the previous sections.

At the sender side, we developed a TailoredRE based Clone which consists of three main components: Chunk Partitioning module, Chunk hash computing module and Chunk caching module. The receiver in our simulation conducts the functions chunk caching and data recovery. The clone and receiver pair consists our simulator.

We evaluate the effectiveness and efficiency of TailoredRE with the changes of time and cache size respectively, and compared TailoredRE with the following RE methods:

- EndRE [2]. The chunk caches at the sender and receiver are tightly synchronized, the same as TailoredRE, so a chunk presented in the sender's cache is guaranteed to be in

the receiver's cache. However, it conducts TRE against the data traces from all the applications.

- Asymmetric Caching (AC) [38]. If the receiver receives a chunk that hits the cache, it will send back the hashes of the chain after that chunk as the feedback to the sender. The sender thus performs RE operations based on its feedback cache storing the hashes and its regular cache that stores chunks.

TailoredRE uses MAXP to find the chunk boundaries and the traffic data are divided into chunks by these boundaries. Least Recently Used(LRU) caching policy is used in our experiment in order to improve the caching efficiency. We limit the chunk sizes to [128,384] bytes in order to balance the redundant chunk recognition accuracy and the transmission overheads. The traces we use to simulate our TailoredRE method are shown in Table 4.1.

We define the activity factor of an application as the probability that data from that application will be sent at a certain time slot. By this way, we can simulate the activity degrees of applications for a certain user. In this simulation, the application activity factors are also shown in Table 4.1.

4.3.1 Performance metrics along time

In this simulation, the cache size is set to 2.5MB. We measured the redundancy hit ratio, bandwidth saving ratio and normalized throughput with time changing. The results are respectively shown in Figure 4.2, Figure 4.3 and Figure 4.4.

From Figure 4.2 we can see that in every time slot, the redundancy hit ratio conforms TailoredRE>EndRE>AC. The reason is that compared with the EndRE, our proposed method considers the profiling of user behavior and redundancy and always choose the applications with high redundancy ratio and activity to conduct redundancy elimination against. Thus, the cache in TailoredRE can be used much more efficiently and TailoredRE has higher cache hit ratio than EndRE. Compared with EndRE, in AC, the chunks are stored in the form of flowlet (chunk chain), and redundant chunks exist between flowlets, which degrades the cache's utilization efficiency. In addition, because of the asynchronization

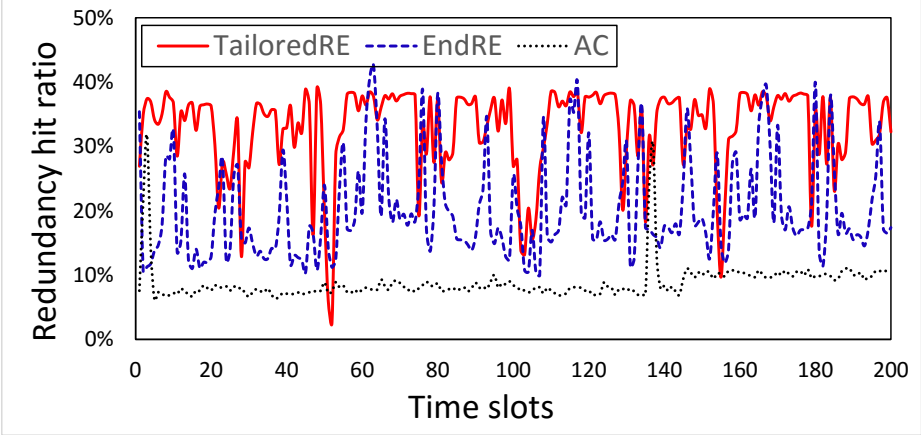


Figure 4.2: Redundancy Hit ratio over time

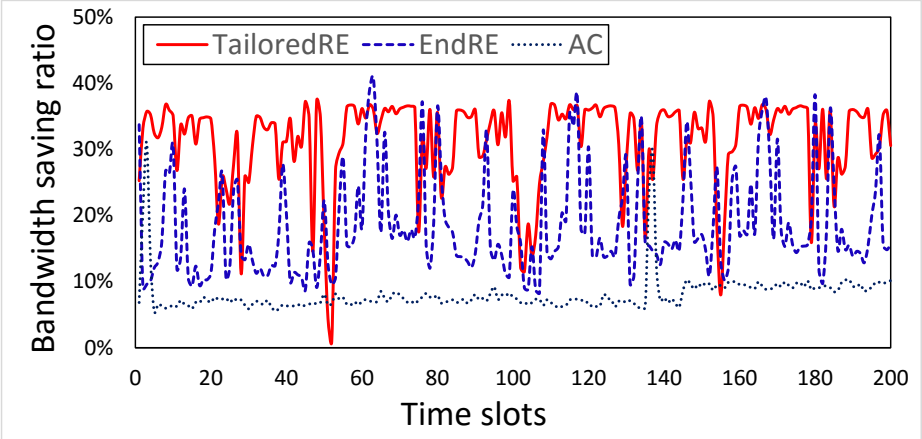


Figure 4.3: Bandwidth saving ratio over time

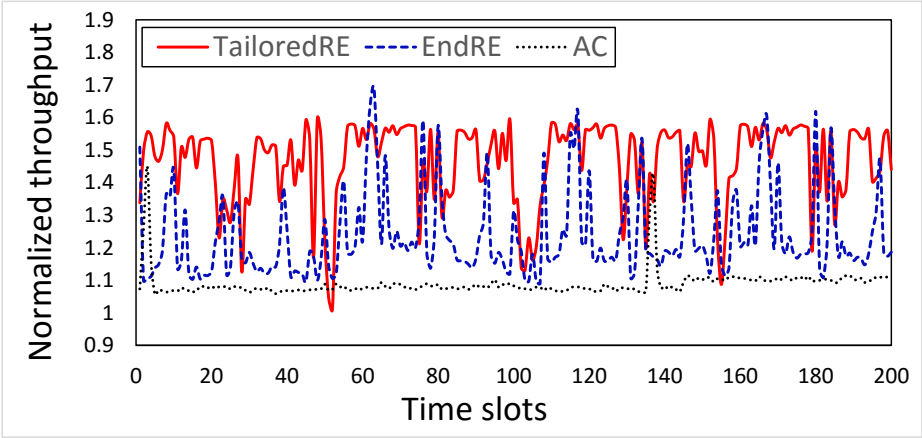


Figure 4.4: Normalized throughput over time

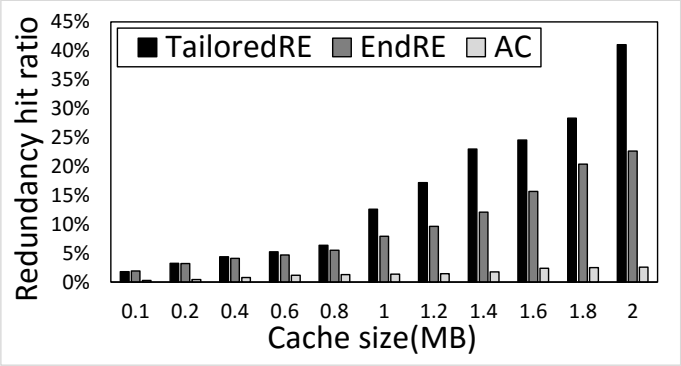


Figure 4.5: Redundancy Hit ratio over Cache Size

between the sender and receiver, the chunk information stored in both the feedback cache and regular cache can only indicate parts of the chunks in receiver's cache. For these two reasons, the redundancy hit ratio achieve lower than EndRE.

Figure 4.3 shows that at each time slot, the redundancy hit ratio also conforms TailoredRE>EndRE>AC. It is because that compared with TailoredRE and EndRE which use offset in the cache to replace chunk with, AC uses hashes to do that, which have larger size than the offset and consume much more bandwidth. Additionally, as shown in Figure 4.2, the redundancy hit ratio conforms TailoredRE>EndRE>AC, which means the bandwidth reduction. Thus, TailoredRE can conduct highest bandwidth saving, and AC the lowest.

As shown in Figure 4.4 that, at each time slot, the normalized throughput also conforms TailoredRE>EndRE>AC. Since TailoredRE can eliminate the most redundant chunks during the data transmission, and reduce the highest transmission time and improve the throughput to the most.

4.3.2 Performance metrics along cache size

We measured the redundancy hit ratio, bandwidth saving ratio and normalized throughput with cache size changing from 0.1 MB to 2MB. The results are respectively shown in Figure 4.5, Figure 4.6 and Figure 4.7.

Figure 4.5 shows that the redundancy hit ratio conforms TailoredRE>EndRE>AC

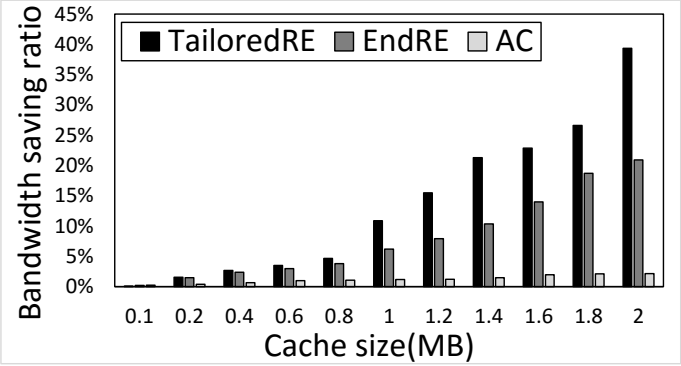


Figure 4.6: Bandwidth saving ratio over Cache Size

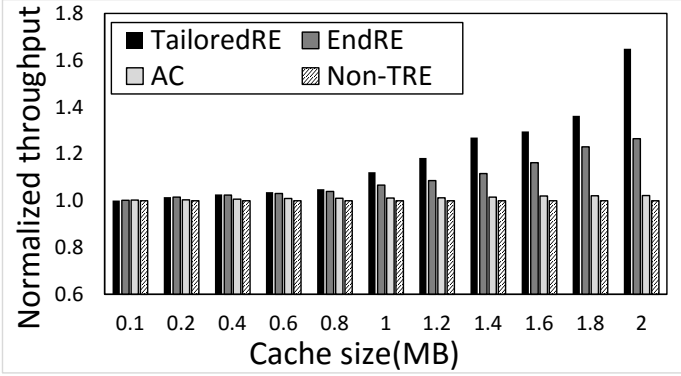


Figure 4.7: Normalized throughput over Cache Size

due to the same reason with 4.2. In addition, with the increase of cache size, the redundancy hit ratios in both TailoredRE, EndRE and AC increase. The reason is that with larger cache size, more chunks can be cached and the chunks in transmission have much higher probabilities to hit the cache.

From Figure 4.6, we can see that bandwidth saving ratio also conforms TailoredRE>EndRE>AC due to the same reason with 4.3. The bandwidth saving ratio also increases with the cache size, since the redundancy hit ratio increase with the cache size.

As is shown in Figure 4.7, that normalized hit ratio conforms TailoredRE>EndRE>AC for the same reason shown in Figure 4.4. The normalized throughput also increases with the cache size as more cache can store more chunks.

