Cloud Assisted Traffic Redundancy Elimination for Power Efficiency in Smartphones

Shenghua He*, Haiying Shen**, Vivekgautham Soundararaj† and Lei Yu††
*Department of CSE, Washington University in St. Louis, St. Louis, MO, USA
**Department of CS, University of Virginia, Charlottesville, VA, USA
†Department of ECE, Clemson University, Clemson, SC, USA
††College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
Email: *shenghuahe@wustl.edu, **hs6ms@eservices.virginia.edu, † vsounda@clemson.edu, ††leiyu@gatech.edu

***Abstract*—The exceptional increase in the usage of smartphones has contributed to a massive increase in data traffic from application servers to the smartphones, which not only strains their computation capacities and batteries but also bogs down the last hop in data transmission. For this problem, traffic redundancy elimination (TRE) is an effective solution, in which a chunk to be transmitted could be directly fetched from the receiver's cache. However, existing TRE solutions either cannot be directly applied to or are not suitable for smartphones due to high computing and energy overhead imposed on smartphones. To address this problem, in this paper, we propose a novel TRE system, called TailoredRE, which consists of three components. First, each smartphone has a clone in the cloud that is responsible for computation intensive tasks including parsing traffic and detecting redundancy. Second, considering that each mobile user has certain applications (e.g., YouTube) to use in daily life, each smartphone's clone selectively chooses the applications that are most frequently used by the user and also have high redundancy ratios to cache data. Third, considering that some users always have common favorite applications, TailoredRE clusters their clones together to cooperatively conduct the redundancy detection task in order to reduce the cache resource consumption in the cloud. We collected traces from eleven applications including Web Browser, YouTube, CNN, Quora, Instagram and Facebook, and used the traces in simulation. We also implemented and open-sourced TailoredRE and conducted prototype-based experiments. Experiment results show that TailoredRE can achieve much higher cache hit rate, end-to-end throughput, bandwidth saving and energy efficiency compared with previous TRE methods.**

***Index Terms*—Traffic redundancy elimination, Mobile device, Smartphone**

I. Introduction

Smartphones have been becoming increasingly popular in recent years. People nowadays use smartphones as a primary computing platform in their daily lives. The increasing capacity and popularity of smartphones stimulate the rapid growth of mobile applications. However, the increasing workload and traffic of mobile applications strain the limited computation capacity and battery on smartphones. Besides computation, the wireless data communication is actually a more significant source of power consumption on the smartphones [1, 2]. It is pervasive for mobile users to use smartphones to browse websites (Facebook, BBC, CNN), online videos (e.g., YouTube) and so on. These user data accesses generate a large amount of traffic, which not only strains the battery of the smartphones but also bogs down the last hop wireless link in the data transmission, resulting in reduced throughput of the network.

Previous studies [3–12] have found that, due to the access to similar data objects from the users, the Internet traffic has significant redundancy. Accordingly, several traffic redundancy elimination (TRE) solutions [13–22] have been proposed. However, *existing TRE solutions either cannot be directly applied to or are not suitable for smartphones due to high computing and energy overhead imposed on smartphones* (smartphones, end-clients, receivers are interchangeable terms in this paper). Because the smartphones have limited computation capacity and battery, TRE must be operation-efficient and the energy cost of TRE on smartphones should be far less than the cost saved from bandwidth usage reduction. Also, considering the limited cache size of smartphones, it is critical to increase the cache hit rate of redundancy detection, so the bandwidth usage reduction can be maximized as much as possible.

Existing TRE methods can be classified into three categories: in-network based, receiver side based and server side based. In-network methods [13–17] cache all data chunks from the flows that traverse network links and encode the chunks in the future flows when they are found in the caches. However, these methods cannot reduce the traffic redundancy over last-hop links to smartphones since they do not conduct TRE at the last hop.

In the receiver-based TRE methods [18–21], each end-client sends to the sender its predicated data or the contents of its cache so that the sender can conduct the TRE accordingly. However, if we deploy such a method in smartphones, it will incur a large amount of computation and up-link transmission overheads on the smartphones as they need to constantly conduct hash computing and make predictions to notify the senders. In addition, as they do not employ fully synchronized caches between the sender and the receiver, the sender is not fully aware of the contents of receiver's cache, which reduces the redundancy hit ratio and the bandwidth savings attainable by TRE.

The sender based TRE method [22] offloads most processing effort and memory cost to servers. It requires tight cache synchronization between the sender and the client, so that the sender can be aware of all the chunks cached at the receiver and does not need to send the chunks already in the cache. However, it is costly for an application server to maintain the cache synchronization with all of its clients. More importantly, in practice, many application servers do not provide the TRE service. Therefore, a solution that does not rely on the servers but enables smartphones to conduct TRE is needed to reduce

the energy consumption of smartphones without generating much computation and energy overhead.

In all previous TRE methods, a client's cache serves all of its applications. That is, EndRE is agnostic to the applications and conducts TRE against the traffic from any application. However, since memory resource in smartphones is limited, the chunks from mobile applications with lower redundancy hit ratio may occupy the cache space, which will reduce redundancy cache hit rate and the bandwidth savings caused by TRE.

In order to reduce the computing and up-link bandwidth overhead on the smartphones while increasing the redundancy cache hit rate by making the best of the limited cache resource at smartphones to improve the efficiency and effectiveness of TRE, we propose a TRE system, called TailoredRE. *TailoredRE is novel in that it leverages cloud resource to create clones to offload TRE overhead from smartphones and leverages social network property to conduct TRE by tailoring to individual user's application activities.* Social network property here means that each user uses a certain number of favorite applications in daily life and some users have common favorite applications [23, 24]. TailoredRE incorporates three components listed below.

Cloud clone assisted TRE. TailoredRE creates a clone (i.e., a program running in a virtual machine) in the cloud for a smartphone and offloads most operation cost from the smartphone to its clone. A smartphone's clone acts as its proxy for all of its data accesses to the Internet. The clone is responsible for computation intensive tasks in TRE operations, including parsing traffic and detecting redundancy. The caches in the clone and in the smartphone are synchronized, so after the clone detects redundancy, it only needs to send the location of the cache to let the smartphone to fetch the redundant contents from its cache accordingly. Fundamentally, our "clone" is different the one in Byung-Gon [25]'s work, which loaded the relevant data and codes to the cloud to leverage the part of the execution on the smartpohone.

