The Dissertation Committee for Zheng Lu
certifies that this is the approved version of the following dissertation:

# Scheduling Wireless Transmissions Exploiting Application Awareness and Knowledge of the Future

Committee:

Gustavo de Veciana, Supervisor

Sanjay Shakkottai

Constantine Caramanis

Sujay Sanghavi

Lili Qiu

# Scheduling Wireless Transmissions Exploiting Application Awareness and Knowledge of the Future

**by**

**Zheng Lu, B.E., M.S.E.**

**DISSERTATION**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**DOCTOR OF PHILOSOPHY**

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2015

Dedicated to my parents.

# Acknowledgments

First I wish to thank my parents for their enormous love and support throughout my life, which is a debt that can never be paid off. I also wish to thank my grandparents for their love and encouragement. I am so fortunate to be born in this family and no matter how far I go I always know there is a home I can return.

I am also very fortunate that I always meet great teachers and mentors in my life. I wish to give special thanks to my PhD supervisor Dr. Gustavo de Veciana, who has always been exceptional in personality, teaching and research. I learned from him the way to think, the way to research and most importantly the way to do things right. I will never forget the many discussions we had and the time we sit at the desk and write down research ideas together, which are all invaluable experience and lead me through a successful PhD study.

I wish to thank the members of my dissertation committee: Dr. Sanjay Shakkottai, Dr. Constantine Caramanis, Dr. Sujay Sanghavi and Dr. Lili Qiu for generously offering their time, guidance, and support throughout my PhD qualifying exam and my PhD dissertation and defense.

I thank Dr. Alan Bovik, Dr. Robert Heath, Dr. Jeffrey Andrews, Dr.Constantine Caramanis, Dr. Xiaoqing Zhu, Dr. Vinay Joseph, Dr. Chao

Chen, and Dr. Sarabjot Singh for all the help and discussions through the VAWN project.

I thank Dr. Kerstin Johnsson, Yicong Wang and Namyoon Lee for the suggestions and discussions through the Intel 5G Research Program, and thank Navid NaderiAlizadeh and Dr. Salman Avestimehr for sharing their early work on ITLinQ.

I thank Dr. Jeffrey Foerster, Dr. V Srinivasa Somayazulu and Dr. Hassnaa Moustafa for their guidance and help during my internship at Intel Labs.

I thank Intel, Cisco and NSF for supporting the work in this dissertation through the VAWN project, and thank Intel and NSF (CNS-1343383) for the support through the Intel 5G Research Program.

I thank Dr. Vinay Joseph for the helpful discussions and his help with the simulations. I thank Yuhuan Du for his help preparing the paperworks for this dissertation. I also wish to thank all my colleagues and friends for their help through my PhD study and life. Hook'em Horns!

# Scheduling Wireless Transmissions Exploiting Application Awareness and Knowledge of the Future

Publication No. ____________________

Zheng Lu, Ph.D.
The University of Texas at Austin, 2015

Supervisor: Gustavo de Veciana

This dissertation explores ways to improve the scheduling of wireless transmissions, by exploiting the application layer information of the ongoing transmissions and exploiting the knowledge of the future capacity variations. First, we consider the design of cross-layer opportunistic transport protocols for stored video over wireless networks with a slow varying (average) capacity. We focus on two key principles: (1) scheduling data transmissions when capacity is high; and (2), exploiting knowledge of *future* capacity variations. The latter is possible when users' mobility is known or predictable, e.g., users riding on public transportation or using navigation systems. We consider the design of cross-layer transmission schedules which minimize system utilization (and thus possibly transmit/receive energy) while avoiding, if at all possible, rebuffering/delays, in several scenarios. For the single-user anticipative case where all future capacity variations are known beforehand; we establish the optimal transmission schedule is a Generalized Piecewise Constant Thresholding (GPCT) scheme. For the single-user partially anticipative case where only a finite window of future capacity variations is known, we propose an online policy: Greedy Fixed Horizon Control (GFHC). An upper bound on the competitive ratio of GFHC and GPCT is established showing how performance loss depends on the window size, receiver playback buffer, and capacity variability. We also consider the multiuser case where one can exploit both future temporal and multiuser diversity. Finally we investigate the impact of uncertainty in knowledge of future capacity variations, and propose an offline approach as well as an online algorithm to deal with such uncertainty. Our simulations and evaluation based on a measured wireless capacity trace exhibit robust potential gains for our proposed transmission schemes.

Second, we consider the design of scheduling algorithms for dynamic D2D networks where links are set up to mediate file transfers among close by users, but with limited 'contact' times or deadlines. Our focus is on improving three performance metrics: (1) overall offload data; (2) number of transfers which complete within their deadlines; and (3) file transfer delays. The design and evaluation of existing D2D scheduling algorithms has mostly focused on optimizing the network's sum rate subject to fairness concerns for a *fixed* set of links. Starting from a dynamic scenario where links share a single collision domain, this dissertation investigates optimal scheduling algorithms which exploit application layer context. These results drive our proposal for application-aware versions of state-of-the-art schedulers such as FlashLinQ and CSMA/CA. Our simulations show that these schedulers could achieve substantial performance improvements depending on the operational scenario, i.e., density of users, file and contact time distributions, etc. We also show that performance can be further enhanced by incorporating both application-aware scheduling and admission control. Finally we investigate scenarios where D2D links have no deadlines. We show that application-aware schedulers result in smaller average file transfer delays and can increase the stability region of the system. This is in part due to their ability to reduce spatial clustering of links resulting from interference coupling in the dynamic setting.

# Table of Contents

# List of Figures

# Chapter 1

# Introduction

We have seen, and continue to see increasing growth in traffic and reliance on wireless networks. The main part of this growth comes from the increasing video traffic such networks support. According to the Cisco Visual Networking Index [1], mobile data traffic will increase 10-fold between 2014 and 2019 and about 80 percent of the consumer Internet traffic will be video including real-time video, video on demand (VoD), video conferencing etc.

Many innovations have been made to accommodate this growth. For example video transport protocols have gone through a major upgrade in the last decade, changing from Real-time Transport Protocol (RTP) which is based on UDP and progressive video download which delivers videos as data files, to a new set of HTTP based video streaming protocols (e.g., DASH, Apple's HTTP live streaming, etc.). HTTP based video streaming protocols provide users with different quality choices and enable adaptation of the rate (and thus video quality) to the level of congestion on the network. They thus are able to improve users' Quality of Experience (QoE) when the networks are limited in capacity.

Software-Defined Networking (SDN) is another technology that promises to help to improve networks ability to manage Quality of Service (QoS). SDN separates the control plane from the data plane and thus enables monitoring, prediction and control on a per flow basis. Applications can talk to the SDN controller via APIs to collect network information and control the resources allocated to its flows (e.g., bandwidth allocation, route selection, etc.), providing new methods and opportunities to improve users' QoE.

Also new technologies are enhancing the ability of systems to predict human behavior. For example, many people traveling in their cars use GPS devices, which are able to predict their future locations. Taxis (e.g., Uber cars) and public transportations (e.g., UT shuttles) provide their real-time locations via smart phone apps. By using a wireless signal strength map or a coverage map along with the predictions of the future location, a network can make a prediction of the wireless capacity a mobile is likely to see in the future. In addition there are some features of SDN (e.g., time series data repository in OpenDaylight) which enable storage, analysis and prediction of the network statistics, and can be easily accessed by applications, adding to the predictability of the future loads.

These new technologies provide both new opportunities and demands to make network schedulers "smarter". This dissertation exploits two key ideas towards improving wireless network schedulers: the first is to let the schedulers exploit application layer information of the traffic flows being scheduled so as to improve both the system performance and the end users' QoE; the second idea is to exploit the knowledge of future capacity variations.

In Chapter 2, we investigate how knowledge of future capacity variations can be used towards reducing system utilization (increasing capacity) while minimizing video rebuffering. We propose a new class of video transport protocol which exploits both application layer information (e.g., playback buffer status) and knowledge of future capacity variations. This is an attempt towards devising anticipative networking, and to our knowledge the first work on devising video transport protocols that exploit knowledge of future capacity variations. We study the case where we have perfect knowledge of future capacity variations, and the case where we only have perfect knowledge for a finite window in the future, and finally the case where the predictions are imperfect. Simulation and evaluations exhibit this approach can provide robust potential gains.

In Chapter 3, we study application-aware opportunistic scheduling in Device-to-Device (D2D) communication networks. D2D is an emerging technology that facilitates wireless devices to directly exchange transmissions among each other, with the potential to enable substantial traffic offloads from wireless infrastructure while increasing area spectral efficiency. Existing D2D schedulers do not consider network dynamics (i.e., users come and go) and thus typically focus on improving sum network throughput without considering flow-level performance (e.g., number of file transfer completions within a given deadline). In Chapter 3 we propose application-aware versions of state-of-the-art schedulers such as FlashLinQ and CSMA/CA. Our simulations show that these schedulers could achieve substantial improvements on flow-level performance without sacrificing the sum throughput in such networks. Another interesting finding is that the proposed application-aware schedulers are able to reduce spatial clustering of links resulting from the coupling of link transmissions through interference in the dynamic setting.

Chapter 4 concludes the dissertation and discusses some future directions.

# Chapter 2

# Video Delivery in Wireless Networks: Value of Knowing the Future

## 2.1 Introduction

Video delivery over wireless networks is expected to grow quickly in the next few years. Recent studies (see [1]) show that mobile data traffic will increase 10-fold between 2014 and 2019 and about 80 percent of the traffic will be video including real-time video, video on demand (VoD), video conferencing etc. The wireless infrastructure can hardly keep pace with such growth, thus it is important to make effective use of the available wireless resources in video delivery.

Even the successor to current cellular systems, 4G broadband, promises not only improvements in overall capacity but also, unfortunately, higher degrees of capacity variability, particularly in the case of mobile users. Our premise in this chapter, is that approaches can be devised that exploit such

This chapter is based on my work in [27] (published) and [26] (in submission to IEEE/ACM Trans. on Networking), which are co-authored by Dr. Gustavo de Veciana.

capacity variations, and the nature of the underlying services. Indeed, mobile devices are increasingly equipped with video playback buffers, giving more flexibility in exploiting capacity variations without interrupting playback.

In this chapter we design application-layer opportunistic transport for stored video over wireless networks with a slow varying (average) capacity. We focus on two key ideas: (1) scheduling data transmission when capacity is high; and (2), exploiting knowledge of *future* capacity variations. The latter is possible when users' future locations are known, which can in turn be used to infer their future wireless coverage/capacity. For example this is the case for users on public transportation buses/trains, or others using navigation systems in their cars. In fact, even without prior knowledge of vehicle routes, one can still infer future vehicle's mobility. Indeed [39] demonstrate the effectiveness on real data of "K Nearest Trajectories" an algorithm to predict future capacity variations for vehicles. More generally, humans' mobility patterns tend to be highly predictable, [40] show a potential 93% average 'predictability' suggesting knowing the future (in this regard) is quite reasonable.

Let us consider a simple example. Suppose a server is delivering a constant-bit-rate stored video to a mobile user. The length of the video is 4 seconds and the streaming rate is 200kbps, thus the size of the video is 800kb. Suppose the capacity variation is as shown in Fig. 2.1. If the video is delivered at a fixed rate of roughly 200kbps then two of the four slots are partially utilized, leading to a 75% system utilization. However if the video is delivered greedily, i.e., at the full available capacity, then transmission completes in 2.5secs, resulting in utilization of 62.5%. However, it is easy to see that an optimal schedule would send at a full rate in Slots 1 and 3, and transmit nothing in Slots 2 and 4, which guarantees smooth video playback and results in a minimal utilization of 50%. This scheme transmits when channel conditions are good, i.e., we set a threshold and transmit only when the capacity is above the threshold. The threshold choice, however, depends on the future capacity variations and video playback requirements. We will call this a 'thresholding scheme' and define it formally in the sequel.

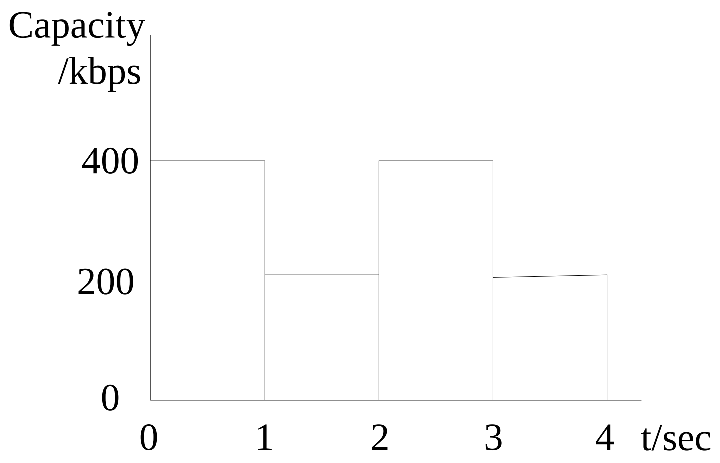


Figure 2.1: The channel variations in four time slots. Transmitting the video at peak capacity in the first and third slots and transmitting nothing in the second and fourth slots, gives the lowest system utilization.

*Related Work.* There has been a substantial body of literature on stored video delivery. Below we focus on a few key works that are closest to ours, in terms of methodology, but differ terms of their objectives. A piecewise constant-rate transmission scheme was developed in [29], wherein dynamic programming was applied to find the optimal schedules for a variety of optimization criteria, in particular minimizing the maximum transmission rate subject to a maximum initial delay, minimizing the maximum transmission rate subject to a maximum user interaction delay, and minimizing the average user interaction delay subject to a constraint on the maximum transmission rate. Similarly [37] suggest several ways to reduce transmission rate variability for stored video delivery and propose an optimal smoothing scheme which is further explored in [9, 17]. However, if one or more users will see highly variable wireless capacity, it should be clear that smoothing transmissions may not be the right objective; indeed, 'bursty' transmissions might be preferable. [35] consider a scenario where video streams share a source server. They propose a centralized prefetching protocol which schedules transmissions to users whose playback buffer queues are shortest. This does not *explicitly* account for capacity variations – present or future. The work in [36], proposes a decentralized protocol targeting a scenario where streams from multiple video servers share a multiplexer and dynamically adjust their transmit windows in a window flow control mechanism to mitigate packet loss.

The key differentiating element of our work from the above is a formal investigation of how knowledge of future capacity variations could be used towards reducing utilization (increasing capacity) while minimizing video rebuffering. The idea of exploiting knowledge of future capacity variations has been studied by a few works, e.g., [41] proposes an optimal proactive resource allocation which assumes users' demand can be tracked, learned and predicted.

And predictable peak-hour requests are served during off-peak time to reduce cost in the networks. This is similar to our work from the perspective of exploiting temporal diversity in the future capacity variations. However it has a much slower time scale, i.e., offloading traffic from peak time to off-peak time considers a time scale of hours, as compared to a time scale of seconds in our work. Also their paper does not consider the delay sensitivity of video delivery (e.g., rebuffering constraints) as we do. In [6] the authors exploit future knowledge of capacity variations to improve the performance of video delivery. Their goal is to trade off video quality with rebuffering, which is different from our objective. Also they did not look into the case as we do, where predictions of future capacity variations have error. [7] has a similar objective as our work. It first proposes an offline optimal resource allocation that minimizes system utilization under perfect capacity predictions, then proposes an online algorithm to deal with uncertainty in the predictions. However, [7] is different from our work in the following aspects: (1) the offline algorithm in [7] only considers the optimization of one video user and treats all the other users as background traffic, but in our work we exploit multiuser diversity by jointly considering video traffic from multiple concurrent users; (2) we perform analysis and show an upper bound on the performance degradation caused by prediction error, which is not studied in [7]; and (3) to deal with prediction error, [7] proposes an online algorithm that iteratively uses the offline algorithm in a finite horizon, which has higher complexity and requires more statistical capacity information, as compared to our BDT algorithm which does not make use of the offline optimal algorithm and only requires the prediction of mean capacity in a finite horizon.

*Contributions and Organization.* This chapter of the dissertation proposes a new class of cross-layer transmission schedules which minimize system utilization (and thus possibly transmit/receive energy) while avoiding, if at all possible, rebuffering/delays. We propose approaches to exploit the knowledge of future capacity variations and study how to deal with error in the predictions of capacity variations. In Section 2.2 we study the single-user anticipative case where all future capacity variations are known beforehand; we formally establish the optimal transmission schedule is a Generalized Piecewise Constant Thresholding (GPCT) scheme. In Section 2.3 we consider the single-user partially anticipative case where only a finite window of future capacity variations is known, we propose an online Greedy Fixed Horizon Control (GFHC). An upper bound on the competitive ratio of GFHC and GPCT is established clearly indicating how performance loss depends on the window size, receiver playback buffer, and capacity variability. In Section 2.4, we consider the multi-user anticipative case, and we develop two multiuser schemes based on GPCT, which are suboptimal, but straightforward to implement, and able to achieve good performance. Simulations described in Section 2.5 explore the performance gains achievable for a typical scenario and explore the impact of correlation in capacity variations on these gains. Our simulations show the potential gains for such opportunistic transmission schemes exhibit an up to 70% reduction in the system utilization. In Section 2.6 we deal with the uncertainty in capacity predictions. We first propose an offline approach and show an upper bound on the performance degradation caused by prediction error. We then propose an online Buffer-Dependent Thresholding (BDT) algorithm that only requires the prediction of mean capacity for a finite window in the future. Simulations show that BDT is effective and robust. Section 2.7 briefly concludes the chapter.

## 2.2 Single-User Anticipative Case

We will first consider the anticipative case where a server is delivering video content to a single user and future variations in wireless capacity are known beforehand.

### 2.2.1 Model Formulation

Consider a video server streaming a stored video to a mobile user via one (or more) base station(s). Suppose the wireless part is the bottleneck so that the application layer throughput mainly depends on the wireless capacity. Further let us focus on slow variations in wireless capacity, e.g., on the order of secs, so that timely end-to-end feedback actions can be realized before the capacity changes too much. Let $c(t)$ be the average[1]of the peak capacity at time $t$, and $r(t)$ be the actual transmission schedule such that $0 \leq r(t) \leq c(t)$. Then $r(t)/c(t)$ can be roughly regarded as the system utilization at time $t$. Let $s(t)$ be the cumulative amount of data sent and received[2], i.e. $s(t) = \int_0^t r(\tau)d\tau$.

Now suppose the video to be transmitted has a finite length $T$ (seconds), a finite size $S$ (bits), and define two functions $l(\cdot)$ and $u(\cdot)$ associated with requirements on the video's transmission. Suppose the transmission begins at time 0 and no interruptions (rebuffering) happen during the transmission. Let $l(t)$ denote the cumulative data consumed if the user watches the first $t$ secs of the video, where $t \in [0, T]$. Note that if the user's playback buffer has finite size, s/he can only receive a limited amount of data before viewing it. We define $u(t)$ to be the maximum cumulative amount of data that can be received by the user over $[0, t]$, where $t \in [0, T]$. Further we assume $l(\cdot)$ and $u(\cdot)$ are both nondecreasing piecewise constant and right continuous functions as depicted in Fig. 2.2, where jumps happen at times $t_0, t_1, t_2, ..., t_n$. The jumps might correspond to individual (or groups of related, e.g., intra-coded/predicted) frames being displayed and which are no longer necessary to reconstruct future content. Note that the jump points are chosen such that $t_0 = 0$, $t_n = T$, and $t_0 \leq t_1 \leq t_2 \leq ... \leq t_n$. Also note that it is possible for $t_i = t_{i+1}$ where $t_i$ is a jump point of $u(\cdot)$ and $t_{i+1}$ is a jump point of $l(\cdot)$. Further let $I_l = \{i : t_i \text{ is a jump point of } l(\cdot)\}$ and $I_u = \{i : t_i \text{ is a jump point of } u(\cdot)\}$ denote the sets of jump point indices for the two piecewise constant functions. Note that if the client has a fixed playback buffer size $b$, then there is a vertical gap of size $b$ between $u(\cdot)$ and $l(\cdot)$. However Fig. 2.2 shows a more general case where there may be time varying buffer allocations for playback.

Next, we define the cost function in our model. The cost is a sum of

[1]The average is taken over periods on the order of seconds to smooth out fast capacity variations.

[2]We assume that transport exploits playback buffering and thus can be made reliable.

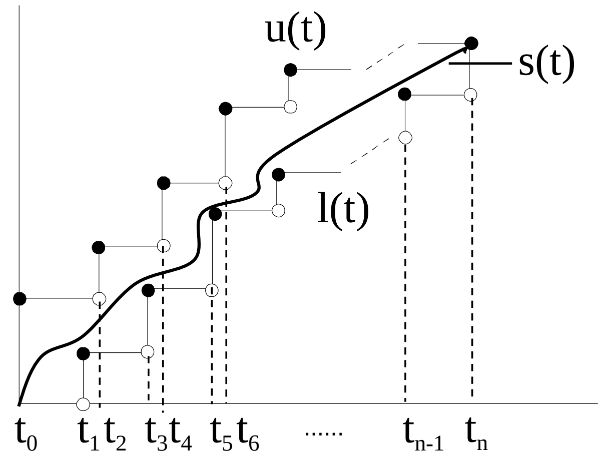


Figure 2.2: The piecewise constant functions $l(\cdot)$, $u(\cdot)$ and the cumulatively transmitted data $s(\cdot)$. $l(t)$ is the cumulative amount of data consumed (i.e. watched) by the user over $[t_0, t]$. $u(t)$ is the maximum cumulative amount of data that can be received by the user over $[t_0, t]$. Jumps happen at times $t_1, t_2, t_3, ..., t_n$. $s(\cdot)$ lies between $l(\cdot)$ and $u(\cdot)$ if there are no playback buffer underflow and overflow.

two terms: the average system utilization $cost_u$ and the rebuffering time $cost_r$. Assuming no rebuffering, the average system utilization during video watch period $[0, T]$ can be defined as:

$$cost_u = \frac{1}{T}\int_0^\infty \frac{r(t)}{c(t)}dt.$$

Note we assume $r(t) = 0$ after the transmission finishes, so the above integration actually has a finite time horizon. We use the above cost function versus $\int_0^T r(t)dt / \int_0^T c(t)dt$ because the former properly captures the reduction in utilization in a system with time varying capacity using opportunistic 'scheduling.'

The rebuffering cost depends on the waiting time a user experiences when buffer underflow occurs during video playback. We assume that playback buffer underflows can only happen at the jump points for $l(\cdot)$, i.e., times in $I_l$. This is a reasonable assumption since the video playback can be paused only after an entire frame is displayed. Further we denote the associated waiting times as $\tau_0, \tau_1, ..., \tau_n$. Note the functions $u(\cdot)$, $l(\cdot)$ and the jump points in Fig. 2.2 are known before transmission, and thus they do not take into account delays due to rebuffering times. In other words, $t_0, t_1, t_2, ..., t_n$ are fixed constants before transmission, which may be shifted by $\tau_0, \tau_1, ..., \tau_n$, which are variables whose values depend on the actual wireless capacities and rebuffering during transmission.

Our rebuffering cost is defined as

$$cost_r = a\sum_{i=0}^{n} \tau_i,$$

where $a$ is a constant. In practice it is natural to put a higher priority on minimizing rebuffering over system utilization. Thus the constant $a$ should be large enough such that one cannot obtain a lower total cost by increasing rebuffering time to reduce system utilization. Suppose we know upper and lower bounds on the wireless capacity, denoted $c_{max}$ and $c_{min} > 0$. Then adding a waiting time $\tau$ can result in a maximum reduction in system utilization of no more than $\frac{\tau(c_{max}-c_{min})}{Tc_{min}}$, which is obtained by assuming video content of size $c_{max}\tau$ is transmitted at rate $c_{max}$ during the waiting time $\tau$ instead of being transmitted at rate $c_{min}$ during a period of $\frac{c_{max}\tau}{c_{min}}$, which results in a reduction of $\frac{c_{max}\tau}{c_{min}T} - \frac{\tau}{T} = \frac{\tau(c_{max}-c_{min})}{Tc_{min}}$ in system utilization. Thus if we set $a = \frac{c_{max}-c_{min}}{Tc_{min}}$, we achieve our goal of strictly prioritizing minimization of rebuffering cost over system utilization.

The above rebuffering cost function is a simplification, since it only captures the cumulative rebuffering time without accounting for how rebuffering periods are distributed over time. In fact, a viewer may prefer that rebuffering happen all at once rather than being spread out and causing several interruptions during playback. To capture this concern one can take a concave function to each waiting time $\tau_i$ before summing them up so as to penalize a higher number of interruptions. In the rest of the chapter, we will focus on the simpler cost function. The "concave-sum" version can be studied similarly.