4.3.3 Performance of shared caching for multiple users

In this section, we developed our TailoredTRE simulation platform which mainly consists of three Java classes: Cloud.java, Cluster.java and Clone.java. In our simulation, each user has a personalized Clone instance, which has the functions including chunk partitioning, hash computing and caching method and forwarding. Cloud instance is responsible for the application traces initialization for each user and grouping their clones based on the our proposed user interest-based grouping algorithm. The grouped clones will consist a cluster and share a common physical cache. The trace pool shown in table 4.2 contains all the traces we use to simulate our cache-shared method in the cloud.

Firstly, we simulate the cache savings over time changing with different cluster size increasing from 2 to 5 in a cluster. The similarity between users in this cluster is described by a similarity factor, which is defined as the probability that the next user has the traces from the same application with the previous user. In this simulation, the similarity factor is set to 0.4. The simulation results are shown in Figure 4.8. From Figure 4.8, with the increasing of the cluster size, the cache savings will increase. That is reasonable, since more clones sharing the cache will incur much more common chunks in the cache, which would cause much more cache savings.

Traces	Traces	Data size (GB)	Redundancy ratio
1	WebBrowser1	0.85	90.18%
2	Techcrunch1	1.11	76.10%
3	Techcrunch2	1.09	74.09%
4	Bloomberg1	1.28	69.47%
5	NYTimes1	1.40	52.60%
6	Bloomberg2	0.90	40.67%
7	Quora1	1.36	26.73%
8	Quora2	0.81	22.26%
9	Quora3	1.12	20.41%
10	Twitter1	1.03	20.10%
11	CNN1	0.98	20.01%
12	Instagram1	1.40	19.57%
13	Spotify1	1.01	18.17%
14	Twitter2	1.13	17.91%
15	Instagram2	0.84	17.68%
16	Youtube1	1.58	17.58%
17	Youtube2	1.02	16.31%
18	Facebook1	0.94	9.80%

Table 4.2: Traces Pool

We assume that all the clones are already geographically grouped into a section and just consider the user interest-based clone grouping. Given the user number, we initialized each user's applications by randomly choosing 4 application traces from the trace pool. The Could instance then groups the clones and build up the Cluster instance based on the grouping algorithm we mentioned in the last section. All the clones within the same cluster will share a common physical cache, but the clones in two different clusters will not. The cluster size is set to [1,8]. The chunk size is set to [256, 768] bytes, and the cache size at the receivers are uniformly set to the 2MB.

We simulate the relationship between the average total cache resource consumption at the cloud side with user number changing from 45 to 1000. We also compare our method with both the randomly grouping method and the one without grouping. In order to compare them clearly, we quantify the average cache consumption with the uniformly configured cache size for each clone. The simulation results are shown in Figure 4.9.

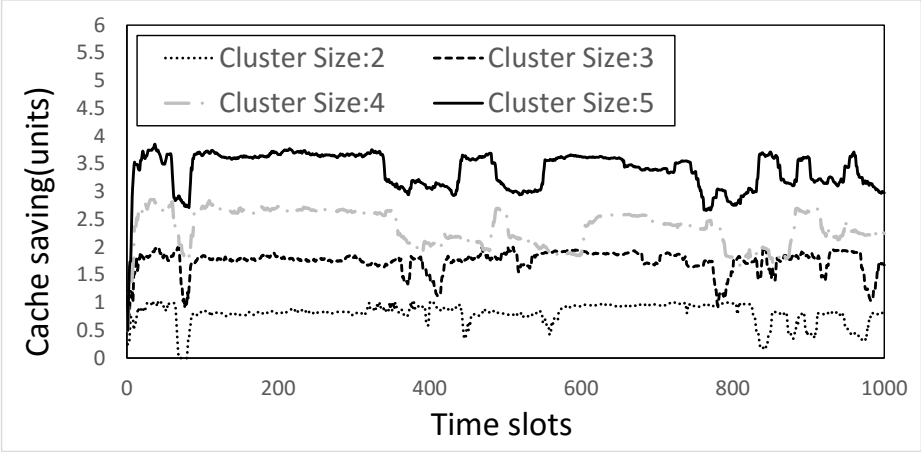


Figure 4.8: Cache saving over time

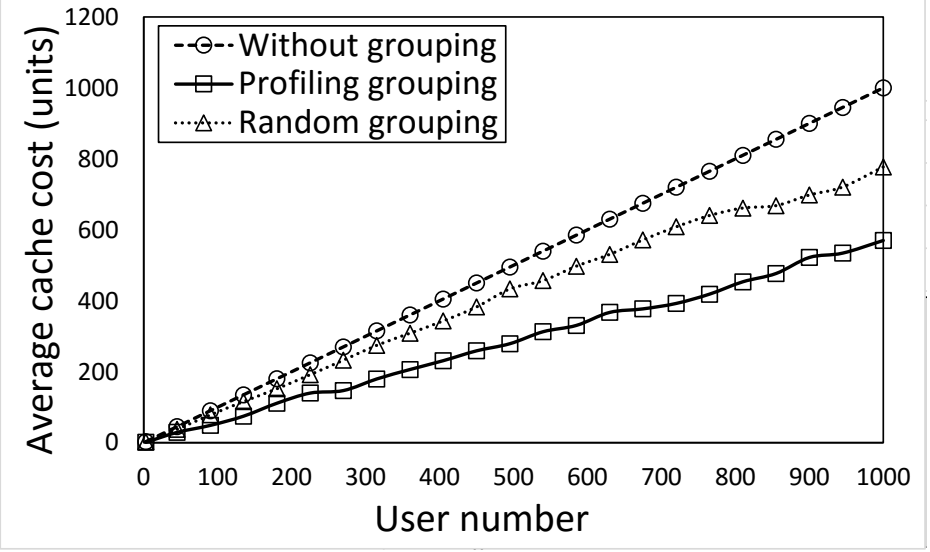


Figure 4.9: Cache consumption over Number of users

Figure 4.9 shows that for a certain user number, the average cache consumption in cloud conforms interest-based grouping¡random grouping¡without grouping. Both interest-based grouping and random grouping method consume less cache resource than method without grouping, since without grouping, the clones will not share the cache resource and it will consume much more cache resource. Our proposed interest-based grouping method can reduce cache consumption by over than 40%, and random grouping can conserve about 20% of the cache consumption. Thus, our proposed TailoredRE method can conserve 100% more than randomly grouping method. The reason is that our proposed grouping method considers the user application usage similarity and the cached chunks between two clones in the same cluster will have much more common chunks and share much more cache resource in the shared cache. In addition, we can see from Figure 4.9 that with the increasing of user number, the cache resource consumption increase linearly, but the cache saving caused by grouping also increases.

4.4 Implementation of TailoredRE prototype

In this section, we outline the implementation details of the TailoredRE system including the platforms and the programming languages we utilized to successfully implement the TailoredRE system both on the client side and the server side. Then we go on to analyse the metrics we measured to show the highlights of TailoredRE system.

4.4.1 Implementation Details

As we have seen in Chapter 3, TailoredRE System can be divided into two subsystems Tailored Clone and TailoredRE client. TailoredRE clone basically constitutes a proxy server that is capable of redundancy deduction and elimination while performing data forwarding. TailoredRE client application includes Data Recovery and Chunks Caching operations.

4.4.1.1 TailoredRE Clone

As we discussed in the last chapter, the proxy or clone of the smartphone constitutes a crucial part of the TailoredRE system. It forwards the data requests arising from smartphone applications to their respective web servers. Upon receiving the data response from web servers, they perform necessary Redundancy detection, elimination, user activity information gathering and profiling routines on the response and forwards the resulting response back to the respective application in the smartphone. The clone or proxy of the smartphone in our prototype is implemented in Python 2.7. In doing so we utilized the extensible library packages of Python like os, sys, thread, socket, hashlib, urllib2, threading to support our implementation.

The clone listens for the socket connections from the smartphones. Upon successfully connecting to the client (the smartphone), it spawns a thread to handle all the data request sent by the client. Whenever the thread spawned from the proxy receives an URL from a smartphone application we open an http URL connection to the web server. Upon successfully connecting to server, we read the data and store it in a buffer. We perform MAXP partitioning and hashing of the data in the buffer to store the hashes and chunks in data structures. Before performing this step, we first determine whether the RE should be performed by referring to a global variable in our Python program. This variable is assigned by a separate monitoring thread which is responsible for monitoring the data requests and responses going through the clone and selecting an application against which the redundancy elimination will be performed. We call this thread *Activity Monitor*. *Activity Monitor* makes use of the Activity Table which is populated during every request and response activity. This activity table is a hash table containing *¡key,value¿* pairs as *¡AppName,Activity¿*. Here *Activity* is a user defined data structure that holds necessary variables such as cumulative response size, average redundancy hit ratio, activity factor and value factor for an *AppName*. The detailed explanation of activity and value factor is described in Section 3.4 in Chapter 3.

Once the data response is parsed through MAXP partitioning and hash computation

modules, it is presented to the encoding module to encode the response according to the marker outputs provided by MAXP algorithm. There are two types of markers called *Hit Markers* and *Cached Markers*, which are outputted as list of tuples from the MAXP module in the program. The list of *Hit Markers* denote the indices in data response buffer at which the chunk marked by the indices are encoded with only *Hash* or *offset* as the chunk is found to be previously cache. The list of *Cached Markers* denote the indices in data response buffer at which the chunk marked by the indices are encoded with *Hash,Chunk* since the chunk is found to be new and cached during the current response. The encoding module uses special symbols called shims to distinguish the hashes from the chunks while encoding the response. The *Hash* or *offset* along with shims are considered as overheads in the encoding process. After the encoding the response, we send the data response back to the smartphone application using the same socket connection. The socket remains open for future requests.

4.4.1.2 TailoredRE Client

Now we look at the Client application implemented in the smartphone. We make use of Android platform to realize the client side subsystem of TailoredRE. According to the system architecture we have observed in Chapter 3, in the client side subsystem, the applications the users use are isolated from the RE application which is responsible for caching newly arrived chunks into the cache and recovering previously cached chunks from the cache. We call this RE application TailoredRE Client. We realize its implementation with the help of Intent service provided by Android framework for passing the data between two applications. It is also to be noted that since all the news and multimedia applications that constitutes and motivates the need for the Redundancy Elimination are not open source, we develop special Android applications that mimics the original applications. This accommodation can be justified by fact that all these news and multimedia applications typically send data request to their respective web servers to get data and then they present it attractive way by parsing the web contents and displaying it in different UI components

provided in the Android framework.

As most of the compute intensive TRE operations in our system has been offloaded from the mobile device to its proxy or clone in the cloud, the modules present in the client are simplistic and does straight-forward operations. The data requests in each individual applications are sent through a common TailoredRE client application. The overall data flow in this process can be explained as follows. When a user using an application clicks a button requesting web contents, the application sends the string of the respective URL to the TailoredRE client application installed on the smartphone. TailoredRE client opens a socket connection to the clone and sends the URL data request. The clone upon receiving the data request forwards the data request to the respective web server.

