Prediction-Based Prefetch Scheduling in Mobile Service Applications

Michael Borkowski, Olena Skarlat, Stefan Schulte, Schahram Dustdar
Distributed Systems Group, TU Wien; Vienna, Austria
{m.borkowski, o.skarlat, s.schulte, dustdar}@infosys.tuwien.ac.at

***Abstract*—The usage of Web or Cloud-based applications on mobile devices suffers from varying link quality, which causes user-perceivable delays and therefore reduces the Quality of Experience (QoE). On the other hand, mobile devices are increasingly feature-rich, allowing to make usage of context data in order to predict network quality based on the user's location.**

In this paper, we propose a prefetch scheduling algorithm based on network quality predictions, and evaluate it using data collected from real-world field tests. We show that our approach fulfills the expected gain in QoE. Using network quality prediction to optimize data prefetching can improve the user-perceived response time by up to 95 %.

Our results not only show the feasibility of the proposed algorithm, but also motivate further research in the field of mobility pattern creation, and undermine the importance of location as a part of user context throughout the software stack.

***Index Terms*—Mobile networks, prefetching, location dependent services, prediction, connectivity**

I. Introduction

In mobile computing, one important parameter of the user-perceived QoE is the round-trip-time between a sender and a receiver [1]. In many mobile application scenarios, certain data is required to be provided in a timely manner: Audio data from a music streaming service will cause playback stutter when the device-side buffer underruns; a turn-by-turn navigation system must provide users with on-time, detailed graphical instructions, possibly generated dynamically by a centralized service in order to reduce computational load of the client devices [2]; live traffic data can be provided to the client in real-time in order to provide the user with dynamic re-routing in case of traffic jams [3]. Also, an increase of applications deployed on mobile devices but using Web- and Cloud-based resources can be observed [1]. This shift towards mobile computing calls for new ways to improve the QoE for mobile users: Mobile devices are often connected to the Internet using mobile data networks like 3G (UMTS, HSDPA/HSUPA) or 4G (LTE), which are by nature subject to fluctuations and instabilities. Some of these variations are accountable to the provider (e.g., caused by over-provisioning), but often it is the location of the device which leads to low connection quality, e.g., because the user is driving through a tunnel or simply since the network coverage in rural areas is inferior to urban environments [4]. Furthermore, it is noteworthy that location influences connection quality not only because of geographical features (hills, tunnels, remote areas), but also because some locations might indirectly cause bad connection quality (a crowded stadium might have decreased mobile network quality, despite being in an otherwise geographically beneficial location, e.g., next to a mobile network base station).

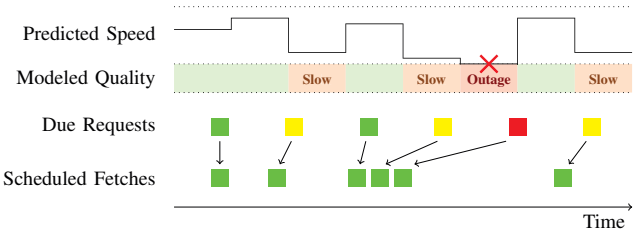

Fig. 1: Using prefetching to overcome periods of slow speed or lack of connectivity

Indeed, the adaptation of communication in order to increase the Quality of Service (QoS) has been a major research topic for many years [5], with recent works also addressing QoE [6], [2]. One particular approach to enhancing the QoE for mobile users is the prefetching of data [7]: Given a user's location, movement direction and possibly the planned route, the network quality over time can be predicted to a certain degree. This information can be used to prefetch data which could be needed during times of poor or intermittent connectivity. As shown in Figure 1, for requests due during such periods, data is then prefetched earlier, when connection quality is sufficient.

To the best of our knowledge, the effort put into developing approaches of smart prefetching based on mobility pattern is limited (see Section II). Hence, in this work, we lay the mathematical foundations of prefetch scheduling, propose a concrete algorithm for such scheduling of multiple requests with known attributes (deadline, size), assuming a model of predicted connection quality. Especially, we address mobility scenarios, i.e., take into account that the location of the user may change quickly and regularly, and therefore the network quality may be subject to rapid changes.

Furthermore, we provide a reference implementation for this algorithm and evaluate its performance using a testbed setup and a real-world recorded data set. Afterwards, we perform regression testing using various parameters.

The results of our work show that creating mobility patterns for users poses a valuable source of information, not only for high-level tasks like providing the user with, e.g., route calculation suggestions, but also on transport-level layers, by

2329-6453/16 $31.00 © 2016 IEEE
DOI 10.1109/MS.2016.16

mangling the fetching schedule for specific data blocks.

Summarizing, the contributions of our work are as follows:

- We formalize the notion of prefetching, its requirements and implications, and provide a discussion of the applicability in real-world situations.
- We formally derive a concrete algorithm for scheduling data fetches under the assumption that certain context knowledge is available.
- We evaluate the algorithm using data acquired in a real-world measurement, and compare its results to both our expectations and to results gained without using our proposed prefetching algorithm. We not only show that the algorithm yields the expected results, but also observe that even inaccurate prediction, allows for increased overall QoE when employing prefetching.
- Examining our results, we motivate a more in-depth research in the field of mobility pattern prediction, by showing that increasing prediction accuracy allows for improved QoE.

