EdgeBuffer: Caching and Prefetching Content at the Edge in the MobilityFirst Future Internet Architecture

Feixiong Zhang*, Chenren Xu†, Yanyong Zhang*, K. K. Ramakrishnan‡,
Shreyasee Mukherjee*, Roy Yates*, Thu Nguyen*
*WINLAB, Rutgers University, North Brunswick, NJ, USA
†Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
‡Department of Computer Science, University of California, Riverside, CA, USA

***Abstract*—The prevalence of mobile devices especially smartphones has attracted research on mobile content delivery techniques. In this paper, we propose to take advantage of the storage available at wireless access points to bring content closer to mobile devices, hence improving the downloading performance. Specifically, we propose to have a separate popularity based cache and a prefetch buffer at the network edge to capture both long-term and short-term content access patterns. Further, we point out that it is insufficient to rely on a device's past history to predict when and where to prefetch, especially in urban settings; instead, we propose to derive a prediction model based on the aggregated network-level statistics.**

We discuss the proposed mobile content caching/prefetching method in the context of the MobilityFirst future Internet architecture. In MobilityFirst, when mobile clients move between network attachment points (e.g., Wi-Fi access points), their network association records are logged by the network, which then naturally facilitates the network-level mobility prediction. Through detailed simulations with real taxi mobility traces, we show that such a strategy is more effective than earlier schemes in satisfying content requests at the edge (higher cache hit ratios), leading to shorter content download latencies. Specifically, the fraction of requests satisfied at the edge increases by a factor of 2.9 compared to a caching only approach, and by 45% compared to individual user-based prediction and prefetching.

I. Introduction

We are currently witnessing tremendous growth in the access of content using mobile devices. In fact, Cisco Visual Networking Index [1] reports that global mobile data traffic reached 1.5 exabytes per month at the end of 2013, and global mobile devices and connections in 2013 grew to 7 billion. The rapid rise of mobile content access has led to a dense deployment of wireless networks especially in urban environments, including WiFi access points (APs) and cellular base stations (macro cells and small cells). Users will now traverse multiple access points (AP) and base stations, encountering frequent network connection transitions within a session (e.g., watching a YouTube video), which in turn makes mobile content delivery while maintaining the user's seamless quality of experience, a daunting task.

The current TCP/IP Internet architecture is poorly suited for content delivery to mobile devices in this manner. A common criticism is the so-called identity-location conflation problem [2], where an IP address in the Internet is used to both identify an interface as well as its network location. As a result, connections break when an endpoint changes network addresses, requiring application-layer workarounds to provide the mobility support. Several future Internet architecture designs have been proposed to refactor naming, addressing and routing for location-independent communication [3], [4], [5], [6]. In this paper, we consider MobilityFirst [3], [7], a mobility-centric architecture that supports large-scale, efficient and robust network services with mobility as the norm. In MobilityFirst, we associate every endpoint with a global unique identifier (GUID), which is decoupled from its network address or location, and rely on a dynamic global name resolution service (GNRS), such as those in [8], [9], [10], [11], to track and resolve the mapping between GUID and its network address(es). MobilityFirst aims to support smooth mobile content delivery exploiting real-time GNRS updates and queries.

The key to mobile content delivery is to ensure seamless delivery when devices move between networks. In MobilityFirst, a mobile device updates GNRS with its latest network address when it moves to (i.e., connects to) a new access point (AP). We envision that these APs, when equipped with storage, can be utilized as a distributed content caching system (similar to the concept of PacketCloud [12] and EdgeClouds [13]), such that a mobile client downloads chunks of content from APs it connects to over time, thus avoiding reaching out to the original content server. The download performance (i.e., latency, throughput) can be greatly improved when the content chunks are available locally at the AP [12], [13], [14], [15]. However, achieving the above goal is not trivial: we need to determine what chunks should be placed at which AP(s) prior to the client's arrival at that AP. There is a need to both anticipate what content users may request (e.g., popularity) as well as take advantage of the individual user's current content access pattern. To effectively achieve this goal, we propose an *EdgeBuffer* framework in which an AP's storage is separated into cache buffer and prefetch buffer, with the former caching popular content chunks while the latter buffering chunks that are likely to be accessed by individual mobile client. That is, while most popular content chunks at an AP are cached at the cache buffer, those most "urgent" content chunks that are expected to be needed for a client in the near future are prefetched and buffered at the AP's prefetch buffer. Caching popular content chunks has been extensively studied

978-1-4799-8461-9/15/$31.00 2015 IEEE

in the literature [16], [17], [18]. In this paper, we focus on prefetching content chunks for individual mobile devices, by predicting the next network the device is moving to, and at what time, and examine how best to partition the storage at the AP.

In the literature, earlier studies [15], [19] have looked at mobility prediction and content prefetching based upon personalized mobility models. These studies reported that individual mobility patterns are highly predictable in certain areas (e.g., suburban locations), due to limited and predictable human habits (e.g., daily commute) and road conditions. Though these observations are true in less populated areas, we find that they do not hold true in urban areas (e.g., San Francisco) due to more complex road networks, frequent traffic congestion, and vehicles such as taxis taking different passengers and thus not following the same route. As a result, personalized (such as daily) mobility patterns are less likely to be useful in these areas. We need to take into consideration the latest mobility information from nearby devices to make accurate predictions. Therefore, in this paper, we develop a network (access point) level mobility model based upon aggregated mobility information collected by the network, which better captures time-varying mobility patterns than personalized mobility models.

MobilityFirst is ideally suited to develop such a network-level mobility model. Since each mobile device updates its latest network address with GNRS when entering a new access network, GNRS naturally observes each device's mobility pattern. Based upon the aggregated mobility patterns from all the devices that pass the network, we build a mobility model for each network – we model user movement as a second-order Markov chain with fallback to first-order [20], and build such network transition probability table (NPT) at each AP. Given the previous AP of a mobile client, the NPT at current AP returns the probable future AP and corresponding transition probability for the client. Each AP also builds a residence time history table (RHT) by extracting statistics from recent residence times observed by the AP. An AP then estimates a particular device's residence time based upon the residence times of past clients.