Personalized and application-adaptive TRE. Each user usually has his/her preferred applications in daily life. For example, a person may log in his/her Facebook and Gmail accounts and turns on the YouTube music play in his/her smartphone before starting to work, and may watch NetFlix or YouTube, or play online game on his/her smartphone after work. Also, traffics from different applications have different redundancy hit ratios [18]. For instance, News sites has 38.7% redundancy hit ratio, shopping sites have 45.5%, social network has 51.5% and finance sites has 63.0% [18]. Therefore, TailoredRE caches the contents of a user's preferred applications that (s)he usually uses and have high redundancy hit ratios to increase the redundancy cache hit rate. This is a novelty of TailoredRE in comparison with the previous TRE methods that are agnostic to the applications. As a result, TailoredRE can always avoid the cache pollution from applications with lower redundancy hit ratios, which improves the cache hit ratio greatly.

Cache sharing among clones. Many users may share similarly visited contents such as those who always watch movies in the same category on NetFlix. Then, their clones in the cloud may cache similar contents and a user may visit new contents that are already in another user's clone. Therefore, in TailoredRE, the clones of the users that have similar interests form a group can share their cached contents between each other. As a result, a clone can quickly fetch contents from another clone in its group rather than fetch the contents from the application server, which reduces cache resource consumption.

We used a smartphone to collect 142GB trace data in total from eleven mobile applications, including CNN, Quora, and Facebook, and analyzed the redundancy hit ratios in these traces. We found that there are indeed considerable redundancies in these traces. Driven by these real traces, we conducted simulations to compare the performance of TailoredRE with previous TRE methods. Experimental results show the superior performance of TailoredRE in terms of redundancy hit ratio, bandwidth savings and throughput. In addition, we conducted a survey on real users' preferred applications. Based on the survey, we implemented TailoredRE clone in Amazon EC2 and TailoredRE client in an Android smartphone to carry out a prototype-based test on the bandwidth savings and power consumption. The experimental results show that TailoredRE can greatly reduce the bandwidth and power consumption. We also open-sourced our implemented TailoredRE in GitHub [26].

For privacy protection, TailoredRE will not profile a user's application preference without the user's permission. The users will be informed that such a permission can save the power of their smartphones but at the cost of possible privacy risk. The rest of this work is organized as follows. Section II discusses related work. Section III describes our system design in detail. Section IV and Section V present the experimental results from the simulation and real implementation. Section VI concludes this paper with remarks on our future work.

II. Related Work

The previous TRE methods can be classified into three categories: in-network based, receiver side based and server side based. In-network TRE solutions, such as Cisco's Wide Area Application Services (WAAS) [13] and RiverBed's Steel-Head [14], place a middle-box across bandwidth-constrained links. Other in-network TRE solutions proposed in [15–17] can remove the redundancy in the traffic by traversing potentially duplicate packets in common routes. However, all of these in-network TRE solutions cannot be used to reduce the traffic redundancy over last-hop links to the smartphones.

Receiver-based TRE solutions [18–21] conduct TRE based on the feedback sent from the receiver to the sender. The feedback can be the predicted data that will be sent from the sender or the contents of the receiver's cache. Then, the sender does not need to send the data shown in prediction or already in the receiver's cache. In Celleration [18], the gateway located at the cellular network Internet entry point, stores the signature of the flows, predicts individual mobile user's future data, and executes flow reduction mechanism for bandwidth savings. In AC [19], the receiver sends the sender the feedback of the data in its cache that is pertinent to the ongoing traffic flow in real time, and the sender then performs RE operations based on its feedback cache storing the received data and regular cache. In

PACK [20], upon receiving the data stream, the receiver parses it into chunks, computes their signatures and retrieves the corresponding chunk chain if a matching signature is found in its cache, and sends a prediction (PRED) message containing the signature of the chunk to the server. The sender will verify the signature of the chunk in the PRED message. If it matched the corresponding range of the data, it replaces the chunk with PRED-ACK. REfactor [21] leverages wireless overhearing to perform sub-packet level redundancy elimination. It maintains a reception probability vector for each chunk entry (stored in access point (AP)'s hash table) that indicates the probability that the receiver has the chunk, based on which, it estimates the expected benefit for the receiver if the packet is encoded and sends the receiver the encoded packet only when the expected benefit is high. Though these receiver-based methods can be deployed in the smartphones to improve the throughput over last-hop link, they demand high computational and up-link bandwidth resources at the receiver. In addition, the sender's cache and receiver's cache are not fully synchronized and the sender must rely on the receiver's predictions to conduct TRE, which will reduce the redundancy hit ratio and bandwidth savings attainable by TRE.

EndRE [22] is a sender-based TRE solution. It keeps the caches in the sender and receiver synchronized, so that the sender is aware of all the chunks cached at the receiver and does not have to send the chunks that are already in the receiver's cache. However, it is costly for an application server to maintain the cache synchronization with all of its clients. Though the sender-based TRE method can solve the problem of limited computation and energy resources of the mobile devices, many application servers do not provide the TRE service in practice. Therefore, it is important to find a solution for the smartphones to conduct TRE while consuming less computation and energy resources on the smartphones.

III. System Design of TailoredRE

Our proposed TailoredRE system consists of three components: *cloud clone assisted TRE, personalized and application-adaptive TRE,* and *cache sharing among clones*. Below, we introduce each component.

A. Cloud Clone assisted TRE

TailoredRE is a TRE service provided by the cloud to the smartphones. As shown in Figure 1, each smartphone has a clone in the cloud, which is a program running in a virtual machine (VM) to conduct TRE for its belonged smartphone. Note that multiple programs can run in one VM. The clone functions as a proxy for its smartphone, and their caches are synchronized. In order to use TailoredRE service, users need to install the TailoredRE client on their smartphones, which is transparent to user apps and works by intercepting the requests sent out by each running app and transmitting them to the clone. The clone then forwards these requests to application servers. Upon receiving the data response from the application servers, the clone performs TRE operations and sends the compressed data to the TailoredRE client. Specifically, when the traffic packets destining to a certain smartphone arrive at the its clone, the clone stripes the header of the packet to get

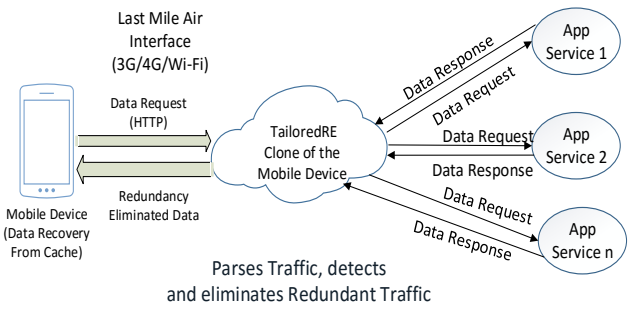


Fig. 1. The procedure in the TailoredRE system.

the application data, and then conducts redundancy detection. That is, the clone partitions the data into chunks, computes their hashes and check whether each chunk exists in its cache. If yes, the clone replaces the chunk with its offset in the cache. The clone also inserts a shim, which is a one-byte binary number, between chunks to separate them in the transmission, so that the client can distinguish chunks. At the client side, it replaces the offset with the chunk in its cache. After removing all the shims in the data stream, the recovered data will be sent to the application process.