TailoredRE Client upon receiving the data request from the clone performs the chunks caching and data recovery. Chunks Caching module in the client involves parsing the data response and identifying the chunks that are to be cached in the memory. If the response is encoded with *Hash,Chunk* pair with the appropriate shim, it stores the necessary chunks and hashes in to the data structure. Data recovery module involves parsing the data response and identifying the hashes encoded with the shim and recovering the chunks that correspond to identified hashes from the memory. After the necessary operation of Chunks Caching and Data Recovery, we now pass on the data response back to the original application. This lets the original application display or store web contents obtained.

4.5 Performance Evaluation of TailoredRE prototype

While introducing the motivation behind TailoredRE, we mentioned two important factors that encourage the need for any RE system for smartphones and mobile devices. The first one being the bandwidth savings that could be obtained by performing redundancy elimination. The second one being power or energy savings obtained through less usage of Wi-Fi and cellular network modems (3G/4G/LTE). Therefore we evaluate our real prototype of TailoredRE using the necessary metrics to accommodate the above factors.

In order to successfully test our real system, we used Amazon's Elastic Compute Cloud's (EC2) virtual server instance to host and run our clone program. The server instance we obtained is powered by Ubuntu 14.04 LTS operating system. The client Android application are run on mobile device named Asus Fonepad powered by Android Version 4.1.2. We used 802.11 Wi-Fi modem in our mobile device to establish communication between the between itself and its clone. The window size w and sampling period p are the two key parameters of MAXP Chunk Partitioning algorithm. In our experiments we set w and p to be 128 and 256 respectively. This setting results in chunks ranging from 128 to 384 bytes which we find it to be optimal and efficient through earlier trace analysis and simulations that we have conducted. The cache size in our experiments are set to 2MB. During our experiment, we measure bandwidth and power consumption over time slots. The experiments are conducted over a period of 2 hours. Two hours are divided into 1 minutes time slots during which we send 50 data requests and receive 50 corresponding data responses.

We conducted a survey about user's application preferences (`http://goo.gl/forms/LZ4YM5ZWzZ`). The survey is performed among 12 students in our research lab. In the survey, we let the students to select the applications they preferred and the time they spend regularly on them in one week. In the survey, we let 12 users to select the applications they preferred and the time they spend regularly on them in one week. They can also enter the applications which are not enlisted in the 'Others' field. We used the survey data obtained to conduct the our evaluation. In order to account for the user preferences that we obtained from the student survey, while performing the experiments, we made sure that the ratio of number of requests sent from each application to the total number of requests equals the ratio of amount of time the user spends on the application to the total amount of time the user spends in all the applications together. For example, say a user has recorded in his/her survey form that he/she spends a total of 10 hours on his smartphone using Internet based applications. This includes 2 hours on YouTube, 2 hours on CNN, 3 hours on Bloomberg and 3 hours on Twitter. The normalized distribution of time the user spend on his pre-

ferred array of applications [YouTube, CNN, Bloomberg, Twitter] can be given as [0.2, 0.2, 0.3, 0.3]. The distribution of the 50 requests sent during every time slot in our experiment also conforms to this normalized distribution of [0.2, 0.2, 0.3, 0.3] (*i.e.*, 10 data requests to YouTube, 10 data requests to CNN, 15 data requests to Bloomberg and 15 data requests to Twitter).

As a means to highlight the advantages behind TailoredRE, we plot Bandwidth savings and Power Consumption obtained in TailoredRE against EndRE and AC. In addition to this, as a baseline measurement we also compare this with Bandwidth savings and Power Consumption when no TRE operations are performed. The following subsections describes our evaluation in detailed manner. As part of our evaluation, we have included the figures 4.10-4.21((a),(b)) which represents Bandwidth savings and Power Consumption plotted over time slots corresponding to each of 12 users who have taken part in the survey. Figure 4.22 and 4.23 represents the average bandwidth savings and power consumptions for each user obtained by employing TailoredRE, EndRE, AC and WithoutRE. Figure 4.24 indicates the energy savings in percentage achieved by TailoredRE, EndRE and AC with respect energy consumed when no TRE operations are performed

4.5.1 Evaluation of Bandwidth savings

In order to be able to perform a good evaluation of Bandwidth Savings obtained by TailoredRE, we conducted our experiments over a period of 2 hours. The experiments are conducted individually for each user in accordance to the data we obtained through survey. To automate the evaluation process the clone automatically calculates and logs all the necessary information for every the data requests and responses it forwards. Bandwidth Savings is denoted in percent and is calculated by $\frac{V_{OriginalResponse}-V_{REencodedResponse}}{V_{OriginalResponse}}$. In this formula, $V_{OriginalResponse}$ is the cumulative sum of data volume of original Response over a period of time slot *i.e.*, the sum of the sizes of the data responses as it is without any TRE operations performed. In TailoredRE and EndRE, $V_{REencodedResponse}$ is the cumulative sum of data volume of encoded Response sent from the clone to its associated

smartphone after performing Traffic Redundancy Elimination over a period of time slot. Here, $V_{REencodedResponse}$ primarily includes sizes of the chunks, offsets and shim if the data response found to comprise of chunks that have not been cached prior (*i.e.*, newly cached chunks) or only the sizes of the offsets and shim for the chunks that have been found in the cache. In the case of Asymmetric Caching (AC), as it also involves feedbacks arising from the smartphone's cache, $V_{REencodedResponse}$ would be the cumulative sum of data volume of encoded Response sent from the clone to its associated smartphone as well as the sizes of the chunks and hashes received by its clone sent from the smartphone. In our experiment, we have set 1 minutes time slot and total of 120 time slots which equates to 2 hours. Therefore data points in the Bandwidth Savings Curve represent aggregate of ratio of total bytes saved from transmitting due to RE to the total bytes in original response in that time slot *i.e.*, the calculations are done for time slot x based on the accumulated data from minute(x-1) to minute x. Cache size of $2MB$ is set in the client and the server while conducting our experiments in TailoredRE and EndRE. For our implementation of Asymmetric Caching, the server's cache size is set at $0.5MB$, a fraction (one fourth) of the client cache size of $2MB$.

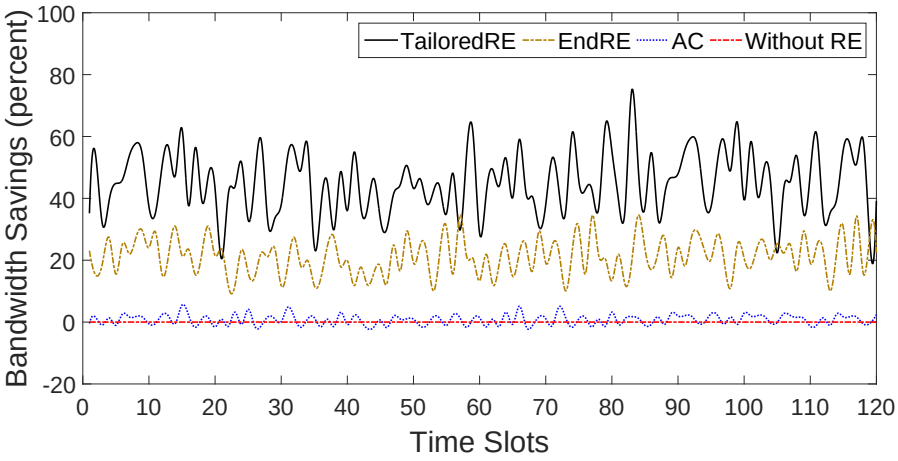
The bandwidth savings plots comprises of Figures 4.10-4.21(a) representing the bandwidth savings obtained for each individual user over time slots in our experiment and Figure 4.22 representing the bar diagram of average bandwidth savings over all the time slots obtained by the users. In Figures 4.10-4.21(a), we can see that TailoredRE provides significantly higher bandwidth savings than EndRE and AC. For a frame of reference, we have plotted bandwidth savings achieved when no TRE is performed as a baseline. It can be seen from the figures that the bandwidth savings obtained by AC is very closer to the baseline. EndRE provides a significantly higher bandwidth savings than AC and it still lags far behind bandwidth savings achieved when TailoredRE is performed. The lower performance of EndRE and AC can be attributed to the overhead that had been incurred due transmission of *Hash or Offset* values over the network. As EndRE and AC performs RE against all data responses in agnostic manner, it results in encoding and caching of chunks

which will not constitute any future transmissions. AC performs worse than EndRE because of the feedback of cache chunks involved while doing redundancy elimination. Unlike EndRE and AC, TailoredRE takes into account heavily used application for conducting personalized TRE resulting in encoding and caching of only chunks that will be utilized in future transmissions. This eventually results in higher bandwidth savings. In Figure 4.22, we can see that the overall bandwidth savings obtained by TailoredRE for every user is almost twice high when compared to EndRE and almost 4 times higher when compared to AC. Thus our Bandwidth Savings evaluation proves the potential benefit of our method.

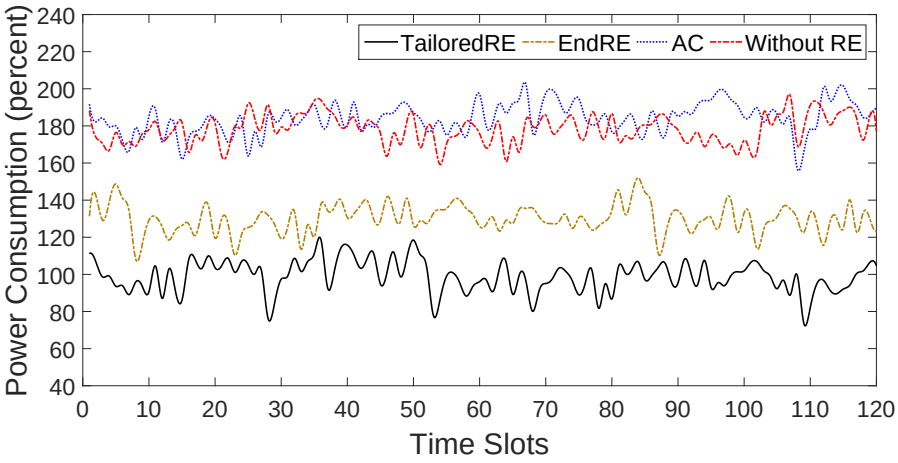
4.5.2 Evaluation of Power Consumption

Minimization of Power Consumption is one of the important motivating factors behind TailoredRE system. Mobile Devices are typically power constrained along with limited battery lifecycle. The devices tend to incur significant amount of power while using radio interfaces such as cellular and Wi-Fi hardware modules. In our experiments, in order to estimate the power consumption too we use PowerTutor, a component power management and activity state introspection based tool. This tool, backed by reasearch work in [47] has been widely used by research community and application developers to estimate the power consumed by their applications. PowerTutor lets us monitor the power consumed by various applications in different hardware components of a smartphone such as Wi-Fi, 3G modules, LCD displays and CPU. PowerTutor utilizes different power models given in [47] and deduces the power consumed by the various components individually for every running application.