For this, the remainder of this paper is organized as follows: First, we will comment on the related work on data prefetching (Section II). Then, we state relevant preliminaries for our work (Section III). Afterwards, we introduce an algorithm for scheduling prefetching (Section IV). A reference implementation of this algorithm is evaluated using a testbed setup and a real-world data set (Section V). Eventually, we conclude this paper (Section VI).

II. Related Work

Prefetching is a well-known approach to increase the performance of an application. It is widely used in processors, databases and file systems [7]. In the following paragraphs, we will discuss according approaches in the field of wireless network services and distributed computing.

A lot of attention in literature is given to video transmission, or multimedia content in general, e.g., [8], [9], [10], [11]. Also, prefetching or caching of search results has gained some attention, e.g., [12], [13], [14]. The same applies to the caching or prefetching of generic Web content [15], [16]. None of these approaches, however, takes into account prefetching for mobile devices. Concepts related to prefetching in the context of location-dependent data, which are related to, but not in the focus of the work at hand, are targeted in [17].

We extend our former work, revisiting the notion of *time criticality* of specific services, or using the difference between *required bandwidth* and *available bandwidth* [18]. Riiser et al. also discuss mobile users and prefetching of data based on bandwidth prediction [9].

While we build on the fundamentals set by [18], [9], our work is – to the best of our knowledge – the first to derive the mathematical basis for prefetch scheduling using calculus, and to formulate an actual algorithm for scheduling fetches.

We assume two major preconditions for our work, namely mobility prediction and network quality (or bandwidth) prediction (see Section III-B). Prediction of user mobility has been researched extensively, e.g., [19], [20], [21]. Network quality prediction has been the subject of research in [22], [23], [24]. A combination of mobility profile creation and network quality prediction can be found in [18], [9], [25].

Reactions to predicted network quality, apart from prefetching as described here, can include offloading data transfers to nearby WiFi stations, either by pushing data to hot-spots [26] or delaying transfers until hot-spots are available [27], [28].

III. Preliminaries

In the following subsections, we provide some background by setting definitions regarding the data to be prefetched.

A. Data Suitability for Prefetching

Not all kinds of data are suitable for prefetching. While we discussed certain types of data regarding suitability in our previous work [29], we herein use a simplified definition: Data is suitable for prefetching if it is cacheable; furthermore, data is cacheable if it does not lose its value or meaning when being fetched ahead of its usage. For instance, prefetching a music song a few minutes ahead does not reduce its value. Prefetching a live surveillance camera image even ten seconds before displaying can render it useless.

B. Predictability

The proposed algorithm requires future network quality, as well as data demand, to be predictable to some degree.

We estimate the future network quality by assuming certain knowledge about the expected user position. As discussed in Section II, for predicting user mobility, different approaches exist. In general, any of these approaches could be used as foundation for the work at hand. As discussed, using mobility patterns for bandwidth prediction has been investigated and applied in [18], [9], [25].

Since the predictability of future network quality can be seen as a requirement which is hard to satisfy, we have conducted extensive research regarding the impact of inaccurate predictions. In fact, the results show that predictions do not necessarily have to be exact in order to increase QoE. The response time is reduced, compared to no location-dependent prefetching, even with very rough (inaccurate) bandwidth prediction. See Section V-B for further details.

The second major requirement is the predictability of the future data demand. This is necessary for deciding when to schedule data (pre-)fetches, and to account for bandwidth required by other applications running on the same client.

Predicting data demand is highly application-specific. A navigation system might have well-predictable fetches, music playing from a playlist is also predictable (unless songs are skipped quickly). On the other hand, demand for fetching e-mail messages is generally not predictable at all. For the purpose of our work, however, we assume that the data items of interest are predictable to a degree that makes our scheduling approach feasible.

We suggest an approach where applications provide demand prediction through APIs. This, however, requires additional engineering by application developers, since the demand must be calculated in real-time.

IV. Scheduling Algorithm

We seek to improve the aforementioned prefetching by taking into account the user context, mainly by predicting network speed from the user's location. In the following section, we formulate a model for such a problem (Section IV-A1), define target variables (Section IV-A2) and finally propose the actual algorithm (Section IV-B and Section IV-C).

A. Model and Requirements

The following sections define the algorithm's problem-specific domain, along with its model and target variables.

1) Model: In our problem domain, the *network quality* is represented by the *network byte rate* (speed), i.e., the rate at which bytes can pass the link and arrive at the receiver. It is important to note that other parameters, mainly the *latency*, also impact QoE, however, for simplicity, we ignore them in our model.

We base our model on the assumptions on the predictability of future network quality and knowledge about future requests, as discussed above:

1) The network byte rate B_{net} is known (estimable) for a future point in time t, denoted as $B_{net}(t)$.
2) A set of scheduled future requests $\mathcal{R}$ is known. A request $R \in \mathcal{R}$ is a tuple $R = (\tau, D_{req})$, where τ is the time at which the data is required (the request's *deadline*) and D_{req} is the amount of data requested.

We furthermore assume that our scheduling decision algorithm is called whenever re-scheduling is necessary, i.e., whenever either the byte rate prediction or the set of scheduled future requests change. Note that both are subject to change whenever more precise data is available.

The outcome of the decision algorithm, for each request, is a scheduling proposal. In other words, the algorithm decides when the fetching of a certain data item should start in order to maximize its utility (see Section IV-A2). We therefore define the output of the algorithm as a set of tuples (R, λ) where λ is the scheduled fetching time for R.