1) Redundancy Detection: The clone of a mobile user has a list of applications that it should conduct TRE against in order to increase the benefit of TRE. We will introduce how to determine these applications for a user in Section III-B. When the clone receives data from an application in this list, it conducts redundancy detection, which includes the following steps.

Chunk partitioning: The clone partitions the application data into chunks with variable sizes by using the MAXP algorithm [27] to find chunk boundaries. By searching the relative maximum value in a range of bytes rather the fixed content or position, MAXP can find much more redundant chunks in the flow with lower computation cost.

Hash computing: After the data is partitioned into chunks, the clone uses a hash function such as SHA1 [22] to compute a 20-byte hash.

Chunk lookup in the cache: As shown in Figure 2, the clone maintains a hash table that stores the previously partitioned chunks and their offsets in the clone's cache. For each chunk of the received data from an application server, if the chunk does not exist in the hash table, it stores the chunk to its own cache and enters the chunk to its hash table along with its offset in its cache. Otherwise, it replaces the chunk with the offset before forwarding the received data to the client. Then, the client can fetch the corresponding chunk from its own cache based on the offset and restores the data.

An offset with 4 bytes can indicate 2^{32} chunks, which is large enough to indicate all the chunks in the cache. In the receiver-based TRE methods [18–21], a duplicated chunk is replaced by its SHA1 hash value, which takes 20 bytes. Thus, by transmitting the chunk's offset instead of the chunk's hash, TailoredRE reduces the transmission overhead to 20%.

Hash collision (i.e., two different chunks may have the same hash) may result in incorrect data recovery at the client. In order to avoid this problem, we can compare the chunks after their hashes are matched. Only when both the hashes and chunks match to each other, the clone ensures that a chunk is redundant and replace the chunk with its offset.

2) Chunking Algorithm: TailoredRE divides the payload of a packet into chunks by a content-based chunking algorithm [20, 22, 27–29]. 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. TailoredRE chooses MAXP [27] because it provides uniformly distributed chunk boundaries across the payload and imposes a lower bound on chunk length and low computational overhead. It can find the local maxima in a range of bytes, without hashing computation. MAXP selects a position as chunk boundary if its value is the maximum (or minimum) over the p-byte region centered at that position. The packet payload is divided into chunks by these boundaries. 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 [30]. The determination of chunk size should consider the trade-off between prediction overhead and bandwidth savings.

B. Personalized and Application-adaptive TRE

Different from previous TRE methods, TailoredRE clone provides personalized TRE, for its smartphone user. That is, each clone selectively chooses applications to conduct TRE based on its mobile user's regular application usage and the different redundancy hit ratios of different applications, so that the benefit of TRE can be increased.

Considering the limited cache size in smartphones, we aim to fully utilize the cache to increase the redundancy hit ratio to eliminate redundant traffic as much as possible. To this end, the clone of a mobile user tries to give higher priority to the data of the applications that have higher redundancy hit ratios and that the mobile user access data more frequently to be stored in the clone's cache. Thus, there are two important metrics that a user's clone analyzes and profiles for the user. The first metric is the redundancy hit ratio distribution over different mobile applications, and the second one is the data transmission volume distribution over the applications of a mobile user.

Offset	Hash	Chunk
1	ksjkd	0x8281891
2	jhsljd	0x8923829

Fig. 2. Cache table in clone.

When a user registers for the TailoredRE service, the TailoredRE client will report to its clone the port number each app uses, by which the clone can tell which traffic belongs to which app. To measure the first metric, the clone records applications user k uses, and maintains an application vestor $U_k = \{app_1, app_2, ..., app_n\}$, along with redundancy hit ratio for app_i ($i = 1, 2, ..., n$), denoted by r_i. $r_i = \frac{V_{hit_i}}{V_{total_i}}$, where V_{total_i} is the volume of the total transmitted data from application i and V_{hit_i} is the volume of the data that hit the clone's cache during a time window T. The clone periodically updates r_i after each T. TailoredRE has a supervisor in the cloud that manages global operations among clones. Each clone reports its measured redundancy hit ratio metric for different applications to the supervisor. The supervisor calculates the average redundancy hit ratio metric for each application and sends the average values to all clones. These global average values can be used for mobile users who first use this TailoredRE service to determine the applications to conduct TRE against (called target applications).

A user's preference or frequency of using different applications can be measured by the volume of data that the user fetches from the applications. Then, we define another metric called a user's activity degree on application i, denoted as a_i, and measure it by the relative data volume of application i, that is, $a_i = \frac{d_i}{\sum d_i}$, where d_i is the data volume of each application in the application vector.

With these two metrics, for application i, the clone calculates the benefit factor, $v_i = r_i \cdot a_i$. It reflects the benefit when the clone conducts TRE against application i for its mobile user. The application with higher redundancy hit ratio and higher accessing activity degree has higher benefit factors, and then has a higher probability to be selected as a target application. We set a threshold of the benefit factor, v^{th}. In selecting the target applications, only when an application's benefit factor is larger than v^{th}, the clone puts it into the target application list and cache its chunks. Since users' preferred applications may change overtime, the target applications are updated periodically (e.g., every month). With the application-adaptive TRE design, our TailoredRE system can always conduct TRE against the applications with highest TRE benefit factor, and improves the redundancy hit ratio.