In order to evaluate power consumption, we performed experiments over a period of 2 hours. In the same manner as Bandwidth Savings measurement, we have set time slots to be 1 minutes each and a total of 120 time slots equating to 2 hours. During this time period, we used the array of applications in the user survey to fetch contents from their respective web server. Cache size of $2MB$ is set while conducting our experiments. The procedure is repeated when there is no TRE enabled, when EndRE is enabled, when

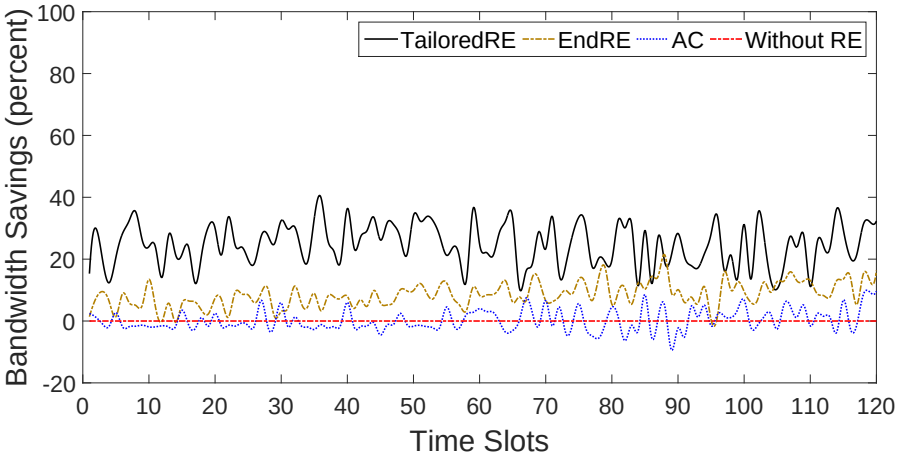


(a) Bandwidth Savings along time

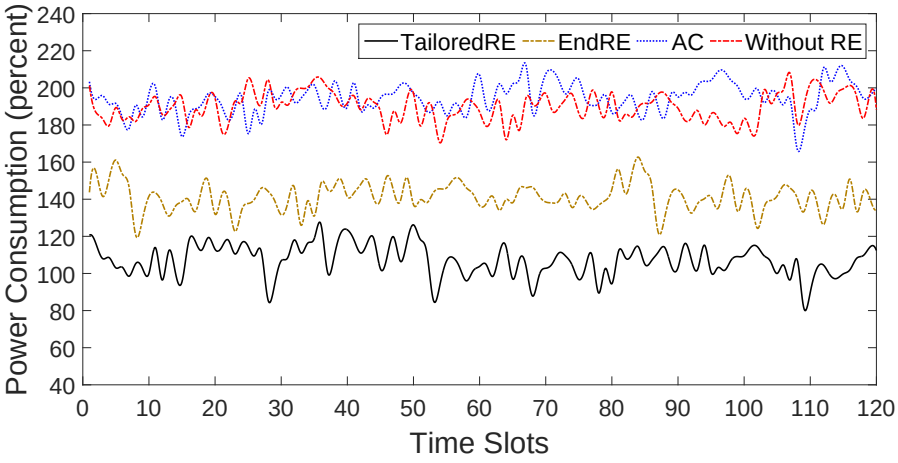


(b) Power Consumption along time

Figure 4.10: Performance Metrics for User 1

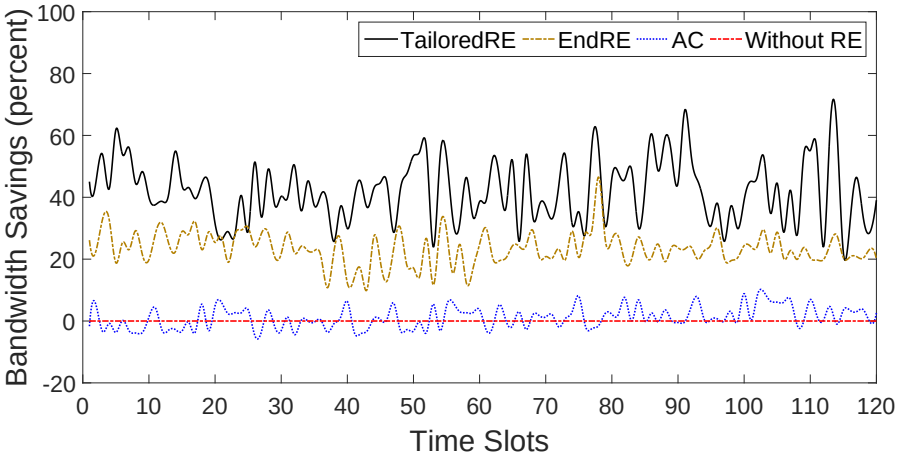


(a) Bandwidth Savings along time

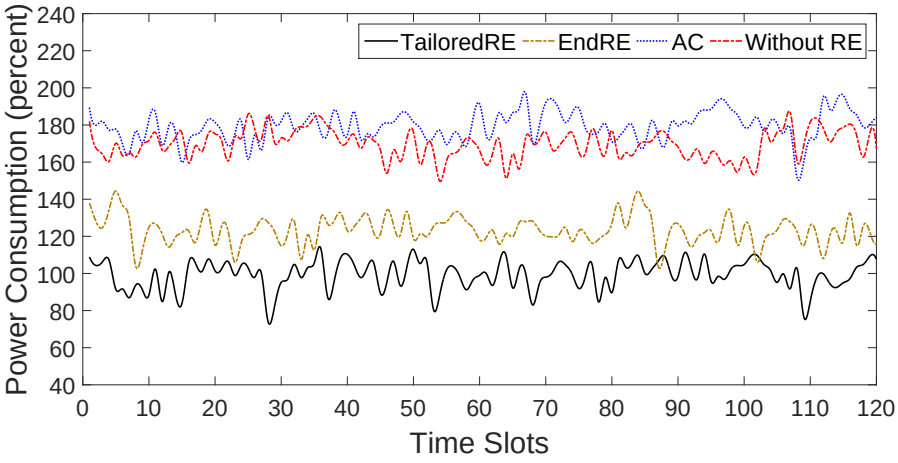


(b) Power Consumption along time

Figure 4.11: Performance Metrics for User 2

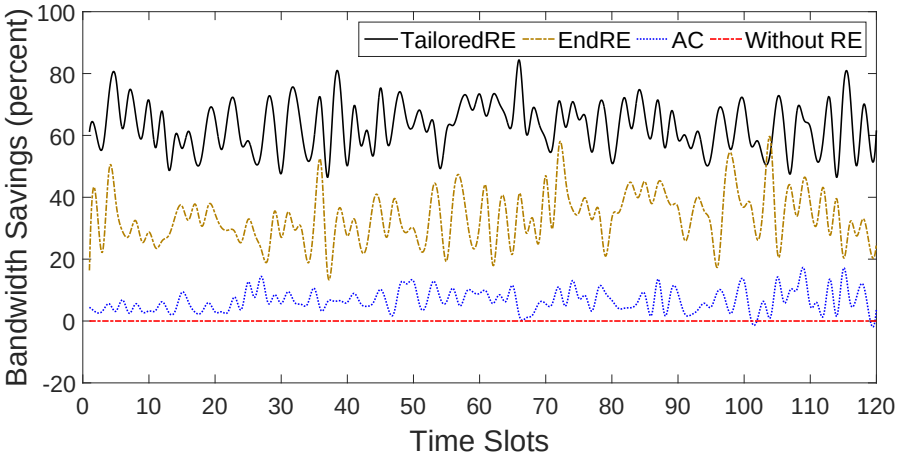


(a) Bandwidth Savings along time

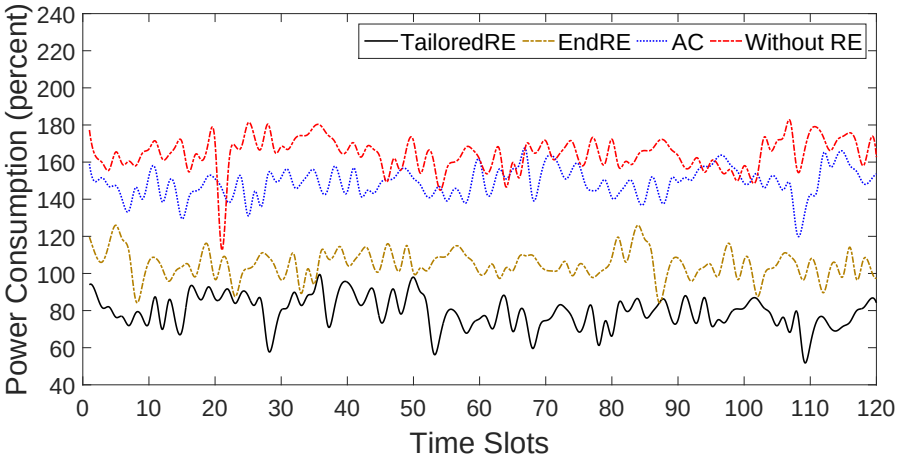


(b) Power Consumption along time

Figure 4.12: Performance Metrics for User 3

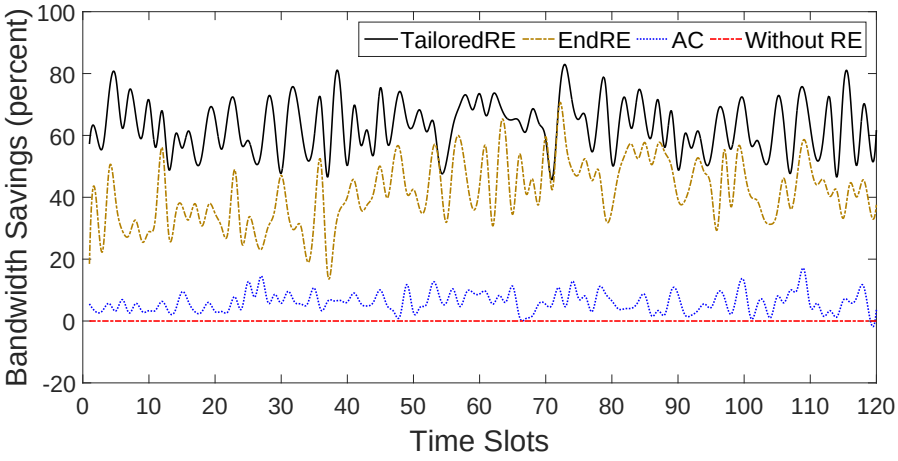


(a) Bandwidth Savings along time

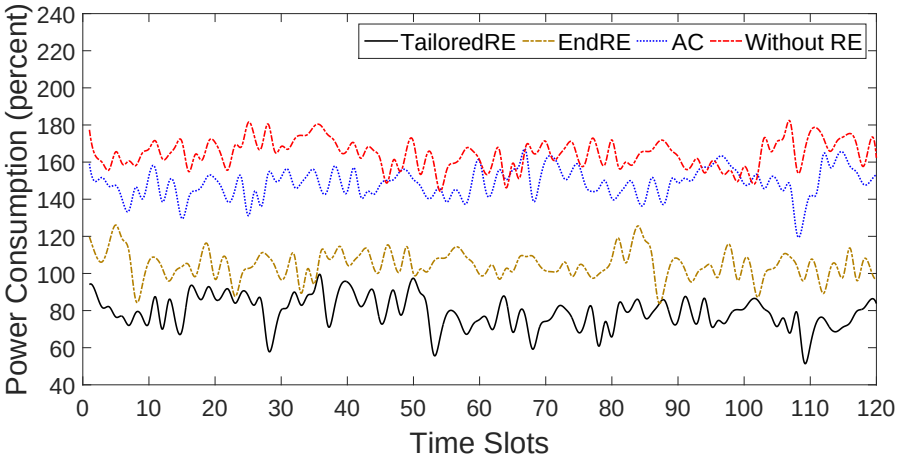


(b) Power Consumption along time

Figure 4.13: Performance Metrics for User 4

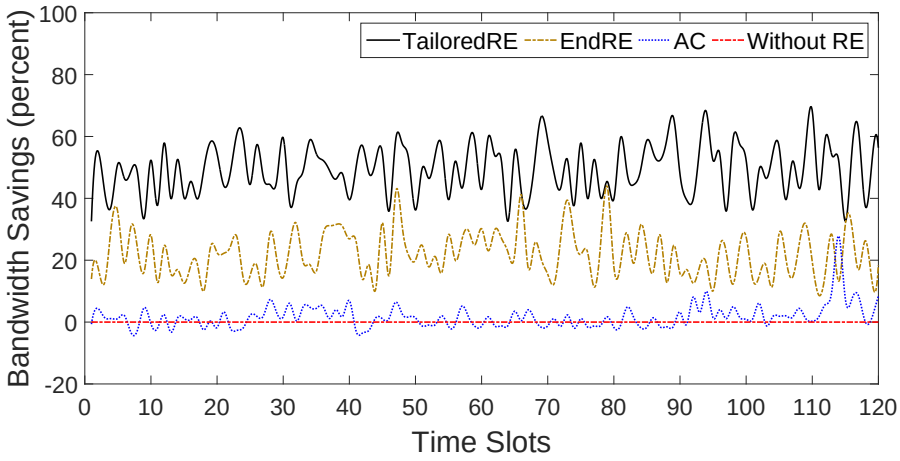


(a) Bandwidth Savings along time

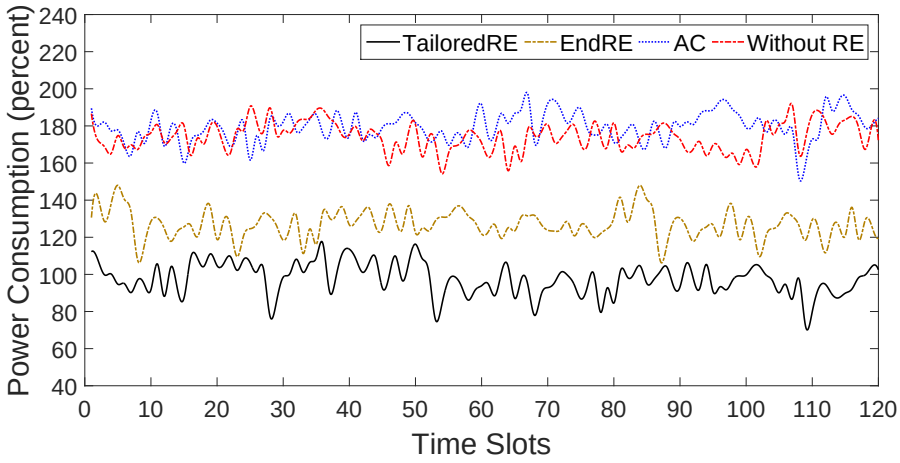


(b) Power Consumption along time

Figure 4.14: Performance Metrics for User 5

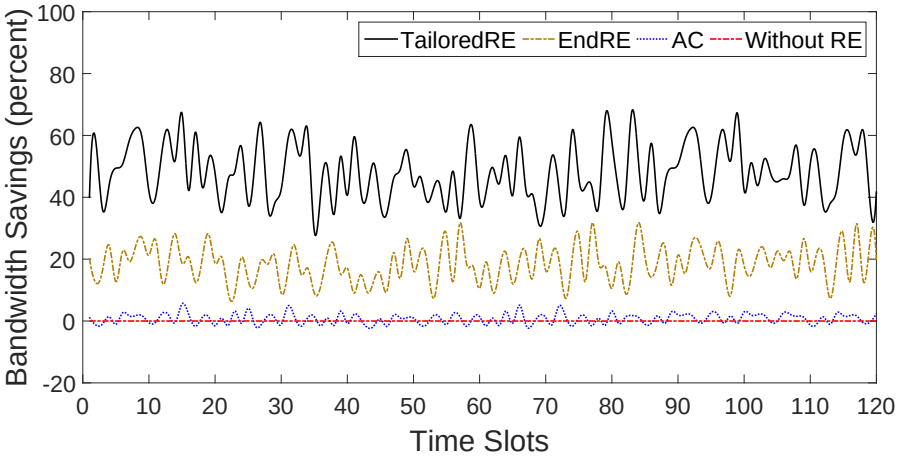


