공학박사 학위논문

Cost-Aware Data Offloading with Throughput-Delay Tradeoffs

처리율과 지연시간의 트레이드오프를 통한 비용
인지 데이터 오프로딩

2014년 8월

서울대학교 대학원
전기 · 컴퓨터공학부
임영빈

Cost-Aware Data Offloading with Throughput-Delay Tradeoffs

지도교수 권태경

이 논문을 공학박사 학위논문으로 제출함
2014년 5월

서울대학교 대학원
전기 · 컴퓨터공학부
임영빈

임영빈의 박사 학위논문을 인준함
2014년 6월

위 원 장	김 종 권	(인)
부위원장	권 태 경	(인)
위 원	최 양 희	(인)
위 원	최 성 현	(인)
위 원	백 상 헌	(인)

Abstract

Cost-Aware Data Offloading with Throughput-Delay Tradeoffs

Youngbin Im
School of Computer Science & Engineering
The Graduate School
Seoul National University

To cope with recent exponential increases in demand for mobile data, wireless Internet service providers (ISPs) are increasingly changing their pricing plans and deploying WiFi hotspots to offload their mobile traffic. However, these ISP-centric approaches for traffic management do not always match the interests of mobile users. Users face a complex, multi-dimensional tradeoff between cost, throughput, and delay in making their offloading decisions: while they may save money and receive a higher throughput by waiting for WiFi access, they may not wait for WiFi if they are sensitive to delay. To navigate this tradeoff, we develop AMUSE (Adaptive bandwidth Management through USer-Empowerment), a functional prototype of a practical, cost-aware WiFi offloading system that takes into account a user's throughput-delay tradeoffs and cellular budget constraint. Based on predicted future usage and WiFi availability, AMUSE decides which applications to offload to what times of the day. Since nearly all traffic flows from mobile devices are TCP flows, we introduce a new receiver-side bandwidth allocation mechanism to practically enforce the assigned rate of each TCP application. Thus, AMUSE users can optimize their bandwidth rates according to their own cost-throughput-delay tradeoff without relying on support from different apps' content servers. Through a measurement study of 20 smartphone users' traffic usage traces, we observe that though users already offload a large amount of some application types, our framework can offload a significant additional portion of users' cellular traffic. We implement AMUSE on Windows 7 tablets and evaluate its effectiveness with 3G and WiFi usage data obtained from a trial with 37 mobile users. Our results show that AMUSE improves user utility; when compared with AMUSE, other offloading algorithms yield 14% and 27% lower user utilities for light and heavy users, respectively. Intelligently managing users' competing interests for cost, throughput, and delay can therefore improve their offloading decisions.

Keywords : Bandwidth management, mobile data, WiFi offloading

Student Number : 2007-21064

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent unprecedented increases in demand for mobile data traffic have begun to stress many mobile operators' networks: Cisco, for instance, predicts that mobile data traffic will grow at 61% annually from 2013 to 2018, reaching 15.9 exabytes per month by 2018 [1]. To cope with this surge in data usage, which is driven by applications such as mobile video, cloud services, and online magazines, many ISPs (Internet service providers) have adopted tiered pricing plans with monthly data caps to discourage heavy usage [2]. To further reduce network traffic, many ISPs have also introduced supplementary networks such as WiFi hotspots or femtocells to offload their cellular traffic [3–5]. Such supplementary offerings introduce new challenges for users as they decide which parts of their traffic can be offloaded at what times.

1.1 Empowering User Decisions

Many data plans, especially in the U.S., charge large overage fees when users exceed a monthly usage cap. While offloading to WiFi reduces cellular data usage, thus saving users money on their data spending, they must also take into account WiFi's intermittent availability and higher throughput performance. At some times, e.g., while out shopping, a user does not have immediate WiFi access and must wait for WiFi connectivity. The user then faces a choice:

- *Don't wait for WiFi:* The user must consume cellular data, using up some of his data cap, and may experience lower throughput than WiFi. However, she need not wait for data access, which is important for urgent applications, e.g. email.

- *Wait for WiFi:* The user can save money and experience higher throughput, but must decide *how long* to wait. Different applications can wait for different periods of time, e.g., cloud backups might be more delay tolerant than photo uploads to Facebook. Given each app's willingness to wait for some period of time, users must anticipate whether WiFi will be available at that time and decide whether the potential savings in data offloading and potential increase in throughput are worth the wait.

  Waiting for WiFi also introduces the risk that apps waiting for WiFi must share the limited 3G bandwidth, should WiFi ultimately not be available. Some apps, such as videos, will require a large amount of bandwidth; their quality can suffer significantly if they must share bandwidth with other apps, e.g., cloud backups.[1]

Most users will not manually balance these competing factors in making offloading decisions. Thus, we propose a user-side, automated WiFi offloading system called AMUSE (Adaptive bandwidth Management through USer EMpowerment) that intelligently navigates these tradeoffs for the user. AMUSE utilizes WiFi access and application usage predictions to decide how long application sessions should wait for

[1]While our systems apply to any form of cellular data, e.g., 3G or LTE networks, we frame our discussion in terms of 3G data. LTE speeds can exceed WiFi, which makes the users' tradeoffs more complicated and AMUSE even more useful.

WiFi and, in case WiFi is not available, to optimally allocate 3G bandwidth among different apps. Building such a system poses both algorithmic and implementation challenges–not only must the user's tradeoff between cost, throughput quality, and delay be quantified and balanced, but we require a way to automatically enforce AMUSE's waiting for WiFi and sharing of 3G bandwidth. In solving these problems, we make the following contributions:

1. We develop a system for cost-aware WiFi offloading that exploits a user's delay tolerances for different applications and makes offloading decisions satisfying her throughput-delay tradeoffs and 3G budget constraints.

2. To enforce AMUSE's bandwidth allocation decisions for each application, we implement a practical receiver-side rate control algorithm for TCP.[2] The algorithm is fully contained on and driven by end-user devices, making it suitable for practical deployment as it requires no modification of the TCP server side.

3. In order to analyze current mobile offloading patterns and the potential to offload more traffic from different apps, we conduct a measurement study using application usage data collected from 20 Android smartphone users for one week.[3] The results reveal several facts that show offloading practice and possibility of smartphone users.

4. We surveyed 100 participants in the U.S. to evaluate users' tradeoff between the cost of 3G usage and their willingness to wait for WiFi access. We incorpo-

[2]We assume that download traffic makes up most of users' usage, so that the receiver is synonymous with the user.

[3]Throughout this work, "app usage data" refers to the volume of data used by each application, not the time duration of application usage.

rate the resulting cost-throughput-delay tradeoff estimates into our model, and evaluate AMUSE's performance using these results and 3G and WiFi usage data collected from a trial with 37 mobile users.

AMUSE is the first WiFi offloading system to fully account for cost, delay, and throughput in offloading traffic from 3G to WiFi. Other works have considered using WiFi offloading to reduce cost within a basic delay constraint, e.g., by using predictions of WiFi connectivity to improve offloading [6] or allocating more WiFi bandwidth to users who are expected to leave the WiFi coverage area in a short amount of time [7]. Mobility can also enhance prefetching data over WiFi [8]. Wiffler [9] considers a more sophisticated model of different applications' delay tolerances, but does not consider different apps' bandwidth needs or their need to share 3G bandwidth.

To fully incorporate cost, delay, and throughput, we build an end-to-end mobile offloading system. In the next section, we describe AMUSE's components and the challenges of developing this end-user system.

1.2 Components of AMUSE

Figure 1.1 gives an overview of AMUSE's components and their interactions. The system architecture comprises four main modules: the User Interface, Bandwidth Optimizer, TCP Rate Controller, and App-Level Session Tracker. The latter two modules reside in the kernel and are accordingly shaded darker in the control flow diagram (Fig. 1.1b); these enforce the offloading decisions made by the User Interface and Bandwidth Optimizer, which reside in the user-space. To illustrate the system's full set of interactions, the Bandwidth Optimizer is split into three components: two prediction modules for app usage and WiFi availability, and an algorithm that computes utility-maximizing offloading decisions.

1.2.1 User Interface

As suggested by its name, AMUSE's User Interface interacts directly with the user, displaying the offloading decisions made as well as the user's app-level usage history. The user may also set her preferences on the user interface, e.g., the maximum budget for 3G usage and delay tolerances for different applications.

1.2.2 Bandwidth Optimizer

The Bandwidth Optimizer makes offloading decisions for the user, given the preferences set by the user on the User Interface. It consists of the three medium-shaded components in Fig. 1.1b: app usage prediction, WiFi access prediction, and a utility maximization algorithm.

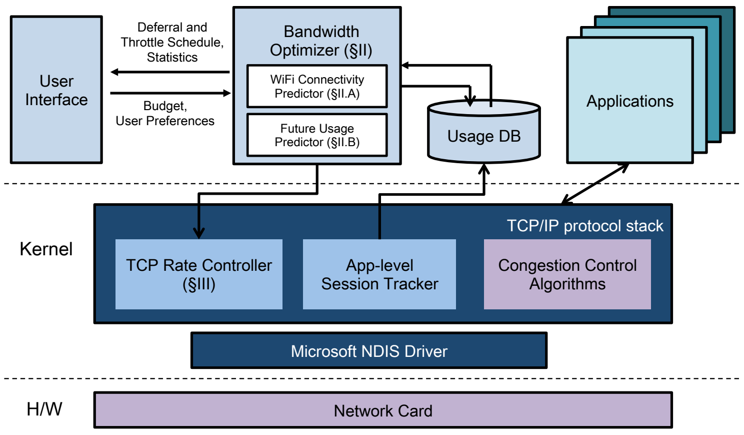
AMUSE uses an adaptive user mobility model to predict WiFi availability at future times (Fig. 1.1b). The app usage prediction component allows AMUSE to calculate the expected savings from offloading an application session and to allocate 3G bandwidth to all active apps at any given time, giving more bandwidth to the apps with higher bandwidth requirements.

The user's offloading decisions at any given time must take into account *future* offloading decisions–for instance, a myopic algorithm may delay all sessions in the morning to 12 noon, if the probability of WiFi access at that time is high. However, should WiFi not be available then, all of the delayed sessions would have to share the limited 3G bandwidth, or else wait even longer for WiFi. Thus, at the beginning of each day, AMUSE optimizes over the entire rest of the day, using estimates of WiFi access probabilities and the size of application sessions at different times (e.g., hours). It then refines this initial solution over the day to reflect the observed usage and WiFi access patterns.

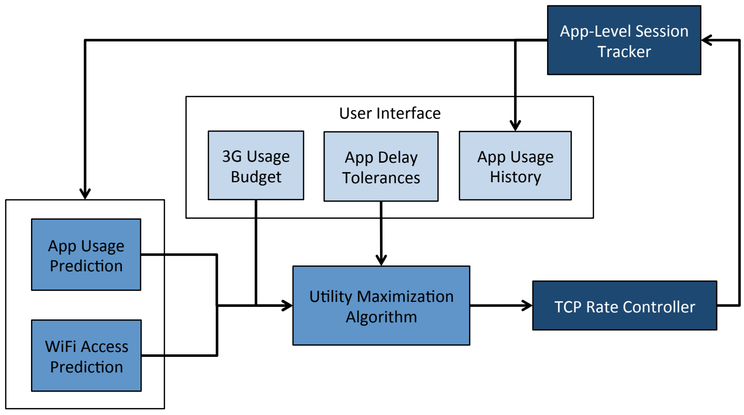
1.2.3 TCP Rate Controller and Session Tracker

Since 99.7% of mobile traffic flows are TCP, we enforce the Bandwidth Optimizer's 3G bandwidth allocations and offloading decisions with a TCP rate controller on end-user devices [10]. To do so, the controller modifies the TCP advertisement window in outgoing acknowledgement (ACK) packets. Unlike typical bandwidth throttling mechanisms, this rate control is completely specified by the *end user* on a *per-application* basis; thus, for example, file downloads may be delayed to wait for WiFi, while streaming videos may receive a higher 3G bandwidth and not be delayed. The app-level session tracker measures the actual usage for each application as the rate controller enforces the Bandwidth Optimizer's decisions. These usage data are then used to update AMUSE's prediction modules, as shown in the control flow diagram (Fig. 1.1b), and are displayed to the user on the User Interface.

In Chapter 2, we discuss prior works that propose functions related to components of the AMUSE system. Chapter 3 discusses the Bandwidth Optimizer in more detail, while Chapter 4 gives an overview of the TCP Rate Controller's algorithm and implementation. In Chapter 5, we observe mobile users' wireless network usage pattern from the viewpoint of WiFi offloading, and find how much and which kind of applications are currently offloaded and can be offloaded more. In Chapter 6, we evaluate AMUSE's effectiveness in improving users' experience, utilizing 3G and WiFi data gathered from 37 mobile users. When compared with two representative offloading algorithms (on-the-spot and delayed [11]), we show that AMUSE increases user utility by intelligently managing the cost-throughput-delay tradeoff for heavy and light users. In Chapter 6, we also evaluate the performance of TCP Rate Controller in various scenarios. We discuss the issues that should be considered to apply AMUSE in real world in Chapter 7. Finally, we conclude the paper in Chapter 8.


User Interface
Deferral and Throttle Schedule, Statistics
Budget, User Preferences
Bandwidth Optimizer (§II)
WiFi Connectivity Predictor (§II.A)
Future Usage Predictor (§II.B)
Usage DB
Applications
Kernel
TCP/IP protocol stack
TCP Rate Controller (§III)
App-level Session Tracker
Congestion Control Algorithms
Microsoft NDIS Driver
H/W
Network Card

(a) System implementation architecture.



(b) Control schematic and main modules.

Fig. 1.1. Overview of AMUSE's components.

Chapter 2

Related Work

Recent studies of 3G and WiFi usage traces, e.g. [11] have showed that offloading 3G traffic to WiFi can significantly benefit mobile ISPs. Other systems have demonstrated offloading's benefits for user experience [6, 8, 9]; other works demonstrate that WiFi offloading can benefit both ISPs and users [12] and even generate more revenue for ISPs [13].

