Algorithms and Design Principles for Rural Kiosk Networks

by

Shimin Guo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The KioskNet project aims to provide extremely low-cost Internet access to rural kiosks in developing countries, where conventional access technologies, *e.g.*, DSL, CDMA and dial-up, are currently economically infeasible. In the KioskNet architecture, an Internet-based proxy gathers data from the Internet and sends it to a set of edge nodes, called "gateways" from which ferries, such as buses and cars, opportunistically pick up the data using short-range WiFi as they drive past, and deliver it wirelessly to kiosks in remote villages. The first part of this thesis studies the downlink scheduling problem in the context of KioskNet. We pose the following question: assuming knowledge of the bus schedules, when and to which gateway should the proxy send each data bundle so that 1) the bandwidth is shared fairly and 2) given 1), the overall delay is minimized? We show that an existing schedule-aware scheme proposed in the literature, *i.e.*, EDLQ [11], while superficially appearing to perform well, has some inherent limitations which could lead to poor performance in some situations. Moreover, EDLQ does not provide means to enforce desired bandwidth allocations. To remedy these problems, we employ a token-bucket mechanism to enforce fairness and decouple fairness and delay-minimization concerns. We then describe a utility-based scheduling algorithm which repeatedly computes an optimal schedule for all eligible bundles as they come in. We formulate this optimal scheduling problem as a minimum-cost network-flow problem, for which efficient algorithms exist. Through simulations, we show that the proposed scheme performs at least as well as EDLQ in scenarios that favour EDLQ and achieves up to 40% reduction in delay in those that do not. Simulation results also indicate that our scheme is robust against the randomness in actual timing of buses.

The second part of the thesis shares some of our experience with building and testing the software for KioskNet. We subjected a prototype of the KioskNet system, built on top of the DTN reference implementation, to stress tests and were able to identify and fix several software defects which severely limited the performance. From this experience, we abstract some general principles common to software that deals with opportunistic communication.

Acknowledgements

I feel rather fortunate to have had Prof. Keshav as my supervisor throughout my Master's, for all the guidance, encouragement, and tolerance I have received from him. I am especially grateful for the critical way of thinking and scientific methods to approach complex problems he has instilled into my mind, which I can benefit from for the rest of my life.

I would also like to thank Prof. Martin Karsten and Prof. Paul Ward for taking the time to read my thesis and provide me with their feedback.

It has been my privilege to work with my intelligent fellow colleagues in the Tetherless Computing Lab: Adi, David, Nabeel, Omer, Darcy, Majid, Sumair, Earl, and Hossein. Thanks to them for making my days at Waterloo enjoyable ones. I am especially grateful for the early discussions I had with Adi and Majid, which directly led to the conception of this thesis.

I would also like to acknowledge Weihan Wang, a true friend, empathic and dependable, in sharing my happiness and frustration on the harsh road of research.

Last but not least, I would like to thank all members of my family, especially my parents, for their constant love, care, and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Rural Internet kiosks in developing countries provide a variety of services such as birth, marriage, and death certificates, land records, and consulting on medical and agricultural problems. Fundamental to a kiosk's operation, among others, is its connection to the Internet. Unfortunately, most existing access technologies, such as dial-up, VSAT, and GPRS, are either not present in those areas, or too costly given local economic conditions.

Delay-tolerant networking (DTN) has emerged as an attempt to extend the reach of networks. It defines a message-oriented overlay above the transport layer and employs the store-and-forward mechanism to deliver messages from senders to receivers without requiring end-to-end connectivity[8]. DTN opens new possibilities in providing Internet access to remote rural communities. The DakNet [17] project was the first to put the theory into practice, where vehicles with onboard wireless routers were used to carry data back and forth between rural kiosks and Internet hubs in neighbouring cities. Based on the same basic idea, the KioskNet [19] project provides a comprehensive solution for rural kiosk networks, encompassing naming, addressing, routing, user mobility management, application support, and security.