2) Target Variables: The decision algorithm primarily optimizes for a minimal *user-perceived response time* for each request R. Given that the input consists of multiple requests, we regard the minimal *sum* of waiting times as optimal. We denote the user-perceived response time as $\mathcal{RT}$, and formulate the first requirement for our scheduling algorithm:

$$\mathcal{RT} \rightarrow \min \tag{1}$$

A trivial solution for (1) would be to fetch all requests at $t = 0$ (i.e., the earliest possible point in time). This involves caching all data for a potentially long time, yielding data that is not *fresh*, i.e., is outdated. To avoid this, we minimize the *data age* ($\mathcal{DA}$) and formulate a second requirement:

$$\mathcal{DA} \rightarrow \min \tag{2}$$

Ultimately, we observe that a trivial solution for (1) and (2) would be to start fetching all requests at $t = 0$ and re-fetch them periodically, until the application layer actually utilizes the data. This would cause continuous data transfer, which unnecessarily increases cost and energy consumption. We therefore introduce a third target variable, *data volume*, i.e., the total amount of transferred data, denote it as $\mathcal{DV}$, and also minimize it. This yields our third requirement:

$$\mathcal{DV} \rightarrow \min \tag{3}$$

Concluding this section, we require data to not be fetched too late (1), but also as late as possible (2), without unnecessarily repeating transfers (3).

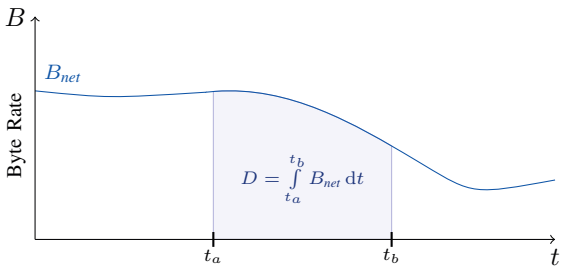

Fig. 2: Transmitting D data (the area under B_{net}) in $(t_b - t_a)$ time, represented as an integral of B_{net}

B. Mode of Operation: One Request

The following section describes the mathematical idea behind the proposed algorithm. We assume a transmission of a defined amount of data, D_{req}, in a time interval $[t_a, t_b]$, with the byte rate B. We also assume that the transmission uses the entire available network byte rate, i.e., $B = B_{net}$. We will refine this notion later (see section IV-D1).

We note that the byte rate is the rate of transferred data per time ($B = \frac{D}{t}$). The momentary byte rate at t is the increase of data transferred up to t. In other words, the byte rate is the *derivative* of transferred data: $B(t) = \frac{\Delta D(t)}{\Delta t}$.

We transform the derivative to an integral to describe the transfer of our request (with D_{req} data and B_{net} byte rate):

$$D_{req} = \int_{t_a}^{t_b} B_{net}(t)\,dt \tag{4}$$

Figure 2 exemplifies this: D_{req}, the area under B_{net}, represents the data transferred between t_a and t_b.

Since data transfer is not truly a continuous process as suggested by this model, but rather a transfer of discrete chunks, we re-formulate the integral as a sum:

$$D_{req} = \sum_{t=t_a}^{t_b} B_{net}(t)\Delta t \tag{5}$$

Note that this model is an approximation, not accounting for deviations from the predicted byte rate B_{net}, or sharing of the wireless connection between applications or devices using the same network.

Since we ultimately seek to finish the fetch on time, i.e., to answer the question on *when to start fetching*, we need to

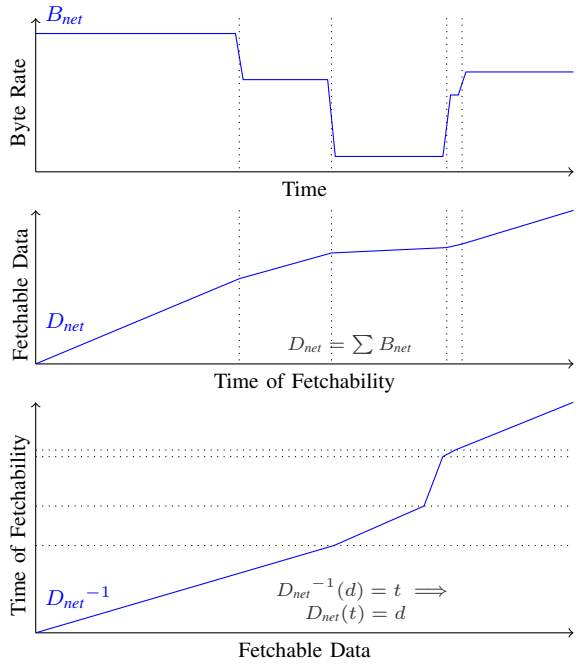

Fig. 3: Relationship between B_{net}, D_{net} and ${D_{net}}^{-1}$

find the lower boundary t_a of the sum. We denote the resulting point in time using the variable λ. Since we cannot solve for λ without a numeric definition of B_{net}, we require an extension to our approach, described in the following.

We define the anti-derivative (integral) of B_{net} and call it $D_{net}(t) = \sum_0^t B_{net}(t)$. In Figure 3, we show the relationship between B_{net} and D_{net}. Since D_{net} is monotonic and increasing[1], we are able to define an inverse function and call it ${D_{net}}^{-1}$, also shown in figure 3. Hence, for any data amount D given, ${D_{net}}^{-1}(D) = t$ returns the smallest[2] point in time t so that $D_{net}(t) = D$.

With ${D_{net}}^{-1}$, we have now defined a function which, for a given data amount D, returns a point in time at which D data will be fetchable (starting from $t = 0$).