Some works have focused on incentivizing users to offload traffic to WiFi. In [14], the authors develop a utility and cost-based formulation to decide the 3G network load that maximizes the user's benefit and apply the decided loads using a modified SCTP implementation in Linux that stripes traffic across multiple interfaces. Win-Coupon [15] takes a slightly different perspective and proposes a reverse-auction scheme to incentivize users to offload their traffic so as to decrease the overall network. Other works, including [16], consider the energy consumption when making an offloading decision. We do not consider energy in this work, but can easily incorporate the battery consumption into our proposed optimization algorithm.

Several research works have analyzed the traffic of smart devices in order to understand their user behavior [17], [18], [19], [20], [21]. In particular, [17] examines users' traffic diversity, relationship to application types, interactivity, and diurnal patterns, while [18] investigates the usage patterns of smartphone apps via network-side measurements. Our work is different from these in that we specifically focus on the potential for WiFi offloading of different applications and incorporate findings on their delay tolerances.

Unlike most offloading works, AMUSE also controls the 3G bandwidth for different applications. AMUSE is unique in using of receiver-side TCP advertisement windows to control application-specific 3G bandwidth from the user side. While several commercial applications (e.g., [22–24]) provide user-side application rate control, most require users to manually specify the desired rates. AMUSE provides automated bandwidth rates and, by conforming to TCP interactions, avoids the TCP timeouts common to existing user-side rate control applications. Although the TCP advertisement window is normally used by the TCP receiver to inform the TCP sender of its available buffer space, other trials [25] have used the advertisement window as a means to control the rate of applications. However, this approach has been mainly applied to the enforcement of different application priorities, rather than direct control of the application rates. Other solutions, such as [26], focus on the edge gateway, rather than the end user.

Chapter 3

Bandwidth Optimizer

In this Chapter, we describe the individual components of AMUSE's bandwidth optimization algorithm. Our design follows two principles: 1) AMUSE's offloading decisions will be implemented in real time on arriving sessions, and 2) AMUSE must use only the data and computational resources available on the end user's device. Thus, we require simple, yet accurate, algorithms to compute concrete offloading decisions that can be communicated directly to the TCP Rate Controller (cf. Fig. 1.1b). In the discussion below, we first introduce practical algorithms to predict WiFi access and application-specific usage (Sections 3.1 and 3.2). We then incorporate these predictions into a mathematical allocation framework in Section 3.3 and propose a heuristic algorithm for computing AMUSE's bandwidth allocations and offloading decisions in Section 3.4.

To consider a user's different delay tolerances on different applications, we group a user's traffic into different application types, e.g., streaming, browsing, and downloads. For practical implementability, we assume that only the most heavily used (e.g., top five) applications are considered, and denote these collectively as a set J. We suppose that the day is divided into n discrete periods of time, e.g., 24 hours, and for each period, we predict both WiFi access and application usage volumes.

Given these predictions, we (i.e., AMUSE) must decide which applications to offload when, subject to a maximum 3G usage budget. By delaying sessions to future periods, users may gain WiFi access and the ability to offload; however, if WiFi is unavailable, the user must send these sessions over 3G, which has a finite bandwidth capacity that must be shared among the different applications. AMUSE therefore computes a 3G bandwidth allocation when deciding whether to wait for WiFi. Following the first principle above, we formulate this decision as a multiple choice knapsack problem, and propose a heuristic solution algorithm.

3.1 Predicting WiFi Connectivity

Since WiFi availability is heavily location-dependent, we predict the probabilities of WiFi access by combining user location prediction with the probabilities of WiFi access at different locations. We define a “location” to be an area with WiFi coverage (e.g., a user’s home). To improve our prediction algorithm’s accuracy, we consider the *functional* availability of WiFi at different locations: while WiFi is always *physically* available at a given location, the user may not access WiFi every time that she is there. For instance, a user may sometimes connect to WiFi at her local Starbucks, but may walk by on weekdays without initiating a connection. These access probabilities also depend on time: when walking by Starbucks in the morning, a user might habitually stop in to have some coffee. We use a training set of empirical WiFi access data to estimate these time-dependent WiFi access probabilities at each location, and modify them as we collect more access data.[1] For a location l, we denote the probability of WiFi access during period k as $v_k(l)$. We use L_k to denote the set of observed locations in period k.

[1]One may refine these calculations by using only weekday or only weekend data, as user mobility will likely differ on weekdays and weekends.

Given the time- and location-specific probabilities $v_k(l)$, we then predict *overall* WiFi access by incorporating predictions of users' future locations. We define w_k to be the overall WiFi probability in period k. We use a second-order Markov chain for the location prediction, which has been shown to be highly accurate [27]. Algorithm 1 summarizes the calculation of overall WiFi access probabilities. We use the notation $p_l^{k+2}(l_k l_{k+1})$ to denote the probability that a user is at location $l \in L_{k+2}$ during period $k+2$, given his locations l_k in period k and l_{k+1} in period $k+1$. To calculate these p^k, we define $N^k(s)$ as the number of times that s is observed, where s is a sequence of locations that ends in period k; the observed location in each period k is denoted by λ_k. We update the p^k values using the empirical probabilities of a user being at location k.

3.2 Predicting Future Usage

At the beginning of each day, we use previous data to predict the size $s_j(k)$ of each application type $j \in J$'s usage in each period k. To accommodate the dependence of session size on the amount of bandwidth allocated, our definitions of session "size" depend on the application: for fixed-volume application sessions such as downloads, in which the volume (MB) does not depend on the available bandwidth, we define the session size as its volume. For fixed-time sessions such as streaming, in which the volume does depend on the bandwidth, we define the size as the time to complete. We use J_v to denote the set of fixed-volume and J_t the set of fixed-time application types. We stress that our prediction algorithms do not depend on the definition of session size; they rely only on users' consistency from day to day. We

Algorithm 1: Computation of WiFi access probabilities over the rest of the day in period i.

```
if $i = 1$ then
    for $k \leftarrow 1$ to $n$ do
        $w_k \leftarrow \sum_{l \in L_k} v_k(l) \frac{N^k(l)}{N}$, $N$ is the number of days of data. // Calculate WiFi
            probabilities for the next $n$ periods.
if $i > 1$ then
    for $k \leftarrow 2$ to $n$ do
        forall the $l \in L_k$, $l_{k-1} \in L_{k-1}$, $l_{k-2} \in L_{k-2}$ do
            if $N^{k-1}(l_{k-2}l_{k-1}) > 0$ then
                $p_l^k(l_{k-2}l_{k-1}) \leftarrow \frac{N^k(l_{k-2}l_{k-1}l)}{N^{k-1}(l_{k-2}l_{k-1})}$
            else
                $p_l^k(l_{k-2}l_{k-1}) \leftarrow \frac{N^k(l_{k-1}l)}{N^{k-1}(l_{k-1})}$
        $w_k \leftarrow \sum_{l \in L_k} p_l^k(\lambda_{k-2}\lambda_{k-1}) v_k(l)$
```

estimate the future usage $s_j(k)$ by taking a moving average of the observed usage sizes $\sigma_j(k)$ of application j in period k over some fixed number of days.[2]

In updating our usage estimates, we modify the moving-average calculation to take into account our deferral recommendations. Since a user may delay application usage to another time in order to offload it to WiFi, we "shift the usage back" in order to evaluate and detect changes in the underlying usage pattern over the day. We perform these adjustments if the observed usage size for application j in period i is much less than the predicted $s_j(i)$, i.e., the user has shifted her usage of application j from period i. To account for the uncertainty in our predictions, we suppose that the actual usage deferred to period k from each period i is proportional to the predicted usage deferred; this assumption ensures that we do not calculate that more usage was deferred to period k than the actual usage observed in that period. Thus, for each

[2]Other prediction methods (e.g., ARIMA) can be substituted for a moving average without affecting the overall structure of our system.

application $j \in J$ and period $i < k$, we adjust the observed usage $\sigma_j(i)$ and $\sigma_j(k)$ by

$$\sigma_j(i) \leftarrow \sigma_j(i) + \sum_{k=i+1}^{n} \frac{c_i^j(k) s_j(i) \sigma_j(k)}{s_j(k) + \sum_{l=1}^{k-1} c_l^j(k) s_j(l)},$$
$$\sigma_j(k) \leftarrow \sigma_j(k) - \sum_{i=1}^{k-1} \frac{c_i^j(k) s_j(i) \sigma_j(k)}{s_j(k) + \sum_{l=1}^{k-1} c_l^j(k) s_j(l)}.$$

Here $c_i^j(k)$ is an indicator variable taking the value 1 if application j is deferred from period i to period k and 0 otherwise.

3.3 User Utility Maximization

In this section, we formulate the user's offloading decision problem, assuming the future WiFi probabilities w_k and usage $s_j(k)$ are known. In the discussion below, the phrase "originally in period i" indicates that the application session(s) under consideration are completed in period i if they are not deferred to a future period.

3.3.1 Utility Functions

To mathematically formulate the user's offloading decision problem, we need a concrete measure of the user's tradeoffs between cost, throughput, and delay. Thus, for a given application type j in period i, we derive expressions for users' *utility* of completing those application sessions over 3G and over WiFi. This utility is determined by the per-volume price p of 3G, the amount of time t the session is deferred, the bandwidth speed r at which the session is completed, and the size s of the session. We use $U_j(p, t, r, s)$ to denote the utility of application $j \in J$.

Though many functions could be used as the U_j, we note that these should be decreasing in p and t (price and time deferred) and increasing in r and s. We use the economic principle of diminishing marginal utility to argue that as s or r becomes larger or t becomes smaller, users' marginal utility from s or r should decrease, and the marginal utility from t should increase. For simplicity, we take the units of t to be the number of periods deferred, and do not consider sessions' timing *within* the period to which they are deferred. Since different users will have different tradeoffs between cost, quality, and delay, we suppose that the U_j functions take the same form, but have different parameters that depend on the particular application and user.

The above guidelines still leave many possible utility functions. To narrow these down, we conducted an online survey of over 100 users, primarily students, faculty and staff from U.S. universities. For each application in Table 3.1, we gave participants the cost to complete one application session over 3G, as well as the speed of WiFi relative to 3G. We then asked the participants how long they would wait for WiFi access instead of immediately completing the session over 3G; for each question, we offered five options for the maximum amount of time participants were willing to wait, ranging from "I won't wait" to "as long as necessary."[3]

We find that our survey data provides a good fit with the functional form

$$\begin{cases} U_j(p,t,r,s) = C_j \exp\left(-\nu + r\nu - \mu t\right) - \eta prs & j \in J_t \\ U_j(p,t,r,s) = C_j \exp\left(-(s/r)\nu - \mu t\right) - \eta ps & j \in J_v, \end{cases} \tag{3.1}$$

where U_j denotes the parameterized utility function for application types j; prs for $j \in J_t$ and ps for $j \in J_v$ denote the cost of each session; and C_j, μ, ν, and η

[3]The survey questions are available in [28].

are nonnegative parameters that depend on j. These functions satisfy several desirable properties: for example, the constants C_j allow for different user priorities for different types of sessions (e.g. a user intrinsically derives more utility from certain applications, even with zero delay or time to completion).[4] For $j \in J_v$, the s/r term in the exponential represents the time to completion, while for $j \in J_t$, the bandwidth r represents the quality of the streaming video.

Table 3.1 shows the parameter values calculated for each session type. To estimate these parameter values, we used the probability that a user would not wait for WiFi as the utility function value, with b, t, r and s measured relative to their values for WiFi. We assume a negligible cost term for low-volume (e.g., email) sessions. We then used nonlinear curve-fitting methods to calculate the utility function parameters, and found a small average squared-error of 0.05 for each survey question, upon comparing the actual answers with our estimates.

We see that the C_j coefficients roughly match our expectations, with email the most important and social networking (photo uploads) the least important applications. The parameter μ represents the amount of time that a user will wait to start an application, e.g., in anticipation of WiFi access or higher 3G speeds: it is largest (i.e., users are least willing to wait) for browsing and email. The importance of available throughput is parameterized by ν, and is highest for video and lowest for social networking.

[4]The $-\nu$ in the exponential for $j \in J_t$ is included for normalization: with maximum bandwidth 1 and no delay, we then have $U_j = C_j$.

Table. 3.1. Estimated parameters for the utility function (3.1).

	C	μ	ν	η
Email	0.9848	0.1527	0.1527	assumed 0
Browsing	0.6865	0.3269	0.0263	assumed 0
Video	0.9399	0.0144	4.3785	0.0986
Social netw.	0.4738	0.006	0.006	0.0986
Downloads	0.6737	0.0097	0.0097	0.0986

3.3.2 Users' Optimization Problem

We now use the utility functions (3.1) to formulate the user's optimization problem. To represent possible 3G and WiFi bandwidth speeds, we normalize the volume units so that the fixed per-second WiFi speed equals 1. The 3G speed γ is chosen from a finite subset of possibilities Γ. For each $\gamma \in \Gamma$ and period $k \geq i$, we define the indicator variables $c_i^j(k,\gamma)$ to be 1 if a session of type j is deferred from period i to period k and assigned 3G speed γ, and 0 otherwise. The user's expected utility from WiFi in period k, for a session of type j originally in period i, is then

$$\left(\sum_{\gamma\in\Gamma} c_i^j(k,\gamma)\right) w_k U_j\left(0, k-i, 1, s_j(i)\right),$$

while the expected utility from 3G in period k is

$$\sum_{\gamma\in\Gamma} c_i^j(k,\gamma)(1-w_k)U_j\left(p, k-i, \gamma, s_j(i)\right).$$

The user wishes to maximize the sum of these utilities over all (original) periods i and application types j:

$$\max_{c_i^j(k,\gamma)} \sum_{i=1}^{n} \left[\sum_{j\in J} \left(\sum_{k\geq i} \left(\sum_{\gamma\in\Gamma} \Big(w_k U_j\big(0, k-i, 1, s_j(i)\big) + \right.\right.\right.$$
$$\left.\left.\left. (1-w_k) U_j\big(p, k-i, \gamma, s_j(i)\big)\Big) c_i^j(k,\gamma) \right)\right)\right] \tag{3.2}$$

$$\text{s.t.} \sum_{k\geq i}\sum_{\gamma\in\Gamma} c_i^j(k,\gamma) = 1;\; c_i^j(k,\gamma) \in \{0,1\}, \tag{3.3}$$

where (3.3) ensures that each application j in period i is deferred to only one period k (we may have $k = i$), with 3G speed γ. This optimization is performed subject to two constraints: a budget constraint on expected 3G usage, and capacity constraints on the 3G bandwidth in each period.