(a) Bandwidth Savings along time

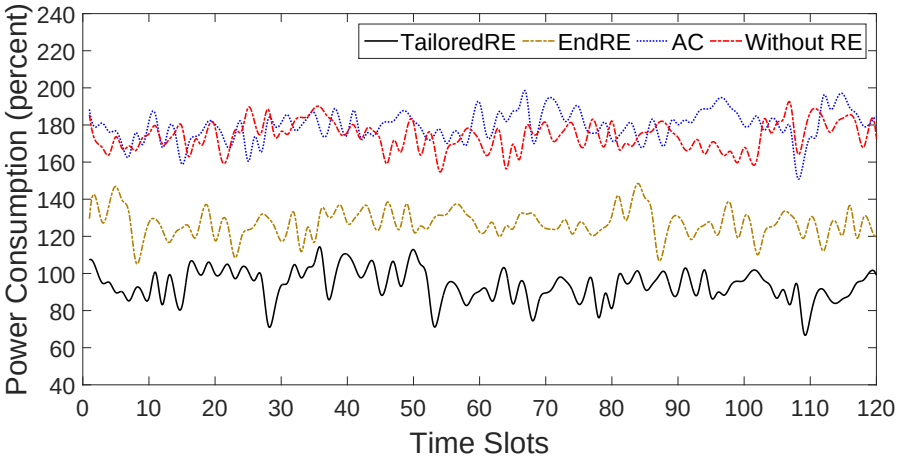


(b) Power Consumption along time

Figure 4.15: Performance Metrics for User 6

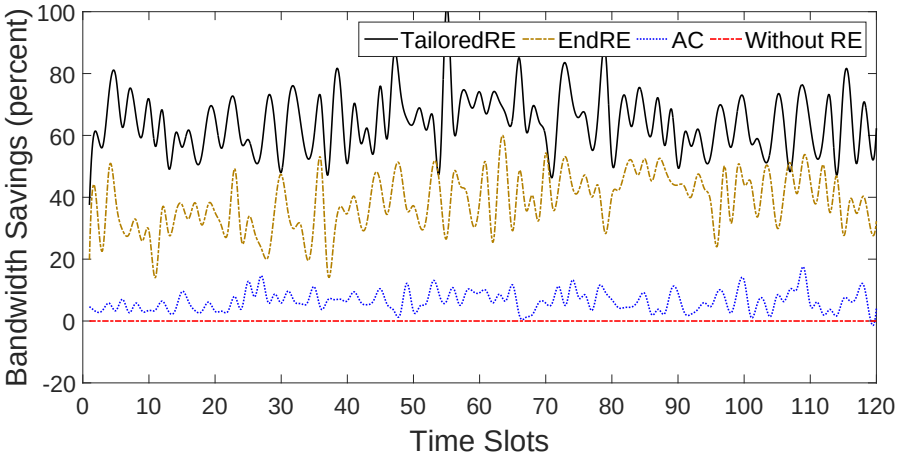


(a) Bandwidth Savings along time

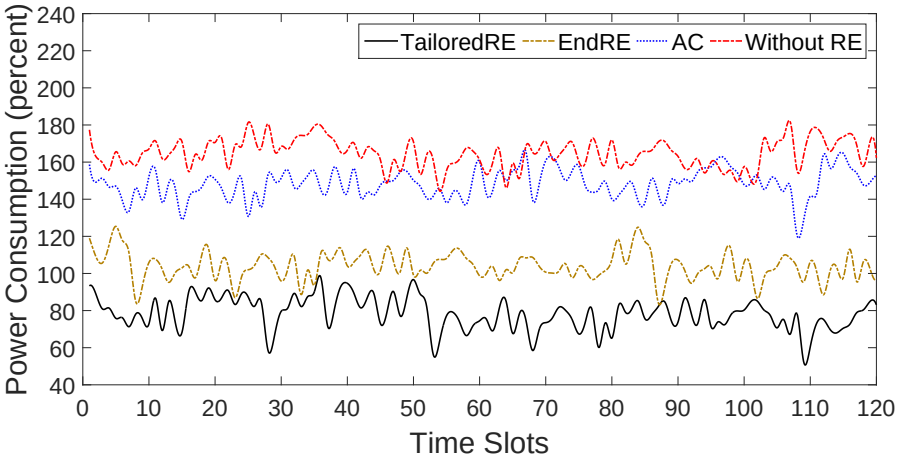


(b) Power Consumption along time

Figure 4.16: Performance Metrics for User 7

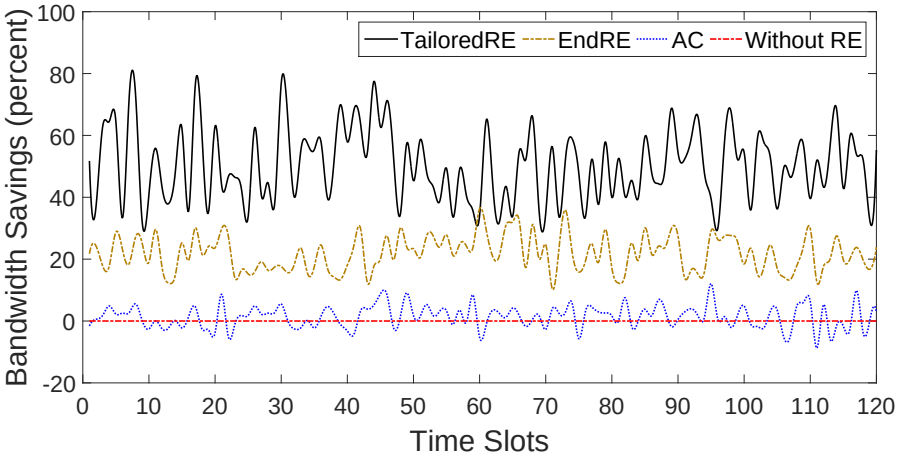


(a) Bandwidth Savings along time

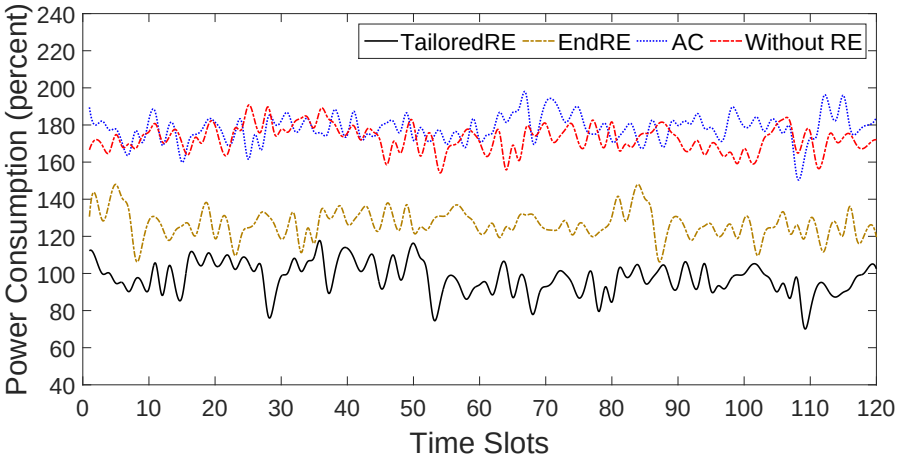


(b) Power Consumption along time

Figure 4.17: Performance Metrics for User 8

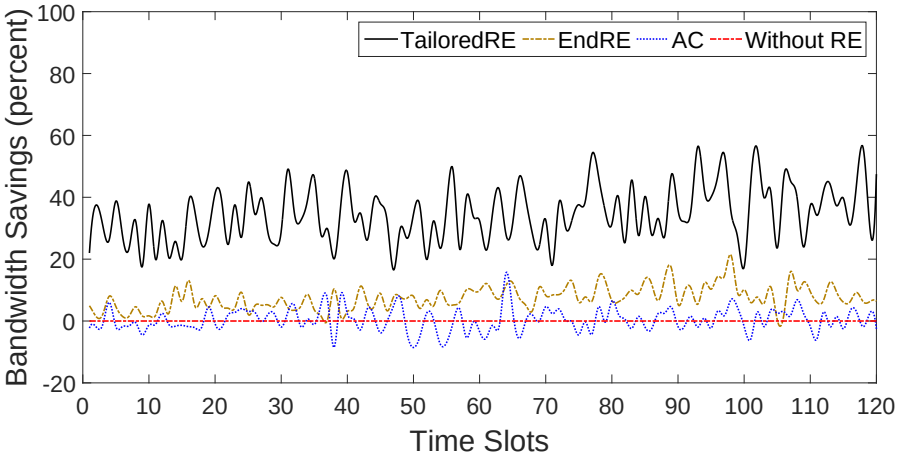


(a) Bandwidth Savings along time

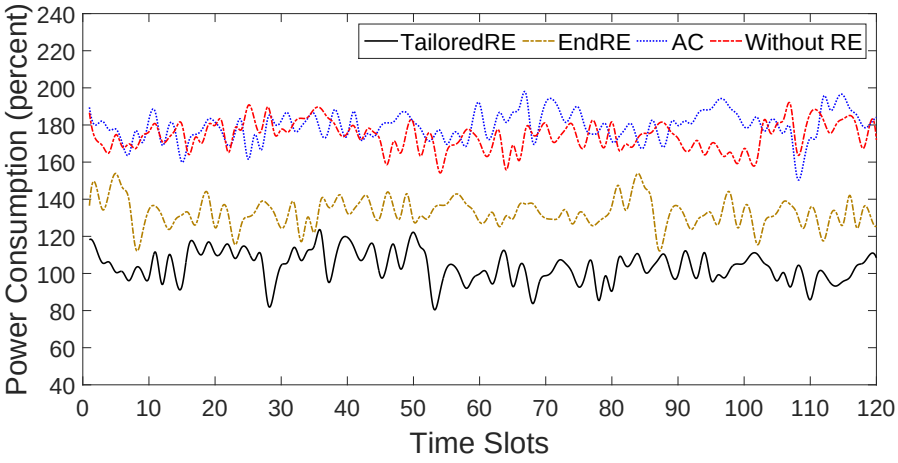


(b) Power Consumption along time

Figure 4.18: Performance Metrics for User 9

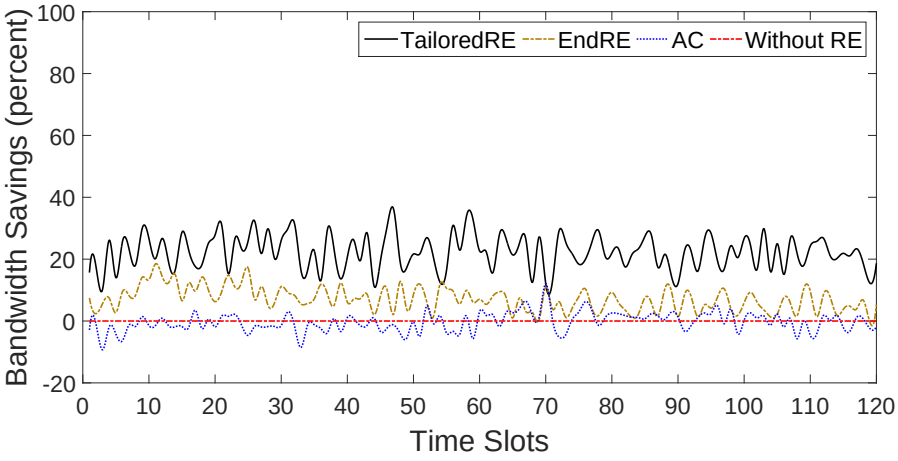


(a) Bandwidth Savings along time

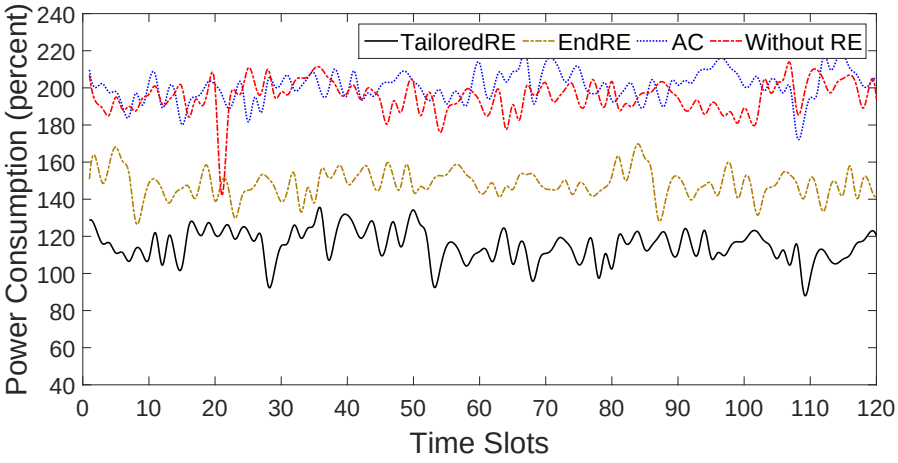


(b) Power Consumption along time

Figure 4.19: Performance Metrics for User 10

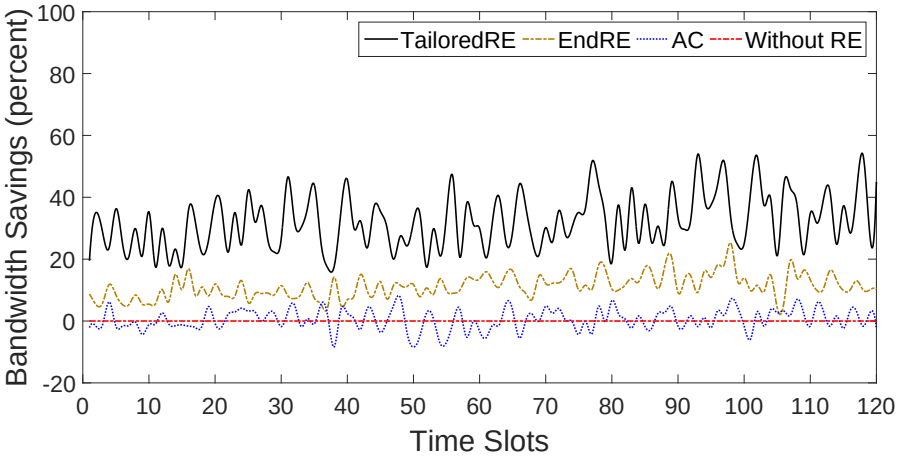


(a) Bandwidth Savings along time

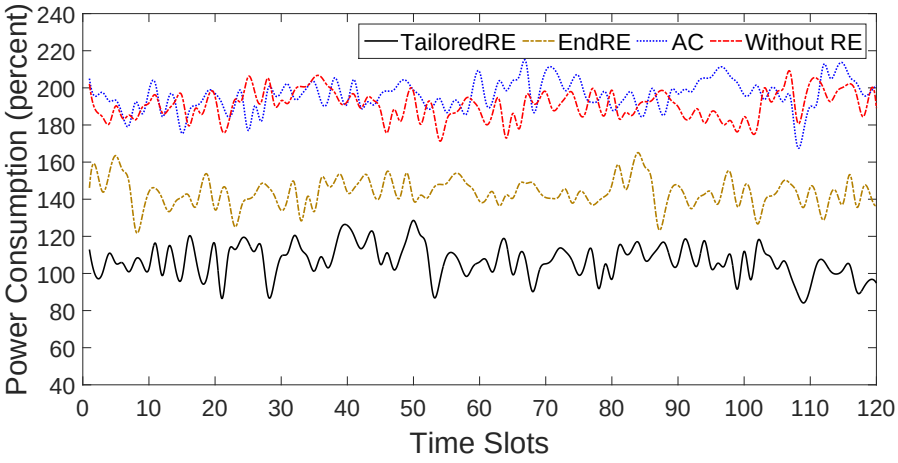


(b) Power Consumption along time

Figure 4.20: Performance Metrics for User 11



(a) Bandwidth Savings along time



(b) Power Consumption along time

Figure 4.21: Performance Metrics for User 12

AC is enabled, and when TailoredRE is enabled. Similar to bandwidth savings curve, data points in the Power Consumption Curve represent average power consumed during a time slot. In order to make a fair comparison of power consumption the number and type of the data requests in each time slots for each measurement is ensured to be same. Total power consumption of all the applications are summed for every time slot. Power measurement does not contain power consumed by display and 3G components and only includes CPU and Wi-Fi components. This is done by deselecting the display and 3G module in the menu of the PowerTutor application. This is because of the fact that display power consumption will be same across all the TRE methods as they dont affect any display component. As we are conducting our experiments with Wi-Fi modem, power consumed by 3G module would be irrelevant and essentially be a noise in our measurement.

Power Consumption of EndRE and TailoredRE would be dominated by the TailoredRE Client application since all sending and receiving of data requests and data responses are performed through this application. The power consumption plots comprises of Figures 4.10-4.21(a) representing the power consumption for each individual user over time slots in our experiment and Figure 4.23 representing the bar diagram of average power consumption obtained by the users. It can be seen that the power consumption of TailoredRE is significantly lower than EndRE's consumption. We can also see from the figures that power consumption of WithoutRE and AC is much higher than the TailoredRE's and EndRE's power consumption. In most cases, AC's power consumption is fairly higher than the power consumed when no TRE is performed. This is mainly attributed to the fact that computational and uplink transmission overhead involved in AC.