In this paper, we have made the following contributions:

- We have developed a content caching and prefetching framework in which an access point has separate cache buffer and prefetch buffer for popularity-based content caching and mobility prediction-based prefetching. Our framework can capture both long-term aggregated content access pattern and short-term individual user access pattern, thus considerably improving the cache hit ratio.
- We have investigated various aspects of the caching and prefetching framework, especially on how to split the storage into a cache component and a prefetch component, and size them correctly, determined by the mobility and access patterns at the AP.
- We have developed a network-level mobility prediction model within the MobilityFirst architecture, taking into consideration latest mobility information from nearby mobile devices. This model is more accurate than prediction based upon individual device traces.

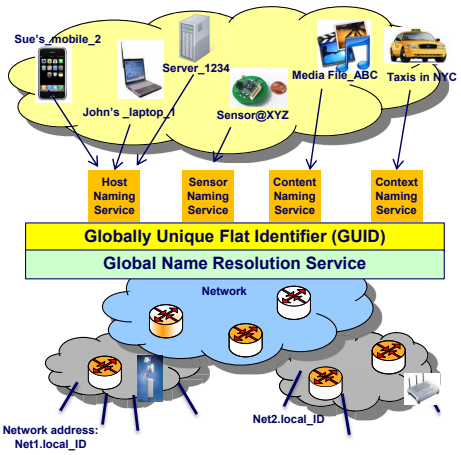


Fig. 1: Name-locator separation and mapping in MobilityFirst

We have generated an access point level user mobility workload based upon a real taxicab trace [21] and the wireless coverage database [22], and use it for simulation. Our evaluation shows that our scheme improves the overall cache hit ratio by 289% and 45% compared to caching only approach and individual user based prediction approach.

II. Background on MobilityFirst

The MobilityFirst architecture [3], [7] is built opon a name-based service layer that serves as the "narrow-waist" of the protocol stack. As shown in Figure 1, this layer uses flat globally unique identifiers (GUIDs) to name all network-attached objects including hosts, content, services and even abstract context.

A GUID can be assigned to a network object by one of multiple name certification services (NCSs), and is derived through a cryptographic hash of the public key that corresponds to that object. The GUID then works as the long-lasting network identifier for the object, and is decoupled from its network address(es).

A dynamic global name resolution service, called GNRS, manages the GUID to network address mappings and provides fast responses to mapping requests and changes, such as inserts, lookups, and updates. Such a service can be implemented in different ways, e.g., DMAP [8], Auspice [9] and DONA [11]. In DMAP, the name resolution infrastructure is designed as an in-network 1-hop DHT with servers distributed all across the Internet; Auspice uses a resolver replica placement engine that automates the placement of geo-distributed name resolvers to facilitate name resolution; DONA relies on a class of resolution handlers (RHs) deployed at each domain to translate names into locations.

Such name-based service model with GNRS naturally supports of mobile content delivery [23]: a client requests content directly by content name, and routers query the GNRS to get updated content location(s); the GNRS mapping of the client is updated whenever the client moves to a new network, thus the content can be delivered to the client correctly later.

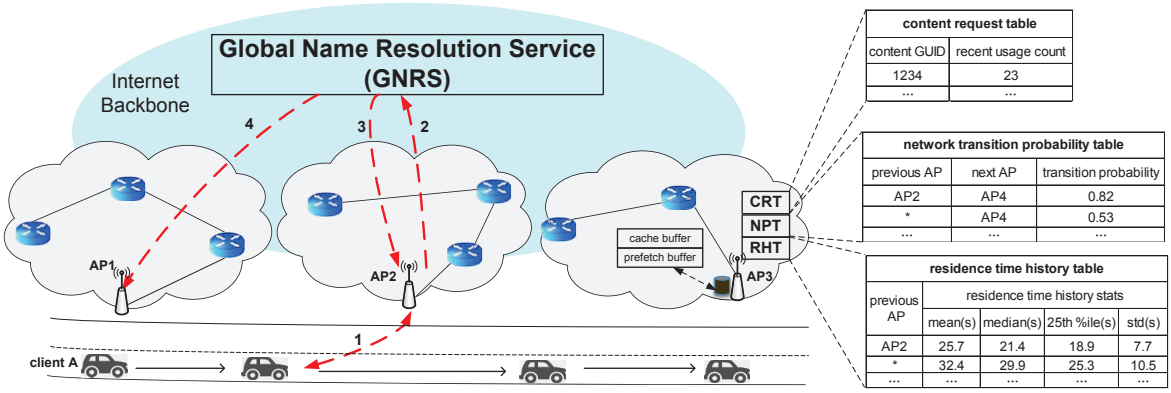


Fig. 2: EdgeBuffer: mobility prediction and content caching/prefetching framework in MobilityFirst

III. EdgeBuffer: Content Caching and Prefetching at the Edge

With the increasing deployment of wireless access points, a mobile client may achieve better connectivity to retrieve content, but at the same time need to associate with multiple access points during a content retrieval session. Caching and prefetching content chunks at these APs can reduce the content download latencies, improve the throughput, and mitigate jitter. For example, when a mobile client connects to an AP, it issues a request (with a specific content chunk number); the request can be satisfied locally if the requested chunk is already cached at the AP due to its popularity. Additionally, if the network can predict which AP the client will connect to next, and at what time, then the predicted next AP can prefetch suitable chunks before the client arrives. In both cases, the client can directly access the content from the network edge, instead of going to the hosting server or CDN. This is the main idea of EdgeBuffer, which uses the distributed storage at APs to capture long-term aggregated content access patterns (i.e., popularity based), as well as the short-term individual client's access pattern based upon the client's mobility information.