We assume that the user specifies a maximum monthly budget $\overline{B}$ for 3G usage. We then calculate a *daily budget* B, taking into account both the number of days remaining in the month (denoted by m) and the amount of budget B_r that has not yet been spent. To allow the user some flexibility, we multiply the average usage B_r/m by the factor $\exp\left(1 - m^{-1}\right)$, which equals 1 only if $m = 1$: at the end of the month, the user cannot exceed the remaining budget. The daily budget B is then defined as $B_r \exp\left(1 - m^{-1}\right)/m$, and the budget constraint is

$$p\sum_{i=1}^{n} \left[\sum_{j\in J_v} \left(\sum_{k\geq i}\sum_{\gamma\in\Gamma} c_i^j(k,\gamma)(1-w_k)s_j(i) \right) + \right.$$
$$\left. \sum_{j\in J_t} \left(\sum_{k\geq i}\sum_{\gamma\in\Gamma} c_i^j(k,\gamma)(1-w_k)\gamma s_j(i) \right) \right] \leq B, \tag{3.4}$$

The 3G bandwidth capacity constraints ensure that the sum of the bandwidth allocated to each application in a given period does not exceed the fixed maximum bandwidth, which we denote as β. Mathematically, this constraint is

$$(1-w_l)\sum_{i\leq l}\sum_{j\in J}\sum_{\gamma\in\Gamma} c_i^j(l,\gamma)\gamma \leq (1-w_l)\beta. \tag{3.5}$$

We include a $1-w_l$ term on each side so that if $w_l = 1$ and all sessions complete over WiFi, any 3G speed γ may be chosen. Putting (3.2-3.5) together, we obtain the optimization problem

$$\begin{aligned}
&\max_{c_i^j(k,\gamma)} \sum_{i=1}^{n}\left[\sum_{j\in J}\left(\sum_{k\geq i}\left(\sum_{\gamma\in\Gamma}\Big(w_k U_j\big(0,k-i,1,s_j(i)\big)+\right.\right.\right.\\
&\qquad\qquad \left.\left.\left.(1-w_k)U_j\big(p,k-i,\gamma,s_j(i)\big)\Big)c_i^j(k,\gamma)\right)\right)\right] && (3.6)\\
&\text{s.t. } p\sum_{i=1}^{n}\left[\sum_{j\in J_v}\left(\sum_{k\geq i}\sum_{\gamma\in\Gamma} c_j^i(k,\gamma)(1-w_k)s_j(i)\right)+\right.\\
&\qquad\qquad \left.\sum_{j\in J_t}\left(\sum_{k\geq i}\sum_{\gamma\in\Gamma} c_j^i(k,\gamma)(1-w_k)\gamma s_j(i)\right)\right]\leq B && (3.7)\\
&(1-w_l)\sum_{i\leq l}\sum_{j\in J}\sum_{\gamma\in\Gamma} c_j^i(l,\gamma)\gamma \leq (1-w_l)\beta\,\forall\, l && (3.8)\\
&\sum_{k\geq i}\sum_{\gamma\in\Gamma} c_j^i(k,\gamma) = 1\,\forall\, j\in J;\ i=1,2,\ldots,n && (3.9)\\
&c_j^i(k,\gamma)\in\{0,1\}.
\end{aligned}$$

We can view the constraints (3.9) as choosing exactly one item from a knapsack, where each (i,j) pair for $i = 1,2,\ldots,n$ and $j \in J$ is associated with a knapsack consisting of items indexed by the variables $k \geq i$ and $\gamma \in \Gamma$. With this interpretation, (3.6-3.9) can be seen as a multidimensional, multiple choice knapsack problem.

3.4 Online Algorithm

In this section, we present an online algorithm to solve the optimization problem (3.6-3.9). At the beginning of each day, the user computes an initial solution, given estimates of the w_k and $s_j(k)$. As the w_k estimates and known usage amounts are updated over the day, this solution is refined.

While various algorithms exist to compute solutions of the knapsack problem (3.6-3.9) to different degrees of accuracy, we use a Lagrange-multiplier based solution [29] that has relatively small computational overhead and generally returns good approximations to the optimum.[5] Given a feasible solution, the algorithm improves the solution while maintaining its feasibility, allowing us to easily update previously computed solutions over the day.

This Lagrange multiplier algorithm starts from a solution that maximizes (3.6) without considering the constraints (3.7-3.9). The solution is then adjusted so that all constraints are satisfied, beginning with the "most violated" (i.e., the constraint with largest Lagrange multiplier). This process repeats until no constraints are violated, and the solution can then be improved by adjusting the solution one variable at a time, so as to most increase the objective value while still not violating the constraints.[6]

[5]Since our estimates of WiFi access and future usage are already approximations, even an exact solution to the optimization (3.6-3.9) will be an approximation of the "true" optimum.

[6]If the constraints are especially tight, the Lagrange multiplier algorithm may not yield a solution. While in practice such a situation is unlikely to occur, we can easily recover from this failure by taking as the initial allocation the worst-case scenario, in which all sessions are given the lowest possible bandwidth; we assume that this is a feasible solution.

Algorithm 2: Bandwidth allocation over a day.

```
$i \leftarrow 1$ // The current period is denoted by $i$.
$B \leftarrow (B_r/m)\exp\left(1 - m^{-1}\right)$ // Calculate the budget for the day.
Calculate WiFi probabilities using Algorithm 1 with $i = 1$.
Calculate predicted usage over all $n$ periods using a moving average.
Allocate bandwidth by approximately solving (3.6-3.9).
for $i \leftarrow 2$ to $n$ do
    $B \leftarrow B - S_{i-1}$ // Remaining daily budget, given the spending $S_{i-1}$ in
        period $i - 1$.
    Update WiFi probabilities using Algorithm 1.
    Update bandwidth allocations by re-solving (3.6-3.9) for the remaining $n - i + 1$ periods.
```

As the user consumes data over the day, we update both the remaining daily budget B and our predictions of future WiFi connectivity $\{w_k\}$. The new optimization problem over the remainder of the day can then be solved by taking the existing solution as the initial point of our Lagrange multiplier algorithm. We note that this solution may well be feasible: the 3G capacity constraints do not change unless WiFi becomes definitely available in some period ($w_k \to 1$), in which case that period k's capacity constraint is removed. Thus, if the existing solution satisfies the new budget constraint, we can skip directly to the "solution improvement" step, which significantly reduces the computational overhead. Algorithm 2 presents this full online algorithm, along with the WiFi and app usage predictions (Sections 3.1 and 3.2).

Chapter 4

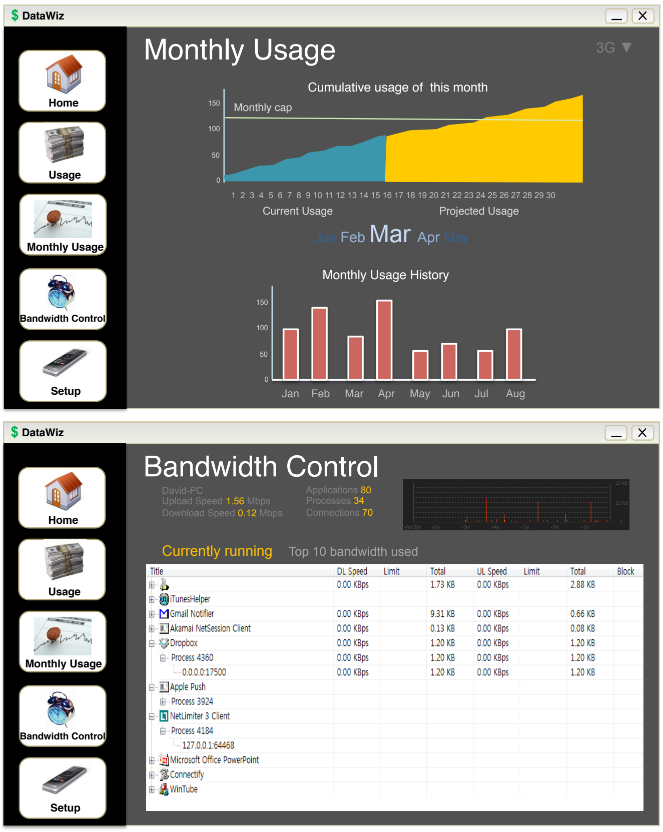
Implementation

We implemented an AMUSE prototype on Windows 7 tablets with the system architecture shown in Fig. 1.1a. We used the Windows Filtering Platform (WFP) [30] to track application usage and implement a user-side TCP rate control algorithm to control each application's download rate.

The AMUSE prototype displays both total usage and the usage of individual applications on a daily, weekly, and monthly basis, as well as the current upload and download rates. We also provide user interfaces from which the user can, if he so chooses, set the download rate of each application and configure his billing starting date and data plan (e.g., 2GB per month). Figure 4.1 shows screenshots of these features.

For the evaluation of our user-side TCP rate control algorithm in Section 6.2, we also implemented the algorithm on Ubuntu 14.04 LTS. We utilized *libnetfilter_queue* [31], a userspace library providing an API to packets that have been queued by the kernel packet filter. Using *libnetfilter_queue*, we can modify the packet in userspace.

We next describe the details of our receiver-side TCP rate control algorithm, and experimental data verifying its efficacy is presented in Section 6.2.



Title	DL Speed	Limit	Total	UL Speed	Limit	Total	Block
	0.00 KBps		1.73 KB	0.00 KBps		2.88 KB	
iTunesHelper							
Gmail Notifier	0.00 KBps		9.31 KB	0.00 KBps		0.66 KB	
Akamai NetSession Client	0.00 KBps		0.13 KB	0.00 KBps		0.08 KB	
Dropbox	0.00 KBps		1.20 KB	0.00 KBps		1.20 KB	
Process 4360	0.00 KBps		1.20 KB	0.00 KBps		1.20 KB	
0.0.0.0:17500	0.00 KBps		1.20 KB	0.00 KBps		1.20 KB	
Apple Push							
Process 3924							
NetLimiter 3 Client							
Process 4184							
127.0.0.1:64468							
Microsoft Office PowerPoint							
Connectify							
WinTube							

Fig. 4.1. Screenshots of the AMUSE prototype. Users can view their monthly usage and set bandwidth rates for individual applications.

4.1 Receiver-Side TCP Rate Control

A TCP sender adjusts a session's rate based on its congestion window size ($cwnd$). The ACK packets from the receiver act as a feedback to the sender on how much has been sent and how much more can be sent to the receiver. We use this ACK clocking to shape the incoming/downloading rates of TCP traffic, by modifying the TCP advertisement window size (rcv_wnd) field in each ACK packet using the WFP driver (*libnetfilter_queue* in Linux). The idea behind this approach is that a

TCP sender cannot send more than $\min(cwnd, rcv\_wnd)$. While one could instead shape the TCP traffic rates by adjusting the round-trip time (RTT) of each flow (i.e., stretching each ACK packet), this latter approach increases the overall response time and renders some interactive or video TCP applications useless. Modifying the advertisement window size does not increase the RTT of each flow, making it suitable for all TCP applications. We verify this with a experiment in Section 6.2.

Unlike current traffic control tools, our proposed control mechanism does not forcibly drop incoming packets, a measure that can induce such undesirable side effects as frequent TCP timeouts. The principle behind our mechanism is as follows: *we increase the size of the advertisement window if the traffic rate recently received is smaller than the target bandwidth, and decrease it if the traffic rate recently received is larger than the target bandwidth.* With this approach, we can implement the bandwidth control entirely at the TCP receiver. The sender is not modified, but it will react to the advertisement window from the receiver according to TCP flow control.[1]

Algorithm 3 presents the pseudo code of our implementation. The algorithm first initializes the adv_wnd to the default value (256 bytes) when the connection is set up, and periodically calculates the traffic throughput for each application.[2] The throughput is obtained by dividing the received bytes ($bytes$) by the interval length ($interval$). Since we initialize the received bytes if $interval$ becomes larger than $rate\_control\_win$, we calculate the throughput of maximum $rate\_control\_win$ period. If the throughput for a given period is smaller than the target bandwidth ($target\_BW$), we increase the advertisement window size by an amount (inc) pro-

[1]In order to not hurt a user's response time with short-lived TCP flows, the algorithm only runs after 5 secs, during which these short-lived flows can complete their transfers.

[2]We set this period to RTT, since the effect of change in the advertisement window will be detected only after one RTT.

Algorithm 3: Receiver-side TCP rate control.

```
Initialization:
  $target\_BW \leftarrow$ // Desired bandwidth (bytes/sec)
  $min\_adv\_win \leftarrow 2\ (bytes)$
  $adv\_win \leftarrow 256\ (bytes)$
  $last\_check\_time \leftarrow current\_time\ (sec)$
  $throughput\_check\_time \leftarrow current\_time\ (sec)$
  $rate\_control\_win \leftarrow 1\ (sec)$
  $bytes \leftarrow 0\ (bytes)$
// accumulated received bytes for current period
  $\alpha \leftarrow 0.1$ // smoothing factor
For each TCP data and ACK packet:
begin
    if a data packet is received then
        $bytes \leftarrow bytes + packet\_len$
        if $current\_time - last\_check\_time > RTT$ then
            $interval \leftarrow current\_time - throughput\_check\_time$
            $throughput \leftarrow bytes/interval$
            $inc \leftarrow adv\_win * \frac{target\_BW - throughput}{throughput} * \alpha$
            $adv\_win \leftarrow adv\_win + inc$
            if $adv\_win > rcv\_buf\_size$ then
                $adv\_win \leftarrow rcv\_buf\_size$
            else if $adv\_win < min\_adv\_win$ then
                $adv\_win \leftarrow min\_adv\_win$
            $last\_check\_time \leftarrow current\_time$
            if $interval > rate\_control\_win$ then
                $bytes \leftarrow 0$
                $throughput\_check\_time \leftarrow current\_time$
    if an ACK packet is ready to be sent then
        set the advertisement window of the ACK to $adv\_win$
```

portional to the deficit throughput. Similarly, if the throughput is larger than the target bandwidth, we decrease the size of the advertisement window by an amount proportional to the surplus throughput. Depending on the increase/decrease of the advertisement window, the TCP sender will increase or decrease the rate of the traffic accordingly, assuming its congestion window is mostly larger than its advertisement window. Here, we multiply the deficit/surplus bandwidth by a ratio α, in order to reduce the oscillation of throughput due to the drastic window size changes. We use $\alpha = 0.1$ after experimentally determining this value's efficacy in achieving the target

Algorithm 4: Calculation of round trip time.

```
Initialization:
  $last\_timestamp \leftarrow 0$
  $timestamp\_sent\_time \leftarrow 0$
  $RTT \leftarrow 0$
  $rttVar \leftarrow 0$
For each TCP data and ACK packet:
begin
    if a packet is received then
        $echo$ ←TSecr of the data packet's timestamp option
        if $last\_timestamp \neq 0$ and $echo \geq last\_timestamp$ then
            $interval = current\_time - timestamp\_sent\_time$
            if $RTT = 0$ then
                $RTT \leftarrow interval$
                $rttVar \leftarrow RTT/2$
            else
                $rttVar \leftarrow rttVar * 3/4 + abs(RTT - interval)/4$
                $RTT \leftarrow RTT * 7/8 + interval/8$
            $last\_timestamp \leftarrow 0$
            $timestamp\_sent\_time \leftarrow 0$
    if a packet is ready to be sent then
        $last\_timestamp$ ← TSval of the ACK packet's timestamp option
        $timestamp\_sent\_time \leftarrow current\_time$
```

bandwidth in several different environments. We prevent the advertisement window size from moving above the maximum buffer size (rcv_buf_size) and below minimum window size (min_adv_win).