C. Cache Sharing Among Clones

Several previous studies [18, 31–33] show that there are two types of redundancy exhibited in network traffic, i.e., intra-user temporal redundancy and inter-user redundancy. Intra-user temporal redundancy means the redundancy occurs when a user accesses the same website repeatedly. Inter-user redundancy occurs when a group of users frequently access the same websites or applications due to their similar interests [23].

Each clone has a cache that is synchronized with the client's cache. Then, it requires tremendously high cache resource with a large number of users (e.g., thousands, millions). In order to reduce the total cache consumption in the cloud, we propose to share the cache resource between clones. However, there are two challenges we need to consider: i) how to group the clones in order to make the cache sharing efficient, and ii) how to share cache in the cloud while keeping the cache in cloud and the cache in the smartphone synchronized. To handle the two challenges, we propose a two-step clone grouping algorithm and a two-layer cache sharing strategy.

1) Two-Step Clone Grouping Algorithm: Only the clones that are i) on the same VM and ii) share similar application interests are grouped into a cluster. Factor i) is to ensure that the chunk sharing between the clones in a group can be efficiently conducted since the clones share the cache space. The cache sharing between the clones located in different VMs will incur large latency and bandwidth overhead within a physical machine or between physical machines. Factor ii) is to ensure that the clones in one group tend to access the same chunks and only one copy is needed to stored in the group

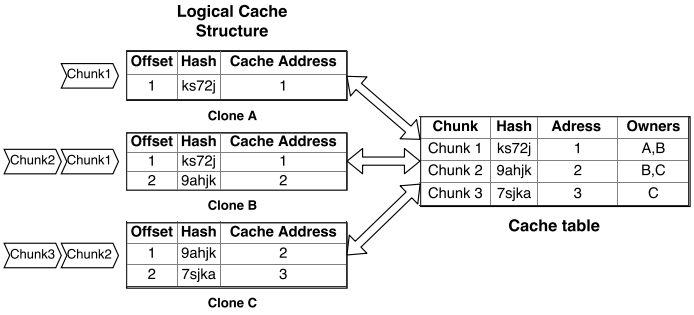


Fig. 3. Cache sharing among clones.

to be shared by the clones, which can save cache resource consumption.

TailoredRE's supervisor in the cloud also manages clone grouping. When a smartphone client k registers for the TailoredRE, (s)he is asked to enter the frequently accessed (i.e., preferred) applications to create U_k initially. Based on these applications, its clone calculates its hashed interest ID using min-wise independent permutation based locality sensitive hash (LSH) scheme [34]. This scheme can quickly and efficiently cluster similar data items together with low overhead. We use the Jaccard set similarity measure [35]: $sim(U_{k_1}, U_{k_2}) = \frac{|U_{k_1} \cap U_{k_2}|}{|U_{k_1} \cup U_{k_2}|}$. As a result, the clients with similar preferred applications will have close interest IDs. Later on, the clone updates the interest ID based on the client's actual behavior on application access. Suppose the hash ID space is $[1, L]$. We evenly slit the ID space to a number of ID range, and put the clones with IDs in the same ID range into a group. The supervisor maintains a list of the clone groups and the members in each clone group. After a newly joined clone calculates its interest ID, it contacts the supervisor, and then receives the clone group where it should join and the members of the clones in the group that it should group with. If a client un-registers the TailoredRE, his/her clone will be removed from the cloud and accordingly, the supervisor removes it from the clone group list. Below, we introduce how to calculate the interest ID for a list of applications.

The similarity between users' application preferences can be calculated by the similarity between user's application vectors. In order to make the grouping operation fast, we use an efficient locality sensitive hash function, called min-wise independent permutation based LSH scheme [34]. We use this hash function on each application vector, and then similar application vectors will have close hash values, which facilitates grouping clones with similar application vectors. We use the Jaccard set similarity measure [35]: $sim(U_{k_1}, U_{k_2}) = \frac{|U_{k_1} \cap U_{k_2}|}{|U_{k_1} \cup U_{k_2}|}$. For example, the similarity is $\frac{3}{5} = 0.6$ between U_{k_1}=CNN | YouTube | Quora | Facebook and U_{k_2}=CNN | YouTube | Quora | Twitter.

2) Two-Layer Caching Sharing Strategy: As the clones are on the same VM, they use the same cache space. In this case, when two clones receive the same chunk, only one copy of the chunk needs to be stored in the cache in order to save the cache space consumption. In TailoredRE, the cache in a client's clone is synchronized with the cache in the client to improve the redundancy hit ratio. However, sharing cache among clones (maintaining only one chunk copy in the clone group) will break such synchronization.

In order to solve this problem, we design a two-layer cache sharing strategy, in which there are two kinds of caches as shown in Figure 3: logical cache for individual clones and physical cache for all the clones in one cluster. The information of each chunk accessed by each clone, including chunk offset in the client's cache, chunk hash and chunk address in the physical cache, are stored in individual logical cache. The real chunks accessed by all clones are stored in the shared physical cache. The logical cache is used to keep it synchronized with the cache in the client, while the physical cache stores the raw chunks received from applications. The supervisor maintains a cache table to show the information of the chunks in the shared physical cache, as shown in the right part of Figure 3. For each chunk, it records its physical address in the shared physical cache, its hash value, the chunk itself, and the clones that the chunk belongs to. For example, for chunk 1, its entry stores the clone IDs of clone A, clone B, and clone C that the chunk belongs to. Each operation in the logical cache, such as chunk insertion and chunk deletion, will incur the chunk update in the physical cache. We present the operations in each clone's logical cache and in the cluster's physical cache below.

Chunk insertion: If the received chunk does not exist in a clone's logical cache, the chunk may or may not exist in the physical cache. First, the clone creates the entry of this chunk in its logical cache, and checks whether the chunk is already in the physical cache inserted by other clones. If the chunk already exists, the physical cache adds this clone's ID into this chunk's owner list. Otherwise, this chunk is inserted into the physical cache and the cache table is also updated.

Chunk deletion: When the clone finds that a chunk is a new chunk, if the clone's logical cache is full, the least recently used chunk in the clone's logical cache will be removed, according to LRU caching scheme [36]. The logical cache finds the chunk in the cluster's physical cache according to its address and checks its owner list. If there are other clones in the list, the supervisor removes this clone from the owner list, otherwise the supervisor removes the chunk entry.

IV. Experimental Simulations

In this paper, we developed a pair of clone (sender) and client (receiver) using Java to simulate our TailoredRE system. All of our simulation experiments are driven by real traces collected by ourselves.