Figure 4.22 and Figure 4.23 show respectively the relationship between the average bandwidth bandwidth saving ratio and different user preferences, and that between the average power consumption and different user preferences. From Figure 4.22 we can see that different user preference have different bandwidth saving ratios, which indicate the necessity to conduct personal TRE for individual user. In addition, compared with EndRE and AC, for each user preference, TailoredRE always has better bandwidth saving than

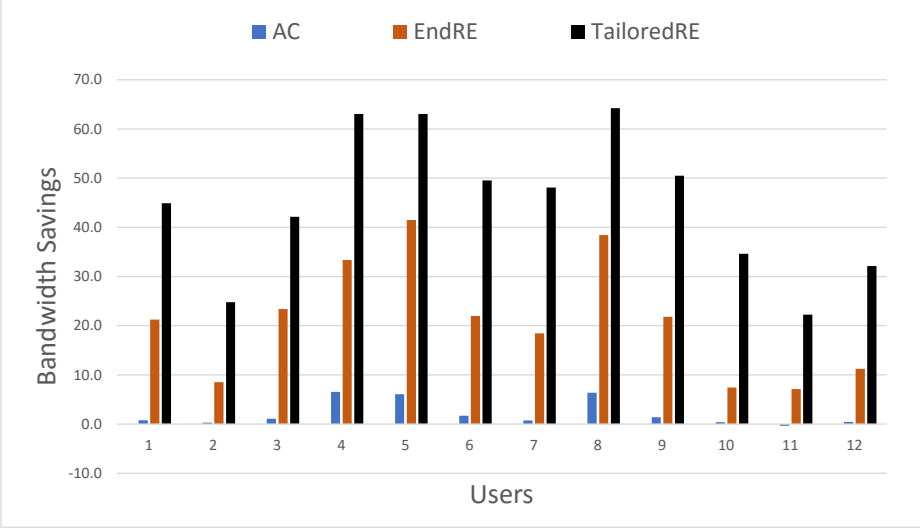


Figure 4.22: Bandwidth Savings among Users

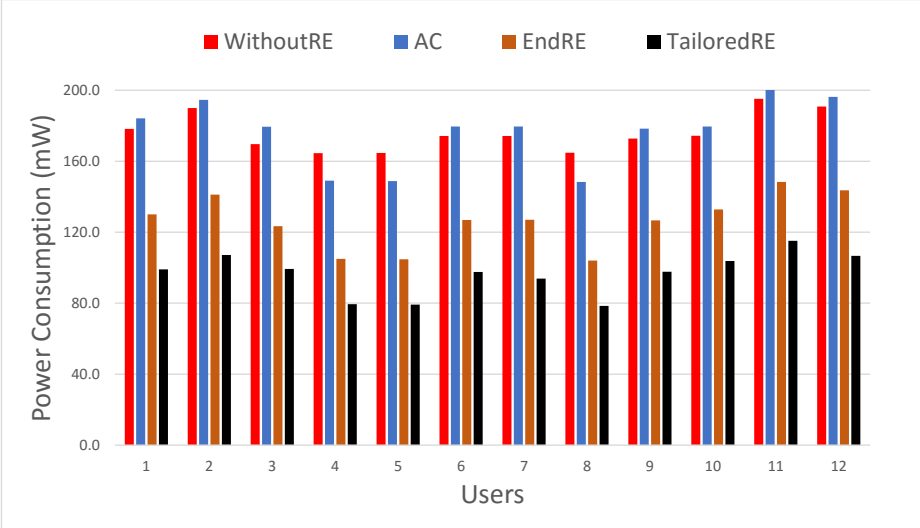


Figure 4.23: Power Consumption among Users

EndRE and AC for the same reason in Figures 4.10-4.21(a). We can also see from Figure 4.23 that with different user preferences, the smartphone's power consumption is varying. In addition, for each user, both TailoredRE and EndRE consume relatively lower power when compared with AC and the case without TRE, for the same reason in Figures 4.10-4.21(b).

In order to further elaborate on the huge upside potential for energy savings achieved by our TailoredRE system, we have plotted overall percentage of energy savings that could be obtained by a typical mobile device in a day. Energy Savings is calculated using the formula $\frac{E_{WithoutRE}-E_{WithRE}}{E_{WithoutRE}}$. In this formula, $E_{WithoutRE}$ is the amount of energy consumed when no TRE operations are performed and E_{WithRE} is the energy consumption observed when any particular TRE technique is used. As we have mentioned earlier, the device we used for experimentation is Asus Fonepad whose battery capacity is 4325 mAh. Through our experimentation we have found that, in a typical day with the regular usage of the device, the battery is found to last for 21 hours at which point it needs to be recharged. When TailoredRE client and clones are put to use, we could achieve energy savings of over 43% - 53% which translates in terms of battery capacity to 1860mAh - 2300mAh. In perspective of battery lifetime, on average the mobile device battery lasted over 40 - 45 hours. In figure 4.24 we have plotted the percentage of overall energy savings that could be achieved by our method along with other compared methods. It can be seen that our method by personalizing the TRE operations could achieve more than twice the energy savings when compared to EndRE and more than four times the saving of Asymmetric Caching (AC) which most of the times has negative effect.

The rationale behind the negative energy savings effect in AC can be explained as follows. In AC, the server upon receiving the data response conducts *MAXP* and *SHA1* hash computation to detect the redundant hashes and chunks. When the chunks are found in its cache it sends the compressed or encoded response (hash and shim) to the client otherwise it sends the original chunks as it is. The mobile client, upon receiving the data from the server, checks whether the data received are hashes or the chunks. If the data received are hashes, it checks if they are in the cache otherwise it conducts *MAXP* and *SHA1*

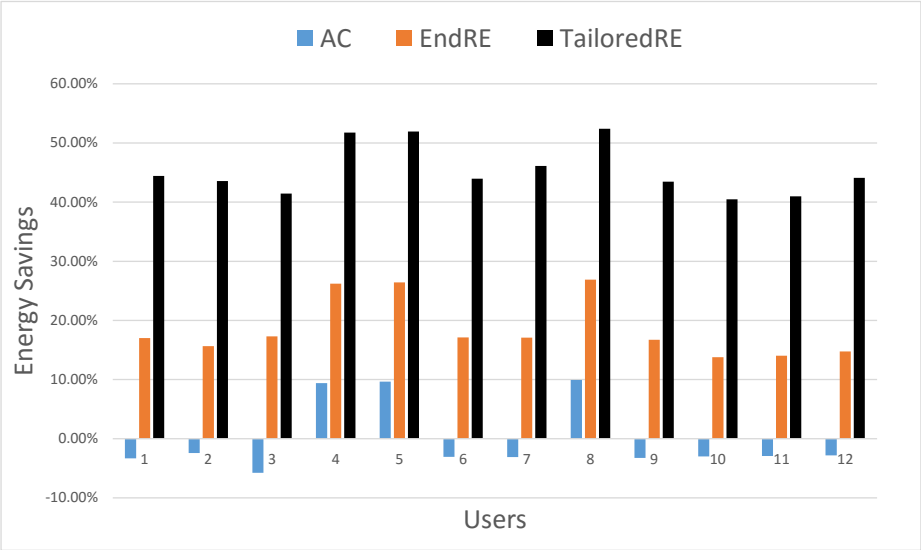


Figure 4.24: Potential Energy Savings

hash computation and then checks if they are cached. In either cases if they are found to be cached, then this will act as a trigger for feedback mechanism. It is to be noted from the survey data that some users use only two or three applications overall while many others use up-to 12 applications. When the number of applications is small, we encounter same chunks often as they are from those few applications. This obviously gives us the higher hit ratio of chunks. However, When the number of applications is large, we encounter much more distinct chunks often as they are from wide variety applications resulting in mediocre hit ratio of chunks. Hit Ratio impacts Bandwidth Savings of AC in two ways. If the hit ratio is high, there will be lot of compressed data transmissions (only hash with shim). Even though there are feedbacks occurring simultaneously, since there are high number reduced data transmissions, it overtakes the overhead incurred due to feedback. Hence, we achieve positive Bandwidth Savings in these cases. Contrarily, if the hit ratio is mediocre, there will be relatively lesser compressed data transmissions. Thus, it cannot offset the overhead incurred due to feedback. Hence, we achieve negative Bandwidth Savings in those cases.

4.6 Summary

This chapter has provided the results and analysis involved in performance evaluation of TailoredRE system. Through simulation, we have evaluated TailoredRE using various metrics of a typical redundancy elimination system, such as the redundancy hit ratio, Bandwidth Savings ratio, and the normalized throughput. Through implementation of real prototype, we were able to highlight the performance of TailoredRE in real time using metrics such as bandwidth savings and power consumption. The following chapter provides summary of our contributions and provides a perspective to readers on the importance of TRE.

Chapter 5

Conclusion

This chapter summarizes and concludes the work we have done in this thesis on TailoredRE. We summarize the critical design elements that make the TailoredRE look distinguishing and advantageous than existing TRE solutions.

Traffic Redundancy Elimination (TRE) systems play a crucial role in improving the throughput performance of any network link especially the last-hop access link which traditionally have scarce bandwidth availability to support all the end users. The growing importance TRE systems has led to the numerous research developments that are directed towards making the TRE system more effective. The motivation for this research is to increase the efficiency of TRE conducted which would result in higher bandwidth savings attainable on the network links. Towards this motivation, we have developed a traffic redundancy elimination system called TailoredRE. TailoredRE, unlike other systems, takes into account various factors like application usage behavior of each individual mobile users, resource constrainment (or otherwise limited resource availability) of the mobile devices for expensive TRE operations so that it can make the design better and more suited for the mobile computing environment. Specifically the distinguishing design elements of TailoredRE can be summarized as follows

(1) **Cloud-Clone based TRE:** TailoredRE system has been designed to leverage the rich computation resources available the cloud to offload the operational cost of per-

forming the TRE from the mobile devices to their clones in the cloud. In the cloud, the TailoredRE clone, acting as a proxy for a mobile device, performs major TRE operations such as redundancy detection in data streams, hash computation, caching and encoding of data streams. Clones and mobile devices maintain a synchronized caches between them.

(2) **Application-adaptive RE:** Different from other TRE systems, TailoredRE clone provides personalized TRE to its mobile device users based on the profile information obtained through Redundancy Profiling. Redundancy profiling involves analysis of redundancy metrics such as the ratio of redundant bytes to total bytes in byte streams of traffic belonging to various application the mobile user uses on his mobile phones. Through this analysis, we select an application against which redundancy is conducted, rather than performing redundancy application agnostically.

(3) **MAXP Chunk Partitioning algorithm:** There exists a plenty of chunking or fingerprinting algorithms to compute a set of representative fingerprints in a data stream. These algorithms pose significant trade-offs between each other. TailoredRE system adopts MAXP algorithm to find the boundaries since it has been found optimal among all because of its minimal overhead and maximal effectiveness of RE.

(4) **Cache sharing among clones:** Mobile traffic data naturally exhibit Inter-User Redundancy since multiple users access same popular contents in the web during a given period of time [48] [26]. In order take advantage of this phenomenon and utilize the cache resource efficiently in the cloud, we design a cache sharing mechanism in which clones are clustered based on the profile information of the mobile users.

TailoredRE incorporating the above four novel design elements to provide an efficient TRE system when compared to the existing methods. To conduct evaluation of our system we have collected real network traffic from the mobile device over a period of two months. We have evaluated our proposed system through the analysis and simulation driven by real traces. We have compared the results of the simulation with two existing methods:

a receiver-based TRE method named Asymmetric Caching and the sender based method called as EndRE. We have found that TailoredRE system performed better in experiments conducted over a wide variety of real data traces and improvements in all necessary performance metrics that evaluates a TRE system. In order to assess the performance of TailoredRE in real time, we implemented a working prototype of our system and measured two key metrics as part of our evaluation. We have shown that the TailoredRE system performs better in both the scenarios. Thus, we are optimistic that TailoredRE system has the potential to become a promising RE system in the future.