To calculate the RTT at TCP receiver, we devised Algorithm 4 whose approach is similar to [32]. The approach in [32] is for estimation of the RTT in the middle of the end-to-end path, while ours is a receiver-side RTT estimation method. We record the timestamp value and sending time when a TCP timestamp option is sent. Then when the echo of the timestamp sent is incoming, we calculate the interval between the timestamp sending time and echo reception time. Using this interval we update the RTT and RTT variation estimations according to the calculation method used in TCP standard [33].

Chapter 5

Measurement

We conduct a measurement study in order to analyze mobile users' network usage pattern from the viewpoint of WiFi offloading. Specifically, in Section 5.2 we show that the application types considered in Table 3.1 comprise a large portion of users' cellular traffic. In Section 5.3, we examine the degree to which these applications are already offloaded and their potential for more offloading.

5.1 Data Collection

To collect empirical traffic data for the measurement study in this chapter, we recruited smartphone users to participate in our trial. We recorded the data by implementing a usage monitoring app and installing it on users' phones. Figure 5.1 shows the screenshots of the usage monitoring app, which informed users of their overall usage over different timescales and usage at different geographical locations.

We collected application usage data from 20 Android smartphone users in Alaska for 7 days, including application package names and categories and upload and download usage amounts for each application in bytes. To compare AMUSE to other offloading algorithms in Chapter 6, we also collected another data set from an additional 12 Android users and 25 iPhone users in the U.S. This second dataset includes participants' 3G and WiFi usage, WiFi availability, and user locations at a ten minute

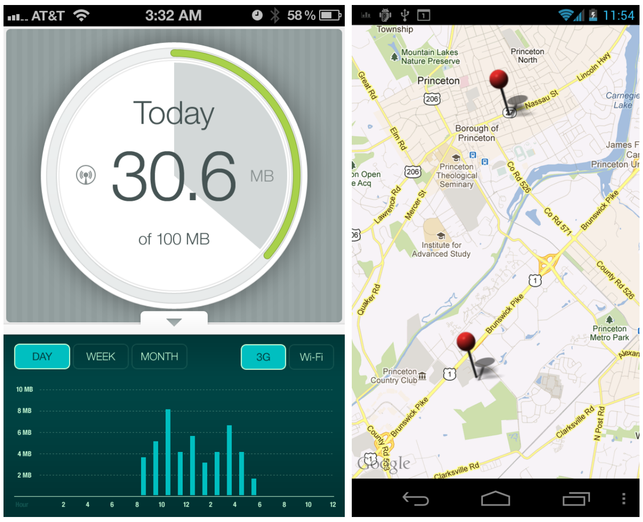


Fig. 5.1. Screenshots of the usage monitoring app.

granularity.[1]

5.2 Application types

[1]We do not collect iPhone application usage data due to iOS implementation restrictions.

Table. 5.1. Top 15 applications in WiFi network.

Index	Package name	Upload(%)	Download(%)	Total(%)	Type
1	Streaming Media	70.20	66.38	68.14	Video
2	android.process.media	3.28	3.26	3.27	Video
3	com.google.android.music:main	3.22	2.77	2.98	Unclassified
4	com.google.android.music:ui	3.20	2.75	2.96	Unclassified
5	com.android.email	2.46	2.59	2.53	Email
6	com.emogoth.android.phone.mimi	2.67	2.29	2.46	Social networking
7	com.marvel.capinstaller	1.77	1.52	1.63	Downloads
8	com.android.browser	1.23	1.69	1.48	Browsing
9	com.clearchannel.iheartradio.controller	1.48	1.27	1.37	Unclassified
10	com.facebook.katana:providers	1.07	1.30	1.19	Social networking
11	com.facebook.katana	0.93	1.31	1.13	Social networking
12	com.motorola.process.system	0.05	2.04	1.12	Unclassified
13	com.rhythmnewmedia.android.e	0.81	1.10	0.97	Unclassified
14	com.ninegag.android.app	0.69	0.65	0.67	Unclassified
	Others	6.93	9.10	8.10	-

Table. 5.2. Top 15 applications in cellular network.

Index	Package name	Upload(%)	Download(%)	Total(%)	Type
1	Streaming Media	5.70	42.64	33.77	Video
2	com.android.email	5.64	5.59	5.60	Email
3	android.process.media	4.03	5.96	5.50	Video
4	com.facebook.katana	7.47	3.86	4.73	Social networking
5	com.android.browser	6.78	3.33	4.16	Browsing
6	com.google.android.music:main	0.04	5.27	4.01	Unclassified
7	com.facebook.katana:providers	6.30	2.51	3.42	Social networking
8	com.rhythmnewmedia.android.e	5.26	1.99	2.78	Unclassified
9	com.pandora.android	4.69	1.49	2.26	Unclassified
10	com.noinnion.android.greader.reader	4.42	1.40	2.12	Unclassified
11	com.motorola.blur.service.main	2.89	1.00	1.45	Unclassified
12	com.motorola.contacts	2.89	0.94	1.41	Unclassified
13	com.alphonso.pulse	2.17	1.07	1.33	Social networking
14	com.motorola.process.system	0.35	1.64	1.33	Unclassified
15	com.clearchannel.iheartradio.controller	1.07	1.25	1.21	Unclassified
	Others	40.28	20.08	24.93	-

In Chapter 3, we classify user's traffic into 5 application types (i.e. Email, Browsing, Video, Social networking, Downloads). In this section, we verify that these application types comprise most of users' traffic by volume, indicating that AMUSE covers most cellular traffic. In Tables 5.1 and 5.2, we list the top 15 applications for WiFi and cellular networks, respectively. To identify the application types, we manually searched for the package names in the Android application market and used the application descriptions there. Packages not found in the Android application market were classified using the name itself (e.g. com.android.email is classified as "Email"). Package names that cannot be identified using these methods are designated as "Unclassified." In the case of cellular network, the top five applications correspond to the application types in Table 3.1, accounting for 54% of the total cellular traffic. From these observations, we can conclude that our offloading mechanism can handle a large portion of mobile users' cellular traffic, demonstrating the possibility of significantly reducing the data cost.

5.3 Offloading practice

From users' usage traces, we find that users are already offloading a large portion of their traffic, but a large amount of video and social networking cellular traffic still runs over WiFi and can be delayed for offloading. In Figure 5.2, we illustrate the amount of upload and download traffic for each application type in cellular and WiFi networks. As expected, the amount of WiFi traffic is much larger than that of cellular, and the amount of download traffic is larger than the upload traffic.

As in [1], video traffic accounts for the largest portion of traffic for WiFi up-

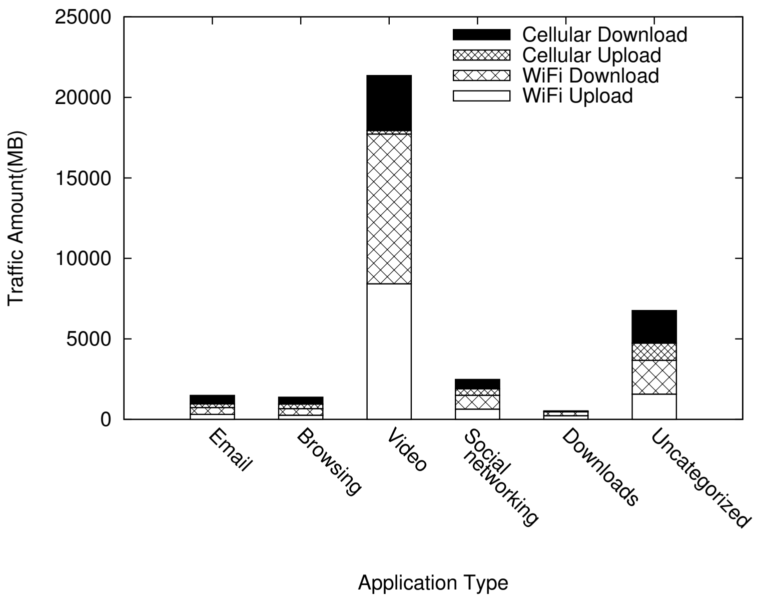


Fig. 5.2. Traffic amounts for each category and network type.

loads/downloads and cellular downloads (74, 70, 49% respectively). For these types of traffic, the order of the application types according to the traffic amount is Video, Social networking, Email, Browsing, and Downloads. In the case of cellular upload, the traffic amount is in the order of Social networking-Browsing-Email-Video-Downloads. We find that 84% of the total upload and 66% of the total download traffic uses WiFi.

To investigate the fraction of each application's traffic that is offloaded to WiFi networks, we calculate the ratio of WiFi to cellular upload and download traffic for all the application types, as shown in Figure 5.3. If this ratio is large, it means that the users already offload their traffic to WiFi for that application type, either because that application type is delay-tolerant or because it requires WiFi's high bandwidth (e.g., video applications).

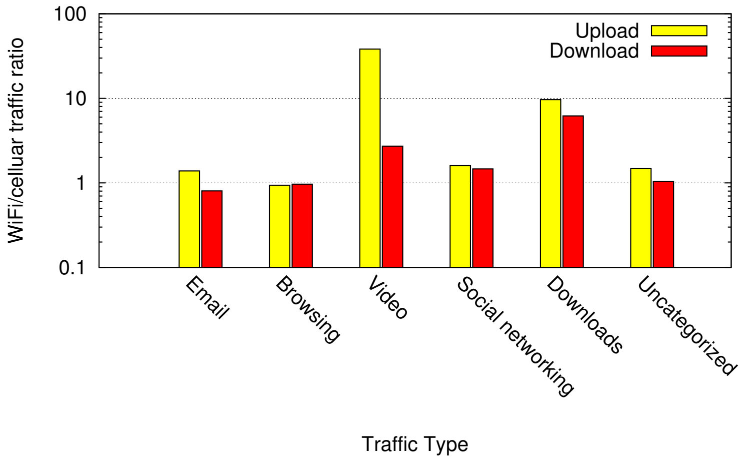


Fig. 5.3. The ratio of traffic WiFi to cellular network traffic for each application type.

From Figure 5.3, we see that different application types show different ratios. In particular, the Video and Download types show large values for both upload and download traffic. This coincides with the small values of μ in Table 3.1 for these application types. Video requires high bandwidth and has high cost generally, incentivizing users to wait for WiFi in order to use high bandwidth. However, by comparing the ratios for Video and Downloads, we observe that users wait more for Video than for Downloads. While a significant fraction of video and downloads are offloaded, the high delay tolerance of Download apps indicates that more offloading is possible.

We also see that the Email and Social networking applications have some offloading potential. Figure 5.4 shows CDFs for the number of periods in which a user uses E-mail and Social networking applications. We observe that about 40% of users use WiFi data for Email and Social networking applications 30% of the time. Over cellular, the data usage frequency decreases, but 20% of users spend significant

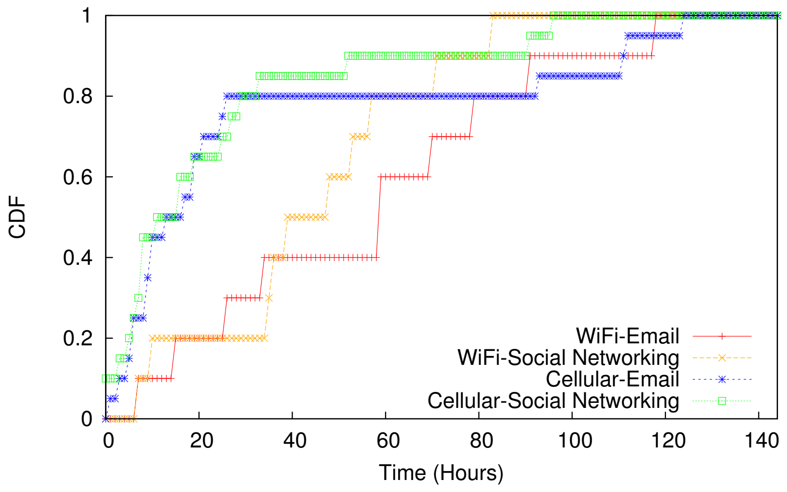


Fig. 5.4. CDF of the number of periods for which each user uses Email and Social networking applications.

amounts of time on email and social networking. This high usage frequency indicates that some Email and Social Networking traffic can be offloaded if a user does not need to wait very long for WiFi access.