We measured the following metrics in our experiment:

- Redundancy hit ratio. It is calculated by $\frac{V_{hit}}{V_{total}}$, where V_{total} is the total number of bytes (i.e., volume) of chunks of accessed data by the client and V_{hit} is the total number of the bytes of the chunks that hit the cache by the client.
- Bandwidth saving ratio. It is calculated by $\frac{V_{hit} - V_{overhead}}{V_{total}}$, where $V_{overhead}$ is the data volume of chunk information including hashes, offsets and shims, incurred by TRE operation and V_{total} is the size of the total transmitted content data from the sender to the receiver.
- Normalized throughput. It is computed by $\frac{r_{RE}}{r_{No-RE}}$ in order to show the final throughput improvement caused by RE, where r_{RE} and r_{No-RE} are the throughputs with TRE and

TABLE I
TRACE POOL

Number	Traces	Size (GB)	Redundancy hit 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%

without TRE respectively. The throughput is the amount of data transmitted successfully from the cloud to the clients, measured in bits per second (bps).

A. Trace Collection and Analysis

1) Trace Collection: We conducted our statistical data analysis on the payloads of packets from real wireless traces. To perform our data collection, we setup a Wi-Fi hotspot (i.e., AP) from a Windows 10 laptop equipped with 802.11ac Wi-Fi cards through an application called mHotSpot [37]. We then captured all the traffic corresponding to an AP connection through Wireshark [38], a packet sniffing tool. We connected a single user to the hotspot and generated traffic by browsing into various popular multimedia and news applications. Then, we segregated packet traces of various applications based on their IP addresses and port numbers using Wireshark's filtering tool [38]. We preprocessed the traffic traces by stripping the packet headers starting from physical, MAC, IP and transport layer headers of each packets.

We used a smartphone to continuously access 18 applications to create two traces. In the first trace, each application trace takes 98 hours to collect, and in the second one, each application trace takes 10 hours to collect. Some of the traces, such as Twitter and Facebook, are HTTPS traffic. We use the first trace to simulate the performance of one clone, that is, it accesses fewer applications and each has a large data size (10GB-28GB). We use the second trace to test the caching sharing method among a large amount of clones (i.e., 1000 in total). For more details of the traces, please refer to [39].

2) Trace Analysis: We have run our simulator to analyze the redundancy hit ratios of all the application data traces. In the MAXP algorithm, we set the sampling parameter, p, to 256 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.

The redundancy hit ratios for different data traces are shown in Table I. We see that different applications indeed have different redundancy hit ratios. First, these results confirm that there exists traffic redundancy in smartphone accessed data [40]. Second, we see that some application data, such as Web Browser and Techcrunch traces, have higher redundancy hit ratios, while others, such as Facebook and Twitter, have lower redundancy hit ratios because of HTTPS encryption. The observations lay the foundation of the design of the application-adaptive TRE in TailoredRE.

B. Trace-driven Simulation for RE Effectiveness

We first evaluate TailoredRE on one smartphone client and then on multiple smartphone clients. The access probability was set to 0.4, 0.2, 0.2, 0.1 and 0.1 for the applications. By this way, we can simulate the activity degrees of applications for a certain user. We compared TailoredRE with EndRE [22] and Asymmetric Caching (AC) [19].

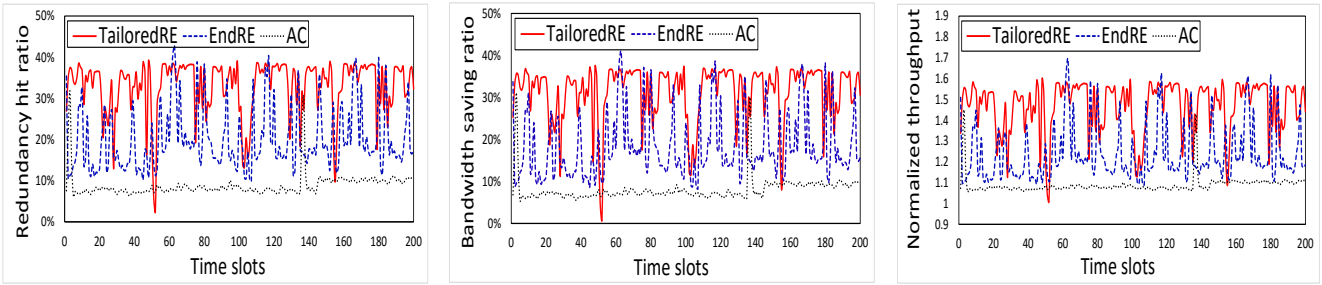
1) Performance Over Time: In this simulation, the cache size was set to 2.5MB and there was no cache sharing module since we only want to measure the effectiveness of redundancy elimination. The client requested for 1MB data block each time. We generated 100 requests in each time slot for 200 time slots.

From Figure 4(a), we see that in every time slot, the redundancy hit ratio conforms TailoredRE>EndRE>AC. The reason is that compared with the EndRE, TailoredRE always chooses the applications with high redundancy hit ratio and activity degree to conduct TRE against. Thus, TailoredRE can more efficiently use the cache and generates higher cache hit ratio than EndRE. Compared with EndRE, in AC, chunks from different servers are stored in different chunk chains. The data stream from a specific server will be checked only in the chain of the server rather than in all existing chains. As a result, with the same cache size at the receiver, AC can cache fewer different chunks compared with EndRE, which reduces hit ratio. In addition, because of the asynchronization between the sender and receiver, AC's redundancy hit ratio is lower than EndRE's.

Figure 4(b) shows that at each time slot, the bandwidth saving ratio also conforms TailoredRE>EndRE>AC. In TailoredRE and EndRE, the sender only needs to replace redundant chunk with an offset (4 bytes) in data transmission. In AC, the receiver needs to send hashes of the predicted chunks to the sender, which needs to replace the correctly predicted chunks with hashes. Each hash value takes 20 bytes, which is much larger than the offset in TailoredRE and EndRE. As a result, AC consumes much more bandwidth in data transmission. Moreover, in AC, the receiver needs to send back predication messages, which contain the future chunk hashes, to the sender and consumes extra feedback overheads. As a result, TailoredRE achieves the highest bandwidth saving, and AC produces the lowest bandwidth saving.