Also note that if we assume $c_{min} > 0$, the normalization factor $T$ in $cost_u$ can be replaced by $\tilde{T} = \max[\frac{S}{c_{min}}, T]$ ensuring the utilization cost remains below 1 without changing the overall character of the objective function. Correspondingly we choose $a = \frac{c_{max}-c_{min}}{\bar{T}c_{min}}$ to prioritize minimization of rebuffering. We will use these from now on.

When minimizing our cost, we are constrained to ensure that the playback buffer does not drop below 0 or exceed the available buffer. This can be captured by the following constraints:

$$\int_0^{t_i+\sum_{k=0}^{i}\tau_k} r(t)dt + b_0 \geq l(t_i),\ \forall i \in I_l$$
$$\int_0^{t_i+\sum_{k=0}^{i}\tau_k} r(t)dt + b_0 \leq u(t_i),\ \forall i \in I_u$$

where $b_0$ is the initial buffer content (in bits) that is not accounted in the transmission schedule (often, it is 0). Note the upper bound in the above integration intervals is the current total time $t_i + \sum_{k=0}^{i}\tau_k$ which consists of the current time of the video $t_i$ and the cumulative rebuffering time $\sum_{k=0}^{i}\tau_k$. We call these buffer underflow and overflow constraints respectively. Also we have the terminal condition:

$$\int_0^{t_n+\sum_{k=0}^{n}\tau_k} r(t)dt + b_0 = l(t_n),$$

which captures the fact that the video transmission must finish prior to final video playback.

Letting $\vec{\tau} = (\tau_0, \tau_1, ..., \tau_n)$ and $r(\cdot)$ denote the rebuffering times and video transmission schedule, we summarize our overall goal in terms of the following optimization problem:

**Optimal Streaming Problem**

$$\min_{r(\cdot),\vec{\tau}} \frac{1}{\tilde{T}}\int_0^\infty \frac{r(t)}{c(t)}dt + \frac{c_{max}-c_{min}}{\tilde{T}c_{min}}\sum_{i=0}^{n}\tau_i, \tag{2.1}$$

$$\text{s. t.} \int_0^{t_i+\sum_{k=0}^{i}\tau_k} r(t)dt + b_0 \geq l(t_i),\ \forall i \in I_l,$$

$$\int_0^{t_i+\sum_{k=0}^{i}\tau_k} r(t)dt + b_0 \leq u(t_i),\ \forall i \in I_u,$$

$$\int_0^{t_n+\sum_{k=0}^{n}\tau_k} r(t)dt + b_0 = l(t_n).$$

---

We say the above optimization problem is a video delivery optimization with initial state $b_0$ and terminal state $l(t_n)$. Note this problem is not convex since the constraints are not convex. However it always has a feasible solution if $c_{min} > 0$. In the sequel, we will first deal with the simpler situation where the optimization has a feasible solution without rebuffering, i.e., $\vec{\tau} = \vec{0}$. Subsequently we generalize the solution to situations where rebuffering is necessary.

### 2.2.2 Piecewise Constant Thresholding (PCT) Algorithm Under No Rebuffering Assumption

Assume $c(\cdot)$ is such that there is a feasible solution to Optimal Streaming (2.1) without rebuffering, i.e., $\vec{\tau} = \vec{0}$. Note in the model above, the constant $a$ is chosen large enough such that $cost_r$ dominates $cost_u$ in the sense that we cannot achieve a lower cost by adding rebuffering time. Thus under the no rebuffering assumption, Optimal Streaming (2.1) is equivalent to the Min Utilization (2.2) given below.

**Min Utilization Problem**

$$\begin{aligned} \min_{r(\cdot)} \ & \frac{1}{\tilde{T}} \int_0^{t_n} \frac{r(t)}{c(t)} dt, \\ \text{s. t. } & \int_0^{t_i} r(t)dt + b_0 \geq l(t_i), \ \forall i \in I_l, \\ & \int_0^{t_i} r(t)dt + b_0 \leq u(t_i), \ \forall i \in I_u, \\ & \int_0^{t_n} r(t)dt + b_0 = l(t_n). \end{aligned} \tag{2.2}$$

In this subsection we determine delivery schedules that solve this problem, i.e., schedules $r(\cdot)$ achieving a minimum utilization while ensuring the cumulative data $s(\cdot)$ lies between $u(\cdot)$ and $l(\cdot)$ as shown in Fig. 2.2. Before introducing our algorithm let us define some terminology.

**Definition 1** *A* single threshold transmission scheme *on an interval* $[t_s, t_e]$ *with initial state* $s(t_s)$ *and terminal state* $s(t_e)$ *is such that for* $t \in [t_s, t_e]$*:*

$$r(t) = \begin{cases} c(t) & \textit{if } c(t) > \alpha \textit{ or } c(t) = \alpha, t \leq \tau \\ 0 & \textit{if } c(t) < \alpha \textit{ or } c(t) = \alpha, t > \tau \end{cases}$$

*where* $\alpha \in [0, \max_{t \in [t_s, t_e]} c(t)]$*, and* $\tau \in [t_s, t_e]$ *are thresholds such that:*

$$\int_{t_s}^{t_e} r(t)dt = s(t_e) - s(t_s).$$

*Further, we denote by* $\beta_{\alpha,\tau,t_s}(t) = \int_{t_s}^{t} r(\tau)d\tau + s(t_s)$ *the cumulative amount of transmitted data for* $t \in [t_s, t_e]$.

Note we refer to this as a “single” threshold scheme although in fact it is a pair: $\alpha$ is a threshold on wireless capacity, and $\tau$ is a threshold on time. Basically the scheme transmits data only when the capacity is above the threshold $\alpha$. Thus continuously decreasing $\alpha$ will increase the cumulative amount of data transmitted with a potential for jumps if the capacity stays constant at some levels. By varying $\tau$ we can further control transmission so that the cumulative data transmitted varies continuously over its range.

The goal of the single threshold transmission scheme is to find $\alpha$ and $\tau$, such that given an initial state $s(t_s)$ and wireless capacity $c(t)$, $t \in [t_s, t_e]$, the terminal state $s(t_e)$ is achieved. The thresholds can be computed by using binary search algorithm. However in practice, we can assume that the capacity $c(\cdot)$ is a piecewise constant function, i.e., we can use a discrete-time system model, in which case we assume the interval $[t_s, t_e]$ is divided into $m$ slots and the capacity function is constant on each slot. We denote the $i$th slot as $[p_{i-1}, p_i]$, $i = 1, 2, ..., m$ and denote the associated capacity as $c_i$. Under these assumptions, the thresholds of the single threshold transmission scheme can be found by using a sorting algorithm, which provides a unique one-to-one mapping $f$: $\{1, 2, ..., m\} \to \{1, 2, ..., m\}$ such that $f(i) < f(j)$ if and only if $c_i > c_j$ or $c_i = c_j, i < j$. Note this mapping sorts the capacities of the slots in descending order. Then we can sum the sorted capacities up from the highest value to the lowest value, such that the sum exceeds $s(t_e) - s(t_s)$. Suppose this happens after we add the $k$th sorted slot, and the sum exceeds $s(t_e) - s(t_s)$ by $s$. Then the thresholds are $\alpha = c_{f^{-1}(k)}$ and $\tau = p_{f^{-1}(k)} - (p_{f^{-1}(k)} - p_{f^{-1}(k)-1}) \frac{s}{c_{f^{-1}(k)}}$.

Also note that under a single threshold transmission scheme, the server only has two choices: send at peak capacity or send nothing. Hence, the corresponding utilization is proportional to the length of time the server is sending data.

We define two types of constraint violations associated with Min Utilization (2.2). We refer to a buffer underflow violation when one of the first set of constraints (buffer underflow constraints) in Min Utilization is not met, and we refer to buffer overflow violation when one of the second set of constraints (buffer overflow constraints) is not met.

Let $u_{[a,b]}$, $l_{[a,b]}$, $c_{[a,b]}$, $r_{[a,b]}$ and $\beta_{\alpha,\tau,t_s,[a,b]}$ denote the values of the functions $u(\cdot)$, $l(\cdot)$, $c(\cdot)$, $r(\cdot)$ and $\beta_{\alpha,\tau,t_s}(\cdot)$ on the interval $[a,b]$ respectively. Then the optimal transmission schedule for (2.2), $r^\star_{[t_0,t_n]}$ can be calculated using Piecewise Constant Thresholding (PCT) algorithm (Algorithm 1) with initial state $s_s = b_0$ and terminal state $s_e = l(t_n)$.

**Algorithm 1** **Piecewise Constant Thresholding (PCT)**

```
Input: $s_s$, $s_e$, $l_{[t_0,t_n]}$, $u_{[t_0,t_n]}$, $c_{[t_0,t_n]}$
 1: $m \leftarrow 0$, $r^\star_{[t_0,t_n]} \leftarrow 0$
 2: $t_s \leftarrow t_0$, $t_e \leftarrow t_n$
 3: repeat
 4:     $(t_b, s_b, r^\star_{[t_s,t_b)}) =$
                Breakpoint$(s_s, s_e, t_s, t_e, l_{[t_s,t_e]}, u_{[t_s,t_e]}, c_{[t_s,t_e]})$
 5:     $m \leftarrow m + 1$
 6:     $s_s \leftarrow s_b$, $t_s \leftarrow t_b$
 7: until $s_s = s_e$
Output: $r^\star_{[t_0,t_n]}$, $m$
```

**Algorithm 2** **Breakpoint**

```
Input: $s_s$, $s_e$, $t_s$, $t_e$, $l_{[t_s,t_e]}$, $u_{[t_s,t_e]}$, $c_{[t_s,t_e]}$
 1: loop
 2:   Apply single threshold transmission scheme on $[t_s, t_e]$ with initial state
      $s_s$ and terminal state $s_e$ to get $\alpha$, $\tau$ and $r_{[t_s,t_e]}, \beta_{\alpha,\tau,t_s,[t_s,t_e]}$.
 3:   if $\beta_{\alpha,\tau,t_s,[t_s,t_e]}$ does not violate any buffer underflow or overflow con-
      straints then
 4:     $t_b \leftarrow t_e$, $s_b \leftarrow \beta_{\alpha,\tau,t_s}(t_e)$
 5:     break
 6:   else
 7:     find the largest $i$, $t_s \leq t_i \leq t_e$, such that the violations on $[t_s, t_i]$ are
        of the same type
 8:     if the violation type is buffer overflow then
 9:       $s_e \leftarrow u(t_i)$, $t_e \leftarrow t_i$
10:     else
11:       $s_e \leftarrow l(t_i)$, $t_e \leftarrow t_i$
12:     end if
13:   end if
14: end loop
Output: $t_b$, $s_b$, $r_{[t_s,t_e)}$
```

The logic underlying PCT is as follows. First try to apply the single threshold scheme on the interval $[t_0, t_n]$ with initial state $s_s$ and terminal state $s_e$. If the resulting transmission schedule $r_{[t_0,t_n]}$ meets the buffer overflow and underflow constraints, then it is the final solution; we call it a 1-piecewise constant thresholding solution. Otherwise if any of the constraints is violated, divide the time interval $[t_0, t_n]$ into two subintervals $[t_0, t_b)$ and $[t_b, t_n]$ at some point $t_b$ (we call it a breakpoint), which is carefully chosen such that we can once again run the single threshold transmission scheme on $[t_0, t_b)$ to obtain a feasible solution which is output as the solution on subinterval $[t_0, t_b)$. Then the remaining subinterval $[t_b, t_n]$ is treated as a new interval and we apply the same procedure to it as on $[t_0, t_n]$. This recursive procedure is realized by a repeat loop in Algorithm 1, where in each loop, the function "**Breakpoint**" is called to calculate the breakpoint $t_b$, and the transmission schedule $r(t)$ for $t \leq t_b$. The function "**Breakpoint**" is described in Algorithm 2, where we use a loop to find the breakpoint, and in each loop the single threshold transmission scheme described in Definition 1 is used. Finally if Algorithm 1 takes $m$ loops to finish, then we end up with $m$ subintervals and we call the solution an $m$-piecewise constant thresholding solution.

The following theorem states that PCT provides an optimal solution for Min Utilization (2.2).

**Theorem 1** *If there exists a feasible solution to Min Utilization (2.2), then PCT determines an optimal transmission schedule.*

### 2.2.3 Proof of Optimality

We prove the optimality of PCT by induction on the total number of jumps $n$ in $u(\cdot)$ and $l(\cdot)$.

First, when $n = 1$, the algorithm is optimal due to the characteristic of the thresholding scheme, i.e., one transmits only at the highest capacities.

As an induction hypothesis, suppose the algorithm is optimal for all the feasible video delivery optimization problems with $n$ jumps where $n \leq k - 1$. We then show that it is also optimal for problems with $k$ jumps.

If PCT results in a 1-piecewise constant thresholding solution, then it is optimal due to the characteristic of the thresholding scheme. Otherwise, if we obtain an $m$-piecewise solution denoted by $s_0(\cdot) = \int_0^{\cdot} r_0(t)dt$, where $m > 1$, then suppose the first breakpoint found by the algorithm is $t_i$, and without loss of generality we suppose $s_0(t_i) = u(t_i)$. (The other possible case is $s_0(t_i) = l(t_i)$, which can be addressed in a similar manner.) In the sequel, we will show that the solution $s_0(\cdot)$ is as good as any other feasible solution $s_1(\cdot)$. We consider two cases separately.

**Case 1.** Suppose $s_1(t_i) = u(t_i)$. In this case, we can change the constraint $s(t_i) \leq u(t_i)$ to $s(t_i) = u(t_i)$ in the original Min Utilization (2.2). Then we will get a new optimization problem denoted by $\tilde{O}$. Note that both $s_0(\cdot)$ and $s_1(\cdot)$ are feasible solutions for $\tilde{O}$. On the other hand, $\tilde{O}$ can be divided into two video delivery optimization problems $\tilde{O}_1$ and $\tilde{O}_2$, where $\tilde{O}_1$ is on the interval $[t_0, t_i]$, $\tilde{O}_2$ is on the interval $[t_i, t_k]$. And we can solve them separately to obtain the optimal solution for $\tilde{O}$. Note that it follows from the characteristic of the proposed algorithm that, if we apply the algorithm on $\tilde{O}_1$ and $\tilde{O}_2$ separately, we will get the same result as $s_0(\cdot)$. Further, by the induction hypothesis, the proposed algorithm gives optimal solutions on $\tilde{O}_1$ and $\tilde{O}_2$, so $s_0(\cdot)$ is an optimal solution on $\tilde{O}$. Thus, $s_0(\cdot)$ is as good as $s_1(\cdot)$.

**Case 2.** Suppose $l(t_i) \leq s_1(t_i) < u(t_i)$. In this case, we can remove the constraint $s(t_i) \leq u(t_i)$ from the original Min Utilization (2.2), to get a new optimization problem $\tilde{O}$. Note that both $s_0(\cdot)$ and $s_1(\cdot)$ are feasible solutions for $\tilde{O}$, and by the induction hypothesis, we can apply the proposed algorithm on $\tilde{O}$ to get an optimal solution $s_2(\cdot)$. Suppose the utilizations of $s_1(\cdot)$ and $s_2(\cdot)$ are $cost_{u1}$ and $cost_{u2}$ respectively, then it should be such that $cost_{u1} \geq cost_{u2}$. Note since $t_i$ is a breakpoint found by PCT algorithm and $s_0(t_i) = u(t_i)$ which corresponds to a buffer overflow violation, we can claim that if we relax the buffer overflow constraints at $t_i$, the associated optimal cumulative transmission at $t_i$ should be greater than $u(t_i)$, i.e., $s_2(t_i) > u(t_i)$. Now we construct a feasible solution $s'_0(\cdot)$ to $\tilde{O}$ by doing time sharing between $s_1(\cdot)$ and $s_2(\cdot)$ as follows,

$$s'_0(t) = \lambda s_1(t) + (1-\lambda)s_2(t), \ t \in [t_0, t_k],$$

where $\lambda$ is a parameter chosen from $(0, 1)$ such that,

$$s'_0(t_i) = s_0(t_i) = u(t_i).$$

Then, the utilization of $s'_0(\cdot)$ can be calculated as

$$cost'_{u0} = \lambda cost_{u1} + (1-\lambda)cost_{u2}.$$

Thus, we have $cost_{u2} \leq cost'_{u0} \leq cost_{u1}$. That means, scheme $s'_0(\cdot)$ is as good as $s_1(\cdot)$. But according to the result in Case 1, $s_0(\cdot)$ is as good as $s'_0(\cdot)$, so we can claim that $s_0(\cdot)$ is as good as $s_1(\cdot)$.

Thus, by induction the optimality of the proposed algorithm is proved.

### 2.2.4 General Piecewise Constant Thresholding (GPCT)

In this subsection, we generalize PCT to solve Optimal Streaming (2.1), i.e., including the possibility of rebuffering. By the results in Subsection II.B, we immediately have the following corollary:

**Corollary 1** *If there exists a feasible solution to Optimal Streaming (2.1) with $\vec{\tau} = \vec{0}$, then PCT solves the optimization.*

However, if Optimal Streaming (2.1) does not allow a feasible solution with $\vec{\tau} = \vec{0}$, then rebuffering must happen and PCT cannot be used directly. Instead we require a modification of PCT to deal with the rebuffering issue. Before we state the new algorithm, we introduce some definitions.

**Definition 2** *We say a transmission scheme is* ***greedy*** *if it sends as much as possible given the capacity and playback buffer constraints, i.e., the greedy transmission scheme tries to keep the buffer full. During the transmission the video plays as long as the playback buffer is not empty and rebuffering happens only when the buffer underflows. Consider a greedy transmission scheme starting at time $t_1$, and for which rebuffering happens at time $t_2$ where $t_2 > t_1$.*

*We say this rebuffering is an **I-rebuffering** if* 100% *utilization was achieved during* $(t_1, t_2)$*. Otherwise, if the greedy schedule on* $(t_1, t_2)$ *was precluded from realizing* 100% *utilization, i.e., due to a limited playback buffer, we call the rebuffering a **B-rebuffering**.*

Note if the playback buffer size is infinite (or at least larger than the video size), then there can only be I-rebufferings. While B-rebufferings are caused by playback buffer overflows. During a video transmission, both I-rebufferings and B-rebufferings can happen. Suppose they happen at the jump points $t_{n_1}, t_{n_2}, ..., t_{n_k}$, denote the corresponding rebuffering times as $\tau_{n_1}, \tau_{n_2}, ..., \tau_{n_k}$, and define $d(n_i) = \sum_{j=1}^{i} \tau_{n_j}$, the cumulative rebuffering time up to time $t_{n_i}$.

With the above definitions we can state our solution to Optimal Streaming (2.1) as General Piecewise Constant Thresholding (GPCT) (Algorithm 3) with initial state $s_s = b_0$ and terminal state $s_e = l(t_n)$.

In GPCT, we first use greedy transmission scheme to find where the rebufferings need to happen and identify all the associated rebuffering types. If no rebuffering happens, then the algorithm degenerates to PCT, and Corollary 1 ensures optimality. Otherwise if an I-rebuffering occurs, we have to use greedy transmission scheme to minimize the waiting time since our rebuffering cost dominates the utilization cost. However if a B-rebuffering happens, we can use PCT before the latest playback buffer overflow, which is denoted as $\tilde{t}(n_i)$ in the algorithm, to achieve the minimum system utilization. Thus we have the following theorem.

**Algorithm 3** **General Piecewise Constant Thresholding (GPCT)**

```
Input: $s_s$, $s_e$, $l_{[t_0,t_n]}$, $u_{[t_0,t_n]}$, $c_{[t_0,t_0+\tilde{T}]}$
 1: Perform greedy transmission on $[t_0,\infty]$ until $s_e - s_s$ of the video is delivered. Suppose the rebufferings under the greedy scheme happen at jump points $t_{n_1}, t_{n_2}, ..., t_{n_k}$, denote the corresponding rebuffering times as $\tau_{n_1}, \tau_{n_2}, ..., \tau_{n_k}$, and identify their rebuffering types (I or B). If a B-rebuffering happens at $t_{n_i}$ for some $i \in [1,k]$, denote $\tilde{t}(n_i) = \max\{t < t_{n_i} + d(n_i) : \text{playback buffer is full at } t\}$
 2: $t_{n_0} \leftarrow 0$, $d(n_i) \leftarrow 0$, $i \leftarrow 1$
 3: while $i \leq k$ do
 4:     if the rebuffering at $t_{n_i}$ is an I-rebuffering then
 5:         do greedy transmission on $[t_{n_{i-1}} + d(n_{i-1}), t_{n_i} + d(n_i)]$. Let the corresponding transmission schedule be the solution $r^\star_{[t_{n_{i-1}}+d(n_{i-1}),t_{n_i}+d(n_i))}$
 6:     else
 7:         run PCT on $[t_{n_{i-1}} + d(n_{i-1}), \tilde{t}(n_i)]$ with input $\max[l(t_{n_{i-1}}), s_s]$, $u(\tilde{t}(n_i) - d(n_{i-1}))$, $l_{[t_{n_{i-1}},\tilde{t}(n_i)-d(n_{i-1})]}$, $u_{[t_{n_{i-1}},\tilde{t}(n_i)-d(n_{i-1})]}$ and $c_{[t_{n_{i-1}}+d(n_{i-1}),\tilde{t}(n_i)]}$; and do greedy transmission during $[\tilde{t}(n_i), t_{n_i} + d(n_i)]$. Let the corresponding transmission schedule be the solution $r^\star_{[t_{n_{i-1}}+d(n_{i-1}),t_{n_i}+d(n_i))}$
 8:     end if
 9:     $i \leftarrow i + 1$
10: end while
11: if $n_k < n$ then
12:     Run PCT on $[t_{n_k}+d(n_k), t_n+d(n_k)]$ with input $\max[l(t_{n_k}), s_s]$, $s_e$, $l_{[t_{n_k},t_n]}$, $u_{[t_{n_k},t_n]}$ and $c_{[t_{n_k}+d(n_k),t_n+d(n_k)]}$, and let the corresponding transmission schedule be the solution $r^\star_{[t_{n_k}+d(n_k),t_n+d(n_k)]}$
13: end if
Output: $r^\star_{[t_0,t_n+d(n_k)]}$
```

**Theorem 2** *The General Piecewise Constant Thresholding (GPCT) solves the Optimal Streaming Problem (2.1).*

An example of GPCT is shown in Fig 2.3, where an I-rebuffering happens at $t_1$ and lasts for $\tau_1$. A B-rebuffering happens at $t_5 + \tau_1$ and lasts for $\tau_5$. The latest buffer overflow associated with the B-rebuffering happens at $t_4 + \tau_1$, which is denoted as $\tilde{t}_5$. Then GPCT performs greedy transmission during $[t_0, t_1 + \tau_1)$ and $[\tilde{t}_5, t_1 + \tau_1 + \tau_5)$; and it runs PCT on $[t_1 + \tau_1, \tilde{t}_5)$.

## 2.3 Single-User Partially Anticipative Case

### 2.3.1 Fixed Horizon Control Scheme

Now we consider a 'partially anticipative' case in which only a finite window of future wireless capacity variations are known beforehand. Assume that at time $t$, we know the capacity $c_{[t,t+w]}$, where $w$ is the future window size, and we do not know the capacity beyond $t + w$. Thus we cannot use an offline scheme like GPCT. However we can apply the GPCT algorithm on $[t, t + w]$, which provides a baseline for online schemes. We propose a Greedy Fixed Horizon Control (GFHC) transmission scheme in Algorithm 4 below.