We return to our initial problem from Section IV-A1, i.e., for each given request $R = (\tau, D_{req})$, to fetch D_{req} data and finish fetching as close to the point in time τ (the deadline) as possible, but not later. Therefore, if we know that at τ, $D_{net}(\tau)$ data will be fetchable (starting from $t = 0$), we know that we seek to determine a point in time λ at which:

$$\underbrace{D_{net}(\lambda)}_{\text{Data fetchable until } \lambda} + \underbrace{D_{req}}_{\text{Request data}} = \underbrace{D_{net}(\tau)}_{\text{Data fetchable until } \tau} \tag{6}$$

It follows from this that D_{req} data is fetchable in the interval $[\lambda, \tau]$. This fits our task of modeling the transfer of D_{req} data.

[1] D_{net} is strictly monotonic if and only if B_{net} is always greater than zero, which we cannot assume; we therefore assume D_{net} is merely monotonic, not strictly monotonic.

[2] Necessary because D_{net} is not *strictly* monotonic.

We can now solve for λ using the introduced inverse function ${D_{net}}^{-1}$:

$$D_{net}(\lambda) = D_{net}(\tau) - D_{req} \tag{7}$$

$$\lambda = {D_{net}}^{-1}(D_{net}(\tau) - D_{req}) \tag{8}$$

Given our derivation, λ satisfies $\sum_{t=\lambda}^{\tau} B_{net}(t) = D_{req}$, i.e., the network can transfer D_{req} data in the interval $[\lambda, \tau]$. Hence, when starting the transfer of D_{req} data at λ, it will be finished at τ. This fits our requirements (1)–(3): First, since the transfer finishes in time (at τ), $\mathcal{RT}$ is zero. Since we select the latest possible point in time λ, $\mathcal{DA}$ is minimal. We also transfer data once, instead of re-transmitting it periodically, which minimizes $\mathcal{DV}$.

Summing up, in this section, we derived a solution to determine an optimal fetching time λ for a single request and formulated it in (8).

C. Mode of Operation: Multiple Requests

Our algorithm is required to schedule multiple requests. In the last section, we have shown how to schedule the fetching time for a single request. In this section, we describe how to extend this functionality to scheduling multiple requests.

As an example, we assume five requests, $R_0 - R_4$, with according deadlines $\tau_0 - \tau_4$. We work our way backwards, and start by scheduling the last request, in our case R_4. By means described in Section IV-B, we find the time for fetching R_4 (named λ_4). Figure 4 shows our example, where λ_4 has no overlap with deadline τ_3. We therefore schedule R_3 accordingly, obtaining the fetch time λ_3.

However, we now see an overlap between the fetching of R_3 (taking place in the time interval $[\lambda_3, \tau_3]$, represented by the blue area R_3) and the next request in line (R_2), since that request's deadline (τ_2) now lies within the fetching time of R_3. We therefore have to start fetching R_2 even earlier, and compensate this by using the same procedure as before, but instead of using τ_2 as the target time, we select λ_3.

In other words, fetching request R_2 must be finished at τ_2, as we have already established, but since this point in time is already known to be busy, we assume an even earlier deadline. For this, we use λ_3, i.e., the latest point in time, in which the network is known not to be busy. Figure 5 shows the result, with the same correction applied to request R_1.

Summarizing, we need a new upper bound instead of τ in (8), and introduce τ', removing overlaps. For a request R_n with the subsequent request R_{n+1}, with a scheduled fetch start λ_{n+1}, a new τ'_n is calculated as follows:

$$\tau'_n = \min(\underbrace{\tau_n}_{\text{No conflict}}, \overbrace{\underbrace{\lambda_{n+1}}_{\text{Conflict with next request}}}^{\text{Next request's fetch start}}) \tag{9}$$

Replacing τ with τ' in (8) eliminates overlaps. This ensures timely fetches of all requests, to the extent of the byte rate prediction accuracy.

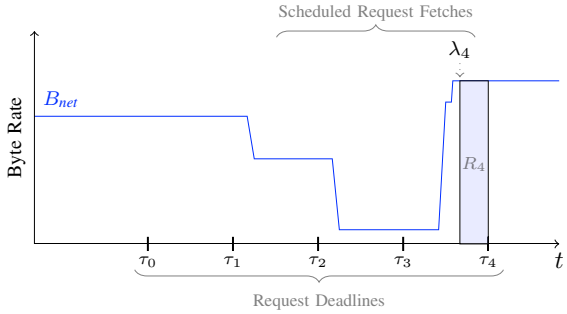

Fig. 4: Requests R_4 and R_3 already scheduled, potential overlap between R_3 and deadline of R_2

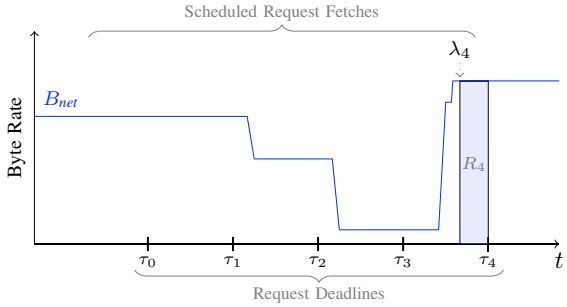

Fig. 5: All requests scheduled, τ_2 and τ_1 adapted to avoid overlaps with R_3 and R_2, respectively

D. Algorithm Parameters

During our preliminary evaluation, we found that certain aspects of the algorithm require further adaption. We therefore introduce two parameters for our algorithm.