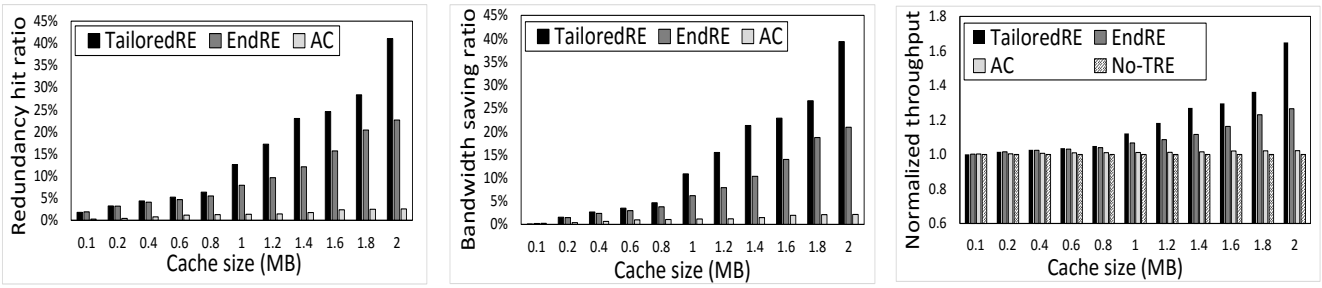
Figure 4(c) shows that the normalized throughput also conforms TailoredRE>EndRE>AC. This is because that TailoredRE can eliminate the most redundant chunks during data transmission, and reduce the highest transmission time, so that it improves the throughput better than other methods. Compared with AC, EndRE has a higher redundancy hit ratio, which reduces more redundant data during transmission, and hence reduces higher transmission time. As a result, EndRE can achieve higher throughput than AC.

In addition, from Figure 4, we see that the results of TailoredRE are stable, but the results of EndRE and AC fluctuate. The reason is that, the cache in TailoredRE just



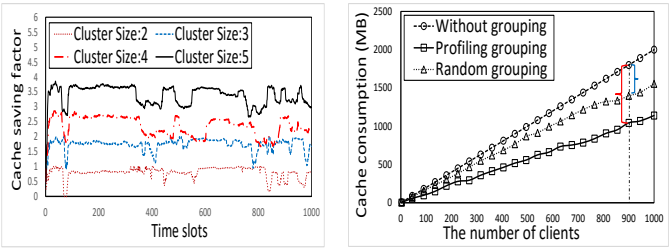
(a) Redundancy hit ratio (b) Bandwidth saving ratio (c) Normalized throughput

Fig. 4. Performance over time.



(a) Redundancy hit ratio (b) Bandwidth saving ratio (c) Normalized throughput

Fig. 5. Performance with different cache sizes.



(a) Cache saving over time (b) Cache consumption with different numbers of clients

Fig. 6. Performance of cache sharing among clones.

caches the chunks from the applications with high benefit factor, there are few chunk replacements among different applications. However, in EndRE and AC, the cache can be populated by the chunks from applications with low benefit factor, which causes performance fluctuation.

2) Performance with Different Cache Sizes: We measured the redundancy hit ratio, bandwidth saving ratio and normalized throughput with cache size changing from 0.1MB to 2MB. The results are respectively shown in Figure 5(a), Figure 5(b) and Figure 5(c). We see that the results of each metric are consistent with those in the previous subsection due to the same reasons. In addition, we see that as the cache size increases, the redundancy hit ratios, bandwidth saving ratios and normalized throughputs of the three systems increase. This is because with a larger cache size, more chunks can be cached and the chunks in transmission have higher probabilities to hit the cache.

C. Simulation of Shared Caching For Multiple Clones

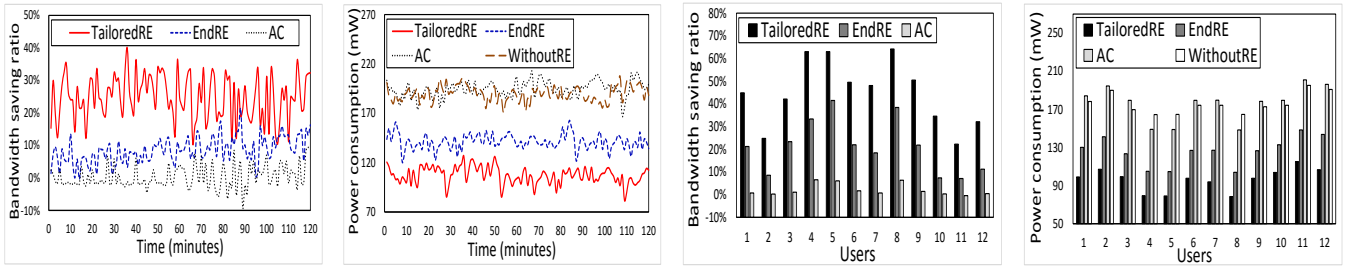
At the initial phase, each user randomly chose 4 applications from the trace pool and their access probabilities are $\{p_1, p_2, p_3, p_4\}$, where p_1, p_2, p_3 and p_4 are randomly chosen and subject to $\sum_{i=1}^{4} p_i = 1$. In the MAXP algorithm, we set the sampling parameter, p, to 512 bytes. We set the cache sizes at the receivers to 2MB and generated once request in each time slot for 1000 time slots in this simulation.

In this simulation, we first created a clone, say clone i. Then, we created other clones in the cluster that have similarity 0.4 with clone i and these clones are grouped with clone i into a cluster. The cache saving factor is computed by $\frac{\sum C_{clone} - C_{cluster}}{C_{clone}}$, where $C_{cluster}$ denotes the total cache consumption (i.e., cache size) in one cluster and C_{clone} denotes the cache consumption for one clone. The simulation results are shown in Figure 6(a).

Figure 6(a) shows one cluster's cache saving factor as a function of time for the cluster with different cluster size. We see that with the increase of the cluster size, the cache saving factor increases. Since more clones sharing the cache will incur more common chunks in the cache, which cause higher cache saving factor.

In addition, we simulate the relationship between the average total cache consumption at the cloud side with the number of clients changing from 1 to 1000. We assume that all the clones are located in the same VM and just consider the user interest for clone grouping. For each time slot, we collected the total cache consumption of all the clusters in the cloud, and finally we calculated the average of the total cache consumption per time slot. We compare our method with both the randomly grouping method and the method without clone grouping, and the simulation results are shown in Figure 6(b).