Chapter 6

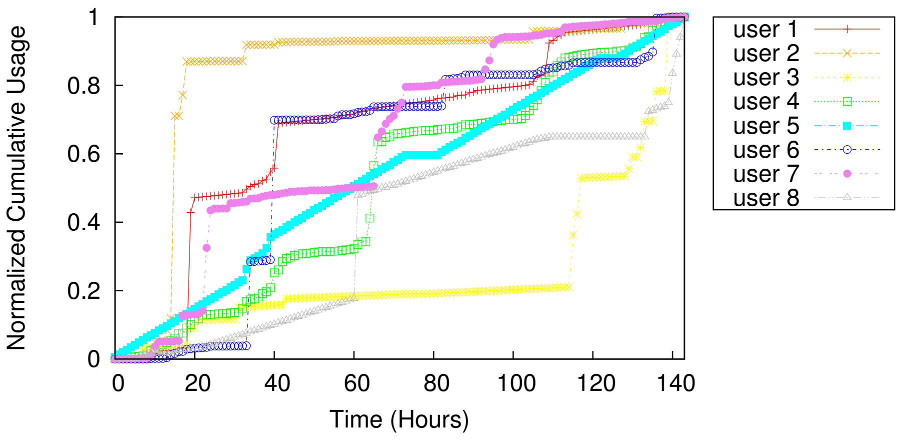
Experimental Evaluation

6.1 Bandwidth Optimizer

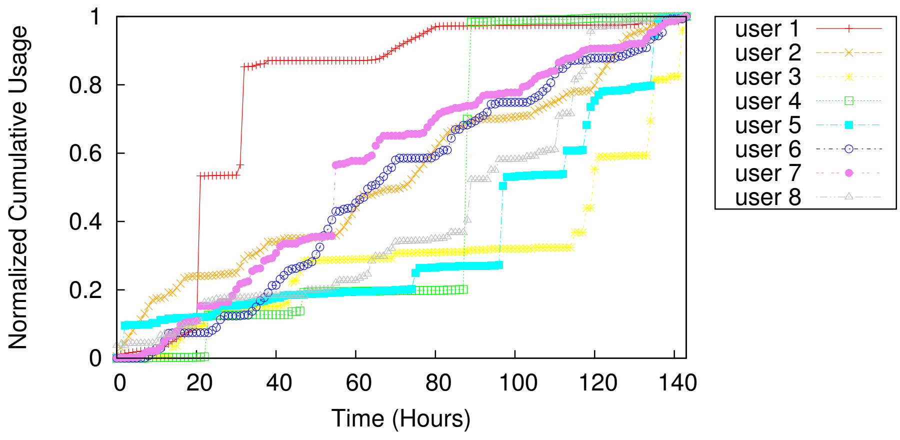
To evaluate the effects of AMUSE's Bandwidth Optimizer (Algorithm 2 in Chapter 3) on users' offloading experience, we collected 3G and WiFi usage and mobility data from an additional 12 Android and 25 iPhone users over a period of 19 days and one week, respectively, as explained in Section 5.1. We then simulate the performance of AMUSE's bandwidth optimizer, taking the recorded usage data as the historical usage, and compare AMUSE's performance with two other known offloading algorithms. Our results show that AMUSE can both reduce users' spending and improve their utility compared with these two benchmarks.

6.1.1 Experimental Data and Settings

Since some of our users exhibited very similar traffic patterns and some showed very limited data usage, we choose sixteen representative users' data on which to run the AMUSE simulation (eight each for iPhone and Android). Figures 6.1a and 6.1b represent the normalized cumulative usage of selected iPhone users for cellular and WiFi, respectively. The normalized cumulative usage of selected Android users for cellular and WiFi networks are shown in Figure 6.2a and 6.2b. We can observe that the chosen representative users show diverse usage patterns in both WiFi and

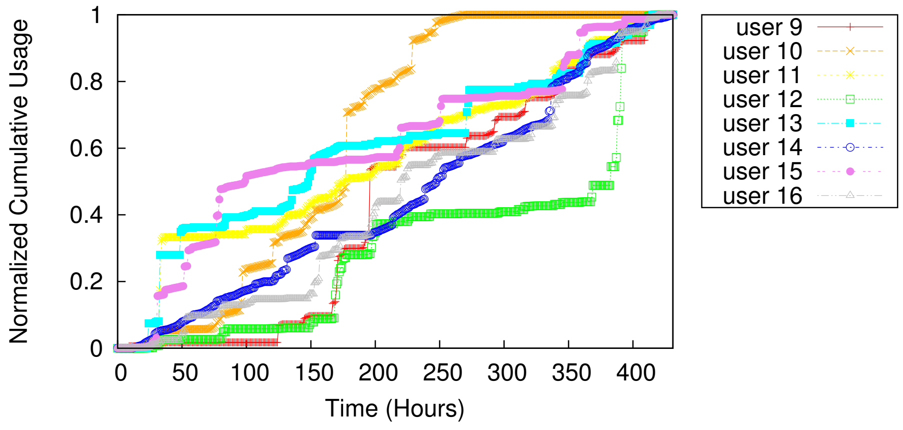


(a) Cellular usage.

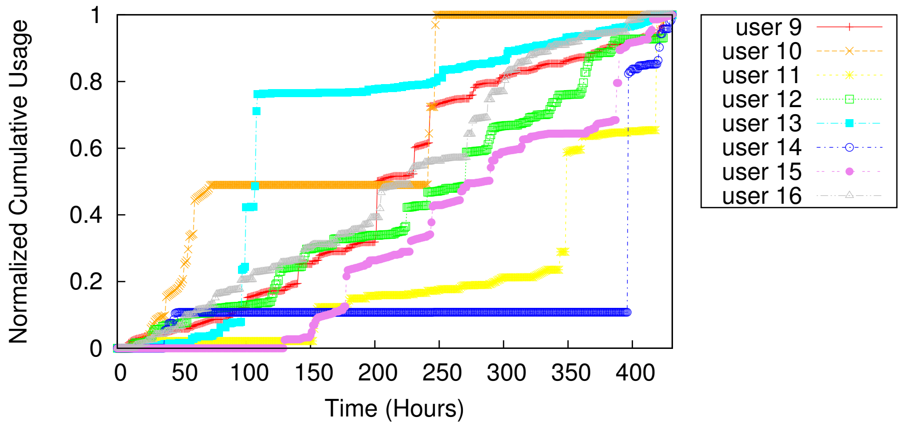


(b) WiFi usage.

Fig. 6.1. Normalized cumulative usage of iPhone users.



(a) Cellular usage.



(b) WiFi usage.

Fig. 6.2. Normalized cumulative usage of Android users.

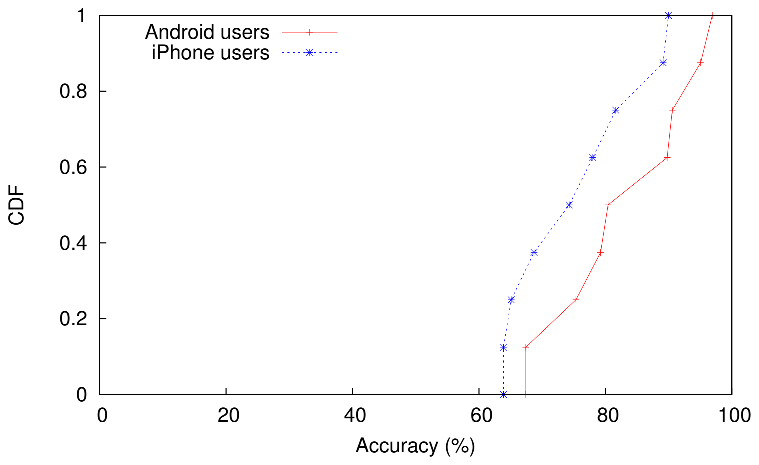


Fig. 6.3. CDF of WiFi prediction accuracy.

cellular networks. We use three days of data as each user's usage history, and run the simulation assuming hour-long periods. The user's monthly budget for 3G data usage is chosen from a truncated normal distribution between \$20 and \$40 (2 to 4 GB at a unit price of \$10/GB).

To verify that our three-day training set of data is sufficient to simulate AMUSE, we test the accuracy of our WiFi prediction algorithm on this dataset. To do so, we define the prediction accuracy as follows: if the probability of WiFi access for a given user in a given future period is greater (respectively less) than 0.5 and we observe (respectively do not observe) WiFi access in that period, we classify the prediction as "accurate." Otherwise, we call the prediction "inaccurate." We then divide the number of accurate predictions by the total number of predictions for each user to find the prediction accuracies. As shown in Figure 6.3, we observe a 64 – 90% prediction accuracy over the eight iPhone users and a 67 – 97% prediction accuracy over

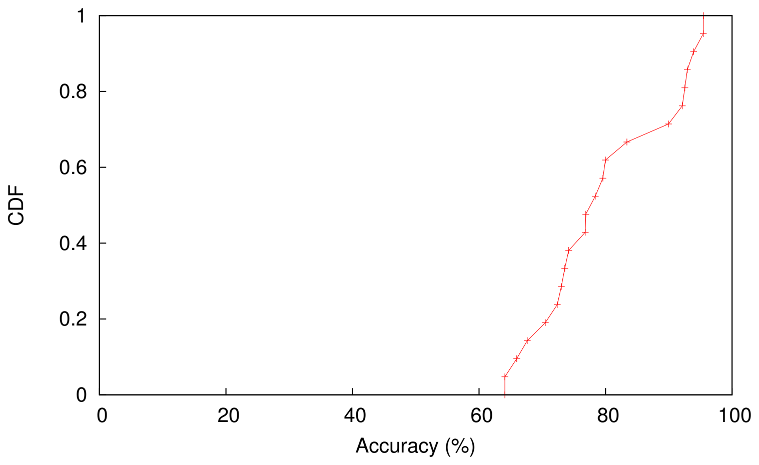


Fig. 6.4. CDF of WiFi prediction accuracy of LifeMap data.

eight Android users, indicating that our data is sufficient for producing meaningful simulation results. To further inspect the prediction accuracy of our WiFi prediction algorithm, we also tested with other open mobility traces. We used the data of 21 users who have enough records needed for prediction from the dataset collected in LifeMap project [34]. We obtained the similar results with our dataset as shown in Figure 6.4.

6.1.2 Baseline Algorithms

We compare AMUSE's performance to two baseline algorithms: on-the-spot offloading and delayed offloading [11].

On-the-spot offloading offloads traffic to WiFi opportunistically: users send their traffic over WiFi if they are connected to a WiFi network at that time, and switch to 3G if they move outside of the WiFi coverage area. No sessions ever wait for WiFi, which

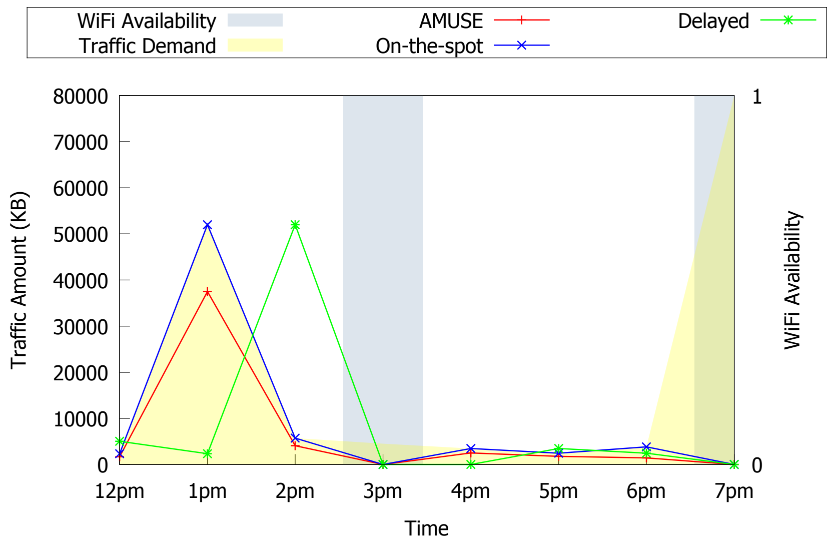


Fig. 6.5. An example traffic demand and cellular usage amount under each offloading algorithm.

may lead to higher spending as compared with AMUSE – AMUSE allows delay-tolerant sessions to wait for WiFi, thus saving users money. The delayed offloading algorithm forces all traffic to wait up to a fixed time limit for WiFi access (1 hour in our simulations). If WiFi becomes available before this time, the waiting traffic is sent over WiFi; otherwise, it is sent over 3G. While this algorithm may offload more traffic than AMUSE and thus save users money, it does not consider users' delay tolerances: even urgent sessions are forced to wait for some time. Moreover, if WiFi is not available at the end of the fixed time limit, the sessions will complete over 3G anyway, costing users money and making them wait.

In Figure 6.5, we show an illustrative example scenario that explains how AMUSE operates compared to other algorithms in making offloading decisions. Figure 6.5 illustrates the cellular traffic demand and WiFi availability of one user for 8 hours. At 1pm, the traffic demand is larger than other times, and WiFi is available at 3pm and 7pm. The amount offloaded with each offloading algorithm can be calculated from the difference in total demand and traffic for each algorithm. Since AMUSE predicts the WiFi availability and delays longer than 1 hour if the utility is increased, it offloads a considerable amount of the traffic in 1pm to 3pm. On the other hand, On-the-spot offloading offloads only the traffic at 3pm and 7pm when the WiFi is available. Delayed offloading offloads more than On-the-spot offloading by delaying the traffic of 2pm and 6pm to 3pm and 7pm, respectively. However, since it cannot predict the WiFi availability of 3pm at 1pm, it cannot delay the traffic of 1pm to 3pm, thus losing the opportunity to offload. Moreover, since it blindly delays the traffic by one hour if the WiFi is not available, the traffic of 12, 1, 4, 5pm is delayed but ultimately transmitted by 3G, decreasing users' utility. AMUSE, on the other hand, does not always prefer offloading to transmitting on 3G network. For example, AMUSE delays only some traffic at 1pm, since the expected utility of transmitting over 3G is larger than that of waiting for higher bandwidth and lower cost WiFi.

6.1.3 Numerical Results

Figure 6.6 plots the distributions of relative utility values under our benchmark algorithms compared to those under AMUSE. For each user, both benchmark algorithms decrease the utility. This decrease is particularly dramatic for one user, whose utility values with on-the-spot and delayed offloading are only 8 and 23%, respectively, of the utility under AMUSE. On average, the utility of on-the-spot offloading is 19% less than that of AMUSE, while that of delayed offloading is 22% lower.