1) Error-Correction Factor α: Since it is assumed that the bandwidth is merely predicted – thus, imprecise – and also mobile bandwidth is shared with other applications running on the same device, as well as with other mobile devices, a certain error margin has to be considered. To account for this, we introduce α as a parameter, with $\alpha \in [0, 1]$. $\alpha = 1.0$ means that no error margin is considered by the algorithm (all predicted available bandwidth is claimed in its scheduling), $\alpha = 0.5$ means that only half of the predicted bandwidth is used. In an extreme case, $\alpha = 0.01$ would cause the algorithm to use only 1 % of the predicted available bandwidth.

2) Look-Ahead Time $t_{horizon}$: The parameter t_{horizon} describes the look-ahead time of the algorithm, i.e., how much into the future the algorithm plans its fetches. This is done for two reasons: First, from a practical point of view, mobile devices are prone to having limited resources, especially memory and processing power. Therefore, it can be assumed that a certain limit has to be established in order to conserve those resources. Second, from a simulation point of view, prediction quality (of both future bandwidth and future data fetches) decreases over time, as the likelihood of changing circumstances increases. The parameter t_{horizon}, by cutting off any scheduling after the look-ahead time, makes the simulation model more realistic.

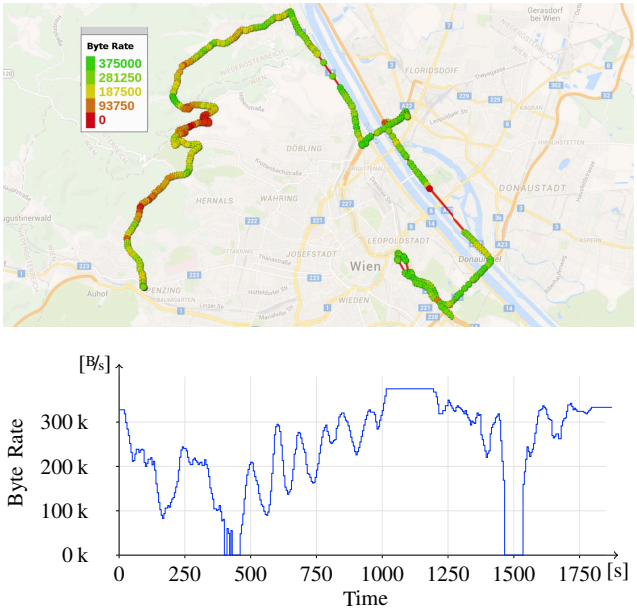

Fig. 6: The route used to record the test dataset, along with the extracted speed information

V. Evaluation

We evaluate the proposed algorithm in three steps. First, we simulate the algorithm using a testbed consisting of virtual machines running Linux. We employ network traffic shaping using the `tc` command (provided by the `dummynet` project[3]), which is a standard traffic shaping tool in the Linux operating system.

Our use case for evaluation is a car driving along a route, fetching data from a service in defined intervals (e.g., directions for turn-by-turn navigation). We aim to resemble this use case, while maintaining the flexibility provided by a testbed. We therefore perform test runs using a data trace recorded while driving along a route in a car. The route contains periods of poor reception due to hills, as well as around 70 s of total connection outage in a tunnel (see Figure 6). Results are presented in Section V-A.

Afterwards, we put our focus onto measuring the impact of prediction accuracy on the results, since the predictability of network speed is undoubtedly the strongest assumption. In Section V-B, we describe our evaluation using simulations and regression analysis, and show that even inaccurate predictions improve the overall performance.

Finally, we analyze the impact of algorithm parameters of the presented approach on the result. Specifically, we measure the impact of the look-ahead-time (t_{horizon}) and the error correction factor (α). The results are shown in Section V-C.

[3] http://info.iet.unipi.it/~luigi/dummynet/

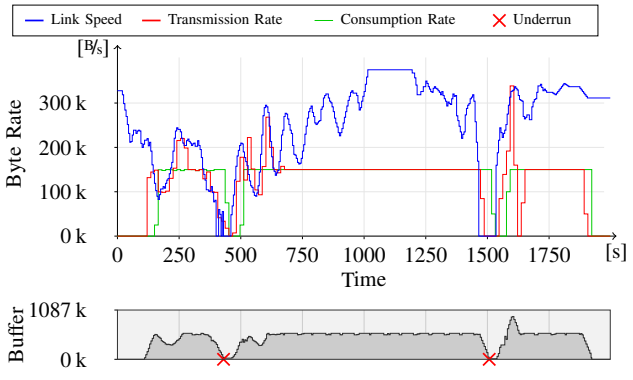

Fig. 7: Resulting timeline using location-independent prefetching, showing two buffer underruns

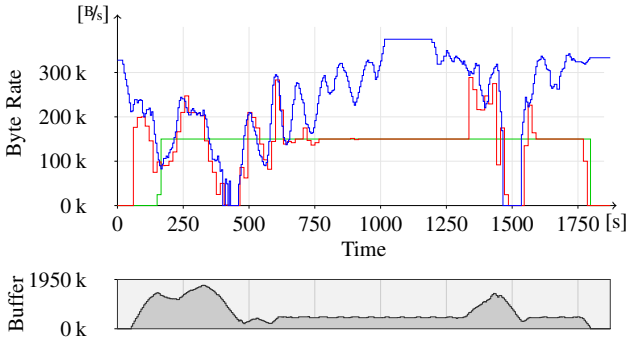

Fig. 8: Resulting timeline using location-dependent prefetching, showing no buffer underruns

A. Evaluation Using Real-World Route

Our simulation consists of service calls with a varying interval and size, which produce an average traffic of 150 kB/s. The total size of the transmitted data is 24.75 MB, and the transmission lasts for 165 s, if no interruption is encountered. Our mobile client software consumes the data stream with a constant byte rate, but builds an initial buffer of 450 kB before starting consumption, and also after having encountered a buffer underrun.