GFHC is an online scheme that successively applies GPCT on a sequence of $w$-sized intervals. For each interval it must set the initial buffer state $s_s$, and target buffer state $s'_e$ at the end of the interval. The latter is done in Step 3 of Algorithm 4, where $s'_e \leftarrow s_s + m_i$, which is the maximum one could achieve at the end of the $i$th interval using greedy transmission. The

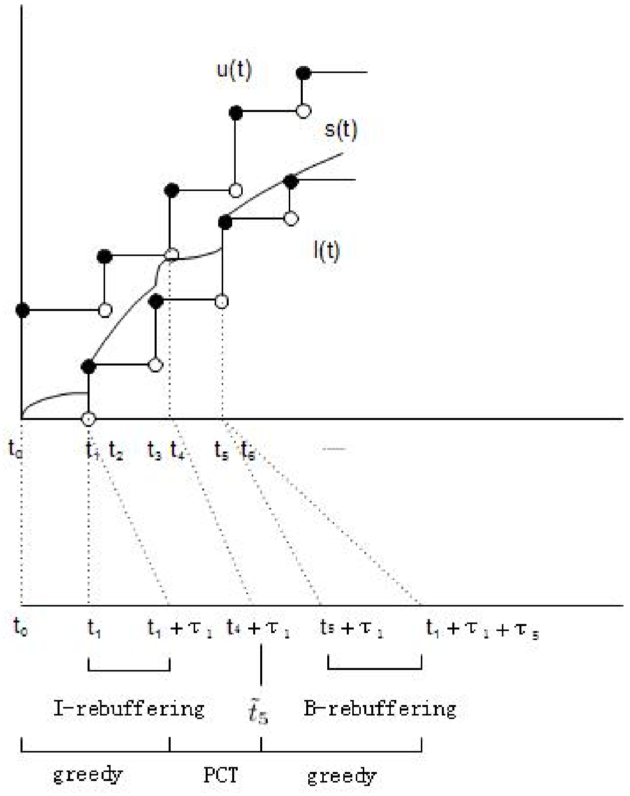


Figure 2.3: An example of GPCT. An I-rebuffering happens at $t_1$ and lasts for $\tau_1$. A B-rebuffering happens at $t_5 + \tau_1$ and lasts for $\tau_5$. The latest buffer overflow associated with the B-rebuffering happens at $t_4+\tau_1$, which is denoted as $\tilde{t}_5$.GPCT performs greedy transmission during $[t_0, t_1+\tau_1]$ and $[\tilde{t}_5, t_1+\tau_1+\tau_5]$; and it runs PCT on $[t_1 + \tau_1, \tilde{t}_5]$.

**Algorithm 4** **Greedy Fixed Horizon Control (GFHC)**

```
Input: $s_s$, $s_e$, $l_{[t_0,t_n]}$, $u_{[t_0,t_n]}$, $c_{[t_0,t_0+\tilde{T}]}$
1: $i \leftarrow 0$, $d \leftarrow 0$
2: repeat
3:     $s'_e \leftarrow s_s + m_i$, where $m_i$ is the maximum amount of data that can be
       delivered (i.e., using greedy transmission) during $[t_0 + iw, t_0 + (i+1)w]$
4:     run GPCT on $[t_0+iw, t_0+(i+1)w]$ with input $s_s$, $s'_e$, $l_{[t_0+iw-d,t_0+(i+1)w-d]}$,
       $u_{[t_0+iw-d,t_0+(i+1)w-d]}$ and $c_{[t_0+iw,t_0+(i+1)w)]}$. Let the resulting transmission
       schedule be $r^\star_{[t_0+iw,t_0+(i+1)w)}$, and the rebuffering time be $\tau$
5:     $d \leftarrow d + \tau$
6:     $s_s \leftarrow s_s + \int_{t_0+iw}^{t_0+(i+1)w} r^\star(t)dt$
7:     $i \leftarrow i + 1$
8: until $s_s = s_e$
Output: $r(\cdot)$
```

initial state $s_s$ is initialized and subsequently updated (Step 6) once the transmission schedule for the current window is determined. GFHC runs GPCT on $w$ intervals, during which it computes a transmission schedule and may incur rebuffering delays. Thus in Step 4, the constraints have been shifted by $d$ the cumulative rebuffering incurred so far. The resulting transmission schedule $r^\star$ is the concatenation of those computed across $w$-windows.

Note in Step 3 shown in Algorithm 4 the target buffer state $s'_e$ is chosen in a "greedy" manner, i.e., as high as possible in order to minimize rebuffering time. This is why we call the proposed scheme "greedy fixed horizon control". In fact, one can in principle define different kinds of Fixed Horizon Control (FHC) schemes by using different strategies in Step 3 of Algorithm 4 to choose $s'_e$. For example, we can define a "resource saving FHC" by setting $s'_e = \max[s_s, l((i+1)w)]$.

### 2.3.2 Competitive Optimality of GFHC

To evaluate the performance of GFHC, we use the "competitive ratio" which is defined as the ratio of the cost of GFHC and the optimal offline cost achieved by GPCT under the same problem settings (i.e., the same capacity variations, $l(\cdot)$, $u(\cdot)$, etc., but they are known ahead of time). The following theorem gives an upper bound on the competitive ratio.

**Theorem 3** *Given a maximum playback buffer size $b$, video length $T$ and size $S$, window size $w$, maximum capacity $c_{max}$ and minimum capacity $c_{min} > 0$, the competitive ratio of GFHC to GPCT satisfies:*

$$\frac{cost^G}{cost^O} \leq 1 + \max\left[\frac{1}{c_{min}}, \frac{T}{S}\right] \frac{b}{w}(\frac{c_{max}}{c_{min}} - 1).$$

Note that when $\frac{S}{T} > c_{min}$: i.e., the overall average video compression rate exceeds the minimum capacity the upper bound in Theorem 3 can be viewed as having two parts.. The first, $\frac{b}{wc_{min}}$, captures the size of the playback buffer relative to the minimum amount of data that will be delivered in future window. If these are close clearly the offline opportunistic scheduling realized by GPCT will not achieve great gains over GFHC. The second, $\frac{c_{max}}{c_{min}} - 1$ captures the worst case variability in capacity. In general, the worst case performance of GFHC is closer to GPCT under a larger minimum capacity, a larger prediction window size, a smaller playback buffer size and a smaller ratio between the maximum and the lowest capacities. The proof of Theorem 3 is as follows.

*Proof:* We denote by $cost^O = cost_u^O + cost_r^O$ and $cost^G = cost_u^G + cost_r^G$ the optimal costs achieved by GPCT and GFHC – the two terms correspond to the system utilization and rebuffering time. We also let $r^O()$ and $r^G()$ denote the optimal transmission schedules for GPCT of GFHC respectively. Suppose $t_0 = 0$ and let $s^O(t) = \int_0^t r^O(\tau)d\tau$ and $s^G(t) = \int_0^t r^G(\tau)d\tau$ be the the cumulative transmitted data for the two schemes. The greedy nature of GFHC ensures that $s^G(t) \geq s^O(t), \forall t \geq 0$, i.e., GFHC is always ahead of GPCT, and so it has a rebuffering cost no higher than GPCT. Since GPCT strictly prioritizes minimization of rebuffering cost over system utilization, it follows that GPCT achieves the lowest possible rebuffering cost. We can conclude that $cost_r^G = cost_r^O$. Next we develop an upper bound for the difference in the utilization costs, i.e., $cost_u^G - cost_u^O$.

Suppose GFHC uses $k$ window intervals, each with length $w$. On the $i$th interval, $1 \leq i \leq k$, the amount of data delivered by GFHC is denoted as $\delta_i^G = s^G(iw) - s^G((i-1)w)$. Also denote $\delta_i^O = s^O(iw) - s^O((i-1)w)$ as the amount of data delivered by the optimal scheme during the $i$th interval. Then we define two index sets as follows:

$$I_1 = \{i : \delta_i^G \leq \delta_i^O\};$$

$$I_2 = \{i : \delta_i^G > \delta_i^O\}.$$

Note that $\forall i \in I_1$, GFHC delivers less data than the optimal scheme on interval $i$. We can define an intermediate transmission scheme on this interval, which starts at initial state $s^G((i-1)w)$, transmits at following rate $r^I(t)$:

$$r^I(t) = r^O(t)\mathbf{1}_{\{t\in[(i-1)w,\tilde{t}]\}},$$

where $\tilde{t} \le iw$ is chosen such that,

$$\int_{(i-1)w}^{\tilde{t}} r^I(t)dt = s^G(iw) - s^G((i-1)w).$$

It is easy to check that such a scheme is well-defined, and we can claim that the utilization cost of the intermediate scheme on interval $i$ is no more than that of GPCT, since it transmits at the same rate as GPCT on $[(i-1)w, \tilde{t}]$, and transmits nothing on $[\tilde{t}, iw]$. Note on interval $i$, GPCT transmits an amount of $\delta_i^O - \delta_i^G$ more than the intermediate scheme. Transmission of the extra data requires an extra utilization no less than $\frac{\delta_i^O-\delta_i^G}{\tilde{T}c_{max}}$, which is obtained by assuming that the data is transmitted at maximum capacity thus achieving the best savings. Thus we have, $cost_{u,i}^I \le cost_{u,i}^O - \frac{\delta_i^O-\delta_i^G}{\tilde{T}c_{max}}$. Also since $s^G(t) \ge s^O(t), \forall t \ge 0$, we have that $s^I(t) \ge s^O(t)$ on interval $i$, which in turn implies $cost_{r,i}^I \le cost_{r,i}^O$ (in fact they are equal). Thus we have $cost_i^I \le cost_i^O - \frac{\delta_i^O-\delta_i^G}{\tilde{T}c_{max}}$.

However note that, the intermediate scheme and GFHC start at the same initial state and end at the same terminal state on interval $i$. Also note that GFHC uses GPCT on interval $i$ and thus is optimal under fixed initial state and terminal state on that interval. As a result, we have $cost_i^G \le cost_i^I$. In conclusion we have

$$cost_i^G \le cost_i^O - \frac{\delta_i^O - \delta_i^G}{\tilde{T}c_{max}}.$$

By similar arguments we have, $\forall i \in I_2$,

$$cost_i^G \leq cost_i^O + \frac{\delta_i^G - \delta_i^O}{\tilde{T} c_{min}},$$

where $c_{min}$ comes from assuming the extra data is transmitted at the lowest capacity giving an upper bound on the additional utilization.

Summing up all the intervals we get:

$$cost^G \leq cost^O - \sum_{i \in I_1} \frac{\delta_i^O - \delta_i^G}{\tilde{T} c_{max}} + \sum_{i \in I_2} \frac{\delta_i^G - \delta_i^O}{\tilde{T} c_{min}} \tag{2.3}$$

Note since GPCT and GFHC transmit the same amount of data at last, there must be:

$$\sum_{i \in I_1} (\delta_i^O - \delta_i^G) = \sum_{i \in I_2} (\delta_i^G - \delta_i^O) \overset{\text{def}}{=} A.$$

Also note that on each interval $i$, the difference between $\delta_i^O$ and $\delta_i^G$ cannot exceed the buffer size, i.e., $|\delta_i^O - \delta_i^G| \leq b$, and since there are at most $\frac{\tilde{T}}{w}$ intervals, we have that

$$A \leq \frac{\tilde{T} b}{w}. \tag{2.4}$$

Then we can rewrite inequality (2.3) as

$$cost^G \leq cost^O + A\left(-\frac{1}{\tilde{T} c_{max}} + \frac{1}{\tilde{T} c_{min}}\right). \tag{2.5}$$

Combining inequalities (2.4) and (2.5), we have

$$\begin{aligned} cost^G &\leq cost^O - \frac{\tilde{T} b / w}{\tilde{T} c_{max}} + \frac{\tilde{T} b / w}{\tilde{T} c_{min}} \\ &\leq cost^O + \frac{b(c_{max} - c_{min})}{w c_{max} c_{min}}. \end{aligned}$$

Finally, dividing by $cost^O$ and noticing that $cost^O \geq cost_u^O \geq \frac{1}{\tilde{T}}\frac{S}{c_{max}}$, we obtain that

$$\frac{cost^G}{cost^O} \leq 1 + \frac{b(c_{max} - c_{min})\tilde{T}}{wc_{min}S},$$
$$\leq 1 + \max[\frac{1}{c_{min}}, \frac{T}{S}]\frac{b}{w}(\frac{c_{max}}{c_{min}} - 1),$$

where we used the fact that $\tilde{T} = \max[\frac{S}{c_{min}}, T]$.

■

## 2.4 Multiuser Anticipative Case

In Section II we proposed GPCT and proved that it solves the Optimal Streaming problem (2.1). However, the algorithm was developed for a single-user case and it is hard to generalize it to the multiuser case. In this section, we develop sub-optimal multiuser schemes based on GPCT which have reasonable complexity.

### 2.4.1 Multiuser Piecewise Constant Thresholding Under Proportional Capacity Allocation (MTP)

Suppose a base station is serving $n$ mobile users and user $i$ has a peak capacity $c_i(t)$ at time $t$, $i = 1, 2, ..., n$. The peak capacities are assumed to be known beforehand. A simple way to deal with the multiuser issue is to make an up-front allocation of resources among the $n$ users in a round robin fashion and thus the allocated capacity for user $i$ is $\tilde{c}_i(t) = \frac{c_i(t)}{n}$, which is proportional to his/her peak capacity $c_i(t)$. Each user $i$ is then assumed to have a capacity $\tilde{c}_i(t)$ which is independent of other users' capacities and thus we can apply GPCT to each user separately. We call this scheme multiuser piecewise constant thresholding under proportional capacity allocation (MTP). MTP is straightforward to implement since every user can independently run a single-user algorithm on his/her own based on knowledge of his/her capacity and the number of users sharing the bottleneck, and requests transmission from server accordingly. Thus the scheme works in a decentralized way and there is no need for a centralized controller. However, although it applies GPCT which exploits temporal diversity in capacity variations well, MTP is based on a proportional capacity allocation which does not directly exploit multiuser diversity. Below we consider how both might be exploited.

### 2.4.2 Multiuser Piecewise Constant Thresholding Under Opportunistic Capacity Allocation (MTO)

We introduce a centralized scheme which exploits both temporal and multiuser diversity. Consider a base station serving $n$ mobile users. To reduce the system utilization, there is a central scheduler at the base station which knows future capacity variations of the mobiles. The system operates in discrete (slotted) time and each time slot the scheduler chooses which of the video users could be served in the slot. In order to exploit capacity variations across different users while ensuring 'short-term' fairness, we use the opportunistic resource allocation scheme with a token counter mechanism proposed in [32]. It works as follows.

Each user $i$ is associated a token counter $T_i$. At the beginning, all the token counters are set to be the same positive integer value $m$, which is referred to as the token limit. Each time slot, the scheduler searches among the users who have a non-zero token counter and chooses the user with the highest capacity[3]. $T_i$ is decremented if user $i$ is served in that time slot. When all the token counters are zero, they are reset to $m$. Using the same token limit $m$ for all the users guarantees that the system allocates the same number $(m)$ of time slots to each user within $m \cdot n$ slots. Subsequently each user has his/her allocated capacity $\tilde{c}_i(t)$ and thus GPCT can again be applied independently to each user. The resulting scheme is denoted the multiuser piecewise constant thresholding under opportunistic capacity allocation (MTO). MTO is of higher complexity than MTP since it is based on a centralized controller. However it can achieve a lower system utilization because it not only exploits the temporal diversity for each user but also exploits the multiuser diversity across all the users via the token counter mechanism. The choice of the token limit $m$ affects the performance of MTO in that a higher token limit allows the system to exploit the multiuser diversity more aggressively and results in a higher decrease in the system utilization, however, a smaller token limit enforces more temporal fairness and results in a shorter rebuffering time. We will see this point later in Section V.

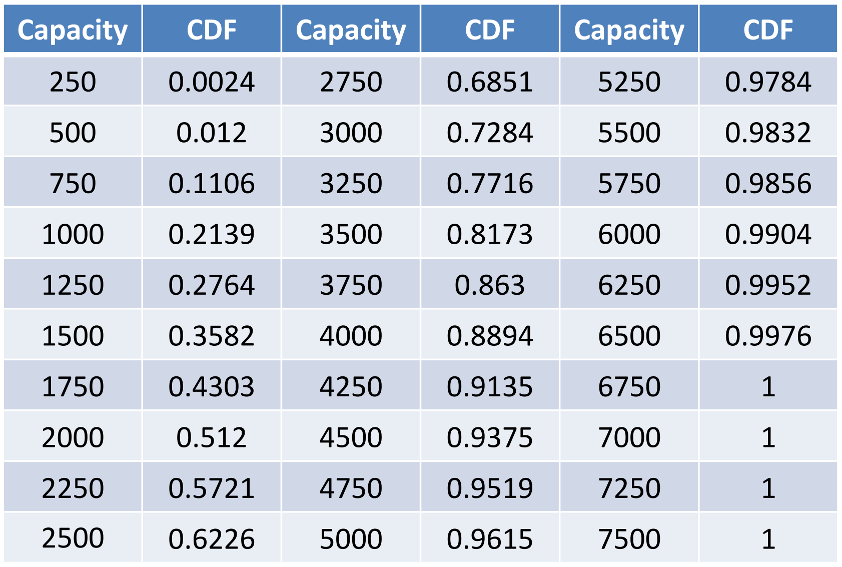
[3]Selection could be weighted or driven by quantiles to address fairness concerns.

## 2.5 Simulation Results

**Performance sensitivity of GFHC.** We ran a simulation to explore the performance sensitivity of GFHC to window, playback buffer and temporal correlation in capacity variations. We considered a server delivering a 10-minute constant bit rate video, 900kbps, to a receiver. We simulated a slotted system, with slots of length 0.1sec. Thus the jump points for $l(\cdot)$ are at 0.1sec, 0.2sec, $0.3sec$,.. with values $l(0.1m) = 9m$, for $m = 1, 2, ..., 6000$. The initial buffer $b_0$ was set to 0.

Capacity variations, were modeled via a discrete time Markov chain whose states represent capacities, specifically $0, 250, 500, \ldots, 7500$kbps. The transition probability matrix for the chain is selected so that the invariant is a pre-defined stationary distribution corresponding to the PDF for the throughput of a randomly positioned user in an realistic HSPA system single antenna equalizer on the receiver under medium system load. The distribution is shown in Fig. 2.4. We consider two such matrices which differ in the speed at which capacity varies. Specifically in the first case transitions can only happen between the two nearest states (e.g., from 250kbps to 500kbps) which results in slow variations (i.e., with temporal correlation); in the second case we simply take iid samples of the throughput PDF (i.e., with no temporal correlation).

Finally we let the receiver's playback buffer size $b$ vary from $1620, 1890, 2160, 2430$ to 2700kb, and the window size $w$ from $10, 20, 50, 100, 300$ to 600sec. Each scenario was simulated 20 times to obtain average results.



| Capacity | CDF | Capacity | CDF | Capacity | CDF |
|---|---|---|---|---|---|
| 250 | 0.0024 | 2750 | 0.6851 | 5250 | 0.9784 |
| 500 | 0.012 | 3000 | 0.7284 | 5500 | 0.9832 |
| 750 | 0.1106 | 3250 | 0.7716 | 5750 | 0.9856 |
| 1000 | 0.2139 | 3500 | 0.8173 | 6000 | 0.9904 |
| 1250 | 0.2764 | 3750 | 0.863 | 6250 | 0.9952 |
| 1500 | 0.3582 | 4000 | 0.8894 | 6500 | 0.9976 |
| 1750 | 0.4303 | 4250 | 0.9135 | 6750 | 1 |
| 2000 | 0.512 | 4500 | 0.9375 | 7000 | 1 |
| 2250 | 0.5721 | 4750 | 0.9519 | 7250 | 1 |
| 2500 | 0.6226 | 5000 | 0.9615 | 7500 | 1 |

Figure 2.4: The stationary distribution (cdf) of capacity (kbps) used in our simulation.

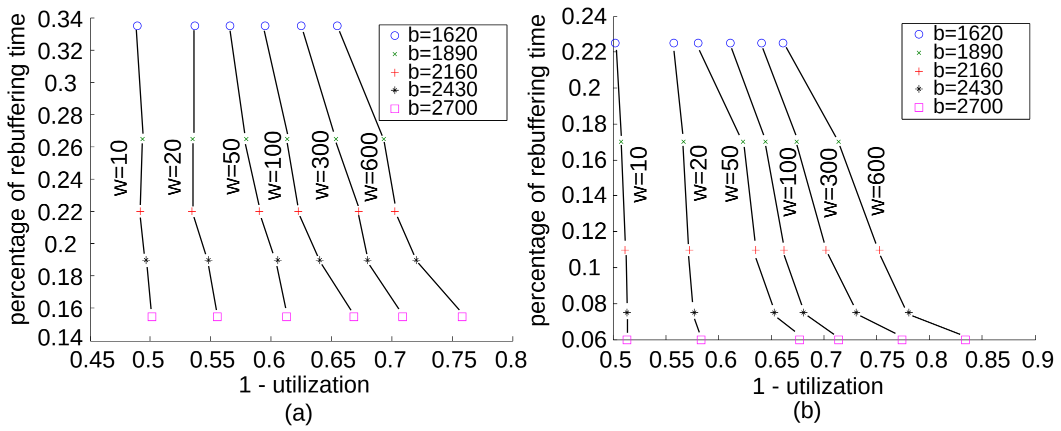


Figure 2.5: (a) The performance of GFHC under correlated capacity variation. (b) The performance of GFHC under iid capacity variation. Each point shows the percent rebuffering time versus (1 − utilization) for a specific playback buffer size and window size. The points on the bottom right of the figures correspond to best performance.

The results shown in Figs. 2.5 (a) and (b) correspond to the scenarios with correlated and iid capacity variations respectively. The figures show the percent rebuffering time versus ($1-$utilization) for varying playback buffer and window sizes. Note the points on the bottom right of the figures correspond to best performance, i.e., lowest system utilization and rebuffering time. The figures exhibit the following three observations.

**1.** For fixed $b$ and capacity variation, increasing $w$ significantly reduces utilization but does not affect the rebuffering time.

**2.** For fixed $w$ and capacity variation, increasing $b$ reduces rebuffering time and results in a marginal decrease in utilization. Note Theorem 3 suggests that a smaller $b$ should result in a better performance, but this corresponded to worst case 'performance' vs the averages considered here.

**3.** For fixed $b$ and $w$, temporal correlation in capacity variation results in increased rebuffering and a higher utilization.

We also evaluated GFHC vs greedy transmission for a wireless capacity trace measured from a vehicle driving through Mountain View, CA. We considered a server delivering a 4min, 900kbps constant bit rate video to a receiver. The capacity trace was rescaled to have an average rate of 2000kbps. The playback buffer size was set to be 1/10 of the video size. The greedy transmission scheme resulted in a 67.69% utilization and 8.1sec of rebuffering time. The GFHC resulted in the same rebuffering time, but the utilization was reduced to 29.00%,48.83%,55.17% and 59.44% when the window $w$ was set to

240, 120, 60, and 30sec respectively. This confirms the benefit of exploiting anticipated capacity variations for a trace from a real wireless network.

**Performance and comparisons for multiuser algorithms.** We ran simulations to compare the performance of MTP and MTO versus that of a proportional rate allocation scheme in which greedy transmission scheme is used and the time slots are assigned to the mobile users in a round robin fashion so that the transmission rate to each user is proportional to his/her peak capacities. In our simulation, we assume a fixed number of users are being served and each user is watching a 10-minute video with a constant bit rate of 90kbps. All the users have the same playback buffer size which was set to be 18000kb (i.e. one third of the video size), and they start watching the videos simultaneously. The model for capacity variations is the slotted model with correlated variations discussed above.

We let the number of users range from 1 to 12 and repeat each one 20 times to obtain average results. In MTO, we test the performance under four token limits which are 1,3,6 and 9. The average system utilization and average percent rebuffering time were computed and are plotted in Figs. 2.6 (a) and (b).

Fig. 2.6 (a) exhibits the utilization as a function of the number of users. As can be seen, proportional rate allocation achieves the highest system utilization; by comparison, MTP and MTO achieve a $60 - 70\%$ reduction. Alternatively, for the same system utilization, MTP and MTO might allow 2x-3x more users. As expected MTO achieves a lower system utilization than

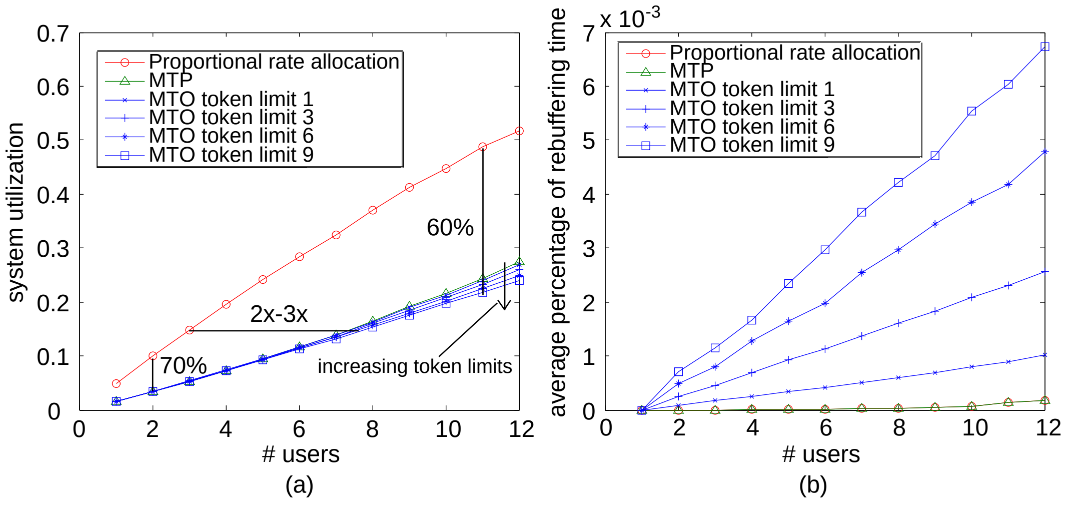


Figure 2.6: (a) The system utilization. The proportional rate allocation has the highest system utilization. MTP and MTO achieve reduced system utilization. MTO schemes do a little better than MTP. An MTO scheme with a higher token limit achieves a lower system utilization. (b) The average percent rebuffering time. The proportional rate allocation and MTP have the same percent rebuffering time. MTO schemes result in higher rebuffering time which increases as the token limit increases.