Figure 6(b) shows that for a certain number of clients, the aggregate cache consumption with our cache sharing algorithm is always less than that with random grouping, and the cache consumption without grouping always is the highest among the three methods. The cache reduction can be expressed by $\frac{C_{w/o-grou} - C_{w/-grou}}{C_{w/-grou}}$, where $C_{w/o-grou}$ and $C_{w/-grou}$ represent the



(a) Bandwidth saving (b) Power consumption (c) Average bandwidth saving (d) Average power consumption

Fig. 7. prototype-based experimental results.

total cache consumption without clone grouping and with clone grouping. Then, our proposed cache sharing method can reduce cache consumption of the method without grouping by more than 40%, while random grouping can reduce the cache consumption of the method without grouping by about 20%. Our proposed grouping method considers the user application usage similarity, thus the cached chunks between two clones in the same group will have much more common chunks and share much more cache resource in the shared cache. We can also see from Figure 6(b) that with the increasing of the number of clients, the cache resource consumption increase linearly. Also, the cache consumption reduction of our method and of the random grouping method increases as the number of clients increases. The results imply that when there are a larger number of clients, our cache sharing method can reduce more cache resource consumption in the cloud.

V. Prototype-based Performance Evaluation

A. Experiment Setup

We developed the Python-based TailoredRE clone and Android-based TailoredRE client prototypes, and open-sourced TaioredRE in GitHub [26]. We developed a test app which mimics the real app to send out real application data requests to the TailoredRE client directly. We deployed our TailoredRE clone in Amazon's EC2 virtual server instance powered by Ubuntu 14.04 LTS operating system. TailoredRE client ran on a smartphone named Asus Fonepad powered by Android Version 4.1.2. We also conducted an online survey about user's application preferences in [41]. In this survey, 12 users on campus are randomly invited to select and enter the applications they preferred and the time they spend regularly on the applications in one week, which can roughly reflect users' daily application usage. We conducted our prototype-based experiments on the bandwidth savings and the power consumption with TailoredRE based on the users' application preferences from the survey. Power measurement includes the power used for CPU and Wi-Fi components caused by the test applications and the TRE method. We also implemented EndRE and AC for performance comparison.

In the MAXP algorithm, we set the sampling parameter, p, to 256 bytes. The cache size for the client was set to 2MB. We ran each experiment for 2 hours. In each minute, the Android-based test applications sent 50 data requests and receive 50 corresponding data responses in total. The data request probability for each application is based on the user entered times from the survey.

B. Prototype-Based Experiments Results

In this part, we first randomly picked one user among all the users in the survey. We show this user's application preference in Figure 8. Figure 7(a) and Figure 7(b) show the bandwidth saving ratio and power consumption for each minute over time for this user, respectively. From Figure 7(a), we see that the bandwidth saving ratio conforms TailoredRE¿EndRE¿AC, which is in coordination with the simulation results in Figure 4(b) due to the same reasons. We can see from Figure 7(b) that, the power consumption conforms TailoredRE<EndRE<WithoutRE<AC. Both TailoredRE and EndRE consume considerably lower power compared with the case without TRE, since TRE causes less bandwidth consumption and less transmitted data, thus it reduces the power consumption caused by data transmission. TailoredRE consumes lower power than EndRE, because TailoredRE has higher bandwidth savings and higher reduction on the amount of transmitted data. It is interesting to see that AC has slightly higher power consumption than the case without TRE. The reason is that AC conducts TRE operations, such as chunk partitioning and hashing, which incur extra computation and CPU consumption in the smartphone. Additionally, AC needs to send back feedback messages to the sender, which incurs extra wireless radio power consumption for the smartphone.

User application preference

Applications	Time in one week (h)
YouTube	0.5-2
Facebook	0
CNN	2.0-4.0
Quora	0.5-2.0
Techcrunch	2.0-4.0
NYTimes	0.5-2.0
Twitter	2.0-4.0
Instagram	2.0-4.0
Spotify	2.0-4.0
Bloomberg	0.5-2.0
Browser	0.5

Fig. 8. User application preference.

We then tested the bandwidth saving ratio and power consumption for all of the 12 users in the survey by configuring the test applications based on their application preferences. Figure 7(c) and Figure 7(d) respectively show the average bandwidth saving ratio and the average power consumption in two hours for each user in the survey. From Figure 7(c), we see that different users, with different application preferences, have different bandwidth saving ratios, which indicate the necessity to conduct personalized TRE for individual users. In addition, for each user, the total bandwidth saving ratio conforms TailoredRE¿EndRE¿AC. Specifically, TailoredRE conserves 15% – 29.7% more bandwidth compared with EndRE, and 24.5%–57.8% more bandwidth compared with AC. The reason for the result is the same to that in Figure 4(b). We can see from Figure 7(d) that with different user preferences, the

power consumption of different smartphones are different. In addition, for each user, both TailoredRE and EndRE consume lower power than AC and the case without TRE. Moreover, TailoredRE consumes much lower power than EndRE, for the same reason as in Figure 7(b). Specifically, TailoredRE conserves 14.1% – 19.3% more energy compared with EndRE, and 42.3% – 49.2% more energy compared with AC.

VI. Conclusion

In this paper, we proposed the TailoredRE system to provide TRE service for smartphone users in accessing Internet applications. Different from previous TRE methods, TailoredRE conducts the TRE against selected applications that a user more frequently accesses and have high redundancy hit ratios for each individual user. In this way, the limited cache resource can always be better utilized to improve the redundancy hit ratio and hence achieve higher TRE. In addition, in order to conserve the cache resource in the cloud, TailoredRE groups the clones in one VM based on user interests, and let the clones share contents with each other. Trace-driven simulation shows that our TailoredRE system greatly improves the bandwidth savings and cache resource saving. Furthermore, prototype-based experiments show that TailoredRE can conserve 14.1% – 19.3% more energy compared with EndRE, and 42.3% – 49.2% more energy compared with AC. We open-sourced our prototyped TailoredRE.

Acknowledge

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751.

References

[1] S. He, Z. Lu, X. Wen, Z. Zhang, Y. Sun, and L. Zhang, "Energy-efficient power allocation with QoS guarantee in OFDMA wireless networks," in *Proc. of WPMC*. IEEE, 2014.