The route traveled is shown in Figure 6, along with the network byte rate information extracted from this test drive. We corrected the route after recording regarding GPS inaccuracies inside the tunnel. Furthermore, we used a different time scale (essentially accelerating the simulation) in order to achieve a feasible simulation time in our testbed. Our simulation thus has a duration of 1800 s (30 min), even though the real drive lasted for over an hour.

In order to have a baseline result without employing our location-aware prefetching approach, we employ a buffering method consisting of prefetching independently of user location (and thus, independent of predicted network speed). This is a method commonly used in media transmission and playback software [30]. As we can see in Figure 7, buffer overruns occur during the first phase of bad reception (between 350 s and 500 s), as well as while traveling through the tunnel (between 1450 s and 1600 s).

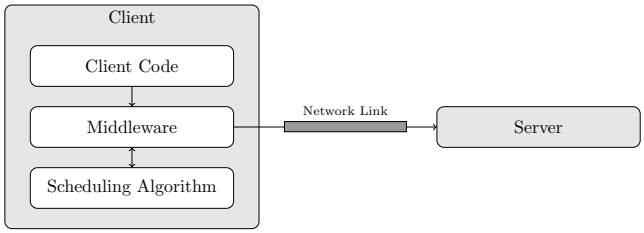

Fig. 9: Architecture setup of simulation environment used for regression tests

We then employ our proposed prefetching algorithm. The algorithm overcomes all periods of bad or lacking reception by pre-buffering data (see Figure 8). We can also see that the algorithm does not prefetch data unnecessarily: during times of good reception (for instance, between 750 s and 1250 s), no prefetching is performed. Before those periods of good reception end, however, the algorithm starts prefetching again – this effect is visible at 1300 s.

We recorded a total of three routes in the Vienna urban area and its surroundings (one of which is shown here) and could consistently verify our findings with all three of them. Further details can be found online.[4]

B. Impact of Prediction Inaccuracy

Naturally, the strongest assumption in our work is the availability of bandwidth prediction in the future. In Section II, we have shown that the subject of creating mobility patterns of users is of ongoing interest. Nevertheless, we want to measure the impact of prediction inaccuracy on the performance of our approach.

To this end, we developed an agent-based virtual-time simulation environment. The environment consists of a client-server setup, where the network link (see Figure 9) is manipulated, in order to artificially induce network bandwidth limit and delay.

We used the simulation environment for regression testing. Specifically, we performed a series of test runs, changing various accuracy parameters. Each parameter constellation was run repeatedly ($n = 500$) and all results of those runs were aggregated.

For all runs, we used three different approaches: No prefetching (A), which constitutes a very naive baseline; location-independent prefetching (B), which represents a commonly used approach; and our proposed location-dependent prefetching algorithm (C). We measure two parameters: response time ($\mathcal{RT}$) and data age ($\mathcal{DA}$).

We identified two major manifestations of bandwidth prediction inaccuracies. Predictions can be either inaccurate regarding amplitude (i.e., an overly high or overly low prediction), or regarding time (i.e., a change in bandwidth actually happens earlier or later than predicted). Other possible inaccuracies can be seen as a combination of those two factors.

[4] http://github.com/michael-borkowski/pf-sched

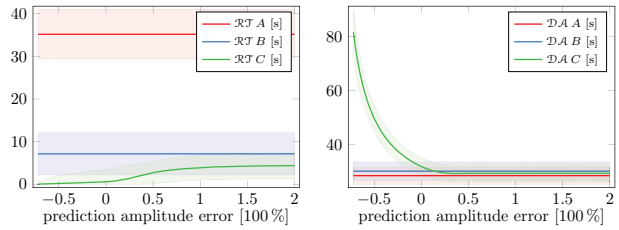

Fig. 10: Impact of prediction amplitude inaccuracy on response time (left) and on data age (right), with no prefetching (A), location-independent prefetching (B) and the proposed location-based prefetching approach (C)

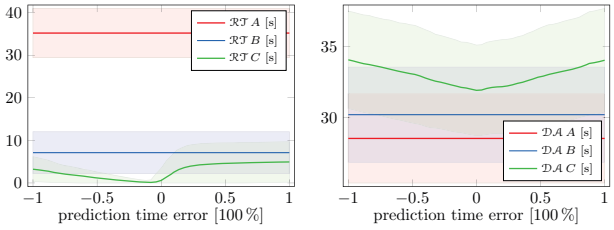

Fig. 11: Impact of prediction time inaccuracy on response time (left) and on data age (right), with no prefetching (A), location-independent prefetching (B) and the proposed location-based prefetching approach (C)