MTP, since it exploits multiuser diversity, and MTO with a higher token limit results in a lower utilizations. However, as shown in Fig. 2.6 (b) this benefit is obtained at the cost of a additional rebuffering. Fig. 2.6 (b) exhibits the percent rebuffering time versus the number of users. It shows that MTP and proportional rate allocation require the same rebuffering time, but MTO results in more rebuffering as does MTO with higher token limits.

## 2.6 Uncertainty in Knowledge of Future Capacity Variations

So far we have developed video delivery policies assuming knowledge of future capacity variations (either in the anticipative case or the partially anticipative case) is perfect. However in practice there will be uncertainty in predictions of future capacity. Such uncertainty may arise from many sources, e.g., the mapping of wireless signal strength on the coverage map to capacity, uncertainty in a user's motion, interference from other mobile users, uncertainty in the number of mobile users contending for resources, etc.

Such uncertainty limits the optimality of our proposed approaches (GPCT and GFHC). For example, GPCT opportunistically chooses when to request video delivery based on future capacity variations. If the true capacity variations are lower than the predicted ones, then the users may experience longer rebuffering times. Similarly one can show that the minimum system utilization will not be achieved under imperfect capacity predictions.

In this section we explore different approaches to address such uncertainty. We first develop an offline approach which is able to achieve a minimum rebuffering time with an upper-bound on system utilization, under the assumption that the uncertainty in capacity variations can be bounded. We then propose an online algorithm that can deal with general types of uncertainty.

### 2.6.1 Offline Approach under Uncertainty in Capacity Prediction

Let $\tilde{c}(t)$ denote the predicted capacity at time $t$, and $c(t)$ the true capacity which is unknown beforehand. If the prediction is good, such that $\tilde{c}(\cdot)$ is uniformly close to $c(\cdot)$, then it seems reasonable to use GPCT on the predicted capacity function to obtain an offline video delivery strategy. We shall denote such a policy by $\tilde{\mathcal{S}}^\star$. This policy can then be used under the true capacity function to mediate video delivery. Note that strategy $\tilde{\mathcal{S}}^\star$ specifies when the user should request video delivery under the predicted capacity function $\tilde{c}(\cdot)$, which is a set of time intervals, e.g., $\tilde{\mathcal{S}}^\star = \{[t_1, t_2]\}$ indicating that the user should request video delivery during $[t_1, t_2]$ under capacity $\tilde{c}(\cdot)$. However, when the user applies $\tilde{\mathcal{S}}^\star$ under the true capacity function $c(\cdot)$, some changes have to be made. For example, if $\tilde{c}(t) \le c(t)$, $\forall t$, the user have more video delivered than expected. To avoid buffer overflow violations, the user needs to adjust policy $\tilde{\mathcal{S}}^\star$ to stop requesting video delivery whenever the playback buffer is full. We denote this modification to $\tilde{\mathcal{S}}^\star$ by $\tilde{\mathcal{S}}^\star_m$.

We can compare the perfromance of $\tilde{\mathcal{S}}^\star_m$ obtained based on the predicted capacity function to the optimal policy (GPCT) $\mathcal{S}^\star$ obtained assuming the true capacity function were known. Since we give higher priority to avoiding rebuffering, it is reasonable to require that $\tilde{\mathcal{S}}_m^\star$ result in no more rebuffering time than $\mathcal{S}^\star$. The following lemma shows that such requirement is met if $\tilde{c}(t) \leq c(t),\ \forall t$.

**Lemma 1** *Suppose $\mathcal{S}^\star$ and $\tilde{\mathcal{S}}_m^\star$ are as defined above, then applying $\tilde{\mathcal{S}}_m^\star$ on the true capacity function $c(\cdot)$ results in no more rebuffering time than $\mathcal{S}^\star$, if for all $t$*

$$0 < \tilde{c}(t) \leq c(t). \tag{2.6}$$

*Proof:* We prove the lemma by contradiction. Suppose the claim is not true, i.e., $\tilde{\mathcal{S}}_m^\star$ results in more rebuffering time than $\mathcal{S}^\star$. This implies that there exists a period $[t, t+\tau] \nsubseteq \tilde{\mathcal{S}}_m^\star$, such that the modified strategy $\tilde{\mathcal{S}}_m^\star \cup [t, t+\tau]$ reduces the rebuffering time of $\tilde{\mathcal{S}}_m^\star$. Note that since $\tilde{\mathcal{S}}_m^\star$ differs from $\tilde{\mathcal{S}^\star}$ only at times when the buffer is full, we can claim that $[t, t+\tau] \notin \tilde{\mathcal{S}^\star}$. Also by Condition (2.6) applying $\tilde{\mathcal{S}^\star}$ to $\tilde{c}(\cdot)$ delivers no more than $\tilde{\mathcal{S}}_m^\star$ on $c(\cdot)$ cumulatively at any time. Thus strategy $\tilde{\mathcal{S}^\star} \cup [t, t+\tau]$ results in less rebuffering time than $\tilde{\mathcal{S}^\star}$ under the capacity function $\tilde{c}(\cdot)$, which contradicts the fact that $\tilde{\mathcal{S}^\star}$ was obtained using GPCT and thus results in the minimum rebuffering time. ■

However in practice, Condition (2.6), i.e., that predictions are always pessimistic, may not hold, and the appropriate offline approaches should depend on the uncertainty in capacity prediction, i.e., the difference between $\tilde{c}(\cdot)$

and $c(\cdot)$. To pursue this further let us assume that the prediction error can be bounded.

**Definition 3** *We say the capacity function prediction satisfies an* $(\alpha, \beta)$ *-* prediction error *if for all* $t$

$$c(t) \in [(1-\alpha)\tilde{c}(t), (1+\beta)\tilde{c}(t)],\ \ \alpha, \beta \in [0, 1). \tag{2.7}$$

Note that under the $(\alpha, \beta)$ - prediction error, the true capacity function is bounded within a certain range of the predicted one. Let $\hat{c}(t) = (1-\alpha)\tilde{c}(t)$, which is thus a lower bound on the true capacity. Let $\hat{\mathcal{S}}^\star$ and $\hat{\mathcal{S}}^\star_m$ denote the strategy obtained by applying GPCT on $\hat{c}(\cdot)$ and the corresponding modified strategy obtained by applying $\hat{\mathcal{S}}^\star$ on $c(\cdot)$ avoiding buffer overflow, respectively. Now according to Lemma 1, $\hat{\mathcal{S}}^\star_m$ should result in no more rebuffering time than $\mathcal{S}^\star$.

To evaluate the performance of $\hat{\mathcal{S}}^\star_m$, we need to compare the system utilization (denoted by $\hat{u}^\star_m$) to the optimal system utilization (denoted by $u^\star$) obtained by $\mathcal{S}^\star$. The following lemma shows that $\hat{u}^\star_m$ can indeed be upper bounded.

**Lemma 2** *Suppose* $\hat{\mathcal{S}}^\star_m$*,* $\mathcal{S}^\star$*,* $\hat{u}^\star_m$ *and* $u^\star$ *are defined as above,* $\tilde{T}$*,* $c_{min}$*,* $S$ *are as defined in Section 2.2, and the true capacity satisfies Condition (2.7), then*

$$u^\star \leq \hat{u}^\star_m \leq u^\star + \frac{\alpha+\beta}{1+\beta}\frac{S}{c_{min}\tilde{T}}. \tag{2.8}$$

*Proof:* The first inequality in (2.8) holds due to the fact that $\mathcal{S}^\star$ is the strategy obtained by applying GPCT to the true capacity function $c(\cdot)$, which according to Theorem 2 should lead to the minimum system utilization among all strategies that result in the minimum rebuffering time for a given capacity function.

We prove the second inequality in (2.8) as follows. Due to the fact that $\hat{\mathcal{S}}_m^\star$ is the modified version of $\hat{\mathcal{S}}^\star$ so as to avoid buffer overflow, and that $\hat{c}(t) \leq c(t),\ \forall t$, we can claim that the system utilization resulting from applying $\hat{\mathcal{S}}_m^\star$ to $\hat{c}(\cdot)$ (denoted as $\hat{u}^\star$) is no less than $\hat{u}_m^\star$, which gives:

$$\hat{u}_m^\star \leq \hat{u}^\star. \tag{2.9}$$

Paralleling the way we obtained the strategy $\hat{\mathcal{S}}_m^\star$, we will construct another strategy $\mathcal{S}_m^\star$ by applying $\mathcal{S}^\star$ to the capacity function $\hat{c}(\cdot)$. However since $\hat{c}(t) \leq c(t),\ \forall t$, $\mathcal{S}^\star$ may end up delivering only a part of the video under $\hat{c}(\cdot)$. We deliver the remaining part of the video (denote its size as $S_r$ bits) using time intervals not included in $\mathcal{S}^\star$ and in a greedy manner (i.e., deliver as early as possible) to keep the rebuffering time as low as possible. We denote this strategy by $\mathcal{S}_m^\star$ and the corresponding system utilization is denoted by $u_m^\star$. Note that according to Theorem 2, it follows that

$$\hat{u}^\star \leq u_m^\star. \tag{2.10}$$

Next we calculate the gap between $u_m^\star$ and $u^\star$. Note that the file size of the video $S$ can be written as:

$$S = \int_{t \in \mathcal{S}^\star} c(t)dt. \tag{2.11}$$

Similarly, $S_r$ can be written as:

$$S_r = S - \int_{t \in \mathcal{S}^\star} \hat{c}(t)dt. \tag{2.12}$$

According to the definition of $\hat{c}(\cdot)$ and Equation (2.7), we have:

$$\frac{1-\alpha}{1+\beta} c(t) \leq \hat{c}(t), \ \forall t. \tag{2.13}$$

Combining (2.11), (2.12) and (2.13), we obtain:

$$S_r \leq \frac{\alpha+\beta}{1+\beta} S. \tag{2.14}$$

Thus the extra time spent delivering $S_r$ is no more than $\frac{\alpha+\beta}{1+\beta}\frac{S}{c_{min}}$. Normalizing the time by $\tilde{T}$ we obtain a bound on the gap of the realized system utilization:

$$u_m^\star - u^\star \leq \frac{\alpha+\beta}{1+\beta}\frac{S}{c_{min}\tilde{T}}. \tag{2.15}$$

Combining (2.9), (2.10) and (2.15), we obtain:

$$\hat{u}_m^\star - u^\star \leq \frac{\alpha+\beta}{1+\beta}\frac{S}{c_{min}\tilde{T}},$$

which proves the second inequality in (2.8). ■

Lemma 2 shows that under errors in capacity prediction if the prediction is good enough (i.e. $\alpha$ and $\beta$ are close to zero), then the offline strategy $\hat{\mathcal{S}}_m^\star$ performs nearly as well as the optimal offline strategy $\mathcal{S}^\star$.

So far we have proposed an offline approach under bounded prediction error. It first uses GPCT on a lower bound of the capacity variation (i.e., $\hat{c}(\cdot)$) and then modifies the obtained strategy to be feasible, i.e., avoiding buffer overflow when it is applied to the true capacity function. We have also showed that the proposed approach results in no more rebuffering time than GPCT, and the resulting system utilization can be bounded if the uncertainty in the capacity prediction is bounded. These results are summarized in the following theorem.

**Theorem 4** *Suppose $\hat{\mathcal{S}}_m^\star$ is the offline video delivery strategy proposed earlier in this section, and $\mathcal{S}^\star$ is the optimal strategy obtained by GPCT. Then under $(\alpha, \beta)$ - capacity prediction error, Eq(2.7), $\hat{\mathcal{S}}_m^\star$ results in no more rebuffering time than $\mathcal{S}^\star$. Also $\hat{\mathcal{S}}_m^\star$ results in a higher system utilization than $\mathcal{S}^\star$, which can be upper bounded as shown in (2.8).*

### 2.6.2 Online Approach under Uncertainty in Capacity Prediction

In the previous subsection we proposed an offline approach, the performance of which, can only be guaranteed when uncertainty is small. If there is substantial uncertainty in the prediction of the future capacity variations, online approaches are preferable since they are able to adjust their strategies based on what they have experienced. In this subsection, we propose an online approach in which the users adaptively decide their thresholds based on their playback buffer status and the average value of their predicted capacity for a finite window into the future. The basic idea of our online approach is that when a user's playback buffer is low, in order to avoid rebuffering, the user should use a small threshold to request video content as soon as possible; when a user's playback buffer is high, in order to reduce the system utilization, the user should use a relatively large threshold based on the prediction of the average future capacity to exploit the temporal opportunism in capacity variations.

To simplify the problem, we assume the requested video has a constant bit rate $r$, and the size of the video is $S$. Suppose we can predict the average capacity for a window into the future of $w_{max}$ seconds. Based on the capacity predictions and the playback buffer status we make sequential decisions on the capacity threshold $\gamma$ (i.e., video will be requested only when the true capacity is above $\gamma$ and the playback buffer is not full). The thresholds are determined by a Buffer-Dependent Thresholding (BDT) strategy displayed in Algorithm 5.

In Algorithm 5, time is divided into intervals and the capacity threshold $\gamma$ is determined sequentially on each interval. At the beginning of the $i$th interval, suppose there is $b_i$ seconds of unwatched video content in the playback buffer, then the length of the $i$th interval is $w_i = \min[w_{max}, b_i]$. Suppose the average predicted capacity in the $i$th interval is $\tilde{c}_i$. We set two thresholds $\overline{b}$ and $\underline{b}$ on the playback buffer with $\overline{b} > \underline{b}$, indicating the fullness of buffer. We consider the buffer is high if $b_i > \overline{b}$ and low if $b_i < \underline{b}$. The capacity threshold of the $i$th interval $\gamma_i$ is then determined based on the buffer status.

First, if the buffer is neither high nor low, i.e., $\underline{b} \leq b_i \leq \overline{b}$, then the

**Algorithm 5 Buffer-Dependent Thresholding (BDT)**

```
Input: $r$, $S$, $w_{max}$, $\bar{b}$, $\underline{b}$
1: $i \leftarrow 0$, $s = 0$
2: $t_i^s \leftarrow 0$
3: repeat
4:     $b_i \leftarrow$ length (sec) of unwatched video in buffer at $t_i^s$
5:     $w_i \leftarrow \min[w_{max}, b_i]$
6:     $t_i^e \leftarrow t_i^s + w_i$
7:     $\tilde{c}_i \leftarrow$ predicted mean capacity in $[t_i^s, t_i^e]$
8:     if $\underline{b} \leq b_i \leq \bar{b}$ then
9:         $\gamma_i \leftarrow \max[\tilde{c}_i - r, 0]$
10:    else if $b_i > \bar{b}$ then
11:        $\gamma_i \leftarrow \max[\tilde{c}_i - r + \frac{r(b_i - \bar{b})}{w_i}, 0]$
12:    else
13:        $\gamma_i \leftarrow 0$
14:    end if
15:    $\gamma(t)|_{t \in [t_i^s, t_i^e]} = \gamma_i$
16:    During $[t_i^s, t_i^e]$, request video only when the true capacity is above $\gamma_i$ and
       the playback buffer is not full.
17:    $s_i \leftarrow$ the amount of video delivered during $[t_i^s, t_i^e]$
18:    $s \leftarrow s + s_i$
19:    $i \leftarrow i + 1$
20:    $t_i^s \leftarrow t_{i-1}^e$
21: until $s = S$
Output: $\gamma(\cdot)$
```

threshold is set to

$$\gamma_i = \max[\tilde{c}_i - r, 0]. \tag{2.16}$$

We explain the rationale of this capacity threshold as follows.

Note that in the thresholding policy (2.16) if $\tilde{c}_i \leq r$, then $\gamma_i = 0$, which leads to the greedy strategy and results in the minimum rebuffering time. Otherwise if $\tilde{c}_i > r$, the following lemma indicates that if the prediction of the average future capacity is accurate, then the thresholding policy (2.16) results in no rebuffering.

**Lemma 3** *In the thresholding policy (2.16), suppose the prediction of the average capacity is accurate, i.e.,*

$$\tilde{c}_i = \frac{1}{t_i^e - t_i^s} \int_{t_i^s}^{t_i^e} c(t)dt, \tag{2.17}$$

*where $t_i^s$ and $t_i^e$ are the start and end time of the ith interval. If $\tilde{c}_i > r$, then the following holds:*

$$\frac{1}{t_i^e - t_i^s} \int_{t \in I_i^+} c(t)dt \geq r, \tag{2.18}$$

*where $I_i^+ = \{t | c(t) \geq \gamma_i, t \in [t_i^s, t_i^e]\}$, which is the set of times where video delivery will be requested.*

*Proof:* Let $I_i^- = [t_i^s, t_i^e] \backslash I_i^+$, which is the set of times where no video will be requested. And let $|I_i^-|$ be the total length of time in the set $I_i^-$. According to (2.17), it follows that:

$$\tilde{c}_i = \frac{1}{t_i^e - t_i^s} \int_{t \in I_i^+} c(t)dt + \frac{1}{t_i^e - t_i^s} \int_{t \in I_i^-} c(t)dt. \tag{2.19}$$

Due to the definition of $I_i^-$, we have:

$$\frac{1}{|I_i^-|}\int_{t\in I_i^-} c(t)dt < \gamma_i = \tilde{c}_i - r. \tag{2.20}$$

Since $|I_i^-| \le t_i^e - t_i^s$, the following can be obtained from (2.20):

$$\frac{1}{t_i^e - t_i^s}\int_{t\in I_i^-} c(t)dt < \gamma_i = \tilde{c}_i - r. \tag{2.21}$$

Then (2.18) can be proved by combining (2.19) and (2.21). ■

Lemma 3 shows that the playback buffer will keep growing when $\tilde{c}_i > r$. If the buffer grows too high, i.e., $b_i > \bar{b}$ it is reasonable to set the threshold $\gamma$ in a more aggressive way than (2.16) so as to further reduce system utilization. So when $b_i > \bar{b}$, we will reduce the amount of requested video so that the playback buffer drops to $\bar{b}$ at the end of the $i$th interval. Thus the total amount of video delivered is $rw_i - r(b_i - \bar{b})$. By analogy with the thresholding policy (2.16), we set the threshold to be:

$$\gamma_i = \max\{\tilde{c}_i - \frac{rw_i - r(b_i - \bar{b})}{w_i}, 0\},$$

which motivates thresholding policy when buffer is high in BDT (Line 11).

Note that Lemma 3 holds under the assumption that the prediction of the average capacity is accurate. However if the prediction is higher than the true capacity, the thresholding policy (2.16) may lead to extra rebuffering time. To mitigate this problem, we use the greedy strategy, i.e., set $\gamma_i = 0$ when $b_i < \underline{b}$, which results in policy when the buffer is low in BDT (Line 13)

Note that BDT does not rely on precise capacity predictions. It only requires reasonable precision in the prediction of the average capacity variations for a finite window into the future.

To test the performance of BDT, we ran simulations where we use the same settings as in Section 2.5. For BDT, we took $w_{max} = 10$s, $\underline{b} = 2$s, $\bar{b} = 12$s, $r = 900$kbps, and the buffer size is 21600kbits, which corresponds to 24 seconds of video. We generate prediction error for the average capacity variations as i.i.d. Gaussian random variables with zero mean and standard deviation $\sigma$ varying within the set $\{0, 0.2c_{avg}, 0.4c_{avg}, 0.6c_{avg}, 0.8c_{avg}, c_{avg}\}$, where $c_{avg}$ is the average of the true capacity. We compare BDT with GPCT and the greedy strategy which always delivers video content as soon as possible. The results are shown in Figure 2.7. As can be seen, BDT achieves up to a 15% reduction in system utilization as compared to the greedy strategy, while the optimal offline solution achieves a 25% reduction in the same simulation scenario. Moreover, BDT only results in a slight increase in rebuffering time (up to 0.05 seconds in the simulations, which is almost negligible as compared to the length of the video). Also as the prediction error grows, the performance of BDT does not drop too much. Thus BDT is effective in terms of reducing system utilization, and simultaneously keeping a low rebuffering time, and can work against uncertainty in future capacity variations.

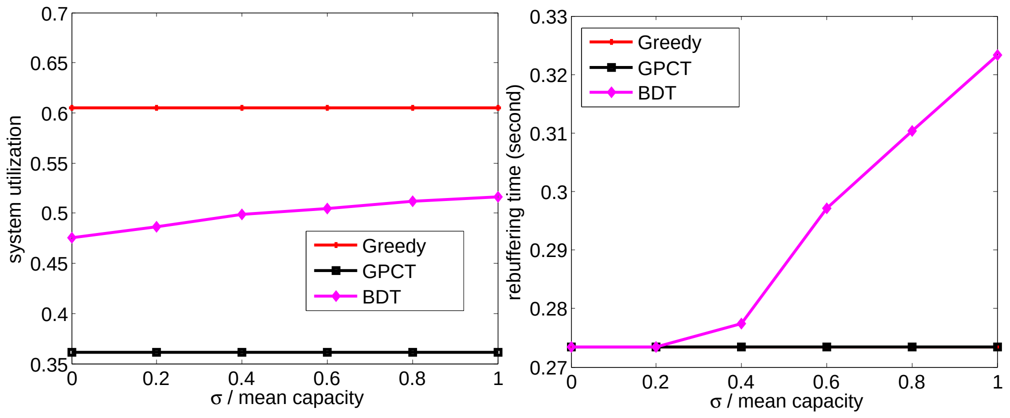


Figure 2.7: Performance comparison among BDT, GPCT and the greedy strategy. BDT achieves up to a 15% reduction in system utilization as compared to the greedy strategy and only results in a slight increase in rebuffering time. Also as the prediction error grows, the performance of BDT does not drop too much.

## 2.7 Conclusion

By leveraging geolocation and contextual information regarding users mobility patterns it is possible to predict the large-scale wireless capacity variations mobile users are likely to see. In this chapter we have developed and analyzed new cross-layer transport protocols that exploit knowledge of future capacity variations to deliver stored video (or other files) efficiently without compromising rebuffering/delays. Our analysis and simulations suggest this has substantial potential to increase the ability of wireless systems to deliver stored video in the case of mobile users seeing high variability in their available capacity.

# Chapter 3

# Application-Aware Scheduling in D2D Networks

## 3.1 Introduction

Device to Device (D2D) communication refers to establishing direct wireless links among mobile devices circumventing relaying through infrastructure, e.g., Wi-Fi and cellular access points. This is expected to be a key technology to improve overall system capacity and users' experience for various proximity-driven services, e.g., file sharing, spatial context sharing and advertising, local voice service, local multicasting, multiplayer gaming, augmented reality, synchronization among personal devices, communication among wearable devices and smart phones, etc. Overviews of D2D technologies can be found in [23] and [24].

D2D links have many advantages. First, short-range links can achieve high data rate at low power thus increasing spatial reuse. Second, such links

This chapter is based on my work in [25] (in submission to IEEE/ACM Trans. on Networking), which is co-authored by Dr. Gustavo de Veciana.

can directly offload traffic that would otherwise need to be relayed on 2 hops through infrastructure, thus reducing overheads, transmission delay, and saving precious infrastructure resources. Third, D2D connectivity also makes cooperation among devices, e.g., a mobile device can function as a gateway or relay for others, enabling flexible usage scenarios. Note that D2D communications can be supported on licensed and/or unlicensed bands, e.g., LTE Direct [34] and LTE Proximity Service (ProSe) [3] are targeted for licensed bands while Wi-Fi Direct [42] operates on unlicensed band, thus the beneficiaries of offloading and thus cost structures may differ.