AMUSE yields higher utility values than on-the-spot offloading due to offloading more traffic onto WiFi: Fig. 6.7 shows the distributions of relative amounts of

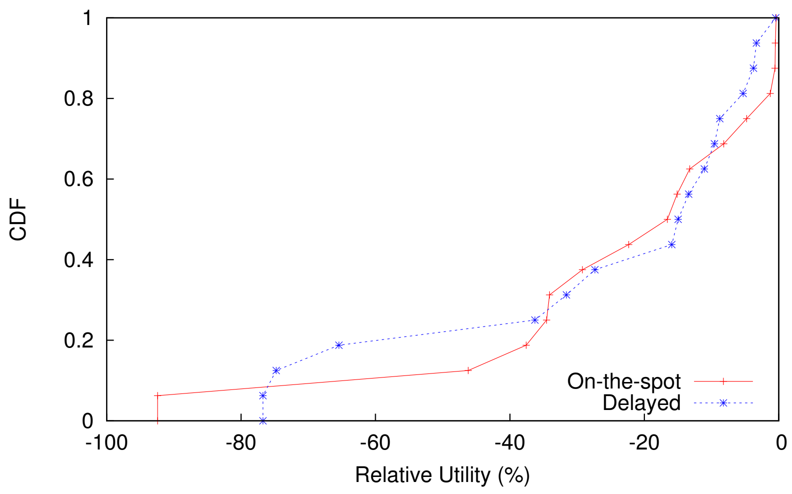


Fig. 6.6. CDF of relative utility function values compared to AMUSE.

traffic offloaded under both benchmark algorithms, as compared to AMUSE. We see that for all users, the amount of traffic offloaded is larger under AMUSE than it is under on-the-spot offloading; thus, AMUSE leverages the delay tolerance of some sessions by allowing them to wait for WiFi access. Users then save money: Fig. 6.7 also compares users' amount spent under the two benchmark algorithms to that spent with AMUSE. Users consistently spend over 20% more with on-the-spot offloading, and on average increase their spending by 33% compared with AMUSE.

Compared to delayed offloading, AMUSE trades off between reducing users' spending by offloading traffic and completing some sessions immediately due to their intolerance of delay. Figure 6.7 shows that delayed offloading offloads more traffic than AMUSE for 10 users: AMUSE sends some sessions over 3G without waiting for WiFi, allowing users to spend more and delay less. One user offloads nearly 160% more traffic under delayed offloading relative to AMUSE. The consequent decrease in cost relative to AMUSE (nearly 80%) is offset by less delay under AMUSE; this user

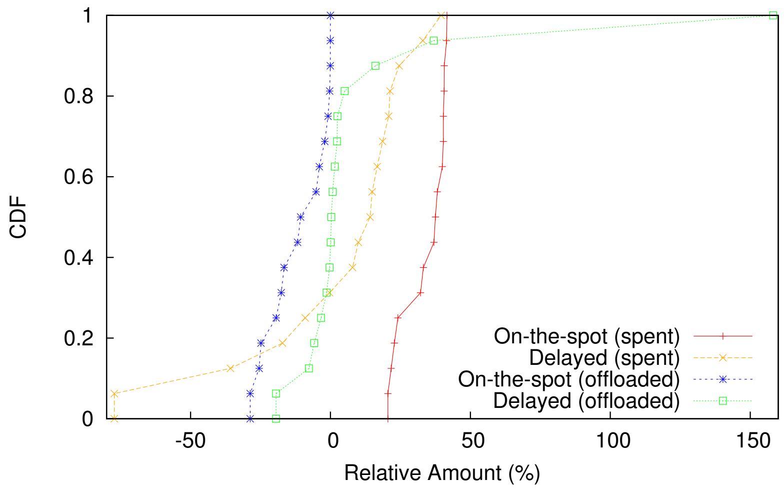


Fig. 6.7. CDF of relative offloaded traffic amount and amount spent compared to AMUSE.

in fact experiences 5% less utility under delayed offloading than that under AMUSE. On the other hand, delayed offloading offloads less traffic than AMUSE for 6 users: AMUSE allows delay-tolerant traffic to wait more than an hour for WiFi. We found that this additional wait for WiFi reduces these users' spending, offsetting the loss in utility from delaying the session.

Finally, we examine AMUSE's benefits for different types of users. We split 16 users into "heavy" and "light" usage groups (8 users for each group), and plot the amount offloaded, utility, and cost of the two benchmark algorithms relative to AMUSE in Fig. 6.8. For heavy users, both benchmark algorithms perform worse than AMUSE: users' utility and amount offloaded decrease under these algorithms, while their cost increases. Thus, AMUSE's cost savings in offloading more traffic offset any loss in utility from waiting more for WiFi access. On-the-spot offloading performs especially poorly compared to AMUSE, indicating that most users' delay

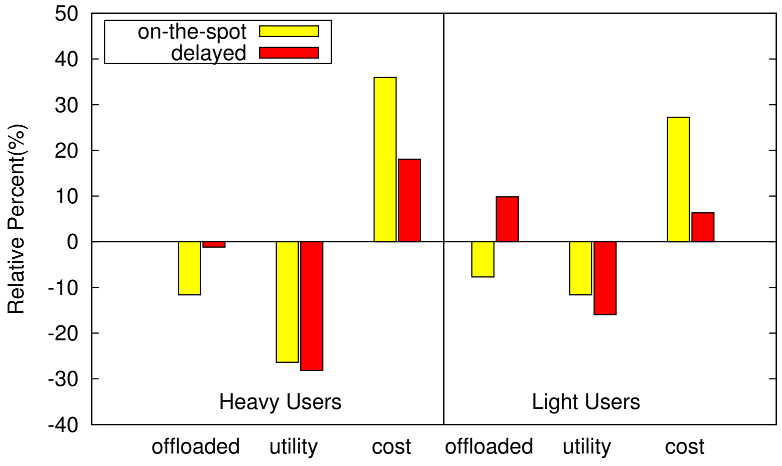


Fig. 6.8. Average offloaded traffic, utility, and cost of heavy and light users in all traces.

tolerance enables them to gain utility under AMUSE by selectively delaying some sessions and sending them over WiFi. For light users, AMUSE weights the cost savings from waiting for WiFi less heavily: some sessions do wait for WiFi, as shown by the decrease in amount offloaded in on-the-spot offloading, but delay-intolerant sessions do not wait, as shown by the increase in amount offloaded in delayed offloading. This likely arises from light users' looser budget constraint: by definition, light users spend less than heavy users on their data consumption. They accordingly benefit less overall: compared with AMUSE, the utility of heavy users decreases by 27% under the benchmark algorithms, while that of light users decreases by 14%.

6.2 Receiver-side TCP rate control

In this section, we evaluate our proposed receiver-side TCP rate control algorithm. Our evaluation approach is two folds. First, we evaluate our algorithm in real

Table. 6.1. Basic rate control test using Iperf. Parentheses denote the standard deviations.

Target rate	100 Kbps	500 Kbps	1,024 Kbps
Ethernet	103.8 (0.42)	506.2 (0.42)	1031.2 (1.81)
WiFi	83.14 (3.63)	459 (6.46)	902.4 (21.67)
3G	95.28 (1.52)	474.7 (11.86)	896 (47.28)

networks. Then we evaluate the algorithm on emulated networks to see the performance in various network environments and scenarios.

6.2.1 Real network experiments

To verify Algorithm 3 in practice, we test our receiver-side bandwidth control algorithm by running `Iperf` over Ethernet, WiFi, and 3G networks. We used target bandwidths of 100 Kbps, 500 Kbps, and 1 Mbps; experimental results for the three cases are shown in Table 6.1. While the bandwidth control algorithm achieves the target rate in each of the three different networks, we observe that the rate over the Ethernet link is much closer to the target rate than the rates over WiFi and 3G: packet loss rate and link jitter are the smallest in Ethernet.

We also test the algorithm with different applications. For this experiment, we set the target bandwidth to 300 Kbps and run two applications (HTTP and FTP). Our results (Table 6.2) show that the rates achieved are similar to the target rate.

6.2.2 Experiments in emulated networks

To investigate the performance of our algorithm in more various network environments, we set up an testbed using a WAN emulator, WANem [35] as in Figure 6.9.

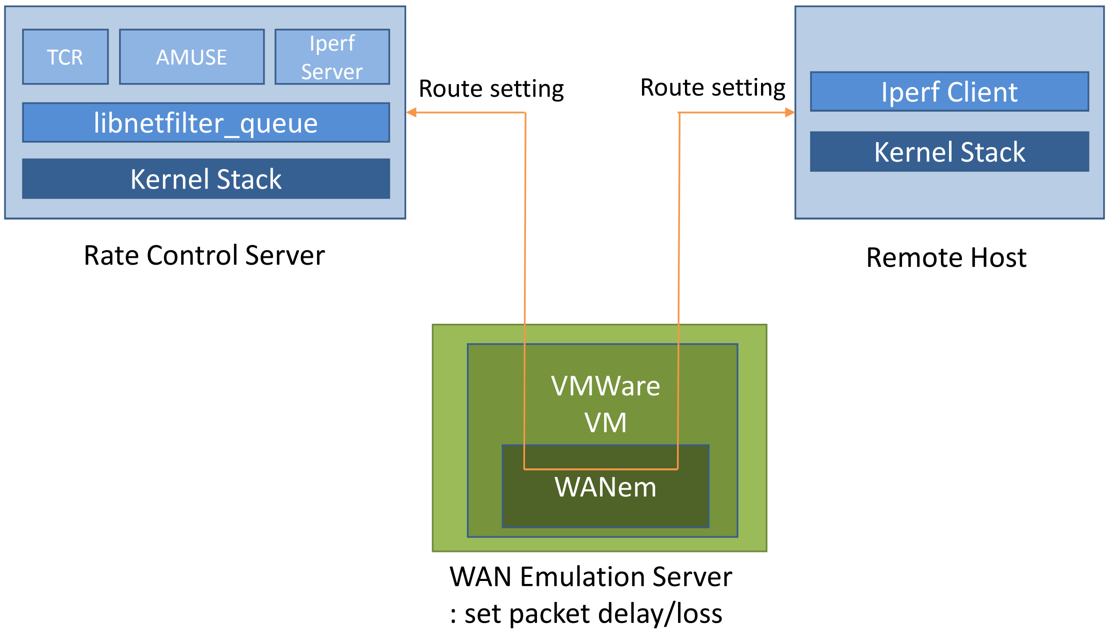


Fig. 6.9. Testbed settings.

Table. 6.2. Application rate control test using HTTP and FTP. Parentheses denote the standard deviations.

Application (rate)	HTTP (300 Kbps)	FTP (300 Kbps)
Ethernet	297.04 (3.54)	291.2 (4.68)
WiFi	271.12 (10.77)	269.28 (8.27)
3G	296.16 (3.33)	279.2 (4.43)

We set three servers: Rate Control Server that controls the TCP rate in receiver-side, Remote Host that transmits the data through TCP connection, and WAN Emulation Server that emulates the WAN characteristics on the TCP connections. To establish TCP connections, we run an Iperf server on Rate Control Server and an Iperf client on Remote Host. By setting the routing path, we let both servers' traffic go through WANem which is installed on the virtual machine in WAN Emulation Server. We use TCP Cubic for all the experiments except the experiment that varies the TCP types. In addition, we ran the Iperf 15 times and show the average and standard deviation on the graph for each experiment except several experiments that show the time-variation of the observed values.

As a baseline algorithm to compare the performance with our receiver-side bandwidth control algorithm, we test TCR (TCP Rate Control) proposed in [36]. TCR adjusts the advertisement window and sets inter-ack spacing time according to the target rate of each flow.

6.2.2.1 Test for accuracy of rate control

To check the accuracy of rate control for various target rates in networks with different delays, we varied the one-way network delay to 10, 50, 100 msec (RTTs

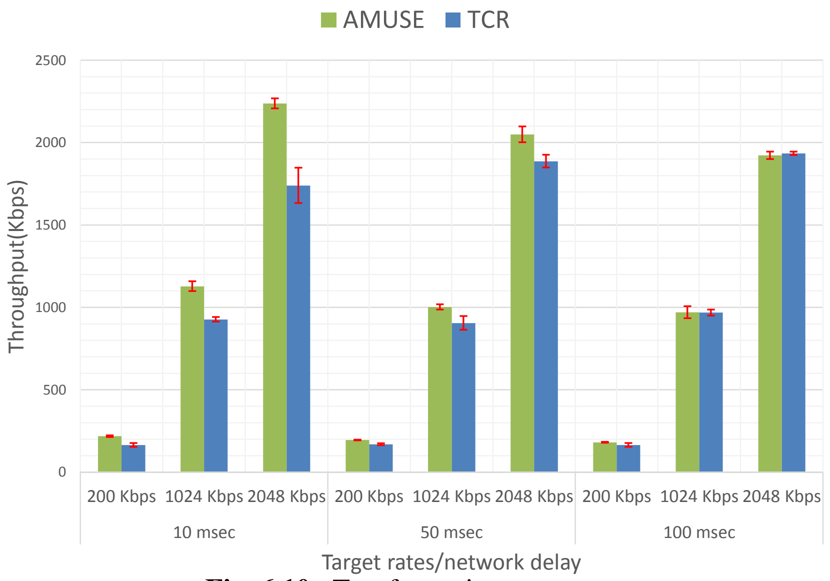


Fig. 6.10. Test for various target rates.

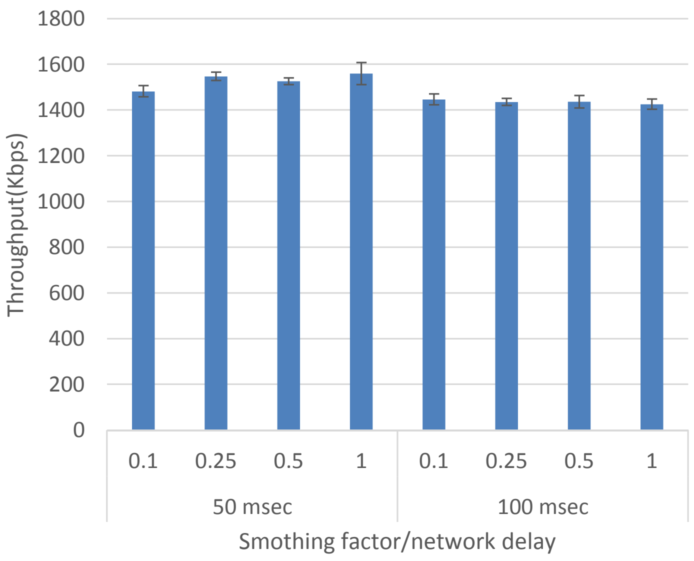


Fig. 6.11. Test for various smoothing fator values.

are 20, 100, 200 msec), and the target rates to 200, 1024, 2048 Kbps. As we can see in Figure 6.10, our algorithm obtains more accurate throughputs than TCR in many cases.