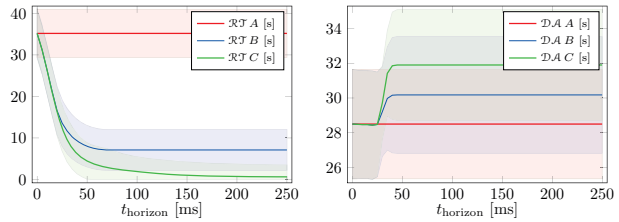

Fig. 12: Impact of the t_{horizon} parameter on response time (left) and on data age (right), with no prefetching (A), location-independent prefetching (B) and the proposed location-based prefetching approach (C)

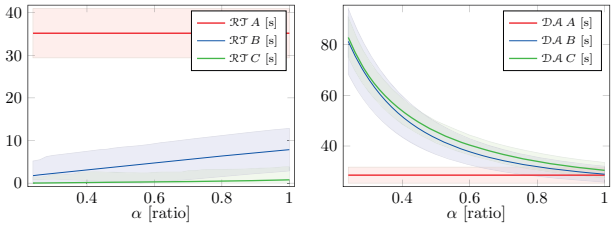

Fig. 13: Impact of the α parameter on response time (left) and on data age (right), with no prefetching (A), location-independent prefetching (B) and the proposed location-based prefetching approach (C)

Therefore, we performed regression analysis for two prediction parameters: *amplitude error* and *time error*. In Figure 10, we observe that negative amplitude inaccuracies (prediction is less than actual bandwidth) cause lower response time (better), while positive amplitude inaccuracies (prediction is more than actual bandwidth) cause higher response time (worse). This matches our expectation, where an overly optimistic prediction causes worse results than when rather conservative (early) fetching is employed. On the other hand, data age increases drastically as prediction becomes more and more pessimistic, which intuitively shows the drawback of overly pessimistic predictions.

Regarding time inaccuracy, Figure 11 shows that response time is impacted in both directions of inaccuracy, i.e., both premature and late predictions impact response time negatively, albeit the patterns for those two directions differ (the graph for $\mathcal{RTC}$ is asymmetric). We have not found a definite answer to this phenomenon as of the time of writing, but can draw a general conclusion stating that time inaccuracy negatively affects prefetching performance, as expected.

Concluding the investigations regarding the impact of estimation, we observe that while prediction inaccuracy naturally affects the performance of our proposed algorithm, it does so in an extent that justifies using prefetching even with partial (inaccurate) prediction. Most notably, our approach is never performing worse than the two presented baselines approaches; in most cases, its results are significantly better with respect to response time and data age.

C. Impact of Algorithm Parameters

In order to investigate the impact of the two algorithm parameters introduced in Section IV-D, we performed regression analysis, similar to the method described in Section V-B. Figure 12 shows the impact of t_{horizon} on the performance. Around 45 ms, response time stops decreasing, and data age stops increasing. This matches the simulation setup used, with data bursts requested at an interval of 45 ms on average. The intuitive notion that the look-ahead time should be higher than the excepted call interval is confirmed in this observation.

Figure 13 shows the impact of the α parameter. Similar to Figure 10, a decrease of response time (better performance) is related to an increase of data age (worse performance). This is caused by the fact that a lower α results in a more conservative (pessimistic) schedule decision, in the same way that those results are caused by negative amplitude error in Figure 10.

VI. Conclusions and Future Work

In this paper, we have discussed and shown the demand for prefetching in mobile networks. We have described how using network quality prediction can be used to optimize prefetching, enhancing the user-perceived QoE.

We have derived and formulated a concrete algorithm for prefetching, based on request deadlines and an approximate connection speed prediction. This algorithm performs as envisioned, which can be seen in our evaluation using real-world data from actual connection speed recordings.

Even though our proposed approach requires assumptions about the predictability of data demand and network quality,

and these assumptions seem strong, we have shown that even with imprecise prediction, our approach provides a decrease in waiting time, leading to a significant increase of overall, user-perceived QoE.

We therefore motivate research in mobility pattern recognition and prediction. This, in conjunction with research presented herein ultimately allows to optimize user experience in context of current and projected location.

The main areas of further research are predictive elements (gaining more precise knowledge about network quality and data demand), as well as the algorithm itself: Predictions can be improved, taking into account additional context (e.g., the user's calendar to gain knowledge about their future location). Tailoring of the algorithm can yield further improvements when taking into account factors like applications rivaling for the mobile connection, i.e., running on the same client.

Furthermore, it is noteworthy that all prefetching approaches, deployed to mobile clients, bear the danger of creating ripple effects: a high amount of clients, prefetching data during times of good network quality, actually decrease the network quality during this time. Deploying such mechanisms on large-scale networks requires appropriate mechanisms for preventing such situations.

ACKNOWLEDGMENT

This work is partially supported by the Commission of the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201) and by TU Wien research funds.

REFERENCES

[1] S. Ickin, K. Wac, M. Fiedler, L. Janowski, J.-H. Hong, and A. K. Dey, "Factors Influencing Quality of Experience of Commonly Used Mobile Applications," *IEEE Communications Magazine*, vol. 50, pp. 48–56, 2012.

[2] A. Papageorgiou, A. Miede, S. Schulte, D. Schuller, and R. Steinmetz, "Decision Support for Web Service Adaptation," *Pervasive and Mobile Computing*, vol. 12, pp. 197–213, 2014.

[3] F. Lécué, S. Tavelli-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. Sbodio, and P. Tommasi, "Smart traffic analytics in the semantic web with STAR-CITY: Scenarios, system and lessons learned in Dublin City," *Web Semantics*, vol. 27–28, pp. 26–33, 2014.