A key feature of D2D networks is link dynamics. Since links are only available when devices are within close proximity of each other, network topology can have substantial variability. D2D links are likely to be initialized and ended frequently depending on users' behavior. For example, people who meet during a conference may have a few hours to share a large file; while meeting on the street may limit link availability to several minutes; vehicle-to-vehicle communications may be available only for a few seconds. The work in [18] presents a study of the distribution for such available times. We refer to the time for which a link is available as its contact time, thus giving a deadline for completing the associated data transmission. The characteristics of traffic loads and network dynamics can thus have a substantial impact on the usefulness of D2D technology.

The focus of this chapter is on understanding how scheduling can improve the performance of such networks. We consider a broad class of applications where links are set up to share files (data, images, videos, etc.) but are only available for a limited 'contact time'. Our focus is on the following performance metrics: (1) total offloaded data; (2) number of transfers completing within deadlines; and (3) average file transfer delays for completed transfers. Offload traffic is a *system* performance metric, since increasing the offloaded data will reduce the traffic on the infrastructure networks. Of course individual users will likely also benefit from offloaded data if such transfers are cheaper or free relative to infrastructure mediated communications. Completion of transfers is desirable from both the users' and system's perspective since in the worst case it reduces partial transfers/lost data and in the best case it reduces overheads/delays associated with switching the transmissions from the D2D link to infrastructure networks. Average delays for completed transfers are also a natural user perceived performance metric. Note that we focus on file transfers because they are the intrinsic components of many applications e.g., web browsing, multi-user gaming, stored video streaming, etc. Thus besides file transfers, our work will provide insights on managing QoS in a broader set of applications.

Many D2D link scheduling algorithms have been proposed over the last decade. Current state-of-the-art examples include FlashLinQ [44] which is intended for D2D transmission on licensed bands, and CSMA-based protocols e.g., IEEE 802.11 family, IEEE 802.15.4, which are used in establishing D2D links via Wi-Fi technology, e.g., SoftAP, Wi-Fi direct, etc. Further recent research building on these standards include ITLinQ [31] and Quantile-based

CSMA [20] aiming to further enhance sum throughput and/or fairness. The focus to date has been on 'greedily' optimizing the sum throughput perhaps subject to fairness concerns for a *static* set of users. Such schedulers may, however, not result in good performance in the dynamic settings we envisage in practice. For example, Figure 3.1(a) exhibits simulation results for the offload traffic and number of completed transfers for a dynamic network with increasing D2D link arrival rates under FlashLinQ scheduling – details are provided in Section 3.5. As expected when the D2D traffic load increases so does the offload traffic, however, the number of successful transfers eventually collapses. This is not really surprising since at higher loads one might expect individual users to have lower throughputs and thus miss their completion deadlines. Could better schedulers further enhance both traffic offload and completed transfers?

Figure 3.1(b) shows a snapshot of the locations of active D2D links, during a simulation of a dynamic network, wherein most of the D2D links are clustered in the upper right corner of the square area, while only a few links are present at the bottom right. This 'clustering phenomenon' is partly caused by the randomness of the underlying arrival process – again see Section 3.5 for details. However another cause for such link clustering is that they experience more interference than those in a less crowded area, which results in lower throughput for the clustered links, which delays their completions. As a result the links in a crowded area will further delay new links in that area. Overall this results in the emergence of increased link clustering which may result in poorer performance. In this chapter we explore the significance of this phenomenon. Could better schedulers mitigate this clustering effect?

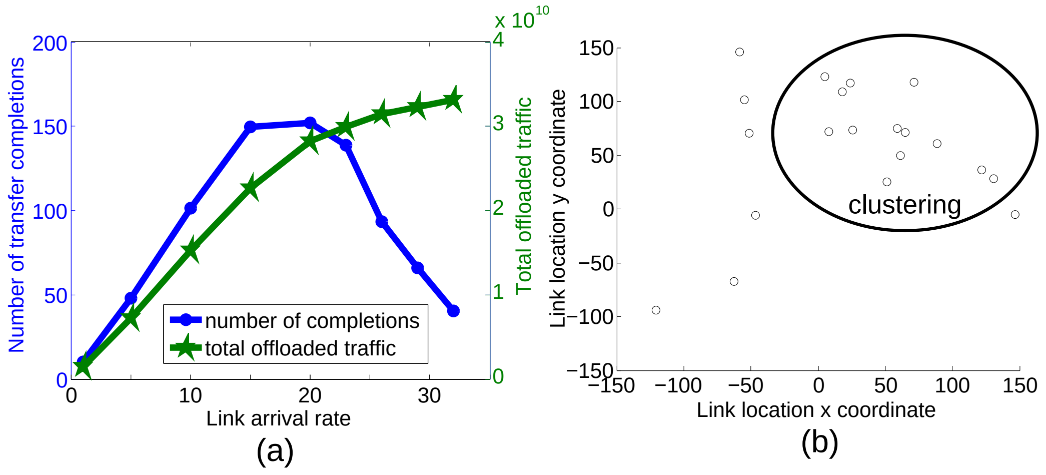


Figure 3.1: (a) The number of file transfer completions and the amount of offloaded traffic under FlashLinQ vs. link arrival rate; (b) A snapshot of the locations of active D2D links during the simulation.

*Related Work.* There has been a substantial body of literature on resource allocation in D2D networks. These works can be categorized in terms of four usage scenarios: (1) non-cooperative D2D networks overlaid on separate frequency bands than the cellular network, where each D2D link is initiated between a transmitter and its associated receiver, and scheduling algorithms (either distributed or with base station assistance) can be designed without consideration of the cellular users. The goal is then to avoid link collisions (or mitigate interference) and optimize the network sum throughput subject to some fairness concerns for D2D users. Examples of such work are [44], [31], [20] and [33]. (2) The second scenario is that of a non-cooperative D2D network underlaid with a cellular network, where D2D transmitter-receiver pairs and cellular users are sharing the same resource. The D2D users usually can have two transmission modes: cellular mode in which the D2D transmissions use the base station as a relay; and D2D mode where the D2D transmitters communicate directly with their receivers. Thus the schedulers need to decide how to allocate the resources between cellular and D2D users, and also decide which mode a D2D pair should be in if mode selection is allowed. Such works, e.g., [4], [46], [47] and [8], mainly focus on maximizing the total throughput or minimizing the total power consumption, again possibly subject to some fairness concerns. (3) The third scenario is that of cooperative D2D networks, where users are divided into cooperating clusters, and in each cluster the users are receiving the same service (e.g., downloading the same file) from base station and/or sharing it amongst each other through D2D links. These works mainly focus on finding optimal clusterings that minimize file transfer delay, maximize total throughput or minimize total power consumption, e.g., [13], [11] and [21]. (4) The fourth scenario is that of D2D relay networks, where D2D links are set up for the purpose of relaying traffic from one user to another or to the base station, in order to mitigate rate degradation for users with poor coverage or in dead spots, or relaying traffic from an overloaded cell to an underloaded cell to balance traffic loads. The goal is to improve overall throughput (or minimize power consumption) and provide fair service to all users. Note that many of the ideas in D2D relay networks can be borrowed from early works on ad hoc networks. Such works have been studied in, e.g., [28], [15] [43], [2]. Moreover, [12] proposes a relaying policy taking advantage of user mobility.

This chapter focuses on the first scenario, i.e., non-cooperative D2D networks overlaid on a cellular network. The difference between our work and the other works in this setting is that, we consider D2D links' dynamics and QoS in terms of perceived per-flow performance. We devise scheduling policies jointly improving offloading and number of successful completions when D2D links' transfers have deadlines. To the best of our knowledge, there is no prior work on this topic. There is however a literature on scheduling traffic with deadline constraints in a broader settings which we discuss next.

Going back to the real-time scheduling literature [30] presents problems, e.g., resource sharing, processor allocation, etc., for distributed systems where tasks have hard real-time constraint. They propose conditions for the feasibility of scheduling policies to meet hard constraints, but do not consider the case where the requirements may be infeasible and one still wishes to maximize the number of successful completions. In [38] the authors study the scheduling of multiple real-time streams with deadlines, over a shared channel. The focus is on maximizing the number of packets successfully transmitted within their deadlines. They propose the Feasible Earliest Due Date (FEDD) scheduling policy and prove optimality under particular assumptions. This work focuses on packet-level deadline constraints, not flow-level (i.e., file transfer) deadlines which are our concern. Also the work assumes an on/off wireless channel model with only two channel states, and thus there is no concern with throughput. The authors in [14] study a wireless network with time-varying capacities and delay constraints. They propose an online scheduling algorithm that optimizes users' long-term average rates under packet delay constraints. The algorithm addresses fairness by guaranteeing each user a certain minimum long-term service rate, and assumes this is strictly feasible. This work also focuses on packet-level deadlines, and thus does not address the flow-level deadline constraints. In [45] the authors propose a family of bandwidth allocation criteria which depends on the residual work of on-going transfers. This is a similar idea to that proposed in this chapter, i.e., giving high priority to the flows with smaller residual sizes. However, [45] focuses on minimizing the user perceived average bit transmission delay, and does not address the issue of completion/failure of flows subject to deadlines. The authors of [5] study a dynamic scenario similar to ours, where users come and go as governed by the arrival and completion of service demands over time. They focus on the performance of channel-aware scheduling algorithms at the flow level and provide analysis on the distribution of the number of active users, the mean service times, the throughput and on the stability of the system. This work however uses a processor-sharing model which does not capture the characteristics of D2D networks. Moreover, it does not address flows with deadlines.

*Our Contributions.* In this chapter we explore the potential benefits of sharing and exploiting application-aware information, e.g., residual file sizes, deadlines etc., in D2D link scheduling. Our specific contributions are as follows. First we analyze a single collision domain setting where all links contend for the same shared resource. In this setting we establish conditions under which there exist performance optimal schedulers maximizing offload traffic or the number of transfer completions. A key result is that if the transfers have deadlines, the Earliest Deadline First (EDF) scheduler, will maximize the total offload traffic when links have homogeneous service rates. Note that EDF is known to either produce a feasible schedule under flow-level delay constraints (e.g., in [30]), or to maximize total number of delivered packets under packet-level delay constraints (e.g., in [38]). However in this chapter we show that it can also maximize throughput under flow-level delay constraints. Another key result is that if transfers have deadlines, the Shortest Remaining Processing Time (SRPT) first scheduler, will maximize the number of completions when specific traffic requirements are met. Note that although SRPT is known to minimize average delays in settings without deadlines, we show, perhaps surprisingly, that in a deadline constrained scenario it also maximizes completions. The requirements for this to be true will tend to be satisfied when the network is heavily loaded, i.e., when these constraints are tight. The key insight is that when loads are heavy SRPT will end up maximizing the number of completions. We also show that the schedulers maximizing the offload traffic may not be compatible with those maximizing completions, and further that in general no online policy is available to optimize the two metrics.

Second we use these insights to develop application-aware versions of FlashLinQ and CSMA-like protocols with a view on improving offload traffic, number of completions and reducing delays in dynamic D2D networks.

Our simulation results validate the theoretical insights suggesting that indeed there are substantial benefits to be gleaned from application-aware D2D link scheduling. We explore the sensitivity of these results to various underlying assumptions on the network scenario. Third we investigate the benefit of applying admission control. Our simulations show that a simple admission control strategy can improve QoS among admitted links, moreover, admission control used together with application-aware schedulers can further enhance its performance. Fourth, we consider a scenario where D2D links' transfers have no deadlines and thus the system can be unstable for high link arrival rates. Our simulation show that application-aware schedulers result in lower file transfer delays and have the potential to increase the system stability region. Finally, we show that application-aware schedulers are able to mitigate spatial clustering of links and thus reduce the overall interference among nearby links, which explains why such schedulers are able to enhance links' QoS as well as system capacity.

*Paper Organization.* The rest of the chapter is organized as follows. Section 3.2 describes the dynamic D2D networks to be considered. In Section 3.3 we analyze a simple scenario where all D2D links contend for the same shared resource, i.e., a single collision domain. We establish optimal schedulers maximizing offloaded data and the number of successful transfers separately under particular assumptions, and obtain insights for designing scheduling algorithms in general scenarios. We exploit these insights and design application-aware schedulers based on FlashLinQ and a CSMA like protocols in Section

3.4. In Section 3.5 we evaluate the performance of our proposed schedulers via simulation and show the merit of application-aware schedulers in various aspects e.g., improving links' QoS, increasing total offload amount, enhancing performance for a system under admission control, reducing transfer delays in the scenario where links have no deadlines, mitigating links' spatial clustering, etc. Section 3.6 briefly concludes the chapter.

## 3.2 Dynamic D2D Network Model

We consider a network where D2D transceiver pairs associated with interactions among users and/or machines are initiated according to a general spatio-temporal arrival process. Upon initiation a D2D link $i$ has both an associated file of size $s_i$ it wishes to transfer and a deadline $d_i$ modeling a limited contact time amongst the associated devices or possibly an application-level requirement. The pairs $(s_i, d_i), i = 1, 2, ..$ can be arbitrary, possibly capturing usage scenarios where the file sizes and deadlines are correlated, e.g. (1), it may be expected if two users of a D2D link initiate a large D2D file transfer, they have a priori knowledge of the time they are likely to be close to each other; or (2), in order for users to experience a consistent D2D throughput of at least $r$ the network may assign a deadline for a file transfer of size $s_i$ given by $d_i = s_i/r$. We assume the D2D links' locations, and thus associated users/machines do not change during their presence in the system. If a link is only available for a limited time, it leaves the system when either the associated file transfer completes or its deadline expires.

We shall also assume a D2D network which is operating on a dedicated frequency band which is orthogonal to the infrastructure network, e.g., cellular network, WLAN, etc. So there is no interference amongst D2D and infrastructure transmissions. For simplicity we assume all D2D links share the same frequency band so their transmissions may interfere with each other. In particular as the density of D2D links becomes large the distance between a typical D2D link and its nearest neighbor will become smaller and thus the interference they experience if they transmit simultaneously will be higher. So scheduling algorithms are needed to avoid transmission failure caused by interference. As should be clear, there will be a complex interaction among the D2D link dynamics, the scheduling of D2D link transmissions, interference, and deadlines, which will drive the performance metrics of interest. In order to develop a basic understanding, in the next section we first consider a simpler system where all the links belong to a single collision domain, i.e., a region where all ongoing D2D links strongly interfere with each other so only one can be scheduled at a time.

## 3.3 Single Collision Domain Model

When D2D links belong to a single collision domain the network can be modeled as a single server queue where links arrive as a random process. Each link $i$ has an associated initial file size $s_i$, transfer deadline $d_i$ and service rate $r_i$ bits/s corresponding to link $i$'s capacity when it is scheduled. We let $t_i^a$ denote the arrival time for link $i$ and denote by $t_i^d$ its departure which occurs when either the file transfer completes or the deadline expires, and thus depends on the link scheduling algorithm. As mentioned earlier in this setting only one link will be scheduled at a time but we assume service given to a link's file can be preempted and subsequently resumed at will. For link $i$, we let $s_i^r(t)$, $d_i^r(t)$ and $\tau_i(t)$ denote the residual file size, remaining time to deadline, and the cumulative service it has been given up to time $t \in [t_i^a, t_i^d]$. In other words, when link $i$ arrives at $t_i^a$ we have $s_i^r(t_i^a) = s_i$ and $d_i^r(t_i^a) = d_i$ and subsequently at some time $t \in [t_i^a, t_i^d]$ we have that $s_i^r(t) = s_i - r_i \tau_i(t)$ and $d_i^r(t) = d_i - (t - t_i^a)$.

As we will see below, under additional specific conditions one can devise online scheduling policies which maximize the total offload traffic and/or number of file transfer completions.

### 3.3.1 Schedulers Maximizing Offload Traffic

In order to maximize the offload traffic, i.e., aggregate D2D bits delivered, one might imagine greedily serving the link with the highest service rate. Unfortunately since each link has a deadline, this greedy strategy is not optimal. The following result exhibits the role of D2D link deadlines for offload optimal schedulers.

**Theorem 5** *Consider the server system modeling a single collision domain described above and suppose further that links' service rates are homogeneous, i.e., $r_i = r\ \forall i$, then Earliest Deadline First (EDF) scheduling policy maximizes the offload traffic at all times for arbitrary arrival processes but without accumulation points. Under the EDF scheduler at any time $t$, the link with* *the earliest deadline is served, i.e., that with smallest $d_i^r(t)$ among all links available for service.*

*Proof:* Since all the links share the same service rate $r$, a scheduler maximizing the system's utilization will also maximize the offload traffic. We shall prove the theorem via an exchange argument. Consider an optimal scheduler $\phi$, we shall show that through a sequence of swaps we can convert $\phi$ into EDF scheduling without decreasing the system utilization.

Suppose EDF and $\phi$ schedule the same links during $[0, t_1)$ but they differ thereafter. If so, without loss of generality since the arrival has no limit points, there is an interval $[t_1, t_1 + \delta)$ where EDF schedules a link, say Link 1 and $\phi$ schedules a different set of links, for simplicity lets say Link 2. [1]

We now modify $\phi$ so that it matches EDF over the interval $[0, t_1 + \delta)$ but still has the same utilization. There are two cases to consider:

**Case 1.** Suppose $\phi$ schedules the bits EDF schedules for Link 1 on $[t_1, t_1 + \delta)$ at some later time, e.g., an interval $[t_2, t_2 + \delta)$ or possibly a set of intervals but with the same length $\delta$. If so, we can exchange the service times that $\phi$ allocates to Link 2 on $[t_1, t_1 + \delta)$ with those it allocates to Link 1 later on, e.g., $[t_2, t_2 + \delta)$. This modification results in no change of utilization for $\phi$ but now it matches EDF on the interval $[0, t_1 + \delta)$. This modification must be feasible, i.e., meet deadlines, since Link 2's deadline must be greater than or equal to

[1]Note for simplicity we assume the scheduler $\phi$ schedules but one link at a time, i.e., if it were continuously time sharing between two or more links one could aways modify it so that it scheduled but one link at a time for the appropriate fractions of times.

that of Link 1.

**Case 2.** Suppose $\phi$ does never schedules some of the bits served by EDF on $[t_1, t_1 + \delta)$ at some later time. If so we can let $\phi$ schedule those bits during $[t_1, t_1 + \delta)$ in lieu of those it was serving. Once again there is no change of system utilization for $\phi$ and this replacement must be consistent with the deadlines of the traffic. Once again $\phi$ has been modified to match EDF without lowering its utilization.

The above procedure can be carried out until $\phi$ has been transformed into EDF without modifying its utilization and thus its associated offload traffic. It follows that EDF is also optimal. ■

The insight behind Theorem 5 is that when links have different deadlines, if we serve the link with the earliest deadline first, we may still be able to serve the links with longer deadlines at a later time. Otherwise if we serve the links with longer deadlines first, the links with early deadline may expire before getting served. Although our optimality result is only guaranteed when all the links have homogeneous service rates, this insight also reflects the role of deadlines in heterogeneous cases, especially when system is not fully loaded. However we should point out that when the links' arrival rate is large, the role of deadlines is less critical. In fact, in such cases there will likely to be a large number of links in the system at any time, and one can always find a link with a high capacity to serve, thus no need to delay the transmission of the links with high rates and long deadlines. As a result, greedily serving the link with the highest service rate may be near optimal when the system is heavily loaded.

### 3.3.2 Schedulers Maximizing Number of File Completions

Schedulers aiming to maximize the number of file completions also face the challenge of not knowing future arrivals and their associated deadlines. Still with an additional condition on the traffic, we can prove the optimality of a simple online scheduler.

**Theorem 6** *Consider the server system modeling a single collision domain described above. If file sizes, transfer deadlines and possibly heterogeneous user service rates are such that for each link $i$ we have*

$$d_i - \frac{s_i}{r_i} < \frac{s_j}{r_j},\ \forall j, \tag{3.1}$$

*then the Shortest Remaining Processing Time (SRPT) first scheduling policy maximizes the number of file completions for all finite times for arbitrary arrival processes but without accumulation points. Under SRPT scheduling at any time $t$, a link $i^*$ is served only if*

$$i^* \in argmin_i\{\frac{s_i^r(t)}{r_i}|s_i^r(t) > 0, \quad d_i^r(t) - \frac{s_i^r(t)}{r_i} \geq 0\}, \tag{3.2}$$

*i.e., has unfinished work, its deadline has not expired and has the shortest residual service time. Ties can be broken arbitrarily.*

*Proof:* In Lemma 4 below we will show that when the link traffic load satisfies Condition (3.1) and links are scheduled according to the SRPT policy then at any time $t$, for any active link $i$, the following condition holds:

$$d_i^r(t) - \frac{s_i^r(t)}{r_i} < \frac{s_j^r(t)}{r_j} \;\; \forall j, \text{ s.t. } \frac{s_j^r(t)}{r_j} > \frac{s_i^r(t)}{r_i}. \tag{3.3}$$

Now suppose rather than scheduling $i^*$ (i.e., an active link with the SRPT) we schedule link $j$. The above condition guarantees that completing the transfer for link $j$ will make it impossible to finish link $i^*$ before its deadline. In other words more time i.e., $\frac{s_j^r(t)}{r_j}$ will be required to complete $j$ than would be required to complete $i^*$, i.e., $\frac{s_{i^*}^r(t)}{r_{i^*}}$. Thus such scheduling policy would be suboptimal.

■

Note that Condition (3.1) is tied to the deadline slack for each file transfer being less than the service time for files. The following lemma establishes that under SRPT such conditions propagate.

**Lemma 4** *Consider the server system modeling a single collision domain described above. If file sizes, transfer deadlines and possibly heterogeneous link service rates are such that for each link $i$, Condition (3.1) is satisfied, then under the SRPT first scheduling policy at any time $t$, for any active link $i$, Condition (3.3) holds.*

*Proof:* First note that SRPT will keep serving a link until the file transfer completes, deadline expires or a new link arrives. We prove the lemma by induction. The condition holds trivially prior to the first link arrival to the system. Now suppose the condition holds at all times during the interval $[0, t)$ and that at time $t$ there are $n$ active links in the system, denoted without loss of generality link $1, 2, ..., n$, and Condition (3.3) holds for these links up to time $t$. Suppose SRPT is scheduling Link $i$ at time $t$. We consider two cases.

**Case 1.** Suppose no new link arrives before Link $i$'s file transfer completes or deadline expires, at time $t_i^d = t + \delta$, then (3.3) will continue to hold for all the $n$ links at all times during the process, and it holds for the other $n - 1$ links in the system when link $i$ exits at time $t + \delta$. This should be clear since when SRPT is serving link $i$, $d_j^r(t) - \frac{s_j^r(t)}{r_j}$ is decreasing, while $\frac{s_j^r(t)}{r_j}$ stays fixed for all $j \in \{1, ..., n\} \backslash \{i\}$. Thus (3.3) will continue to hold for $[0, t + \delta]$.

**Case 2.** A new link say $n + 1$ arrives at time $t_{n+1}^a = t + \delta$, before the file transfer of link $i$ completes. On one hand for any link $j \in \{1, ..., n\}$ where $\frac{s_{n+1}}{r_{n+1}} > \frac{s_j^r(t_{n+1}^a)}{r_j}$, we have:

$$d_j^r(t_{n+1}^a) - \frac{s_j^r(t_{n+1}^a)}{r_j} \leq d_j - \frac{s_j}{r_j} < \frac{s_{n+1}}{r_{n+1}} = \frac{s_{n+1}^r(t_{n+1}^a)}{r_{n+1}}. \tag{3.4}$$

Note the first inequality in (3.4) is due to the fact that the slack (i.e., $d_j^r(t) - \frac{s_j^r(t)}{r_j}$ for link $j$ at time $t$) of a link is always non-increasing. The second inequality in (3.4) is Condition (3.1). On the other hand, if $\frac{s_{n+1}}{r_{n+1}} < \frac{s_j^r(t_{n+1}^a)}{r_j}$, similarly we have:

$$\begin{aligned} d_{n+1}^r(t_{n+1}^a) - \frac{s_{n+1}^r(t_{n+1}^a)}{r_{n+1}} &= \\ = \quad d_{n+1} - \frac{s_{n+1}}{r_{n+1}} < \frac{s_{n+1}}{r_{n+1}} &< \frac{s_j^r(t_{n+1}^a)}{r_j}, \end{aligned} \tag{3.5}$$

where the first inequality is derived from Condition (3.1). Combining (3.4) and (3.5) it follows that (3.3) holds for all the $n + 1$ links over the interval $[0, t + \delta]$. The induction hypothesis thus holds for the larger time interval

$[0, t+\delta]$. Since the arrival process has no accumulation points it should be clear that this guarantees it holds for all subsequent times. ■

### 3.3.3 General Online Optimal Scheduling Algorithms Do Not Exist

The restrictions (homogeneous service rates, tight deadline constraints) for the optimal schedulers developed in the previous two subsections leave open the question whether there may exist more general optimal online algorithms for the single collision domain model. In this section we show that general online optimal scheduling algorithms do not exist, in other words to achieve optimality one will need knowledge of the future. Consider the following simple example.