A. Popularity-based Caching

The access point can build a content request table capturing long-term, local content request patterns as shown in Figure 2. The table maintains a recent usage count (RUC) for each chunk of content that has been requested through the AP within a time window. The RUC value is managed using an explicit aging method with a periodic aging function as discussed in [16]. For a given aging parameter $\alpha(0 < \alpha < 1)$, suppose the aging period is T and there are n requests for content x in the period, the aging function is:

$$C(x)|_{t+T} \leftarrow \alpha * (C(x)|_t + n) \quad (1)$$

where $C(x)$ is the RUC value of content x. Normally, every time when a content is requested, its RUC value increments by one; periodically (i.e., every aging period), each RUC value $C(x)$ is multiplied by α.

When cache buffer is full and a new chunk arrives, the AP compares its RUC value with that of cached chunks. Cache replacement takes place when the new arriving chunk's RUC value is higher than the lowest RUC value of cached chunks.

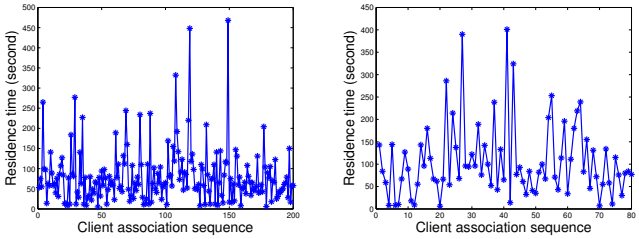


Fig. 3: Residence times for the same taxi at the same AP vary significantly.

B. Mobility Prediction and Prefetching

In addition to caching popular content chunks, we also focus on prefetching chunks for mobile clients. The key to prefetching suitable content chunks before the client arrives at an AP is accurate mobility prediction – as soon as a client associates to an AP, the AP needs to predict how long the client will stay in the current network, and which network it is moving towards.

Previous work has proposed to conduct mobility prediction based upon each individual user's mobility trace [15], [19], with the assumption that user mobility/commute trace is highly regular and predictable. However, we argue that this assumption only holds true in less populated areas, but not true in urban areas with complex road networks and frequent traffic congestions. In such urban areas, the time, as well as the route, taken to travel between two locations may vary considerably. To support our point, from the San Francisco (SF) taxicabs mobility dataset [21] that consists of 536 cabs' GPS trace, we randomly picked two cabs and plotted their residence times at a specified AP in Figure 3. Figure 3 shows that the residence time for the same cab at the same AP varies significantly in crowded areas. In this paper, we therefore investigate mobility prediction at the network/AP level using aggregated mobility traces.

1) Spatial and Temporal Prediction: Conducting mobility prediction at the network/AP level is challenging on today's Internet due to the lack of aggregated mobile network association

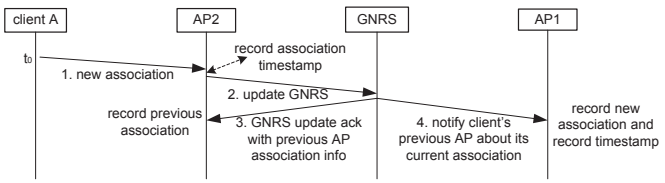


Fig. 4: Sequence diagram when client A transits to AP2 from AP1

traces. GNRS in MobilityFirst, however, naturally keeps track of each mobile client's network association trace – whenever a mobile client gets associated to a new AP/network, it updates the name resolution service about the association, thus making network-level mobility prediction feasible.

In this work, we consider mobility prediction in two orthogonal dimensions: spatial prediction (which AP will the client connect to next) and temporal prediction (how long will the client stay at the current AP). That is, spatial prediction determines where to prefetch content chunks, and temporal prediction determines what content chunks to prefetch. To facilitate these two types of prediction, we assume that each AP maintains the corresponding network association records. Each record consists of the following three fields: (1) previous AP, (2) residence time, and (3) next AP. Based upon the association records, the AP builds a network transition probability table and residence time history table, as illustrated in Figure 2.

Our spatial prediction predicts a client's next AP given the combination of its previous and current APs. The key data structure for spatial prediction is what we call network transition probability table (NPT). We build NPT from the AP's association records through a second-order Markov predictor with fallback to first-order. NPT takes the (previous AP, next AP) tuple as input, and returns the transition probability to the corresponding next AP.

Our temporal prediction predicts a client's residence time given the combination of its previous and current APs. From an AP's association records, we build its residence time history table (RHT) – we first classify the records into different bins according to the previous AP, and then calculate the statistics of residence times in each bin, including the mean, median, 25 percentile and standard deviation. If the previous AP information is not available, then the prediction falls back to look at all the residence times without binning.

Figure 2 and Figure 4 illustrate how these two tables are built using an example. In the example, client A was associated with AP1, AP2, and then AP3. After leaving AP1, it connected to AP2, and sent out an update message to GNRS through AP2. GNRS then responded to AP2 with the client's previous location (AP1 in this case), and notified AP1 with the user's current location (AP2). AP2 then logged the association record (without the next AP field) as shown in Algorithm 1, and AP1 filled the corresponding association record's next AP field with AP2, as shown in Algorithm 2. The same process took place after the client left AP2 and connected with AP3. In this way, AP2 figured out each association record's previous AP, next AP, and residence time. Using these association records, AP2 can construct its NPT and RHT.