Appendices

Appendix A Preferred Applications Survey

Google form provided below in the following page preferred applications survey form that we created using Google for collecting user information from research scholars in order to assist in our experimentation of real prototype of Tailored RE.

TailoredRE: A Personalized Cloud-based Traffic Redundancy Elimination for Smartphones

Preferred Applications survey

Choose the applications or website you use, and select the time you spend in one week
If an application is accessed by browser but is listed in the application list, we can put them in to that application.
However, if applications are accessed by browser but not listed in the application list, we put them into browser.
Additionally, if some applications are not accessed by browser, we can list them (and their usage time in one week) as an answer in other application item.
Thank you.

* Required

Youtube

- ○ Don't use
- ○ <30 minites
- ○ 0.5h-2h
- ○ 2h-4h

- 4h-7h
- >7h
- Other :

Facebook

- Don't use
- <30 minites
- 0.5h-2h
- 2h-4h
- 4h-7h
- >7h
- Other :

CNN

- Don't use
- <30 minites
- 0.5h-2h
- 2h-4h

- 4h-7h
- >7h
- Other :

Quora

- Don't use
- <30 minites
- 0.5h-2h
- 2h-4h
- 4h-7h
- >7h
- Other :

Techcrunch

- Don't use
- <30 minites
- 0.5h-2h
- 2h-4h

- ○ 4h-7h
- ○ >7h
- ○ Other :

NYTimes

- ○ Don't use
- ○ <30 minites
- ○ 0.5h-2h
- ○ 2h-4h
- ○ 4h-7h
- ○ >7h
- ○ Other :

Twitter

- ○ Don't use
- ○ <30 minites
- ○ 0.5h-2h

- 2h-4h
- 4h-7h
- >7h
- Other :

Instagram

- Don't use
- <30 minites
- 0.5h-2h
- 2h-4h
- 4h-7h
- >7h
- Other :

Spotify

- Don't use
- <30 minites
- 0.5h-2h

- ○ 2h-4h
- ○ 4h-7h
- ○ >7h
- ○ Other :

Bloomberg

- ○ Don't use
- ○ <30 minites
- ○ 0.5h-2h
- ○ 2h-4h
- ○ 4h-7h
- ○ >7h
- ○ Other :

Browser *

- ○ Don't use
- ○ <30 minites
- ○ 0.5h-2h

- ○ 2h-4h
- ○ 4h-7h
- ○ >7h
- ○ Other :

Other applications you prefer and the regular using time in one week *

Your answer

SUBMIT

Never submit passwords through Google Forms.

Bibliography

[1] Mikhail Afanasyev, David G Andersen, and Alex C Snoeren. Efficiency through eavesdropping: Link-layer packet caching. In *Proc. of NSDI*, 2008.

[2] Bhavish Agarwal, Aditya Akella, Ashok Anand, Athula Balachandran, Pushkar Chitnis, Chitra Muthukrishnan, Ramachandran Ramjee, and George Varghese. Endre: An end-system redundancy elimination service for enterprises. In *Proc. of NSDI*, 2010.

[3] Mohammed Al-Laham and Ibrahiem MM El Emary. Comparative study between various algorithms of data compression techniques. *IJCSNS*, 7(4):281, 2007.

[4] Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *Proc. of SIGCOMM*, 2008.

[5] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, and Ramachandran Ramjee. Redundancy in network traffic: findings and implications. In *Proc. of SIGMETRICS*, 2009.

[6] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, and Ramachandran Ramjee. Redundancy in network traffic: findings and implications. In *Proc. of SIGMETRICS/Performance*, 2009.

[7] Ashok Anand, Vyas Sekar, and Aditya Akella. Smartre: an architecture for coordinated network-wide redundancy elimination. In *Proc. of SIGCOMM*, 2009.

[8] Ulrich Appel and Achim V Brandt. Adaptive sequential segmentation of piecewise stationary time series. *Information sciences*, 29(1):27–56, 1983.

[9] Nikolaj Bjørner, Andreas Blass, and Yuri Gurevich. Content-dependent chunking for differential compression, the local maximum approach. *J. Comput. Syst. Sci.*, 76(3-4):154–203, 2010.

[10] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proc. of EuroSys*, 2011.

[12] Cisco. Cisco visual networking index. http://newsroom.cisco.com/press-release-content?type=webcontent&articleId=668380. [Accessed in March 2016].

[13] Cisco. Cisco wide area application acceleration services. http://www.cisco.com/en/US/products/ps5680/Products_Sub_Category_Home.html. [Accessed in March 2016].

[14] Computerworld. Wan optimization continues growth. http://www.computerworld.com.au/article/141254/wan_optimization_continues_growth/. [Accessed in March 2016].

[15] Philipp B. Costa, Paulo A. L. Rego, Lincoln S. Rocha, Fernando A. M. Trinta, and José N. de Souza. Mpos: A multiplatform offloading system. In *Proc. of the 30th Annual ACM Symposium on Applied Computing*, 2015.

[16] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services*, 2010.

[17] Fahad R Dogar, Amar Phanishayee, Himabindu Pucha, Olatunji Ruwase, and David G Andersen. Ditto: a system for opportunistic caching in multi-hop wireless networks. In *Proc. of MobiCom*, 2008.

[18] Donald Eastlake and Paul Jones. Us secure hash algorithm 1 (sha1), 2001.

[19] IT Facts. Wan optimization revenues grow 16 http://itfacts.biz/wan-optimization-market-to-grow-16/1205. [Accessed in March 2016].

[20] Aaron Gember, Ashok Anand, and Aditya Akella. A comparative study of handheld and non-handheld traffic in campus wi-fi networks. In *Passive and Active Measurement*, pages 173–183. Springer, 2011.

[21] Archit Gupta, Aditya Akella, Srinivasan Seshan, Scott Shenker, and Jia Wang. Understanding and exploiting network traffic redundancy. *UW-Madison, Madison, WI, USA, Tech. Rep*, 1592, 2007.

[22] Emir Halepovic, Majid Ghaderi, and Carey Williamson. On the performance of redundant traffic elimination in wlans. In *Proc. of ICC*, 2012.

[23] Emir Halepovic, Carey Williamson, and Majid Ghaderi. Dynabyte: A dynamic sampling algorithm for redundant content detection. In *Proc. of ICCCN*, 2011.

[24] Emir Halepovic, Carey Williamson, and Majid Ghaderi. Enhancing redundant network traffic elimination. *Computer networks*, 56(2):795–809, 2012.

[25] Sunghwan Ihm, KyoungSoo Park, and Vivek S. Pai. Wide-area network acceleration for the developing world. In *Proc. of the 2010 USENIX Conference on USENIX Annual Technical Conference*, 2010.

[26] Ram Keralapura, Antonio Nucci, Zhi-Li Zhang, and Lixin Gao. Profiling users in a 3g network using hourglass co-clustering. In *Proc. of MobiCom*, 2010.

[27] Cristian Lumezanu, Katherine Guo, Neil Spring, and Bobby Bhattacharjee. The effect of packet loss on redundancy elimination in cellular wireless networks. In *Proc. of SIGCOMM*, 2010.

[28] Udi Manber et al. Finding similar files in a large file system. In *Usenix Winter*, volume 94, pages 1–10, 1994.

[29] mHotSpot. http://www.mhotspot.com/. [Accessed in March 2016].

[30] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. In *Proc. of SIGOPS*, 2001.

[31] A. Blass N. Bjorner and Y. Gurevich. Content-dependent chunking for differential compression, the local maximum approach. Technical Report 109, Microsoft Research, 2007.

[32] PeerApp. Video and p2p caching. `http://www.peerapp.com/Technology/VideoCaching.aspx`. [Accessed in March 2016].

[33] Pennysleuth. Invest in cell phone infrastructure for growth in 2010. `http://pennysleuth.com/invest-in-cell-phone-infrastructure-for-growth-in-2010/`. [Accessed in March 2016].

[34] Diego Perino, Matteo Varvello, and Krishna P. N. Puttaswamy. Icn-re: Redundancy elimination for information-centric networking. In *Proc. of the Second Edition of the ICN Workshop on Information-centric Networking*, 2012.

[35] Feng Qian, Junxian Huang, Jeffrey Erman, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. How to reduce smartphone traffic volume by 30%? In *Passive and Active Measurement*, pages 42–52. Springer, 2013.

[36] Sean C. Rhea, Kevin Liang, and Eric Brewer. Value-based web caching. In *Proc. of the 12th International Conference on World Wide Web*, 2003.

[37] Riverbed. Riverbed networks: Wan optimization. `http://www.riverbed.com/solutions/optimize/`. [Accessed in March 2016].

[38] S. Sanadhya, R. Sivakumar, K. Kim, P. Congdon, S. Lakshmanan, and J. Singh. Asymmetric caching: improved network deduplication for mobile devices. In *Proc. of MobiCom*, 2012.

[39] Shan-Hsiang Shen, Aaron Gember, Ashok Anand, and Aditya Akella. Refactor-ing content overhearing to improve wireless performance. In *Proc. of MobiCom*, 2011.

[40] Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, and Ellen Zegura. Cosmos: Computation offloading as a service for mobile devices. In *Proc. of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2014.

[41] Yang Song, Katherine Guo, and Lixin Gao. Redundancy-aware routing with limited resources. In *Proc. of ICCCN*, 2010.

[42] Neil T Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proc. of SIGCOMM*, 2000.

[43] Squid. Web proxy cache. `http://www.squid-cache.org/`. [Accessed in March 2016].

[44] Wireshark. https://www.wireshark.org/. [Accessed in March 2016].

[45] Alec Wolman, Geoff Voelker, Nitin Sharma, Neal Cardwell, Molly Brown, Tashana Landray, Denise Pinnel, Anna Karlin, and Henry Levy. Organiz at ion-basedanalysisof web-objectsharingandcachi ng. 1999.

[46] Daniel Zeng, Fei-Yue Wang, and Mingkuan Liu. Efficient web content delivery using proxy caching techniques. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(3):270–280, 2004.

[47] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of Hardware/Software Codesign and System Synthesis*, 2010.

[48] Yan Zhang and Nayeem Ansari. On protocol-independent data redundancy elimination. *Communications Surveys & Tutorials, IEEE*, 16(1):455–472, 2014.

[49] Zhenyun Zhuang and Raghupathy Sivakumar. Wireless memory: Eliminating communication redundancy in wi-fi networks. In *Proc. of WoWMoM*, 2011.

[50] Eyal Zohar, Israel Cidon, and Osnat Mokryn. Pack: Prediction-based cloud bandwidth and cost reduction system. *IEEE/ACM Transactions on Networking (TON)*, 22(1):39–51, 2014.

[51] Eyal Zohar, Israel Cidon, and Osnat Ossi Mokryn. Celleration: loss-resilient traffic redundancy elimination for cellular data. In *Proc. of HotMobile*, 2012.