6.2.2.2 Test for the parameter optimization

In order to optimize the parameter values of the algorithm, we tested the throughput of the our algorithm by changing the values of smoothing factor (α) and rate control window ($rate\_control\_win$). As shown in Figure 6.11, the throughput of Iperf approaches the target rate (1500 Kbps) most closely when the smoothing factor is 0.1. From Figure6.12, we can find that the accuracy of the rate control is the highest when the rate control window value is 1 sec.

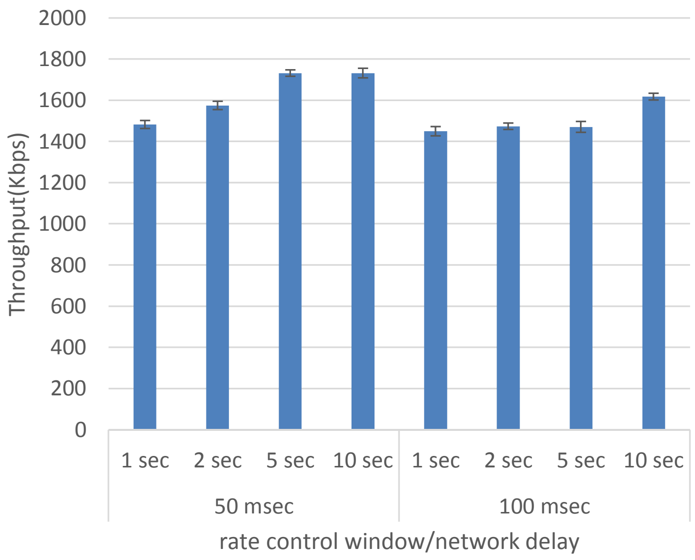


Fig. 6.12. Test for various rate control window values.

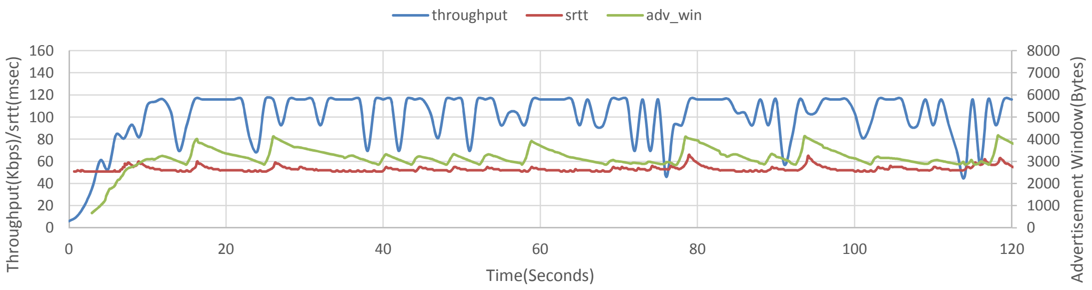


Fig. 6.13. Time variation of proposed rate control algorithm.

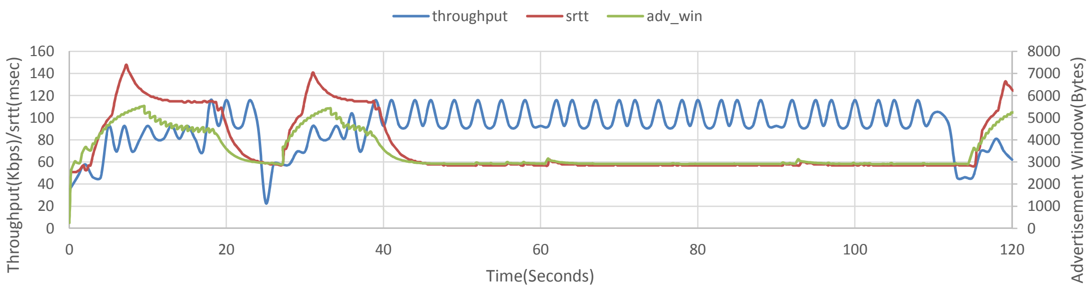


Fig. 6.14. Time variation of TCR.

6.2.2.3 Test for the stability

To check the stability of our algorithm and TCR, we observe the time variation of the throughput, $srtt$ (smoothed RTT) at TCP sender, and advertisement window size. We set the target rate to 100 Kbps and session time to 120 seconds. From Figure 6.13 and 6.14, we can see that the degree of the time variation of $srtt$ and advertisement window for the proposed algorithm is much smaller than that of TCR. When the $srtt$ value becomes large, the TCP sender will regard the network between the sender and receiver as a slow network, and set the retransmission timeout (RTO) to a large value. This will make the detection of the packet loss and retransmission at TCP sender late, so the performance of real-time applications can be degraded. We presume that this increased $srtt$ can be explained as follows: when the data packets are sent in bursts (in cases such as the slow start), some ACK packets are delayed due to ACK pacing, so the time between sending the data packet and receiving the

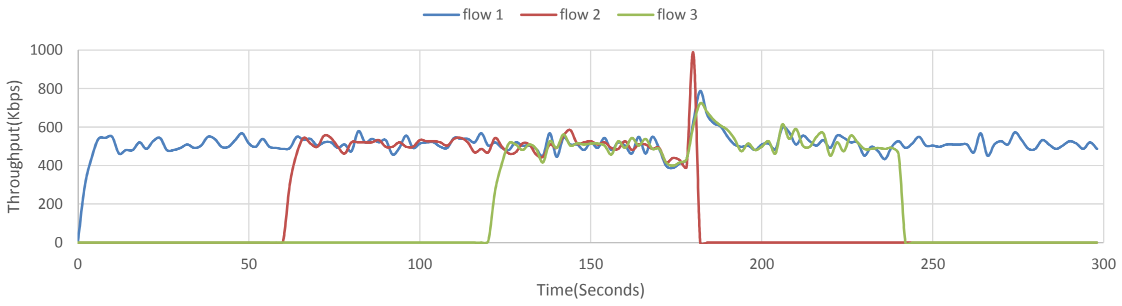


Fig. 6.15. Test for coexistence of rate controlled flows in proposed rate control algorithm.

ACK packet for the data packet gets longer. After receiving these spaced ACK packets, the sender will send packets with spaces, then the $srtt$ will be get smaller after some time since the TCP receiver does not need to put an extra space between ACKs. The increase of the advertisement window in TCR is obvious, since it is set to be proportional to the RTT values.

6.2.2.4 Test for the coexistence among rate controlled flows

To see the performance when rate controlled flows coexist, we sequentially created 3 rate controlled flows with the target rates of 500 Kbps and 60 seconds intervals between flows. The first flow runs for the whole experiment period of 300 seconds, while other two flows run for 120 seconds, respectively. From Figure 6.15 and 6.16 we can observe that both algorithms do not show any performance degradation due to the interference among flows.

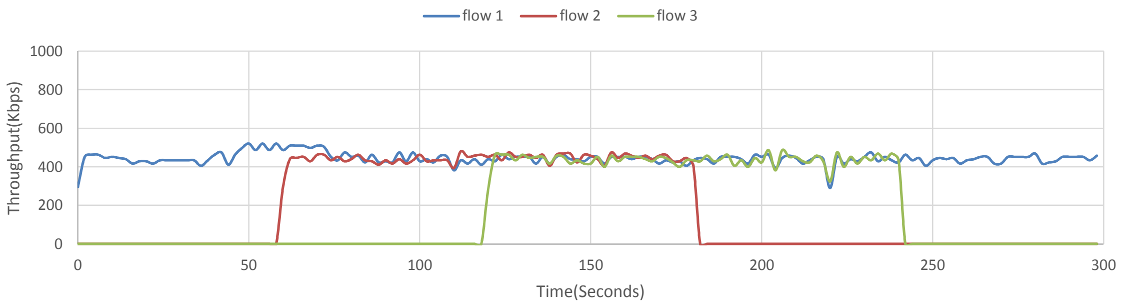


Fig. 6.16. Test for coexistence of rate controlled flows in TCR.

6.2.2.5 Test for coexistence of a rate controlled flow with non-rate controlled flows

We set one rate controlled flow with the target rate of 1500 Kbps and two non-rate controlled flows, in order to see the performance when rate controlled and non-rate controlled flows coexist. We created the non-rate controlled flows at 60 and 120 seconds, respectively, and retained them for 120 seconds. We set the total bandwidth to 2 Mbps. As we can see in Figure 6.17 and 6.18, both algorithms show a similar behavior: the throughput of the rate controlled flow drops, but is higher than other flows when there are non-rate controlled flows. After the non-rate controlled flows disappear, the throughput of the rate controlled flow is recovered immediately.

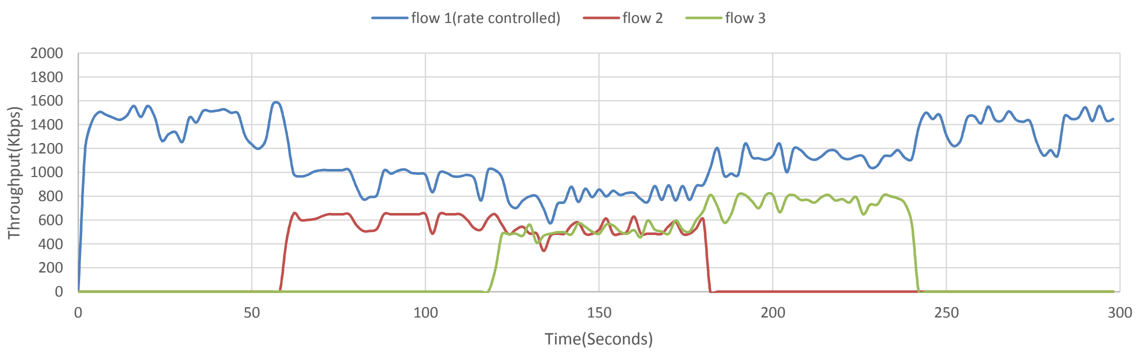


Fig. 6.17. Test for coexistence of a rate controlled flow with non-controlled flows in proposed rate control algorithm.

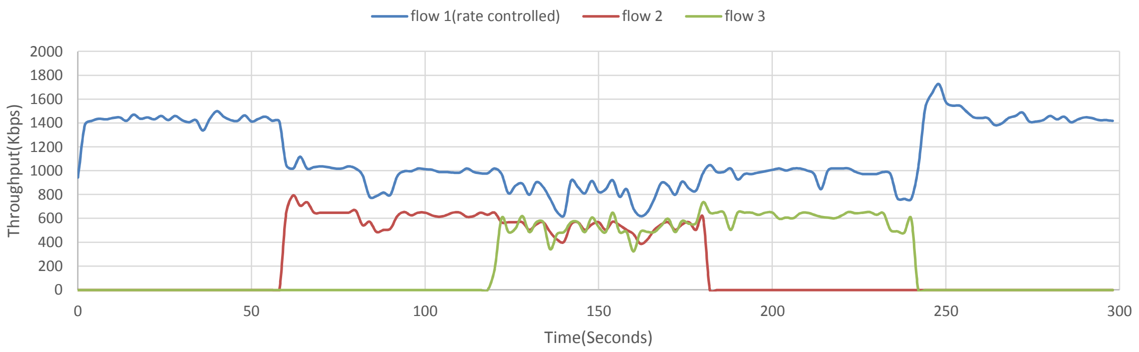


Fig. 6.18. Test for coexistence of a rate controlled flow with non-controlled flows in TCR.

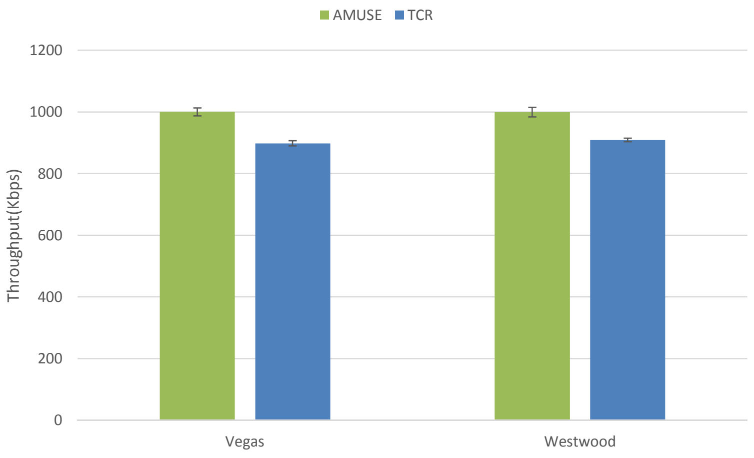


Fig. 6.19. Test for various TCP variants.

6.2.2.6 Test for various TCP variants

We ran Iperf using TCP Vegas and Westwood on both TCP sender and receiver, to see the performance of rate control algorithms in different TCP variants. We set the target rate to 1024 Kbps. From Figure 6.19 we observe that our proposed algorithm performs more accurate rate control than TCR for both TCP Vegas and Westwood.

6.2.2.7 Test for lossy networks

To observe the performance in lossy network environments, we checked the throughputs by varying the packet loss rates to 0.3, 0.6, 0.9%. We set the target rate to 1024 Kbps. From Figure 6.20, we can see that while the throughput of TCR drops drastically as the packet loss rate increases, the proposed algorithm shows a slightly smaller throughput than the target rate only in the case of 0.9% loss rate. We presume that this is because while our algorithm, by nature, tries to recover the throughput when the throughput drops due to packet loss, TCR does not have such a recovery

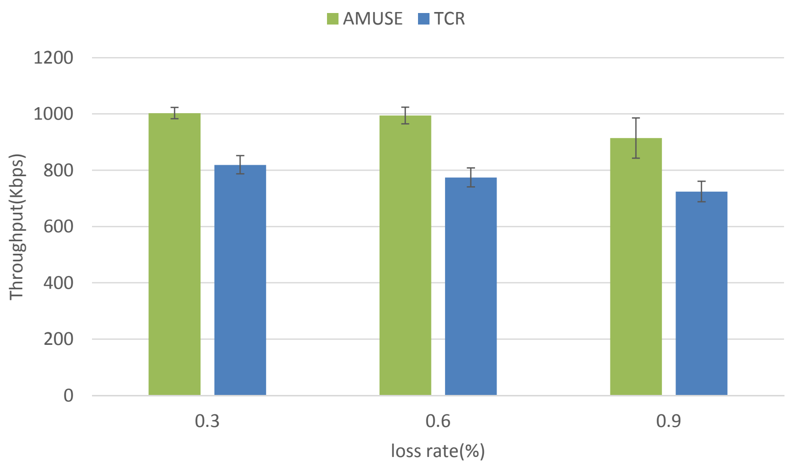


Fig. 6.20. Test for the performance in lossy networks.

mechanism.