[2] S. He, Z. Lu, X. Wen, Z. Zhang, J. Zhao, and W. Jing, "A pricing power control scheme with statistical delay QoS provisioning in uplink of two-tier OFDMA femtocell networks," *Mobile Networks and Applications*, vol. 20, no. 4, pp. 413–423, 2015.

[3] Y. Zhang and N. Ansari, "On protocol-independent data redundancy elimination," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 455–472, 2014.

[4] Z. Li, W. Wang, T. Xu, X. Zhong, X.-Y. Li, Y. Liu, C. Wilson, and B. Y. Zhao, "Exploring cross-application cellular traffic optimization with baidu trafficguard." in *NSDI*, 2016, pp. 61–76.

[5] E. Halepovic, C. Williamson, and M. Ghaderi, "Enhancing redundant network traffic elimination," *Computer Networks*, vol. 56, no. 2, pp. 795–809, 2012.

[6] K. A. Hua, N. Jiang, J. Kuhns, V. Sundaram, and C. Zou, "Redundancy control through traffic deduplication," in *Proc. of INFOCOM*, 2015.

[7] H. Shen, S. He, L. Yu, and A. Sarker, "Prediction-based redundant data elimination with content overhearing in wireless networks," in *Proc. of PerCom*, 2017.

[8] A. Sarker, C. Qiu, and H. Shen, "Quick and autonomous platoon maintenance in vehicle dynamics for distributed vehicle platoon networks," in *Proc. of IoTDI*, 2017.

[9] A. Panda, O. Lahav, K. J. Argyraki, M. Sagiv, and S. Shenker, "Verifying reachability in networks with mutable datapaths." in *NSDI*, 2017.

[10] K. Matsuzono, H. Asaeda, and T. Turletti, "Low latency low loss streaming using in-network coding and caching," in *IEEE INFOCOM*, 2017.

[11] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, and G. Zhang, "Accurate recovery of internet traffic data: A tensor completion approach," in *IEEE INFOCOM*, 2016.

[12] A. Sarker, C. Qiu, and H. Shen, "A decentralized network with fast and lightweight autonomous channel selection in vehicle platoons for collision avoidance," in *Proc. of MASS*, 2016.

[13] Cisco. (2018, April) Cisco Wide Area Application Acceleration Services. [Online]. Available: http://www.cisco.com/en/US/products/ps5680/Products_Sub_Category_Home.html

[14] Riverbed. (2018, April) Riverbed Networks: WAN Optimization. [Online]. Available: https://www.riverbed.com/

[15] Y. Song, K. Guo, and L. Gao, "Redundancy-aware routing with limited resources," in *Proc. of ICCCN*, 2010.

[16] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *Proc. of SIGCOMM*, 2008.

[17] A. Anand, V. Sekar, and A. Akella, "SmartRE: an architecture for coordinated network-wide redundancy elimination," in *Proc. of SIGCOMM*, 2009.

[18] E. Zohar, I. Cidon, and O. O. Mokryn, "Celleration: loss-resilient traffic redundancy elimination for cellular data," in *Proc. of HotMobile*, 2012.

[19] 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.

[20] E. Zohar, I. Cidon, and O. Mokryn, "PACK: Prediction-based cloud bandwidth and cost reduction system," *IEEE Transactions on Networking*, vol. 22, no. 1, pp. 39–51, 2014.

[21] S.-H. Shen, A. Gember, A. Anand, and A. Akella, "REfactoring content overhearing to improve wireless performance," in *Proc. of MobiCom*, 2011.

[22] B. Agarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: An end-system redundancy elimination service for enterprises," in *Proc. of NSDI*, 2010.

[23] R. Keralapura, A. Nucci, Z.-L. Zhang, and L. Gao, "Profiling users in a 3G network using hourglass co-clustering," in *Proc. of MobiCom*, 2010.

[24] D. J. A. Fast and B. Levine, "Creating social networks to improve peer-to-peer networking," in *Proc. of KDD*, 2005.

[25] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. of EuroSys*, 2011.

[26] V. Soundararaj. (2018, April) TailoredRE. [Online]. Available: https://github.com/papersubmit/MASS2018.git

[27] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proc. of SIGMETRICS/Performance*, 2009.

[28] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proc. of SIGCOMM*, 2000.

[29] F. R. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. G. Andersen, "Ditto: a system for opportunistic caching in multi-hop wireless networks," in *Proc. of MobiCom*, 2008.

[30] A. B. N. Bjorner and Y. Gurevich, "Content-dependent chunking for differential compression, the local maximum approach," Microsoft Research, Tech. Rep. 109, 2007.

[31] Z. Zhuang and R. Sivakumar, "Wireless memory: Eliminating communication redundancy in wi-fi networks," in *Proc. of WoWMoM*, 2011.

[32] C. Lumezanu, K. Guo, N. Spring, and B. Bhattacharjee, "The effect of packet loss on redundancy elimination in cellular wireless networks," in *Proc. of SIGCOMM*, 2010.

[33] E. Halepovic, M. Ghaderi, and C. Williamson, "On the performance of redundant traffic elimination in wlans," in *Proc. of ICC*, 2012.

[34] H. Shen, Y. Lin, and T. Li, "Combining Efficiency, Fidelity, and Flexibility in Resource Information Services," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 353–367, 2015.

[35] Y. Chen, W. He, Y. Hua, and W. Wang, "CompoundEyes: Near-duplicate detection in large scale online video systems in the cloud," in *Proc. of INFOCOM*, 2016.

[36] A. Gharaibeh, A. Khreishah, and I. Khalil, "An O(1)-competitive online caching algorithm for content centric networking," in *Proc. of INFOCOM*, 2016.

[37] mHotSpot. (2018, April) mHotSpot. [Online]. Available: http://www.mhotspot.com/

[38] Wireshark. (2018, April) Wireshark. [Online]. Available: https://www.wireshark.org/

[39] "TailoredRE: Cloud assisted social network property based traffic redundancy elimination for smartphones," Tech. Rep. [Online]. Available: https://1drv.ms/b/s!Aie-NDW91wHkbJ9ihnqv41Zw0xI

[40] L. Yu, H. Shen, K. Sapra, L. Ye, and Z. Cai, "Core: cooperative end-to-end traffic redundancy elimination for reducing cloud bandwidth cost," *TPDS*, vol. 28, no. 2, pp. 446–461, 2017.

[41] Preference. (2018, April) The survey of user preference. [Online]. Available: http://goo.gl/forms/LZ4YM5ZWzZ