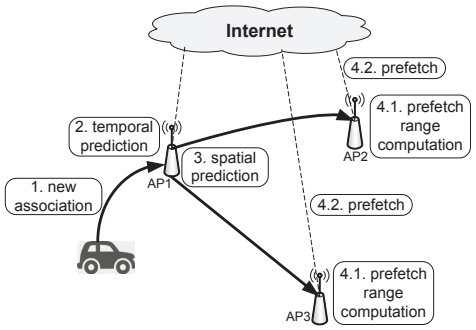


Fig. 5: Content prefetching protocol

Algorithm 1: Recording client association history

Input: GNRS update ack message M_u, association timestamp t_{assoc}
Output: client association record R_{assoc}
Data: client c, previous AP AP_{prev}

$(AP_{prev}, c) \leftarrow$rcvGNRSAckMsg$(M_u)$
R_{assoc}.**insertPreviousAP**(c, AP_{prev})
R_{assoc}.**insertAssocTime**(c, t_{assoc})

Algorithm 2: Building network transition probability table (NPT) and residence time history table (RHT)

Input: GNRS notification message M_n, the time receiving the msg t_{leave}, client association record R_{assoc}
Output: NPT, RHT
Data: client c, previous AP AP_{prev}, next AP AP_{nxt}

$(AP_{nxt}, c) \leftarrow$rcvGNRSNotiMsg(M_n)
$AP_{prev} \leftarrow$ getPreviousAP(R_{assoc}, c)
$t_{assoc} \leftarrow$ getAssocTime(R_{assoc}, c)
$\Delta t \leftarrow t_{leave} - t_{assoc}$
R_{assoc}.**remove**(c)
NPT.**update**(AP_{prev}, AP_{nxt})
RHT.**update**$(\Delta t, AP_{prev})$

C. Content Prefetching Protocol

Using NPT and RHT, an AP predicts when a mobile client will leave the current network (temporal prediction), and to which network it will move (spatial prediction). Following prediction, the predicted next network conducts content prefetching. Below we explain our prefetching protocol using the example shown in Figure 5:

1) *Association:* Let us assume that at time t_0, a client leaves AP0 and makes an association with AP1.
2) *Temporal Prediction:* As soon as the association is made, AP1 predicts the client's residence time, T_r, as the median value of the residence time samples that are

considered. Since we need to start prefetching before the estimated departure time ($t_0 + T_r$), AP1 sets T_p as the 25th percentile of the residence time samples. As such, the system will start prefetching around time $t_0 + T_p$. Please note that past residence time statistics are available in the RHT.

3) *Spatial Prediction:* At time $t_0 + T_p$, if the client is still associated with AP1, AP1 then predicts the client's next AP by looking up the NPT and picking the K (usually $K < 3$) most likely next APs (as shown in Algorithm 3). In this example, we have $K = 2$, and next APs are AP2 and AP3.
   At the same time, AP1 examines which chunk the client is going to download next at the estimated departure time ($t_0 + T_r$), which will also be the first prefetch chunk.
   Finally, AP1 sends out a *Prefetch message* to K next APs. The Prefetch message contains the following information: content ID, chunk ID, and the transition probability.
4) *Prefetching:* Suppose the Prefetch message reaches AP2 and AP3 after a short delay. Both APs predict the residence time for the client, and calculate what chunks the client will request during the estimated residence time as shown in Algorithm 4. If these chunks are not yet cached in the popularity-based cache, the APs will fetch them right away. The prefetch buffer uses LRU as the replacement policy.

Algorithm 3: Predicting client mobility at an AP

Input: client c, elapsed time since client's association t_0, previous associated AP AP_{prev}, current transferring chunk ID chk_0 of content $cont_c$, transition probability threshold P_{thres}
Output: Prefetch message

if *checkClientAssoc*$(c) ==$ *true* **then**
  $t_1 \leftarrow$getResiMedianbyRHT(AP_{prev})
  $\Delta t \leftarrow t_1 - t_0$
  $chk_s \leftarrow$calcPrefetchStartChk$(cont_c, chk_0, \Delta t)$
  $P_v \leftarrow$getNxtAPVectorbyNPT(AP_{prev})
  $P = 0$
  $i = 0$
  while $P < P_{thres}$ **do**
    $(AP_{nxt}, P_{transit}) \leftarrow$getAPwithTopProb$(P_v, i)$
    SndPrefMsg$(AP_{nxt}, cont_c, chk_s, P_{transit})$
    $P \leftarrow P + P_{transit}$
    $i \leftarrow i + 1$
  end
end

IV. Evaluation

We have developed a trace-driven simulator and conducted detailed evaluations of the proposed mobile content delivery scheme.

Algorithm 4: Calculating prefetch range

Input: Prefetch message M_p, popularity cache buffer B_{cach}, prefetch buffer B_{pref}
Output: prefetch chunk range

$(AP_{src}, cont_c, chk_s) \leftarrow$rcvPrefMsg$(M_p)$
$\Delta t \leftarrow$ getResiMeanbyRHT(AP_{src})
$chk_e \leftarrow$ calcPrefetchEndChk$(cont_c, chk_s, \Delta t)$
$chk_range \leftarrow$checkMem$(B_{cach}, B_{pref}, cont_c, chk_s, chk_e)$
if $chk\_range \neq 0$ **then**
  prefetch$(chk\_range, cont_c)$
end

A. Simulation Setup and Workload Generation

We developed a discrete event-driven simulator to evaluate EdgeBuffer. We generated realistic taxi mobile access workload by integrating a GPS trace of taxicabs in San Francisco (SF dataset) [21] and a wireless AP coverage database in SF (WiGLE database) [22]. The SF dataset contains GPS coordinates of 536 taxis for around 3 weeks in the Bay Area. In the original trace, the location update interval was typically 1 minute, and we inserted finer-grain location updates (with an update interval of 1 second) to the original dataset. The WiGLE database provides the AP deployment map in SF, and we cleaned up the database by merging APs that are within a certain distance of each other. In this study, we focus on two distance thresholds: 250m and 100m, referring to the resulting workloads as SF-250 and SF-100 respectively. Specifically, there are 687 APs in SF-250, and 2589 APs in SF-100 workload.