Suppose at time $t = 0$, there are two links in the system, with corresponding residual file sizes, service rates and deadlines given by $s_1^r(0) = 2,\ r_1 = 1,\ d_1^r(0) = 2,\ s_2^r(0) = 2,\ r_2 = 2,\ d_2^r(0) = 3$. If there are no future arrivals, the optimal schedule is to first serve Link 1 until time $t = 2$, then schedule Link 2 until time $t = 3$. The maximum offload is 4, and the maximum number of completions is 2. However, if two new links arrive at time $t = 1$, with $s_3 = s_4 = 2,\ r_3 = r_4 = 2,\ d_3 = d_4 = 2$, then the previous schedule cannot achieve optimality in terms of either the offload or number of completions, even if we change the schedule upon the new arrivals at $t = 1$. Indeed it can be easily seen that the optimal schedule is to serve Link 2 during $t \in [0, 1]$, then serve Link 3 during $t \in [2, 3]$, and serve Link 4 during $t \in [3, 4]$. But note that serving Link 2 during $t \in [0, 1]$ cannot achieve optimality in the previous case. Thus any online algorithm which is agnostic of future arrivals cannot guarantee optimality.

## 3.4 Application-Aware FlashLinQ and CSMA-Like Schedulers

In the previous section, we obtained some insights on how to schedule links in a dynamic D2D network by analyzing a single server model. Although we proposed optimal schedulers under some specific conditions, we should note that there is no easy way to generalize these to a general D2D network, due to the complex interactions among: the link dynamics, the scheduling of link transmissions, interference, and deadlines. However the insights we obtained can be used to devise good schedulers.

Theorem 5 suggests that to maximize offload traffic it is beneficial to serve the links with early deadlines first, especially when the links have similar capacities or when the system is not heavily loaded. Theorem 6 suggests that in order to improve the number of file transfer completions it is better to first schedule the links associated with files having short remaining processing times. Note that Theorem 6 is proved assuming Condition (3.1), which requires that the deadline slack for each file transfer is less than the service time for files. One can expect this might hold in a network with heavy loads since due to interference the service rate per link might be low.

Although these insights were obtained under special assumptions, we think they should still work well under general situations. Below we show

how to modify two state-of-the-art D2D schedulers, FlashLinQ and CSMA like protocols, based on these insights.

### 3.4.1 Application-Aware FlashLinQ

FlashLinQ is a synchronous peer-to-peer wireless PHY/MAC network architecture with many interesting innovations [44]. One of the innovative aspects of this protocols is an signaling mechanism which enables each receiver to know the channel gains from neighboring transmitter and transmitter to subsequently know the channel gains to neighboring receivers, again see [44] for details. Here we shall focus on link scheduling. The basic idea is to schedule a maximal feasible set of links based on measured channel conditions. On any time slot, FlashLinQ determines the set of links it would schedule as follows. First all links are *randomly* assigned a priority for that slot– this is actually computed based on the link identifier and a shared seed, so all links have a consistent view of the priorities of their neighboring links on each slot without exchanging additional information. Second, each receiver decides if it should yield and does so by checking if the interference from higher priority links would push its signal to interference ratio below a threshold $\gamma_{rx}$. Subsequently each transmitter decides if it should yield to higher priority links that have been scheduled (i.e., receiver has not yielded), by checking that it does not cause undue interference to those links. In principle this process could be repeated to efficiently achieve a dense packing of links.

Later in Section 3.5 we are going to see a simulation result for a dynamic D2D network under FlashLinQ, as in Figure 3.3. It exhibits reasonable performance in terms of offload traffic but a poor performance in terms of number of file transfer completions. This makes sense since FlashLinQ is agnostic of both file sizes and link deadlines – so we can ask whether performance could be substantially improved by making the scheduler application-aware. Recall that in FlashLinQ, the links with higher priority have a higher chance to be scheduled. Thus, instead of assigning random priorities to links, we might consider assigning priorities, based on application-layer state.

We briefly discuss our proposed Application-aware FlashLinQ (A-FlashLinQ) scheduler. Each active D2D link, say Link $i$ in the system maintains a moving window average estimate of its application layer throughput $\bar{r}_i(t)$ at time $t$, corresponding to the throughput seen in the window $[t - t_{pwnd}, t)$. Link $i$ also updates its residual file size $s_i^r(t)$ and remaining time to deadline $d_i^r(t)$ if known. Links are scheduled every $t_{slot}$ second, which is on the order of the wireless frame time 2ms.

*Setting priorities for link scheduling.* We propose to set priorities for link scheduling in an application-aware manner. For now lets us discuss this as a centralized process. On each time slot $t$, links are classified into two groups: Group 1 contains links which are *tightly constrained* while Group 2 contains those which are *loosely constrained* and *infeasible*. These are defined as follows:

• *Tightly constrained* links have either been in the system less than $t_{pwnd}$ second, or have residual time to deadlines close to the estimated residual processing time, i.e.,

$$\{i|d_i^r(t) \in [\alpha_l, \alpha_h]\frac{s_i^r(t)}{\bar{r}_i(t)}\},$$

where $\alpha_l < 1$, $\alpha_h > 1$ are two thresholding parameters. Note that these links are close to violating their deadlines.

• *Loosely constrained* links are such that their residual time to deadlines exceeds the estimated remaining processing time, i.e.,

$$\{i|d_i^r(t) > \alpha_h\frac{s_i^r(t)}{\bar{r}_i(t)}\},$$

i.e., they have some time before the deadline expires.

• *Infeasible* links are such that their residual time to deadlines is less than the estimated residual processing time, i.e.,

$$\{i|d_i^r(t) < \alpha_l\frac{s_i^r(t)}{\bar{r}_i(t)}\},$$

so, such links are unlikely to complete.

In addition to avoid instability we include hysteresis, whereby once a link is classified to be of a given type, it must wait $t_{pwnd}$ second before changing its classification.

Priorities are set as follows. First we give Group 1 links higher priority than Group 2. This is because the tightly constrained links are those likely to fail in the near term but may still be able to finish if getting served in time. Meanwhile loosely constrained links can wait since they have more slack, – this matches the insight from Theorem 5. Finally infeasible links have little hope of completing their file transfers prior to their deadlines and so are given low priority. In summary we will prioritize resource allocation to tightly constrained but feasible links in order to improve file transfer completions.

Furthermore amongst Group 1 links priority is given in ascending order according to their estimated remaining processing times, i.e., $\frac{s^r(t)}{\bar{r}_i(t)}$ , so these are at the top of the priority list. Similarly among Group 2 links prioritization is based on their estimated remaining processing time $\frac{s^r(t)}{\bar{r}_i(t)}$. Note we use residual processing time as the key criterion for prioritization so as to improve the number of file transfer completions, – this is based on the insight given in Theorem 6. Given this overall priority list, we let the other parts of FlashLinQ to proceed unchanged.

*Reducing overheads in application-layer information exchanges.* As mentioned above FlashlinQ requires all links to have a consistent priority list, thus the above prioritization needs established and shared amongst all (neighboring) links. This appears to require quite a bit of information to be exchanged, e.g., share residual processing times and possibly deadlines amongst all neighboring links. To reduce such overheads, we assume that only 3 bits of application-layer information are broadcast by each transmitter with its receiver and the transmit/receiver of all neighboring links. To that end we require links to quantize their residual processing times. The first bit indicates if the link is in Group 1 or 2. The second and third bits reflect its quantized residual processing time. With the above application-level info and a random prioritization of the links (as currently available in FlashlinQ) an app-aware prioritization of links can be determined as follows. Links in Group 1 have higher priority than those in Group 2. Within each group, links are prioritized according to an ascending order of their quantized residual processing time. Furthermore if two links have the same quantized value, then they are ordered based on their current random priorities. Figure 3.2 shows an example of this process. Each link, say L1, determines its neighbors which are in Group 1 and have smallest quantized residual processing time, e.g., L2,L3. These are then ordered based on their current random priorities, i.e., L3 before L2. This is also easily done for other residual processing time intervals, and then for neighboring links in Group 2 also based on their quantized residual processing times. Note that the application layer information really only needs to be exchanged when there is a change in the quantized status of the link, thus further reducing overheads.

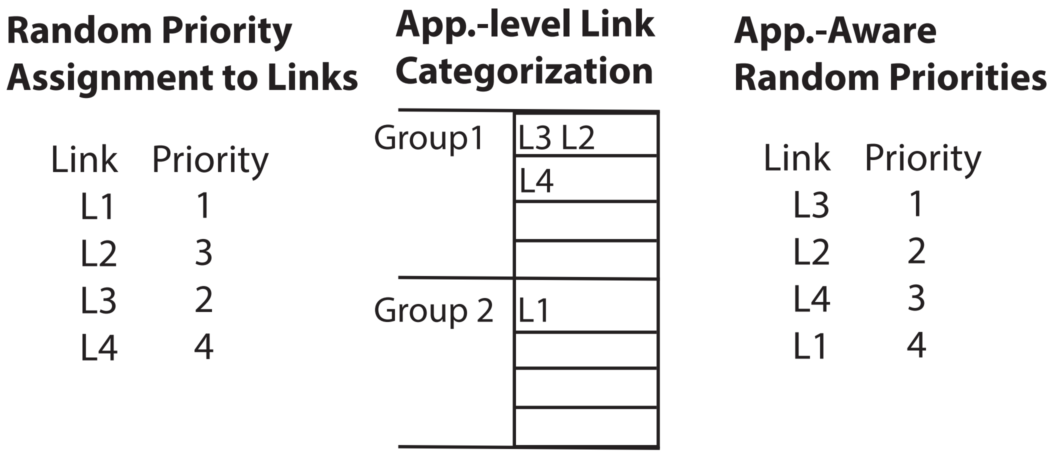

Random Priority Assignment to Links
Link Priority
L1 1
L2 3
L3 2
L4 4
App.-level Link Categorization
Group1
L3 L2
L4
Group 2
L1
App.-Aware Random Priorities
Link Priority
L3 1
L2 2
L4 3
L1 4

Figure 3.2: Example of how application level groups and quantized residual service times along with a random priority assignment to all links can be combined to get an application aware prioritization of links.

Note that in most cases it is easy for applications to track the residual file size and average application layer throughput, and thus estimate remaining processing time. However our prioritization procedure also requires the applications to have knowledge or estimates of their deadlines. This is straightforward if deadlines are driven by application requirements. However if deadlines are modeling link contact times which mainly depend on the users' behavior, it may not easy for the applications to determine their deadline slack in this case. If this is the case one can skip the grouping procedure and simply sort all links based on their quantized remaining processing times (where all 3 bits are used to represent quantized remaining processing time). We refer to this strategy as A-FlashLinQ without grouping.

### 3.4.2 Application-Aware CSMA-Like Protocols

Application-aware scheduling can be similarly realized in other D2D link scheduling protocols. Carrier Sense Multiple Access (CSMA) is the MAC scheduling protocol widely used in random access networks, e.g., CSMA/CD in early local area networks (Ethernet), CSMA/CA in IEEE 802.11 wireless local area networks, slotted CSMA/CA in IEEE 802.15.4 low-rate wireless personal area networks, etc. D2D networks, especially those based on Wi-Fi technology can use CSMA-like protocols to realize link scheduling. Our goal in this section is not to cover the substantial details of CSMA implementation, but instead we focus on a simplified model for CSMA-like protocol and explore the benefits of incorporating application-aware scheduling.

We consider a slotted CSMA/CA model similar to the beacon-enable mode in IEEE 802.15.4, but we make several simplifications. A detailed description and evaluation of slotted CSMA/CA can be found in [22]. In our model we assume availability of a central node (e.g., PAN coordinator) sending beacon signals every $t_{slot}$ second to all the D2D transmitters in the network so all D2D transmissions are synchronized at the start of each slot. Each time slot is further subdivided into a contention period of length $t_{cp}$, followed by a transmission period with length $t_{tp}$, where $t_{cp} + t_{tp} = t_{slot}$.

The transmitter of each link, say Link $i$ maintains a contention window of length $cw_i$, and chooses a random backoff timer uniformly distributed on $[0, cw_i]$. During each contention period, the transmitter of Link $i$ counts down its backoff timer if the wireless medium is sensed to be free. If the timer reaches 0, the transmitter starts to transmit immediately. The transmission continues through the rest of the contention period and the transmission period. Subsequently, the transmitter of Link $i$ picks a new random backoff timer on $[0, cw_i]$, and repeats the same counting down procedure in the next contention period. The count down procedure is suspended if during the transmission period or if the medium is sensed to be busy by the transmitter of Link $i$ during the contention period.

By contrast with traditional CSMA, we shall update the value of contention window $cw$ based on application layer information. We do this in a similar manner to our proposed application-aware FlashLinQ. Links are self-classified into Group 1 or 2, based on the criteria described in Section 3.4.1. If Link $i$ is in Group 1, then at time $t$, it picks a contention window of size

$\frac{1}{2\beta_{RPT}}\frac{s_i^r(t)}{\bar{r}_i(t)}t_{cp}$, where $\frac{s_i^r(t)}{\bar{r}_i(t)}$ is the estimated remaining processing time at $t$ for Link $i$, and $\beta_{RPT}$ is a normalization factor which is larger than $\frac{s_j^r(t)}{\bar{r}_j(t)}$, $\forall j$. If Link $i$ is in Group 2, then at time $t$, it picks a contention window of size $\frac{1}{2\beta_{RPT}}\frac{s_i^r(t)}{\bar{r}_i(t)}t_{cp} + \frac{1}{2}t_{cp}$. Note that by doing so, the links in Group 1 will have smaller contention windows and thus higher transmission priority than others. Also within each group links with small residual service times will have a relatively high transmission priority. We refer to such a scheduler as an Application-aware Slotted CSMA-like (A-SCSMA) scheduler.

Note that the above A-SCSMA scheduler does not require the links to share individual application layer information. However all links need to know the normalization factor $\beta_{RPT}$ which may require the knowledge of the maximum remaining processing time among the concurrent links. In practice links can have a rough knowledge of such value based on estimation or historical statistics, or the central node (PAN coordinator) can collect and periodically broadcast this value via its beacon signals on a crude scale.

As with A-FlashLinQ, A-SCSMA requires a link to be aware of the remaining deadline of its file transfer. When such information is not available, we can skip the grouping policy in A-SCSMA, and treat all links as if they are in Group 1. We refer to this A-SCSMA without grouping.

Note that in our proposed A-SCSMA we skip many details as in conventional CSMA, e.g., DCF inter-frame space (DIFS) and short inter-frame space (SIFS) time intervals, RTS/CTS access mode, etc. We do not consider the overhead from frame headers or acknowledge packets (ACK). Again, we would like to point out that our goal is to explore the benefit of making the scheduler application-aware by studying this simplified model. Test bed or real case simulations are not the focus of this dissertation.

### 3.4.3 Managing FLow-Level Performance Through Deadlines

One of the challenges with devising scheduling algorithms for D2D networks is heterogeneity among links and users' needs, e.g., the performance seen by the users is highly dependent on their link lengths and their local environment. So there is a tradeoff between the sum rate and the fairness users will see. Ideally we would like to guarantee a certain user-level quality of service. One way to do this is to have the applications set deadlines for their transfers, and then have the system try to meet these deadlines. In addition the application could also let the users specify deadlines to reflect the users' preferences and/or contact times if they have such prior knowledge.

This approach is essentially equivalent to applications communicating to the network their QoS requirements. In such networks users might game the system, e.g., an application may manipulate its deadline (so that the corresponding link is scheduled as one that is "tightly constrained") and try to take unfair advantage over other transfers. Another example is one where an application may choose to break up its file into several smaller files to transmit in order to obtain higher transmission priority due to the smaller remaining processing times of the subfiles. This is not fair to the users who are not breaking up their files, and moreover, breaking up a file into pieces might also be problematic for the application level, e.g,. having more overhead associated with the initialization of file transfers. These concerns are typical in any system that schedules the users based on their reported QoS requirements. Such systems may need to let the users pay a well-designed “cost” to disincentivize them from gaming/cheating. While we do not study them in this dissertation, different approaches to disincentivize selfish behavior can be found in literature e.g., the auction mechanism proposed in [14].

Our study of deadline constrained file transfers also provides insights on managing QoS for other applications. For example, stored video delivery can be viewed as sequentially transferring a number of subfiles (in HTTP-based video streaming they are referred to as the video segments), where each subfile has to be delivered within a deadline to avoid rebuffering. By setting proper deadlines for these subfiles, one can manage users’ video quality of experience. Similar arguments can be made for other types of applications in D2D networks by setting appropriate deadlines on file transfers.

## 3.5 Performance Evaluation

We ran simulations of our proposed A-FlashLinQ and A-SCSMA schedulers for dynamic D2D networks using Matlab. D2D links are initiated according to a homogeneous spatio-temporal Poisson point process with intensity $\lambda$ arrivals/($\text{m}^2 \cdot$second) in a 300m $\times$ 300m region. The length of each link may be different, according to a specified distribution but the orientation is rotationally invariant. Each link is set up to mediate a file transfer from a designated transmitter side to a receiver, although this will of course require bidirectional communications (i.e., ACKs) we focus on the direction in which the majority of the data will flow. Once a D2D link is initiated it attempts to transmit its file within a specified deadline. A link leaves the system either when its transfer completes or its deadline expires.

We let both A-FlashLinQ and A-SCSMA operate over a 5 MHz spectrum, and all the links transmit on the same spectrum so simultaneous transmissions will cause interference to each other. We use a transmit power of 20 dBm for each transmitter along with an antenna gain of $-2.5$ dB and a noise figure of 7 dB. The simulation uses a wrap-around model to calculate the distance between any two devices and eliminate edge effects. Based on these distances we model the pathloss between two devices according to the radiowave propagation model suggested in ITU-R Recommendations P. 1411 [16] with an antenna height of 1.5 meters. We do not model fast fading but let each channel experience independent shadowing with a standard deviation of 10 dB. The transmission capacity of a D2D link is determined by the Shannon capacity function taking into account the aggregated interference from other scheduled D2D links.

We test the performance of A-FlashLinQ and A-SCSMA in two cases: (1) homogeneous case where links have homogeneous link distances, file sizes and deadlines; and (2) heterogeneous link distances.

### 3.5.1 Dynamic D2D Networks with Homogeneous Links

We assume each link has a distance 20 meters, file size 150 Mbits and 30 seconds of deadline. For A-FlashLinQ we use the same parameters, e.g., 9dB transmitter and receiver yielding thresholds as in [44]. A link schedule is performed once every 2 ms, i.e., $t_{slot} = 2$ms. The parameters used for grouping in both A-FlashLinQ and A-SCSMA are that $t_{pwnd} = 0.5$ second, $\alpha_l = 0.7$, and $\alpha_h = 2$.

Remember that A-FlashLinQ allows each link to use 3 bits to share their application layer information, of which the first bit indicates whether or not a link is in Group 1, and the other 2 bits represent the link's remaining processing time, which basically requires a quantization function mapping the residual service time into one of 4 intervals. The intervals were chosen to be $[0, 10)$, $[10, 20)$, $[20, 30)$ and $[30, \infty)$ seconds for the links in Group 1, and for those in Group 2 the intervals are $[0, 10)$, $[10, 50)$, $[50, 100)$ and $[100, \infty)$ seconds. For A-FlashLinQ without grouping, all 3 bits are used for the remaining processing time, and thus links need to map their residual processing time into 8 intervals. In the simulations the 8 intervals are $[0, 5)$, $[5, 15)$, $[15, 25)$,$[25, 35)$, $[35, 45)$, $[45, 55)$,$[55, 65)$, and $[65, \infty)$ seconds. The priority list is then constructed based on the 3-bit information as described in Section 3.4.

For A-SCSMA, we let $t_{slot} = 2$ms. We further assume the contention periods are much shorter compared to the transmission periods, i.e., $t_{cp} << t_{tp}$, and thus we can ignore the contention overheads.

A-FlashLinQ and A-SCSMA are simulated separately as two independent systems. The baseline schedulers in the two systems are FlashLinQ and Simplified Slotted CSMA (SCSMA) which are oblivious to application-level information. For FlashLinQ we use the same parameters as A-FlashLinQ except that the priority list is constructed randomly, agnostic of application layer information. For SCSMA, the contention window of each link is fixed as $cw = t_{cp}$.

For both systems, we simulated various arrival rates $\lambda$ varying from $1/(300^2 \times 30)$ to $32/(300^2 \times 30)$ arrivals/($\mathrm{m}^2 \cdot$ second). Note that since the simulation space has an area of $300\mathrm{m}^2$ and each link can stay in the system for no longer than 30 second, an arrival rate of $x/(300^2 \times 30)$ implies there are on average no more than $x$ links simultaneously in the system. For each value of $\lambda$, we simulate both systems for 300 second. We repeat this procedure for 10 iterations and collect average results.

Figure 3.3 exhibits simulation results for FlashLinQ, A-FlashLinQ and A-FlashLinQ without grouping. Figure 3.3(a) shows that as we increase link arrival rate, all the schedulers will at first have increment in the number of file transfer completions, but when the arrival rate exceeds some value (e.g., 20 on the x-axis), the number of completions will drop. This is not surprising because when the system is heavily loaded, each individual link will have reduced throughput and is thus likely to miss its deadline. However A-FlashLinQ is able to achieve a significantly higher number of completions (up to 2.5x increase) than FlashLinQ at high arrival rates, which shows the benefit of

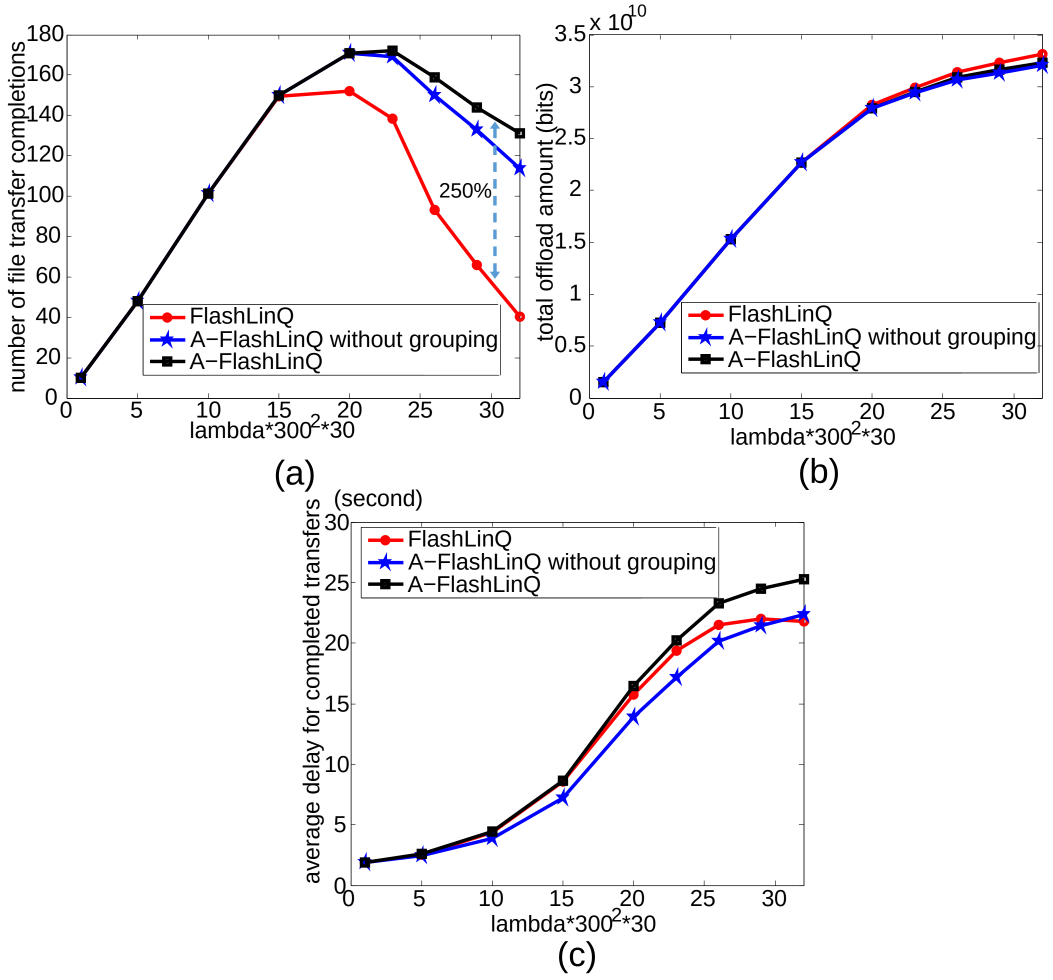


Figure 3.3: Simulation results for A-FlashLinQ under homogeneous scenario. (a) number of file transfer completions vs. link arrival rate; (b) total amount of offloaded data vs. link arrival rate; (c) average file transfer delay for completed transfers vs. link arrival rate.

incorporating application awareness into the scheduler. Also A-FlashLinQ is able to achieve a higher number of completions than A-FlashLinQ without grouping, which exhibits the benefit of knowing the file transfer deadlines.