6.2.2.8 Summary

In a lot of scenarios, the proposed algorithm shows that it can provide more accurate rate control than TCR. In addition, the TCP retransmission timeout is not increased, so it does not hurt the performance of real-time applications. Also the proposed algorithm is more robust to packet loss than TCR, and it can avoid the drawbacks when we use packet pacing. According to [37], the pacing fragments the packet losses, so more SACK blocks are needed to convey loss information. Also when used with TCP Reno, pacing results in lower throughputs and increased latencies in many scenarios.

Chapter 7

Discussion

7.1 Application of AMUSE in various data plans

We have assumed a usage-based data plan for AMUSE algorithm. According to [5], ISPs provide various data plans for wired and wireless Internet services such as Fixed Flat Rate, Usage Based, Priority Pricing, Time of Day, Cumulus Pricing, App-Based Pricing, and Congestion Based. We claim that AMUSE is general so that it can be applied to a number of variants of usage-based data plan by adjusting the cost, bandwidth of the utility function and the cost constraint, etc.

The data plans in which AMUSE can be applied includes “Cap, then metered” which is prevalent worldwide, and “monthly fee, unlimited”, “monthly fee, flat to a cap, then throttle”. In “Cap, then metered” data plan, a user pays a flat price (basic charge) up to a predetermined traffic volume (basic data volume), beyond which the user is charged in proportion to the amount of data he consumes. AMUSE can be applied in two ways for this data plan. First, we can assume that the user has a budget that he sets regardless of the basic charge. In this case, even though the user uses the data under the basic data volume, the cost will be applied in his utility. Therefore, AMUSE can be utilized as it is. Second, we can set the cost to zero and remove the cost constraint when the user uses the data under the basic data volume since the actual cost is not increased. Then we consider the cost in utility function, and apply the cost constraint with the budget of (the budget that the user sets - basic charge) only when the usage amount exceeds the basic data volume.

7.2 Overhead of location sensing

According to [38], the energy consumption of one GPS location sensing operation is about 143.1 and 166.1 mW, respectively, with internal antenna enabled and external antenna enabled. We claim that AMUSE's energy consumption for location sensing is reasonably low: several times of GPS operations per hour are enough since AMUSE requires the overall location not the exact movement path during each time slot. However, to minimize the energy consumption, we can further utilize energy efficient location sensing techniques proposed by [39], [34].

Chapter 8

Conclusion

In this paper, we propose AMUSE, a cost-aware WiFi offloading system that maximizes the *end user's* utility under her 3G budget constraints. AMUSE consists of two main components: a bandwidth optimizer and a TCP rate controller. By predicting future usage and WiFi availability, the bandwidth optimizer chooses how long an application should wait for WiFi access, as well as a 3G data rate should WiFi not be available. These choices are optimized so as to balance the user's tradeoffs between the cost of sending an application's traffic over 3G, the higher throughput received over WiFi, and the delay inherent in waiting for WiFi. The TCP rate controller practically enforces the 3G rates chosen for each application by controlling the TCP advertisement window from the user side. AMUSE also allows for end-user interaction by providing a user interface through which users can set their bandwidth allocation preferences and view the offloading decisions made. Through a measurement study, we show that though a large amount of some applications' traffic is offloaded already, our offloading framework can offload a larger portion of mobile users' cellular traffic.

We prototyped AMUSE and evaluated its performance with mobile traces from 37 users. Our results show that AMUSE can improve both heavy and light data users' utility from offloading; for heavy users, two other representative WiFi offloading algorithms achieve 27% lower utility than AMUSE on average. Heavy users' costs were on average 18 and 36% higher under these benchmark algorithms compared to

AMUSE, a savings realized by offloading more traffic onto WiFi. Though our results are based on data from a limited number of users, we expect similar performance from a wider range of users.

Bibliography

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2013 - 2018," Feb 2014, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf.

[2] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang, "Incentivizing time-shifting of data: A survey of time-dependent pricing for Internet access," *IEEE Communications Magazine*, vol. 50, no. 11, pp. 91–99, 2012.

[3] T. Ricker, "AT&T making tourists even more annoying with free Times Square WiFi," Engadget, May 2010, http://www.engadget.com/2010/05/25/atandt-making-times-square-tourists-even-more-annoying-with-free-w/.

[4] V. Chandrasekhar, J. Andrews, and A. Gatherer, "Femtocell networks: A survey," *IEEE Communications Magazine*, vol. 46, no. 9, pp. 59–67, 2008.

[5] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang, "A survey of smart data pricing: Past proposals, current plans, and future trends," *ACM Computing Surveys*, vol. 46, no. 2, p. 15, 2013.

[6] A. J. Nicholson and B. D. Noble, "BreadCrumbs: Forecasting mobile connectivity," in *Proceedings of ACM MobiCom*. ACM, 2008.

[7] N. Santhapuri, J. Manweiler, R. Choudhury, and S. Nelakuditi, "BytesToGo: Offloading 3G via WiFi prioritization," Duke University, Tech. Rep., 2011, http://synrg.ee.duke.edu/papers/bytes.pdf.

[8] V. A. Siris and M. Anagnostopoulou, "Performance and energy efficiency of mobile data offloading with mobility prediction and prefetching," in *Proc. of WoWMoM*. IEEE, 2013, pp. 1–6.

[9] A. Balasubramanian, R. Mahajan, and A. Venkataramani, "Augmenting mobile 3G using WiFi," in *Proceedings of ACM Mobisys*. ACM, 2010, pp. 209–222.

[10] A. Rahmati, C. Shepard, A. Nicoara, L. Zhong, and J. P. Singh, "Mobile tcp usage characteristics and the feasibility of network migration without infrastructure support," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 14, no. 4, pp. 10–12, 2011.

[11] K. Lee, I. Rhee, J. Lee, S. Chong, and Y. Yi, "Mobile data offloading: How much can WiFi deliver?" in *Proceedings of ACM CoNEXT*. ACM, 2010.

[12] J. Lee, Y. Yi, S. Chong, and Y. Jin, "Economics of wifi offloading: Trading delay for cellular capacity," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 3309–3314.

[13] C. Joe-Wong, S. Sen, and S. Ha, "Offering supplementary wireless technologies: Adoption behavior and offloading benefits," in *Proc. of IEEE INFOCOM 2013*. IEEE, 2013, pp. 1061–1069.

[14] X. Hou, P. Deshpande, and S. Das, "Moving bits from 3g to metro-scale wifi for vehicular network access: An integrated transport layer solution," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, oct. 2011, pp. 353 –362.

[15] X. Zhuo, W. Gao, G. Cao, and Y. Dai, "Win-Coupon: An incentive framework for 3G traffic offloading," in *Proceedings of IEEE ICNP*, Oct. 2011, pp. 206 –215.

[16] N. Ristanovic, J.-Y. Le Boudec, A. Chaintreau, and V. Erramilli, "Energy efficient offloading of 3g networks," in *Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, ser. MASS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 202–211. [Online]. Available: http://dx.doi.org/10.1109/MASS.2011.27

[17] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 179–194.

[18] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, “Identifying diverse usage behaviors of smartphone apps,” in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 329–344.

[19] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, “A first look at traffic on smartphones,” in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 281–287.

[20] A. Gember, A. Anand, and A. Akella, “A comparative study of handheld and non-handheld traffic in campus wi-fi networks,” in *Passive and Active Measurement*. Springer, 2011, pp. 173–183.

[21] G. Maier, F. Schneider, and A. Feldmann, “A first look at mobile hand-held device traffic,” in *Passive and Active Measurement*. Springer, 2010, pp. 161–170.

[22] “NetLimiter,” http://www.netlimiter.com/.

[23] “SoftPerfect Bandwidth Manager,” http://www.softperfect.com/.

[24] “PRTG Network Monitor,” http://www.paessler.com/prtg/.

[25] V. Raisinghani, A. Singh, and S. Iyer, “Improving TCP performance over mobile wireless environments using cross layer feedback,” in *Proceedings of IEEE Conference on Personal Wireless Communications*, Dec. 2002, pp. 81 – 85.

[26] H.-Y. Wei, S.-C. Tsao, and Y.-D. Lin, “Assessing and improving TCP rate shaping over edge gateways,” *IEEE Transactions on Computers*, vol. 53, pp. 259–275, 2004.

[27] L. Song, D. Kotz, R. Jain, and X. He, “Evaluating next-cell predictors with extensive Wi-Fi mobility data,” *IEEE Transactions on Mobile Computing*, vol. 5, no. 12, pp. 1633–1649, 2006.

[28] Y. Im, C. Joe-Wong, S. Ha, S. Sen, T. T. Kwon, and M. Chiang, "A survey of cost, quality and delay tradeoffs in WiFi offloading," Princeton University, Tech. Rep., 2012, http://www.princeton.edu/$\sim$cjoe/AMUSE_Survey.pdf.

[29] M. Moser, D. Jokanovic, and N. Shiratori, "An algorithm for the multidimensional multiple-choice knapsack problem," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 80, no. 3, pp. 582–589, 1997.

[30] "WFP Callout Driver," http://msdn.microsoft.com/en-us/library/windows/hardware/gg463267.aspx.

[31] "libnetfilter_queue," http://www.netfilter.org/projects/libnetfilter_queue/.

[32] B. Veal, K. Li, and D. Lowenthal, "New methods for passive estimation of tcp round-trip times," in *Passive and Active Network Measurement*. Springer, 2005, pp. 121–134.

[33] V. Paxson, M. Allman, and C. T. R. Timer, "Rfc 2988," *Computing TCP's Retransmission Timer*, 2000.

[34] Y. Chon, E. Talipov, H. Shin, and H. Cha, "Mobility prediction-based smartphone energy optimization for everyday location monitoring," in *Proceedings of the 9th ACM conference on embedded networked sensor systems*. ACM, 2011, pp. 82–95.

[35] "Wanem," http://wanem.sourceforge.net/.

[36] S. Karandikar, S. Kalyanaraman, P. Bagal, and B. Packer, "Tcp rate control," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 1, pp. 45–58, Jan. 2000. [Online]. Available: http://doi.acm.org/10.1145/505688.505694

[37] D. Wei, P. Cao, S. Low, and C. EAS, "Tcp pacing revisited," in *Proceedings of IEEE INFOCOM*. Citeseer, 2006.

[38] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, 2010, pp. 271–285.

[39] Z. Zhuang, K.-H. Kim, and J. P. Singh, “Improving energy efficiency of location sensing on smartphones,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 315–330.

초 록

최근의 모바일 데이터 수요의 급격한 증가에 대처하기 위해, 무선 인터넷 서비스 제공자들(ISPs)은 새로운 요금제를 점차 도입하고 있으며, 모바일 트래픽을 오프로딩 하기 위해 WiFi 핫스팟을 설치하고 있다. 하지만, 이러한 인터넷 서비스 제공자 중심의 트래픽 관리를 위한 방안들은 모바일 사용자들의 이익과 항상 일치하는 것은 아니다. 사용자들은 그들의 오프로딩 결정을 위해 비용, 처리율, 지연시간 간의 복잡하고 다차원적인 트레이드오프(tradeoff)에 직면하게 된다. 즉, WiFi를 사용하기 위해 기다림으로써 비용을 절약하고 높은 처리율을 제공받을 수 있지만, 지연시간에 민감한 사용자의 경우 WiFi가 접근 가능할 때까지 기다리지 않을 수 있다. 이러한 트레이드오프를 처리하기 위해 우리는 사용자의 처리율, 지연시간 트레이드오프와 데이터 예산 상의 제약을 고려하는 실용적인 비용인지 WiFi 오프로딩 시스템의 기능적 프로토타입인 AMUSE(Adaptive bandwidth Management through USer-Empowerment, 사용자 중심의 적응적 대역폭 관리기법)를 제안한다. 예측된 미래의 데이터 사용량과 WiFi 이용가능 여부를 바탕으로, AMUSE는 어떠한 어플리케이션을 하루 중 어떤 시간으로 오프로딩 할지를 결정한다. 또한 모바일 장치의 대부분의 트래픽이 TCP 트래픽이기 때문에, 각 TCP 어플리케이션의 할당된 레이트(rate)를 적용하기 위한 새로운 수신자 기반 대역폭 할당 기법을 제시한다. 따라서, AMUSE는 다양한 어플리케이션 컨텐트 서버의 도움 없이 비용-처리율-지연시간 트레이드오프에 따라 대역폭 할당을 최적화할 수 있다. 20명의 스마트폰 사용자의 트래픽 사용량 데이터에 대한 측정 연구를 통해, 사용자들은 몇몇 종류의 어플리케이션에 대해 트래픽의 많은 부분을 이미 오프로딩 하고 있지만,

우리의 기법을 사용하여 이동통신 트래픽의 상당 부분을 추가적으로 오프로딩 할 수 있음을 발견하였다. 우리는 AMUSE를 Windows 7 테블릿 상에 구현하고, 37명의 모바일 사용자로부터 얻은 3G 및 WiFi 사용량 데이터를 통해 AMUSE의 성능을 평가하였다. 실험 결과는 AMUSE가 사용자의 만족도를 향상시킴을 보여준다. AMUSE와 비교해서 다른 오프로딩 알고리즘은 사용량이 적은 사용자와 많은 사용자에 대해 각각 14% 와 27% 낮은 사용자 만족도를 보여준다. 결론적으로, 비용, 처리율, 지연시간에 대한 사용자의 상충된 이해관계를 지능적으로 관리함으로써 오프로딩 결정을 향상시킬 수 있음을 알 수 있다.

주요어: 대역폭 관리, 모바일 데이터, WiFi 오프로딩
학번: 2007-21064