In the simulator, we use one node to represent the content hosting server and another node to represent the name resolution server. Each AP (corresponding to a node in simulator) is connected to both servers through a 50Mbps link with 20 millisecond delay. We further assume that a client is always associated with the closest AP, and sequentially downloads content chunks from the connected AP with an average link bandwidth of 2Mbps and 4Mbps for SF-250 and SF-100, respectively. The content server contains 100,000 files. Each file is 100 MB in size and has 100 chunks. The popularity of the content follows a Zipf distribution with a default parameter of 0.75, consistent with the observations in the web request traces [24]. In the simulations, we assume each AP has 10GB of storage for content caching and prefetching, with 8GB and 2GB allocated for each by default.

B. Prediction Accuracy

We first evaluate the accuracy of spatial prediction and temporal prediction for our network-level prediction (netPredict) and individual user based prediction (userPredict). Note that to implement user based prediction, we use a taxi's AP transition history to predict the next AP(s) that it will move to, and use the taxi's history residence time at an AP to predict its residence time at this AP.

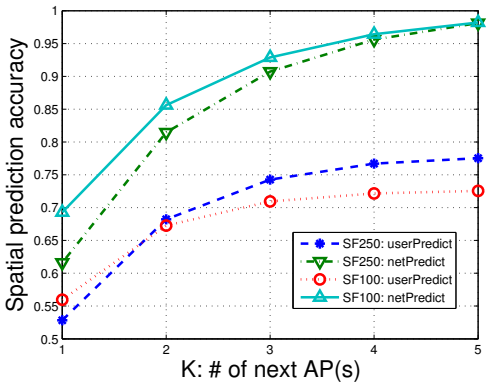
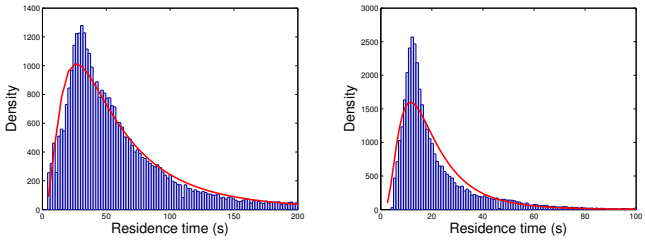


Fig. 6: Spatial prediction accuracy when picking different number of next AP(s). Predicting next AP can achieve high accuracy with a small K value using our network-level prediction.



(a) SF-250 (b) SF-100

Fig. 7: Histograms of residence times of an AP. The smooth curve is a log-normal density distribution.

We measure the spatial prediction accuracy as the percentage of instances in which a client indeed moves to one of the K predicted AP(s). We report the result in Figure 6. The figure shows that our network-level prediction has a much higher prediction accuracy than the user based prediction. Moreover, using our network-level prediction method, we can achieve a very high spatial prediction accuracy with a small K value (e.g., the value is 0.929 and 0.907 with $K = 3$ for SF100 and SF250). In the following simulations, unless otherwise specified, we set the transition probability threshold P_{thres} in Algorithm 3 as 0.8, which corresponds to $K = 2.46$ in average.

Unlike spatial prediction, accurate temporal prediction is much harder due to the high variation in residence times. To support the point, we randomly picked one AP from SF-250 and SF-100 workloads and plot the histogram of the AP's residence times in Figure 7. The histogram shows that the residence time at an AP follows a log-normal distribution, which is in agreement with the conclusion in [25], in which the authors discovered that the residence times in a campus environment follow a log-normal distribution.

In our temporal prediction, we predict a client's residence time as the median value of history residence time samples. We then measure the temporal prediction error as the average difference between the estimated residence time and the actual value. Using this definition, we investigate the estimation error at different history sample size, as seen in Figure 8. The results show that our prediction method is consistently more accurate

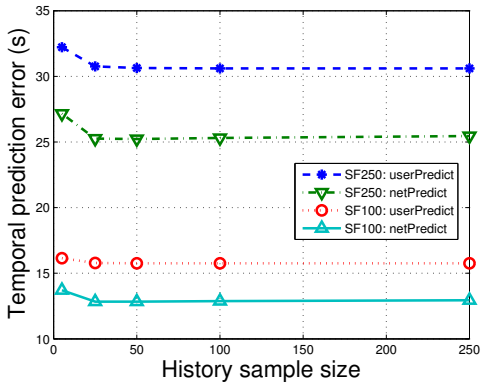


Fig. 8: We evaluate temporal prediction error when the residence time history table uses different history sample size.

than user based prediction because it can dynamically capture the traffic condition. Moreover, a modest history sample size is needed to get the optimal prediction accuracy. Specifically, at sample size of 50, the estimation error for our network-level prediction is 25.2s and 12.8s, while the estimation error for individual user based prediction is 30.6s and 15.8s, for SF-250 and SF-100. The prediction for SF-100 is better than SF-250 because of a smaller AP range. We use this history sample size as the default value.

We have also studied the prediction accuracy for a Beijing taxi drive trace [26], and the results show very similar trends. In the interest of space, we do not present these results here.

C. Hit Ratio at the Edge Buffers

We next evaluate the benefits of having caching and prefetching in the system. When a content request is satisfied at the AP's edge buffer, the multi-hop Internet path is replaced by a one hop path, the round-trip time is thus much smaller, and a higher throughput can generally be obtained. Therefore, the hit ratio at the edge buffers characterizes the performance gain of caching and prefetching regardless of different network settings (i.e., network topology, backhaul bandwidth and delay, wireless condition, etc), we thus focus on the cache hit ratio as the main metric here. Note that this metric is the "cache" hit ratio in terms of content chunks which includes hits from both the cache and the prefetch buffer.

1) Comparison of Four Edge Caching Strategies: We first compare the performance of four edge caching strategies: (1) *LRU*, in which we use the entire AP storage as an LRU cache, (2) *popCache*, in which we use the entire AP storage as a popularity-based cache, (3) *popCache+userPredict*, in which we partition the storage at an AP into two parts: a popularity-based cache and a prefetch buffer based upon user-level mobility prediction, (4) *popCache+netPredict*, in which we partition the storage at an AP into two parts: a popularity-based cache and a prefetch buffer based upon AP-level mobility prediction. In EdgeBuffer, we adopt the fourth strategy, i.e., popCache+netPredict.