[4] W. L. Tan, F. Lam, and W. C. Lau, "An empirical study on 3g network capacity and performance," in *26th International Conference on Computer Communications (INFOCOM)*. IEEE, 2007, pp. 1514–1522.

[5] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "QoS-aware Middleware for Ubiquitous and Heterogeneous Environments," *IEEE Communications Magazine*, vol. 39, no. 11, pp. 140–148, 2001.

[6] K. P. Demestichas, A. A. Koutsorodi, E. F. Adamopoulou, and M. E. Theologou, "Modelling user preferences and configuring service in b3g devices," *Wireless Networks*, vol. 14, pp. 699–713, 2007.

[7] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed Mobile Prefetching," in *10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012, pp. 155–168.

[8] K. Evensen, A. Petlund, H. Riiser, P. Vigmostad, D. Kaspar, C. Griwodz, and P. Halvorsen, "Mobile video streaming using location-based network prediction and transparent handover," in *21st International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. ACM, 2011, pp. 21–26.

[9] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video streaming using a location-based bandwidth-lookup service for bitrate planning," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 8, no. 3, pp. 24:1–24:19, 2012.

[10] F. H. Fitzek and M. Reisslein, "A prefetching protocol for continuous media streaming in wireless environments," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 10, pp. 2015–2028, Sep. 2006.

[11] V. Singh, J. Ott, and I. Curcio, "Predictive buffering for streaming video in 3g networks," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Jun. 2012, pp. 1–10.

[12] S. Jonassen, B. B. Cambazoglu, and F. Silvestri, "Prefetching query results and its impact on search engines," in *35th International Conference on Research and Development in Information Retrieval (SIGIR)*. New York, NY, USA: ACM, 2012, pp. 631–640.

[13] S. Ding, J. Attenberg, R. Baeza-Yates, and T. Suel, "Batch query processing for web search engines," in *4th International Conference on Web Search and Data Mining (WSDM)*. New York, NY, USA: ACM, 2011, pp. 137–146.

[14] B. Cambazoglu and R. Baeza-Yates, "Scalability challenges in web search engines," in *Advanced Topics in Information Retrieval*, ser. The Information Retrieval Series, M. Melucci and R. Baeza-Yates, Eds. Springer Berlin Heidelberg, 2011, vol. 33, pp. 27–50.

[15] H. Huang, H. Sun, G. Ma, X. Wang, and X. Liu, "Poster: A framework for instant mobile web browsing with smart prefetching and caching," in *20th International Conference on Mobile Computing and Networking (MobiCom)*. ACM, 2014, pp. 367–370.

[16] Z. Jiang and L. Kleinrock, "Web prefetching in a mobile environment," *Personal Communications, IEEE*, vol. 5, no. 5, pp. 25–34, 1998.

[17] G. Cho, "Using predictive prefetching to improve location awareness of mobile information service," in *International Conference on Computational Science (ICCS)*. Springer-Verlag, 2002, pp. 1128–1136.

[18] W. Hummer, S. Schulte, P. Hoenisch, and S. Dustdar, "Context-aware data prefetching in mobile service environments," in *4th International Conference on Big Data and Cloud Computing (BdCloud)*. IEEE, 2014, pp. 214–221.

[19] M. Kim, D. Kotz, and S. Kim, "Extracting a mobility model from real user traces," in *25th International Conference on Computer Communications (INFOCOM)*. IEEE, 2006, pp. 1–13.

[20] B. Liang and Z. J. Haas, "Predictive distance-based mobility management for multidimensional pcs networks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 718–732, 2003.

[21] I. F. Akyildiz and W. Wang, "The predictive user mobility profile framework for wireless multimedia networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 1021–1035, 2004.

[22] A. J. Nicholson and B. D. Noble, "Breadcrumbs: Forecasting mobile connectivity," in *14th International Conference on Mobile Computing and Networking (MobiCom)*. ACM, 2008, pp. 46–57.

[23] J. Yao, S. Kanhere, and M. Hassan, "Quality improvement of mobile video using geo-intelligent rate adaptation," in *Wireless Communications and Networking Conference (WCNC)*, 2010, pp. 1–6.

[24] J. Yao, S. S. Kanhere, and M. Hassan, "An empirical study of bandwidth predictability in mobile computing," in *3rd International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*. ACM, 2008, pp. 11–18.

[25] J. Ghosh, H. Q. Ngo, and C. Qiao, "Mobility profile based routing within intermittently connected mobile ad hoc networks (icman)," in *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing*, ser. IWCMC '06. New York, NY, USA: ACM, 2006, pp. 551–556.

[26] N. Imai, H. Morikawa, and T. Aoyama, "Prefetching architecture for hot-spotted networks," in *IEEE International Conference on Communications*, vol. 7, 2001, pp. 2006–2010 vol.7.

[27] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting mobile 3g using wifi," in *8th International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, 2010, pp. 209–222.

[28] V. A. Siris and D. Kalyvas, "Enhancing mobile data offloading with mobility prediction and prefetching," in *7th ACM International Workshop on Mobility in the Evolving Internet Architecture*. ACM, 2012, pp. 17–22.

[29] M. Borkowski, "Smart prefetching for mobile users under volatile network conditions," Master's thesis, Technische Universität Wien, Vienna, Austria, 2015.

[30] W. chi Feng and K. Ramakishnan, "A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video," in *16th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 1, Apr 1997, pp. 58–66 vol.1.