Figure 3.3(b) shows that all three schedulers achieve basically the same total amount of offloaded data. In fact FlashLinQ itself is able to achieve a high offload. Thus 3.3(a) and (b) suggest that our proposed A-FlashLinQ can greatly improve the number of completions and simultaneously result in an offload amount as good as that in FlashLinQ.

Figure 3.3(c) exhibits the average file transfer delay among the completed file transfers. It is not surprising to see A-FlashLinQ without grouping result in the lowest file transfer delay since it uses an SRPT-like strategy to prioritize the links' transmissions. However note that A-FlashlinQ results in the highest delay. This is because A-FlashLinQ classifies into Group 2 the *loosely constrained* links which are likely to have small remaining processing time, and delay their transmissions by giving them lower priority. This prevents them from finishing early and thus increases the average delay. This result along with that in Figure 3.3(a) shows a tradeoff between number of completions and average transfer delay – with grouping strategy A-FlashLinQ achieves higher number of completions but without grouping it results in lower transfer delay.

Figure 3.4 exhibits the simulation results for SCSMA, A-SCSMA and A-SCSMA without grouping. The results and intuitions are similar to those as in Figure 3.3, which suggests that our application-aware strategy can benefit

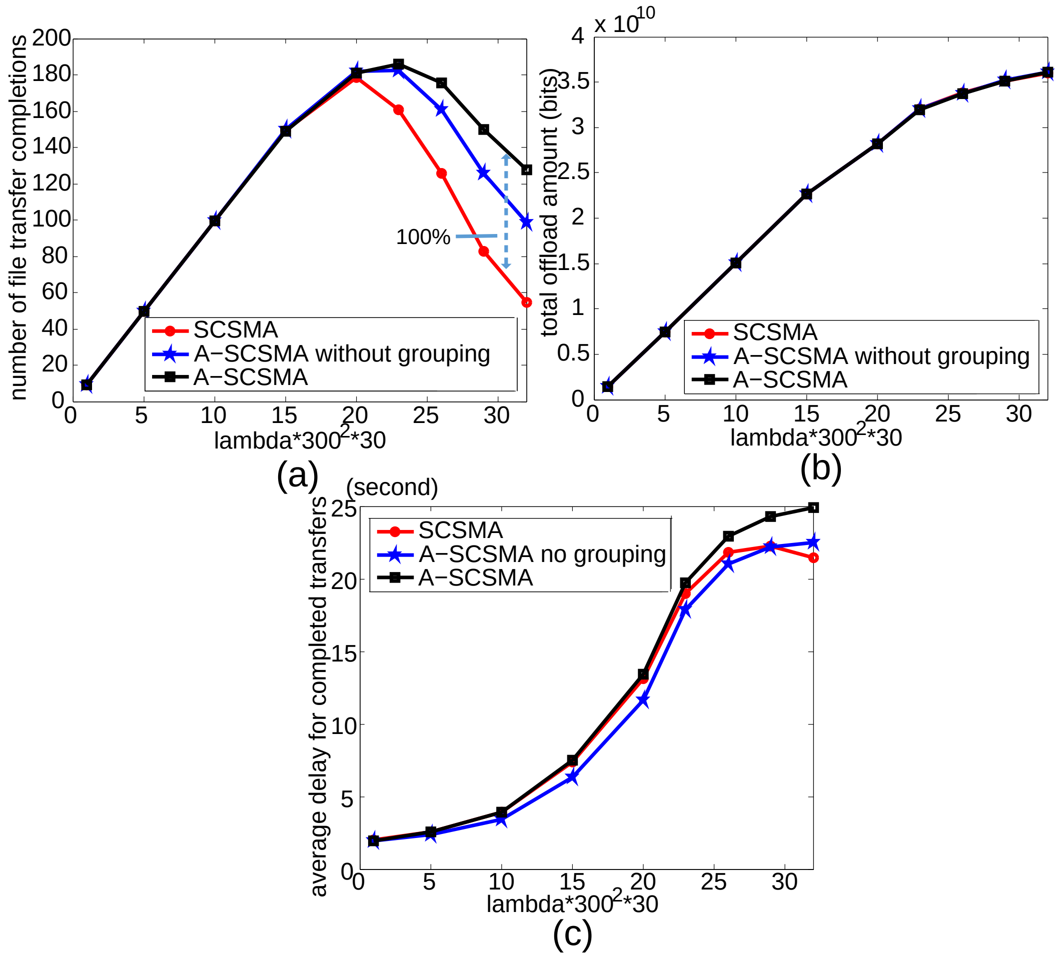


Figure 3.4: Simulation results for A-SCSMA under homogeneous scenario. (a) number of file transfer completions vs. link arrival rate; (b) total amount of offloaded data vs. link arrival rate; (c) average file transfer delay for completed transfers vs. link arrival rate.

not only a D2D system using FlashLinQ scheduler but also one that uses CSMA-like protocols. Thus we expect it to work well in a broad set of D2D systems.

### 3.5.2 Dynamic D2D Networks with Heterogeneous Links

#### 3.5.2.1 Independent File Sizes and Deadlines

We also simulated a heterogeneous case where link lengths are independent and uniformly distributed on $[5, 35]$m. The file sizes associated with the links are generated independently and uniformly on $[20, 1000]$ Mbits. The deadlines are generated independently and uniformly on $[1, 60]$ seconds. The rest of the simulation setup is as described in Section 3.5.1.

Figure 3.5 exhibits simulation results for FlashLinQ, A-FlashLinQ and A-FlashLinQ without grouping. One major difference from the results for homogeneous case (Figure 3.3) is that in Figure 3.5(a), the number of file transfer completions does not collapse when $\lambda \in [20, 60]/(300^2 \times 30)$, as it does in the homogeneous case. This is because in heterogeneous case, when the user density is large, there are likely more 'lucky' links, i.e., those with low link distances, small file sizes and long deadlines, which can complete within their deadlines, and thus increases the number of completions. In other words, heterogeneity brings a kind of "opportunism". Another difference from homogeneous case is that in Figure 3.5(b), A-FlashLinQ (either with or without grouping) results in higher offload amount (up to 15% increase) than FlashLinQ, which along with Figure 3.5(a) suggests that A-FlashLinQ is able to

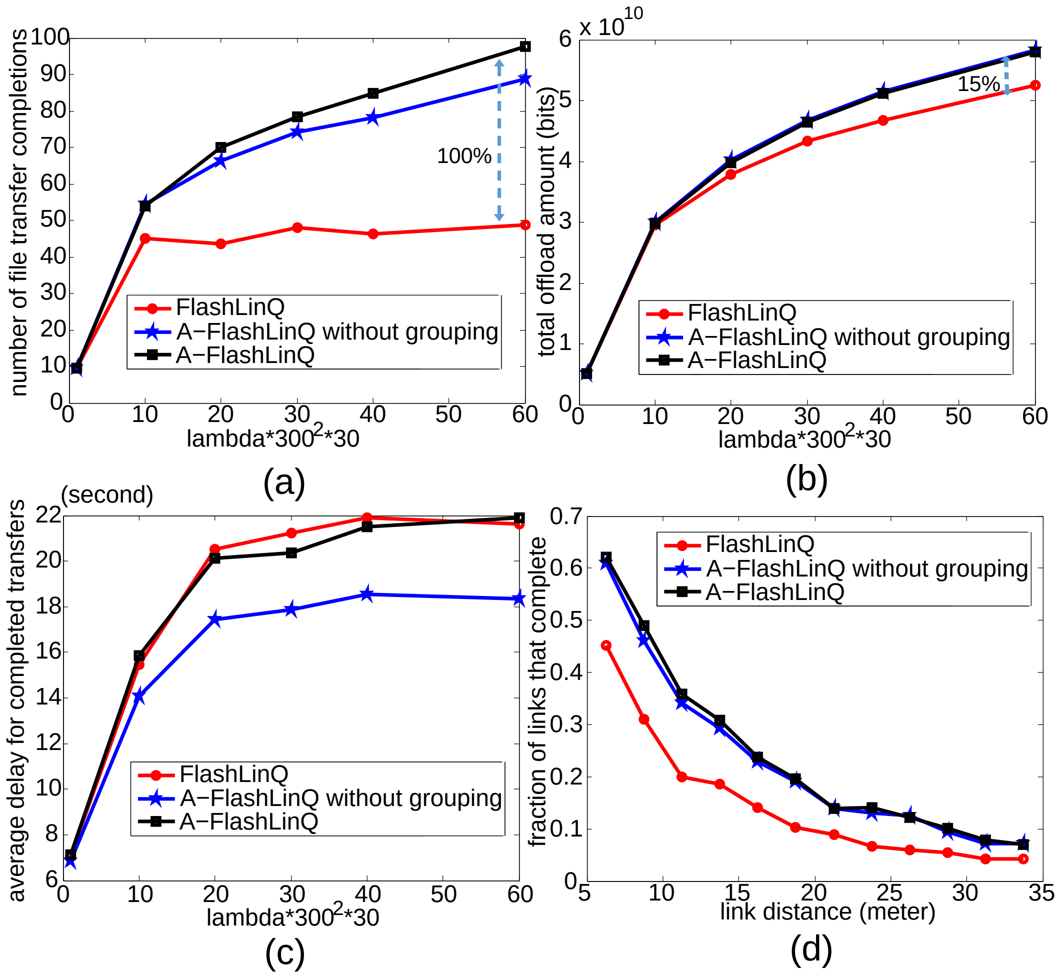


Figure 3.5: Simulation results for A-FlashLinQ under heterogeneous scenario with independent file sizes and deadlines. (a) number of file transfer completions vs. link arrival rate; (b) total amount of offloaded data vs. link arrival rate; (c) average file transfer delay for completed transfers vs. link arrival rate; (d) fraction of links that complete within deadlines vs. link distance.

achieve a significant gain in the number of completions and at the same time has a moderate gain in the offloaded data. Also note that in Figure 3.5(c), A-FlashLinQ results in lower average delay for completed transfers than FlashLinQ, which suggests that A-FlashLinQ dominates FlashlinQ in all the three performance metrics.

Note that A-FlashLinQ schedules links giving higher priority to the ones which have higher transmission rates and/or smaller file sizes. Although it achieves better overall performance than FlashLinQ, it is reasonable to ask whether these gains have been achieved by degrading unfavorable, i.e., long links. To address this issue, we group the links based on link distances, e.g., we put links whose link distances are in $[5, 7.5)$m into one group, and links whose link distances are in $[7.5, 10)$m are put into another group, etc. For each group we calculate the fraction of links that finish their file transfers before their deadlines. Figure 3.5(d) shows that the fraction of completed links for A-FLashLinQ (either with or without grouping) dominates that of FlashLinQ in all groups of link lengths. This is surprising because it implies that, A-FlashLinQ benefits not only the users with high transmission rates, it benefits the users with low transmission rates as well.

Figure 3.6 exhibits simulation results for SCSMA, A-SCSMA and A-SCSMA without grouping in heterogeneous case. The results are similar to those for A-FlashLinQ (i.e., Figure 3.5), except that the gains in total offload amount are smaller and that A-SCSMA with grouping strategy results in a higher average delay than SCSMA and A-SCSMA without grouping strategy.

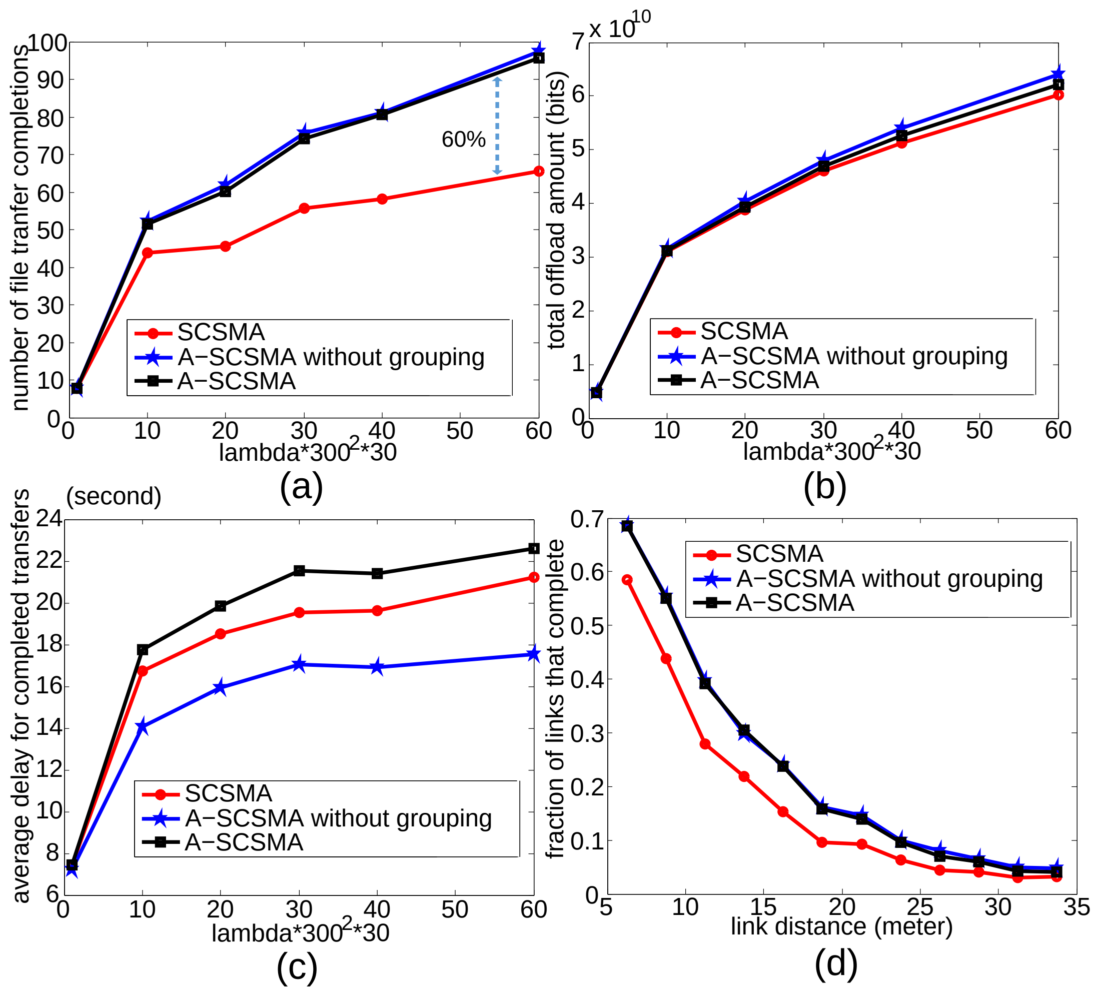


Figure 3.6: Simulation results for A-SCSMA under heterogeneous scenario with independent file sizes and deadlines. (a) number of file transfer completions vs. link arrival rate; (b) total amount of offloaded data vs. link arrival rate; (c) average file transfer delay for completed transfers vs. link arrival rate; (d) fraction of links that complete within deadlines vs. link distance.

In fact as compared to Figure 3.4, we can see that A-SCSMA with grouping strategy performs better in the homogeneous case, while A-SCSMA without grouping strategy is more suitable for the heterogeneous case.

#### 3.5.2.2 Correlated File Sizes and Deadlines

We have investigated the case where the file sizes and deadlines are independent. However in practice, there are usage scenarios where the file sizes and deadlines may be correlated. For example, a link with file size $s$ to transmit may be assigned a deadline $d = s/r$ if a minimum throughput of $r$ is desired. Another example is one where users have a priori knowledge of their contact times, and thus may choose to transfer files of "compatible" sizes.

We simulate the correlated case where file sizes $s_i$ associated with the links $i = 1, 2, ...$ are generated independently and uniformly on $[20, 1000]$ Mbits, and deadlines $d_i$ are set to $d_i = 6 \times 10^{-8} s_i$ second. Other simulation settings are the same as the previous case.

Figure 3.7 and 3.8 exhibits simulation results for the correlated case. Most of the results are similar to the independent case except those in Figure 3.7(c), where the average delay for FlashLinQ drops with increasing load. This is due to the fact that under FlashLinQ only a small number of file transfers can finish, and most of the finished transfers are the ones with small file sizes and short deadlines. However A-FlashLinQ is able to finish more transfers including the ones with relatively large file sizes and long deadlines, which results in a higher average delay than FlashLinQ. Also from Figure 3.8

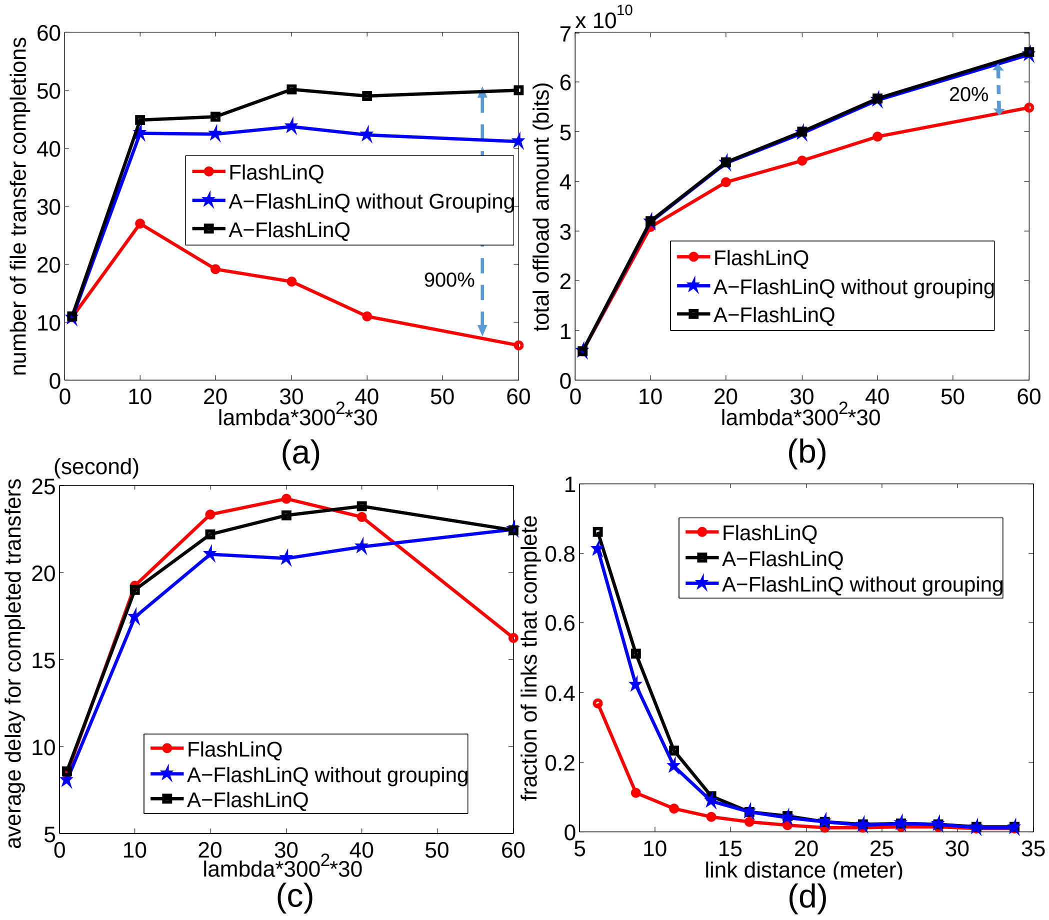


Figure 3.7: Simulation results for A-FlashLinQ under heterogeneous scenario with correlated file sizes and deadlines. (a) number of file transfer completions vs. link arrival rate; (b) total amount of offloaded data vs. link arrival rate; (c) average file transfer delay for completed transfers vs. link arrival rate; (d) fraction of links that complete within deadlines vs. link distance.

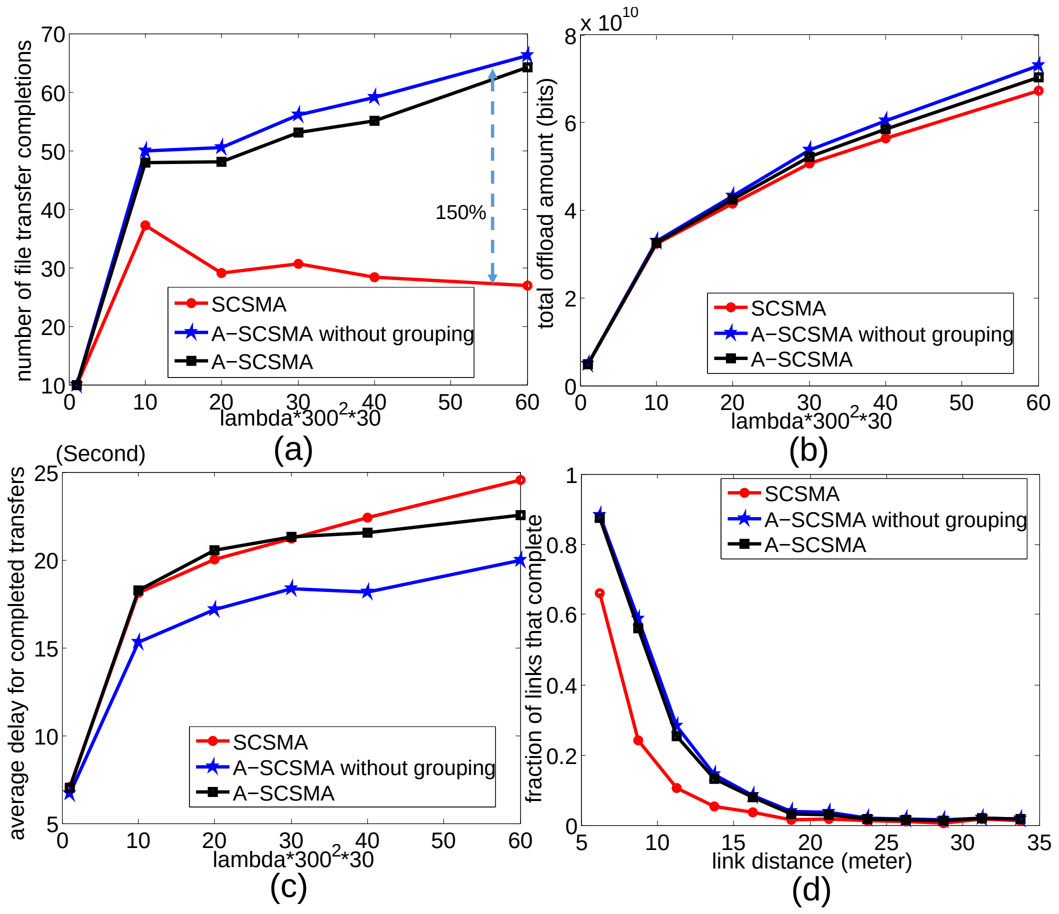


Figure 3.8: Simulation results for A-SCSMA under heterogeneous scenario with correlated file sizes and deadlines. (a) number of file transfer completions vs. link arrival rate; (b) total amount of offloaded data vs. link arrival rate; (c) average file transfer delay for completed transfers vs. link arrival rate; (d) fraction of links that complete within deadlines vs. link distance.

we can see that, A-SCSMA without grouping achieves a better performance than A-SCSMA, which suggests that grouping strategy is not beneficial in this particular scenario.

### 3.5.3 Admission Control

The simulation results shown in the above subsections exhibited that both FlashLinQ and SCSMA have a poor performance in the number of file transfer completions when subjected to high arrival rates. Part of the reason lies in that the schedulers try to maintain a certain degree of throughput fairness while maximizing the spatial packing of links. As a result, when link density is high, each link is able to get some throughput, but few of them have enough throughput to finish their file transfer within their deadlines. This is essentially a problem caused by many links contending for limited resources. Instead of having everyone fail, one might prefer to block some links so that others can succeed. Admission Control (AC) usually plays this role.