We report the performance of the four strategies with different Zipf exponent parameters using SF-100 workload

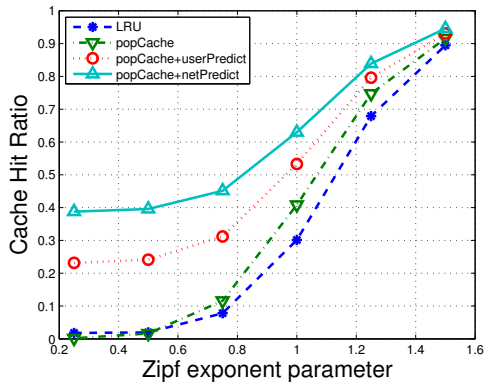


Fig. 9: Cache hit ratio for four edge caching strategies (with SF-100 workload).

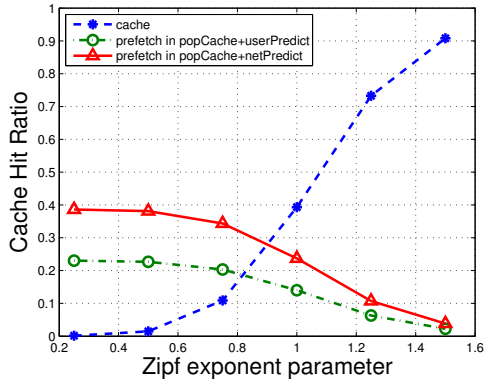


Fig. 10: For popCache+userPredict and popCache+netPredict, we break down the total cache hits into hits from cache and hits from prefetch buffer. Since the contributions from the cache are almost identical, we only show the cache contribution for popCache+netPredict here.

in Figure 9. Further, for popCache+userPredict and popCache+netPredict, we break down the total cache hits into hits from cache and hits from prefetch buffer, and show the breakdown in Figure 10. In fact, we have results for both SF-100 and SF-250 workloads, but choose to show only one because the trends are very similar. From the results, we have the following observations. First, with reasonable Zipf exponent values, prefetching is more effective than caching. As the Zipf exponent becomes larger, the content access popularity distribution becomes much sharper, caching becomes much more important than prefetching, and the difference between LRU cache and popularity-based cache becomes less important. As a result, all four strategies converge when we have large Zipf exponent values. Second, our strategy always fares better than the other three in achieving local hits at the edge. The detailed breakdown results show that our AP-level prefetching is more effective than individual user based prefetching.

Next let us look at the detailed performance when the Zipf exponent parameter is 0.75, which is a common value in web server access [24]. Here, the hit ratios are 0.078, 0.116, 0.312(0.109 from cache and 0.203 from prefetching), and 0.451(0.108 from cache and 0.343 from prefetching)

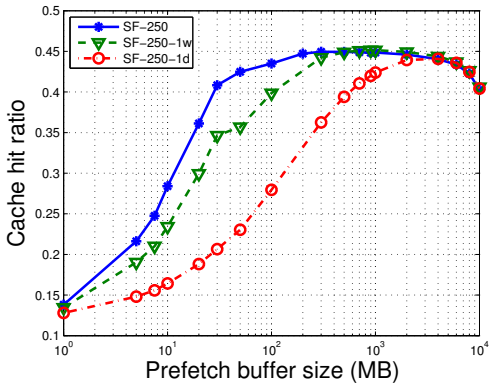


Fig. 11: Hit ratio with different prefetch buffer sizes. Note that an AP has a total of 10GB storage.

respectively for SF-100. The results show that the average hit ratio in our scheme offers 478% improvement over the traditional LRU cache scheme. Compared with popularity cache scheme, there is 289% improvement in hit ratio, which shows the importance of prefetching. Compared to individual user based prefetching, our AP-level prediction can lead to a 45% overall improvement. If we only focus on the hits due to prefetching, our scheme improves 69% over user-based prediction, further proving that our network aggregated prediction is more effective in urban settings.

2) Size Allocation between Cache and Prefetch Buffer: Next, we look at the impact of different cache and prefetch buffer size combinations, given that the sum of these two is fixed. In this set of simulations, we set the Zipf exponent parameter to be 0.75, and vary the load in the system. The original SF-250 workload has around 3 weeks of data with 536 taxis and 687 APs. We increased the load by compressing the total number of taxis that appeared across three weeks into one week and further into one day, thus getting workloads SF-250-1w and SF-250-1d.

We report the results in Figure 11, and have the following observations. First, there exists an "optimal" size for the prefetch buffer – after the prefetch buffer reaches a certain size, cache hit ratio starts to drop. Second, the optimal size varies with the load of the system. A lighter load requires a smaller prefetch buffer: SF-250 shows an optimal prefetch buffer at 3% of the total storage capacity, while SF-250-1d shows an optimal prefetch buffer at 40% of the total storage capacity. Third, the penally of having a prefetch buffer that is too small is much larger than the penalty of having a prefetch buffer that is too big. This is because when the Zipf exponent is 0.75, and the content space is reasonably large, prefetching plays a bigger role than caching.

We have also derived a guideline on how to size the prefetch buffer: $B_{pref} \geq \Delta t * K * \rho * n * M$, where Δt is the interval between the time when a chunk is prefetched and the time when the chunk is requested by the client ($\Delta t = t_1 - t_0$ as in Algorithm 3), K is the number of next AP(s) conducting prefetching (2.46 in our case), ρ is the mobile client arrival rate, n is the number of chunks prefetched for an arrival client,

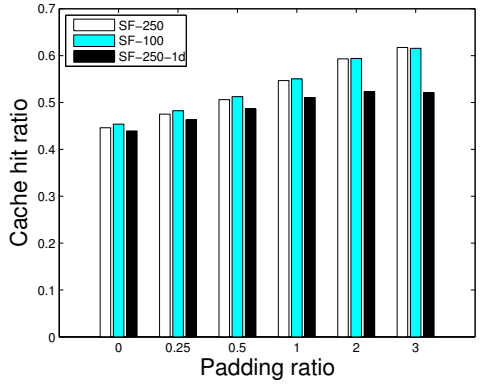


Fig. 12: Hit ratio at different padding level.

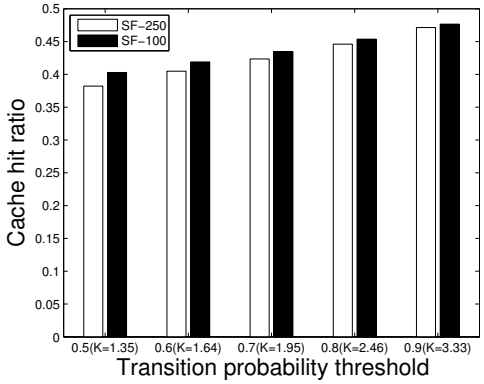


Fig. 13: Hit ratio at different transition probability threshold. We also calculate the average value of K at each threshold.

and M is the size of a chunk. This guideline ensures the prefetch buffer is large enough so that the prefetched chunks will not be replaced before they are consumed by the client. The resulting buffer size matches our simulation results well.

3) Prefetching More Chunks: In this set of results, we explore the impact of prefetching more chunks than what is given by Algorithm 4. Prefetching additional chunks may increase the hit ratio if our residence time estimation is erring on the small side, but it may evict useful chunks if the prefetch buffer is full[1]. Here, we introduce the parameter, *padding ratio*, to measure the amount of extra chunks that are prefetched. Assuming that we are supposed to fetch N chunks, a padding ratio of p means that $\frac{pN}{2}$ extra chunks are prefetched at both sides of the prefetch range.

We report the cache hit ratio at different padding levels in Figure 12, which shows the hit ratio goes up as we fetch more chunks. The hit ratio is about 38.4%, 35.7% and 18.5% higher with 300% padding ratio for SF-250, SF-100 and SF-250-1d. The main reason is that under the default simulation configuration, the prefetch buffer is large enough to accommodate prefetching extra chunks for SF-250 and SF-100. However, with heavier traffic load, prefetching too many chunks for each client will hurt the overall hit ratio, as seen from the degrading performance of SF-250-1d after padding ratio of 2.

4) Network Transition Probability Threshold: We investigate the impact of different transition probability threshold on the caching performance. The transition probability threshold determines the number of top probable next AP(s) that will conduct the prefetching, thus vastly affect the prefetching overhead and caching hit ratio.

The results are shown in Figure 13. The values for SF-250 are: 0.382, 0.405, 0.423, 0.446, 0.471; the values for SF-100 are: 0.403, 0.419, 0.435, 0.454, 0.477. The results show that we can achieve good performance with a small K value (like 1.35), such that the prefetch overhead is kept modest.

[1]Prefetching consumes backhaul bandwidth in addition to AP's buffer storage, thus causes bandwidth waste if the chunks prefetched are not used. Such impact is not evaluated in this paper.

V. Related Work

A. Future Internet Architecture

A number of clean-slate future Internet architecture designs have been proposed recently to address challenges faced by today's IP network. Such designs are generally centered around a name-based service – they address a network object by its unique name/identifier instead of a network address. Thus, the communication with any network object appears to be no different than that if it were a fixed endpoint, resulting in a location-independent communication paradigm.

These Internet architectures differ from each other in how they realize name-based service, especially in the presence of mobility. Several information-centric architectures (such as TRIAD [27] and NDN [4]) and flat-label routing architectures (such as ROFL [28]) propose a name-based routing approach which includes an object name into the routers' forwarding table and forwards packets directly based on the name, without using network-level addresses. While this approach is conceptually simple, the size of the forwarding table becomes overwhelmingly huge, and the propagation of routing updates usually causes substantial overhead and delays within a large network. It may be a challenge for mobile content delivery where clients frequently transit from one network to another.

On the other hand, some other architectures (like MobilityFirst [3], XIA [5], HIP [6] and AIP [29]) places object names outside of the routing plane and uses a name resolution service to translate names to addresses. In particular, MobilityFirst enables name-based service through dynamic in-network name resolution (GNRS) where both end-hosts and routers can query, and a GNRS mapping update when an endpoint moves. Such approach is well suited for mobile content delivery especially in case of fast mobility of hosts, as it only introduces a round-trip update or lookup latency without significantly lengthening the data path or causing too many routing updates during mobility. Moreover, GNRS oversees each mobile client's network movement trace, thus enables a network-level mobility prediction. Our work is based on such a mobility-centric architecture design.

B. Caching and Prefetching

Traditionally, content caching [16], [17] and CDN [18] are utilized to facilitate content distribution and retrieval. They normally measure the long-term content access pattern and cache the most frequently used or most recently used contents. However, most of such work doesn't consider the impact of short-term user mobility on caching performance.

On the other hand, prefetching [30], [31], [32] has been proposed to proactively preload data from servers instead of passively waiting on content requests. However, it is often used in the context of prefetching related content that is expected to be used in the near future according to users' access patterns or the content's structural relationship. Authors in [14], [15], [19] propose to prefetch parts of large content objects to different AP locations by utilizing a client's mobility pattern. While Sprinkler [14] assumes that mobile client's route is known as prerequisite, other work [15], [19] requires mobility prediction. These studies propose a personalized mobility models where each mobile client predicts its future connection based on its own history. Alternatively, authors in [25], [33] have demonstrated how to use traces of several users to build user mobility model in a college campus environment. We argue that a network-level aggregated prediction model is preferred for a complicated urban environment. Incidentally, in [19], [20], [33], authors all use a second-order Markov model to predict mobile client's future location.