Here we propose a simple admission control in which each newly initiated link monitors its throughput for 0.5 second, and then decides whether to quit (i.e., be blocked) or stay in the system. This is a distributed admission control policy akin to that developed in [19]. Specifically, a link with file size $s$, deadline $d$ and average throughput $\bar{r}$ during the initial 0.5 second makes its decision based on the following criterion: the link is

- *admitted*, if $\bar{r} \geq s/d$;
- *blocked*, if $\bar{r} < s/d$.

To test the performance of the proposed admission control strategy, we simulate FlashLinQ and A-FlashLinQ, both with and without the proposed admission control, under the correlated case in the heterogeneous scenario introduced in Section 3.5.2 (Note the results in the other scenarios are similar, so we do not show them here). The results are shown in Figure 3.9, which exhibits that FlashLinQ with Admission Control (AC) can achieve a significant improvement in the number of file transfer completions, and also reduction in average delay among the completed transfers. Moreover with admission control, additional improvements can be obtained by applying application-aware schedulers, as indicated by the superiority of A-FlashLinQ+AC over the other schemes in Figure 3.9(a) and (c). However since a number of links are blocked by admission control and they hardly get any throughput, the system does not see any improvement in the total amount of offload under admission control.

### 3.5.4 Dynamic D2D Networks without Link Deadlines

Last we considered a scenario where the D2D links have no deadlines, i.e., the links will not leave the system until they finish their file transfers. Note that such a system will become unstable as we increase its link arrival rate, i.e., as time advances, the number of concurrent links in the system will increase to infinity, and so will the average file transfer delay. Although possibly unstable, this scenario is of interest particularly in terms of evaluating the file transfer delays and evaluating the "capacity" of the dynamic network under various

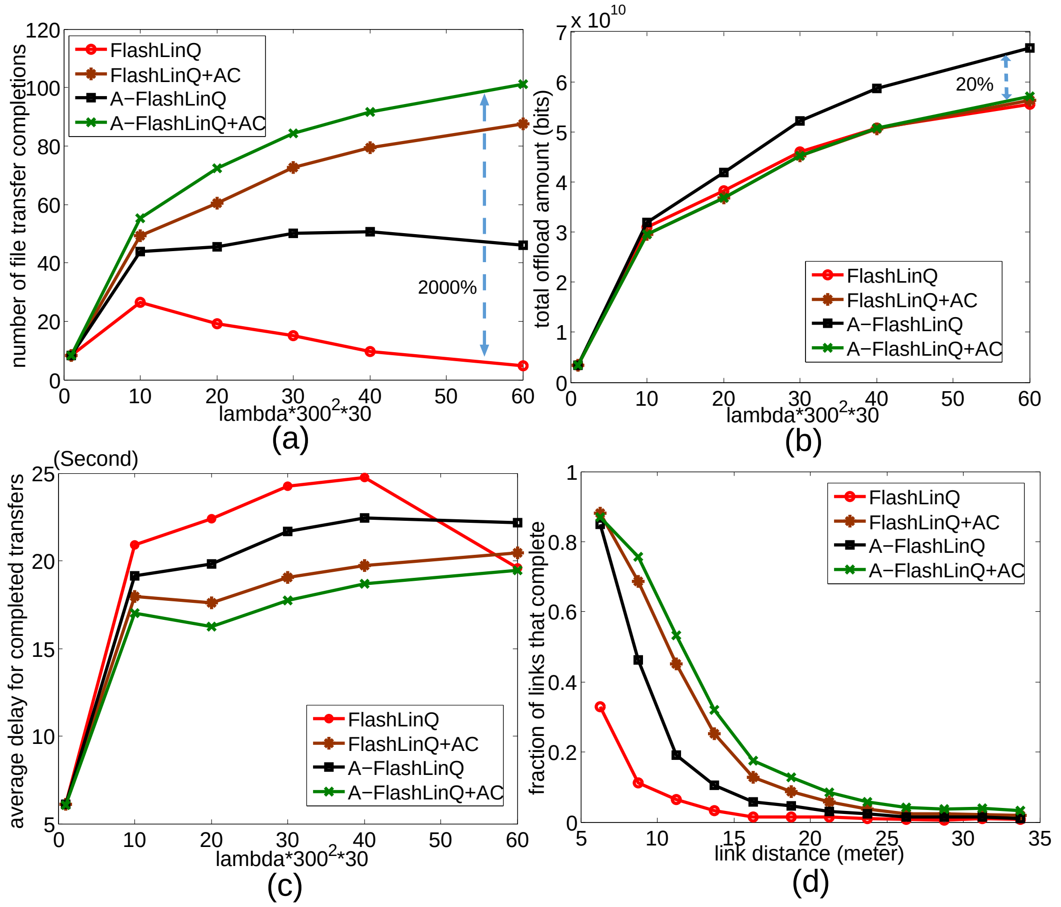


Figure 3.9: Simulation results for Admission Control (AC) strategy under heterogeneous scenario with correlated file sizes and deadlines. (a) number of file transfer completions vs. link arrival rate; (b) total amount of offloaded data vs. link arrival rate; (c) average file transfer delay for completed transfers vs. link arrival rate; (d) fraction of links that complete within deadlines vs. link distance.

schedulers, i.e., what is the maximal spatial arrival rate at which the system becomes unstable.

It is hard to simulate such system if unstable because there is no good stopping point, e.g., if we stop simulation at 300 second as we did in the previous subsections, most of the links that finished their file transfers may be the 'fortunate' ones, i.e., links that have small file sizes and good transmission capacities, and thus if we collect the average file transfer delay for the successful transfers, the result is likely to be biased. In our simulations, we approximate the system by stopping the link arrival process after a certain time $\tau_a(\lambda) = 30\lambda \times 300^2$second, and stopping the simulation when all the links complete their transfers. Note that this approach guarantees all links will finish at the end. However the system is always stable, so we enforce longer simulation times for large arrival rates by letting $\tau_a(\lambda)$ be proportional to $\lambda$, in hopes that this would capture the system's instability, if present.

We simulated FLashLinQ and A-FlashLinQ without grouping for such networks. Note that since links do not have deadlines, we did not simulate A-FlashLinQ with grouping. Other unmentioned system parameters are the same as those used in Section 3.5.2.

Figure 3.10 exhibits the average file transfer delay vs. link arrival rate $\lambda$. A-FlashLinQ achieves a lower average file transfer delay and appears to have a larger stability region.

A phenomenon we conjectured and then observed via simulation was

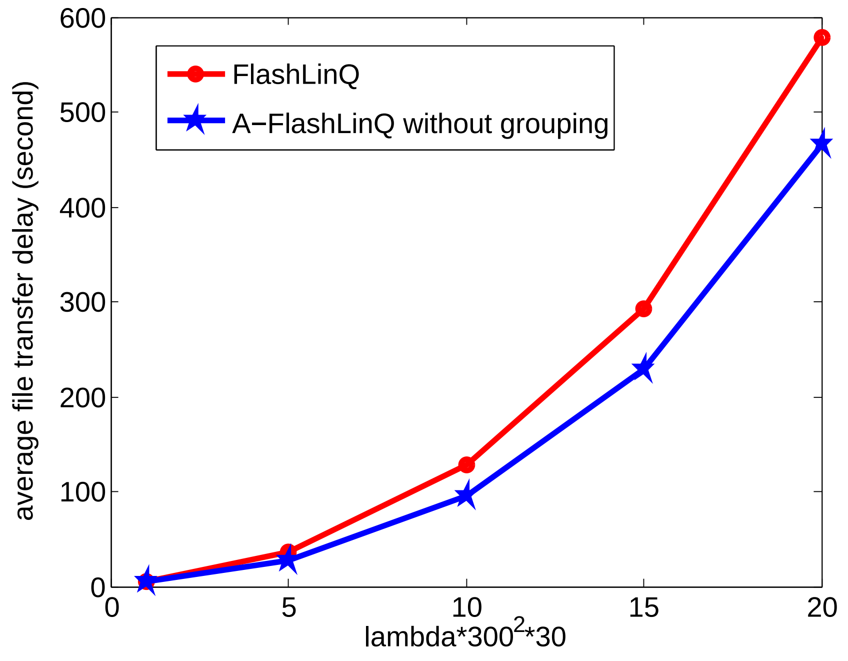


Figure 3.10: Average file transfer delay vs. link arrival rate in the scenario where links have no deadlines.

that in a dynamic network scenario the active links are likely to cluster, even though the arrival process is a homogeneous Poisson point process. An example of the clustering phenomenon was shown in Figure 3.1(b), which suggests the emergence of increased link clustering which may result in poorer performance.

To formally verify the presence of such clustering we use the Ripley's $K$ function. A detailed description of Ripley's $K$ function can be found in [10]. In our case, for a given set of $N$ links distributed in a region of area $A$, the estimated $K$ function is given by:

$$\hat{K}(t) = \frac{A}{N} \sum_{i} \sum_{j \neq i} \frac{\mathbf{1}(d_{ij} < t)}{N}, \tag{3.6}$$

where $\mathbf{1}(\cdot)$ is the indicator function and $d_{ij}$ is the distance between the receiver of link $i$ and the transmitter of link $j$. Note that the distance should be calculated according to the wrap-around model. Note that the two-fold summation in Equation (3.6) calculates the empirical expectation of the following quantity: for a randomly chosen receiver, the number of transmitters (excluding the corresponding transmitter of the chosen receiver) within distance $t$ of the chosen receiver.

If the links were uniformly distributed, we would have $\hat{K}(t) \approx \pi t^2$. So if we define an estimated $L$ function as $\hat{L}(t) = \sqrt{\hat{K}(t)/\pi}$, then we have $\hat{L}(t) \approx t$, and one way to measure the spatial clustering is to test the value of $\hat{L}(t) - t$: $\hat{L}(t) - t > 0$ indicates spatial clustering and $\hat{L}(t) - t \leq 0$ indicates spatial regularity. Again more details of this test method can be found in [10].

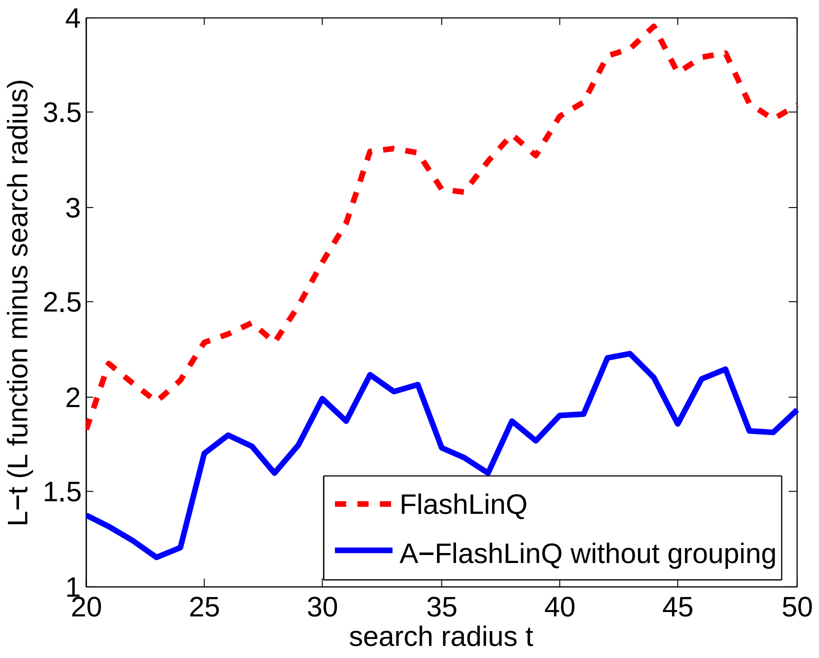


Figure 3.11: L function minus search radius $\hat{L}(t) - t$ vs. search radius $t$. $\hat{L}(t) - t > 0$ indicates spatial clustering and $\hat{L}(t) - t \leq 0$ indicates spatial regularity.

Figure 3.11 shows a plot of $\hat{L}(t) - t$, which is the average value calculated based on the snapshots of the link locations at time 100, 200 and 300 second in 7 independent simulations under an arrival rate of $10/(300^2 \times 30)$ arrivals/($\mathrm{m}^2 \cdot$ second). Note that the fact that the curves are above zero indicates the existence of clustering. However the curve of A-FlashLinQ is below that of FlashLinQ, which implies that our application-aware scheduling mitigates the clustering effect. This may be because A-FlashLinQ speeds up the transmission of links with short remaining processing time and lets them finish faster, resulting in fewer close-by concurrent active links contending for system resources. This suggests another merit of incorporating application-awareness in D2D link scheduling: the potential for mitigating clustering, which improves spatial reuse and reduces overall interference among the links that are close by.

## 3.6 Conclusion

In this chapter we explored the potential benefit of incorporating application-awareness when scheduling deadline constrained file transfers in D2D networks. We propose approaches to modify priorities in FlashLinQ and the contention window mechanism in a CSMA-like protocol so as to increase successful file transfer completions. Our analysis and simulations suggest that if D2D networks are dense this approach has substantial merit towards increasing the number of completions and even provides moderate increase in offload traffic. Moreover, we show that our application-aware schedulers have several other benefits, e.g., enhancing performance of the systems under admission control, increasing system stability region when the links have no deadlines, mitigating links' spatial clustering and thus reduce the interference among close-by links, etc. We expect that D2D schedulers providing more uniformity on user perceived performance metrics, e.g., completions and/or flow-level throughput seen on transfers, and have an important role to play in future.

# Chapter 4

# Conclusion and Future Work

The development of new technologies provides opportunities to make the schedulers in wireless networks “smarter” in two ways. On one hand, HTTP based video delivery protocols and software-defined networking provide the servers, users and network devices with easier access to application layer information, which can be exploited to improve the scheduling policies. On the other hand, by leveraging geolocation and contextual information regarding users mobility patterns it is possible to predict the large-scale wireless capacity variations mobile users are likely to see. In this dissertation we studied how application-awareness and knowledge of future capacity variations can be exploited in network schedulers to improve performance from both the system perspective and the users’ point of view.

We proposed scheduling schemes for video delivery in wireless networks that exploit knowledge of future capacity variations to minimize system utilization without compromising rebuffering/delays. We also proposed application-aware D2D schedulers that improve flow-level performance under link dynamics. Simulation and performance evaluation show that the proposed schedulers can achieve significant gains in various scenarios.

For future work, one might consider generalizing the thresholding policies in Chapter 2 to fit an HTTP based video delivery framework, where optimizing video quality is considered as an important objective along with minimizing system utilization and minimizing rebuffering time. We also believe it would be worthwhile to study the dynamic D2D networks that use mmWave transmissions, where more aspects need to be considered, e.g., directional transmissions, human body absorption, and reflections of mmWaves.

# Bibliography

[1] Cisco visual networking index: Forecast and methodology, 2014-2019. `http://www.cisco.com/c/en/us/solutions/collateral/ns341/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf`.

[2] 3GPP. 3rd generation partnership project; technical specification group radio access network; opportunity driven multiple access. 3GPP TR 25.924, V1.0.0, Dec. 1999.

[3] 3GPP. 3rd generation partnership project; technical specification group services and system aspects; feasibility study for proximmity based services (ProSe) (release 12). 3GPP TR 22.803, V12.2.0, Jun. 2013.

[4] M. Belleschi, G. Fodor, and A Abrardo. Performance analysis of a distributed resource allocation scheme for d2d communications. In *Proc. IEEE GLOBECOM Workshops (GC Wkshps)*, pages 358–362, December 2011.

[5] S. Borst. User-level performance of channel-aware scheduling algorithms in wireless data networks. *IEEE/ACM Trans. on Networking*, 13(3):636–647, June 2005.

[6] N. Bui, S. Valentin, and J. Widmer. Anticipatory quality-resource allocation for multi-user mobile video streaming. In *Proc. the 2nd Workshop*

*on Communication and Networking Techniques for Contemporary Video, in conjunction with the 34th IEEE International Conference on Computer Communications (INFOCOM),*, April 2015.

[7] N. Bui and J. Widmer. Mobile network resource optimization under imperfect prediction. In *Proc. IEEE 16th International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2015.

[8] Tao Chen, G. Charbit, and S. Hakola. Time hopping for device-to-device communication in LTE cellular system. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, April 2010.

[9] G. Van der Auwera and M. Reisslein. Implications of smoothing on statistical multiplexing of H.264/AVC and SVC video streams. *IEEE Trans. on Broadcasting*, 55(3):541–558, September 2009.

[10] Philip M. Dixon. Ripley's K function. In *Encyclopedia of Environmetrics*, volume 3, pages 1796–1803. John Wiley & Sons, Ltd, Chichester, 2002.

[11] Frank H.P. Fitzek, Marcos Katz, and Qi Zhang. Cellular controlled short-range communication for cooperative p2p networking. *Wireless Personal Communications*, 48(1):141–155, 2009.

[12] Matthias Grossglauser and D.N.C. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Trans. on Networking*, 10(4):477–486, August 2002.

[13] Han Han, Hao Wang, and Xiaokang Lin. A low-cost link-selection strategy for cellular controlled short-range communications. In *Proc. IEEE International Conference on Communication Technology (ICCT)*, pages 1332–1335, November 2010.

[14] I. Hou and P. R. Kumar. Utility-optimal scheduling in time-varying wireless networks with delay constraints. In *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 31–40, 2010.

[15] Hung-Yun Hsieh and R. Sivakumar. On using peer-to-peer communication in cellular wireless data networks. *IEEE Trans. on Mobile Computing*, 3(1):57–72, January 2004.

[16] ITU. Recommendation ITU-R P.1411-1 propagation data and prediction methods for the planning of short-range outdoor radio communication systems and radio local area networks in the frequency range 300 MHz to 100 GHz. Tech. Rep., 2001.

[17] J.Rexford and D.Towsley. Smoothing variable-bit-ratevideo in an internetwork. *IEEE/ACM Trans. on Networking*, 7(2):202–215, April 1999.

[18] T. Karagiannis, J.-Y. Le Boudec, and M. Vojnovic. Power law and exponential decay of intercontact times between mobile devices. *IEEE Trans. on Mobile Computing*, 9(10):1377–1390, October 2010.

[19] F.P. Kelly, P.B. Key, and S. Zachary. Distributed admission control. *IEEE Journal on Selected Areas in Communications*, 18(12):2617–2628, December 2000.

[20] Y. Kim, F. Baccelli, and G. de Veciana. Spatial reuse and fairness of Ad Hoc networks with channel-aware CSMA protocols. *IEEE Trans. on Infromation Theory*, 60(7):4139–4157, July 2014.

[21] T. Koskela, S. Hakola, Tao Chen, and J. Lehtomaki. Clustering concept using device-to-device communication in cellular system. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, April 2010.

[22] A. Koubaa, M. Alves, and E. Tovar. A comprehensive simulation study of slotted CSMA/CA for IEEE 802.15.4 wireless sensor networks. In *Proc. 6th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2006.

[23] Lei Lei, Zhangdui Zhong, Chuang Lin, and Xuemin Shen. Operator controlled device-to-device communications in lte-advanced networks. *IEEE Wireless Communications*, 19(3):96–104, June 2012.

[24] Xingqin Lin, Jeffrey G. Andrews, Amitabha Ghosh, and Rapeepat Ratasuk. An overview of 3GPP device-to-device proximity services. *IEEE Communications Magazine*, 52(4):40–48, April 2014.

[25] Z. Lu and G. de Veciana. Application-aware opportunistic D2D link schedulers: Traffic offloading and user perceived QoS. *IEEE/ACM Trans. on Networking*. In submission.

[26] Z. Lu and G. de Veciana. Optimizing stored video delivery for mobile networks: The value of knowing the future. *IEEE/ACM Trans. on Networking*. In submission.

[27] Z. Lu and G. de Veciana. Optimizing stored video delivery for mobile networks: The value of knowing the future. In *Proc. IEEE INFOCOM*, April 2013.

[28] Xiran Ma, Rui Yin, Guanding Yu, and Zhaoyang Zhang. A distributed relay selection method for relay assisted device-to-device communication system. In *Proc. IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, pages 1020–1024, September 2012.

[29] Jean McManus and Keith W. Ross. A dynamic programming methodology for managing prerecorded VBR sources in packet-switched networks. *Telecommunication Systems*, 9:133–152, 1998.

[30] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Cambridge, MA, USA, 1983.

[31] N. Naderializadeh and A. S. Avestimehr. ITLinQ: A new approach for spectrum sharing in device-to-device communication systems. *IEEE Journal on Selected Areas in Communications*, 32(6):1139–1151, June 2014.

[32] S. Patil and G. de Veciana. Managing resources and quality of service in heterogeneous wireless systems exploiting opportunism. *IEEE/ACM Trans. on Networking*, 15(5):1046–58, October 2007.

[33] A Pyattaev, K. Johnsson, S. Andreev, and Y. Koucheryavy. 3GPP lte traffic offloading onto wifi direct. In *Proc. IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 135–140, April 2013.

[34] Qualcomm Research. LTE direct: Always-on, autonomous proximal discovery at scale. `http://www.qualcomm.com/system/files/document/files/lte-direct-proximal-discovery-wireless-networks.pdf`.

[35] M. Reisslein and K. W. Ross. A join-the-shortest queue prefetching protocol for VBR video on demand. In *Proc. IEEE International Conference on Network Protocols (ICNP)*, pages 63–72, October 1997.

[36] Martin Reisslein, Keith Ross, and Vincent Verillotte. A decentralized prefetching protocol for VBR video on demand. In *Multimedia Applications, Services and Techniques-ECMAST'98*, volume 1425 of *Lecture Notes in Computer Science*, pages 388–401. Springer Berlin / Heidelberg, 1998.

[37] J.D. Salehi, Z.L. Zhang, J.F. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. In *Proc. ACM SIGMETRICS*, pages 222–231, May 1996.

[38] S. Shakkottai and R. Srikant. Scheduling real-time traffic with deadlines over a wireless channel. *Wireless Networks*, 8(1):13–26, January 2002.

[39] Upendra Shevade, Yichao Chen, Lili Qiu, Yin Zhang, Vinoth Chandar, Mi Kyung Han, Han Hee Song, and Yousuk Seung. Enabling high-bandwidth vehicular content distribution. In *Proc. ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, number 23, pages 1–12, 2010.

[40] Chaoming Song, Zehui Qu, Nicholas Blumm, and AlbertLaszlo Barabasi. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.

[41] J. Tadrous, A. Eryilmaz, and H. El Gamal. Proactive content download and user demand shaping for data networks. *IEEE/ACM Trans. on Networking*, PP(99), August 2014.

[42] Wi-Fi Alliance Technical Committee, P2P Task Group. Wi-Fi Peer-to-Peer (P2P) technical specification, ver. 1.2. WiFi Alliance, Tech. Rep., 2010.

[43] H. Wu, Chunming Qiao, S. De, and O. Tonguz. Integrated cellular and ad hoc relaying systems: iCAR. *IEEE Journal on Selected Areas in Communications*, 19(10):2105–2115, October 2001.

[44] X. Wu, S. Tavildar, S. Shakkottai, T. Richardson, J. Li, R. Laroia, and A. Jovicic. FlashLinQ: a synchronous distributed scheduler for peer-to-peer ad hoc networks. *IEEE/ACM Trans. on Networking*, 21(4):1015–1228, August 2013.

[45] Shanchieh Jay Yang and G. de Veciana. Enhancing both network and user performance for networks supporting best effort traffic. *IEEE/ACM Trans. on Networking*, 12(2):349–360, April 2004.

[46] Chia-Hao Yu, K. Doppler, C.B. Ribeiro, and O. Tirkkonen. Resource sharing optimization for device-to-device communication underlaying cellular networks. *IEEE Trans. on Wireless Communications*, 10(8):2752–2763, August 2011.

[47] M. Zulhasnine, Changcheng Huang, and A Srinivasan. Efficient resource allocation for device-to-device communication underlaying LTE network. In *Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 368–375, October 2010.

# Vita

Zheng Lu received his B.E. degree in Electronics Engineering from Tsinghua University in China, and his M.S.E. degree in Electrical and Computer Engineering from The University of Texas at Austin in 2011. He has been working under the supervision of Prof. Gustavo de Veciana in the Wireless Networking and Communications Group (WNCG) in the Department of Electrical and Computer Engineering at the University of Texas at Austin since 2010. He interned at Intel Labs, Hillsboro during summer 2013.

Email: zhenglu@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.