VI. Concluding Remarks and Future Directions

In this paper, we present the design of EdgeBuffer, which aims to improve mobile client's content download performance by leveraging aggregated network-level prediction and prefetching as well as popularity-based caching at the network edge. Through detailed simulations, we show that such a scheme can significantly increase the fraction of content requests that can be satisfied at the edge, avoiding long latencies involved in reaching out to the original hosting server or CDN server. Our results also shed light on important issues such as how to allocate the size between cache buffer and prefetch buffer, when to prefetch more chunks, and by how much.

EdgeBuffer shares the same vision as many projects (e.g., PacketCloud, EdgeClouds) that advocate pushing network services towards the edge of the Internet. It is enabled by a fast emerging class of future Internet architectures that focus on addressing the challenges caused by mobile clients. In this paper, we sketch out the initial design of EdgeBuffer, and show that it is a promising approach in supporting mobile content delivery. Towards realizing its full potential, in our future direction, we will investigate more sophisticated prediction algorithms (e.g., leveraging detailed client's information like speed and direction together with network-level statistics), and also build large-scale prototypes and conduct real-world experiments to improve its design and quantify its benefits.

References

[1] Cisco: Global Mobile Data Traffic Forecast Update, 2013-2018.

[2] J. Saltzer, "On the naming and binding of network destinations," *RFC 1498, August*, 1993.

[3] MobilityFirst project, http://mobilityfirst.winlab.rutgers.edu/.

[4] V. Jacobson *et al.*, "Networking named content," in *Proc. of ACM CoNEXT*. ACM, 2009, pp. 1–12.

[5] D. Han, A. Anand, F. R. Dogar *et al.*, "Xia: Efficient support for evolvable internetworking." in *USENIX NSDI*, 2012.

[6] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Host identity protocol," *RFC 5201, April*, 2008.

[7] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet," *ACM SIGMOBILE Mobile Computing and Communications Review*, 2012.

[8] T. Vu *et al.*, "Dmap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet," in *IEEE ICDCS*, 2012.

[9] X. Tie, A. Sharma, and A. Venkataramani, "A global name service for a highly mobile internet," Technical report, UMASS, Tech. Rep., 2013.

[10] A. Venkataramani, A. Sharma *et al.*, "Design requirements of a global name service for a mobility-centric, trustworthy internetwork," in *IEEE COMSNETS*, 2013.

[11] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy *et al.*, "A data-oriented (and beyond) network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 181–192, 2007.

[12] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi, "Packetcloud: an open platform for elastic in-network services," in *Proc. of ACM MobiArch*. ACM, 2013, pp. 17–22.

[13] A. Chandra, J. Weissman, and B. Heintz, "Decentralized edge clouds," *Internet Computing, IEEE*, vol. 17, no. 5, pp. 70–73, 2013.

[14] S. K. Dandapat, S. Pradhan, N. Ganguly, and R. Roy Choudhury, "Sprinkler: distributed content storage for just-in-time streaming," in *Proc. of CellNet*. ACM, 2013, pp. 19–24.

[15] P. Deshpande, A. Kashyap, C. Sung, and S. R. Das, "Predictive methods for improved vehicular wifi access," in *ACM MobiSys*, New York, NY, USA, 2009, pp. 263–276.

[16] J. Zhang, R. Izmailov, D. Reininger, and M. Ott, "Web caching framework: analytical models and beyond," in *IEEE Workshop on Internet Applications*, 1999.

[17] C. Bernardini, T. Silverston, and O. Festor, "Mpc: Popularity-based caching strategy for content centric networks," in *ICC*. IEEE, 2013.

[18] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *Internet Computing, IEEE*, vol. 6, no. 5, pp. 50–58, Sep 2002.

[19] A. J. Nicholson and B. D. Noble, "Breadcrumbs: Forecasting mobile connectivity," in *ACM MobiCom*, 2008.

[20] L. Song, D. Kotz, R. Jain, and X. He, "Evaluating location predictors with extensive wi-fi mobility data," in *IEEE INFOCOM*, 2004.

[21] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD data set epfl/mobility (v. 2009-02-24)," Downloaded from http://crawdad.org/epfl/mobility/, Feb. 2009.

[22] WiGLE - Wireless Geographic Logging Engine, https://wigle.net/.

[23] F. Zhang, K. Nagaraja, Y. Zhang, and D. Raychaudhuri, "Content delivery in the mobilityfirst future internet architecture," in *Sarnoff Symposium (SARNOFF), 2012 35th IEEE*. IEEE, 2012, pp. 1–5.

[24] L. Breslau *et al.*, "Web caching and zipf-like distributions: evidence and implications," in *IEEE INFOCOM*, Mar 1999.

[25] M. Kim, D. Kotz, and S. Kim, "Extracting a mobility model from real user traces." in *IEEE INFOCOM*, 2006.

[26] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: driving directions based on taxi trajectories," in *ACM SIGSPATIAL GIS*, 2010, pp. 99–108.

[27] M. Gritter and D. R. Cheriton, "An architecture for content routing support in the internet." in *USITS*, vol. 1, 2001, pp. 4–4.

[28] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "Rofl: routing on flat labels," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 363–374.

[29] D. G. Andersen, H. Balakrishnan, N. Feamster *et al.*, "Accountable internet protocol (aip)," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 339–350.

[30] J. Xu, J. Liu, B. Li, and X. Jia, "Caching and prefetching for web content distribution," *Computing in science & engineering*, vol. 6, no. 4, 2004.

[31] E. P. Markatos, C. E. Chronaki *et al.*, "A top-10 approach to prefetching on the web," in *Proceedings of INET*, vol. 98, 1998, pp. 276–290.

[32] D. Duchamp *et al.*, "Prefetching hyperlinks." in *USENIX Symposium on Internet Technologies and Systems*, 1999, pp. 12–23.

[33] J. Yoon, B. D. Noble, M. Liu, and M. Kim, "Building realistic mobility models from coarse-grained traces," in *ACM MobiSys*, 2006.