An architecture for rural kiosk networks is proposed in [19]. As depicted in Figure 1.1, the architecture contains four major components: rural kiosks, buses, Internet gateways, and a proxy server. Kiosks are where end users send and receive data. Buses serve as *mechanical backhaul* [19], ferrying data between the kiosks and Internet gateways, using short-range WiFi to download and upload data as they drive past. Internet gateways (or gateways for short), usually located in nearby

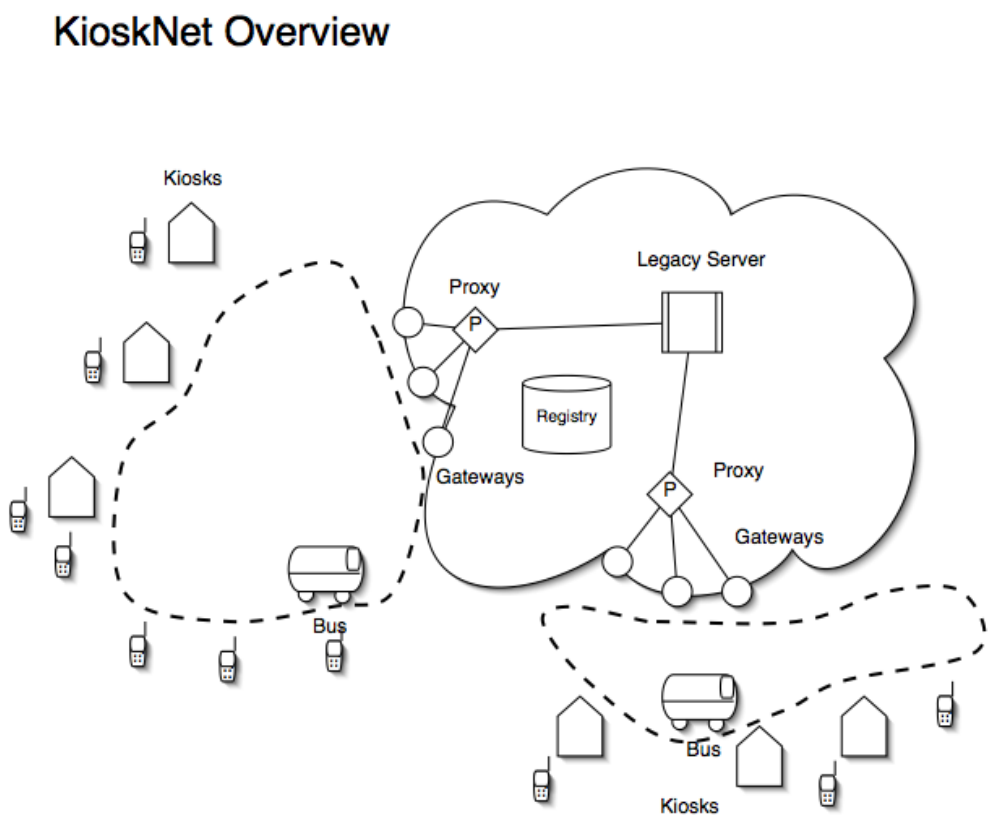


Figure 1.1: An architecture for rural kiosk networks proposed in the KioskNet project.

towns or cities, have persistent Internet connections such as dial-up or DSL, whose job is to forward data to and receive data from the proxy, which is a well-provisioned machine on the Internet. The proxy communicates on behalf of users with legacy servers such as web servers, FTP servers, and mail servers. In the uplink direction (*i.e.*, from kiosks to servers), the proxy receives requests from users and initiates communication with legacy servers. In the downlink direction (*i.e.*, from servers to the kiosks), the proxy buffers data it receives from legacy servers and forwards the data to the gateways, from which the buses opportunistically pick up the data and eventually deliver it to the destination kiosks.

The first part of this thesis studies the downlink scheduling problem in the context of KioskNet. In the downlink direction, the proxy essentially acts as an

application-layer switch. Its incoming (from other servers to the proxy) and outgoing (from the proxy to gateways) links are logical links, typically TCP connections. Data arriving from the Internet to the proxy is fragmented and encapsulated into fixed-sized bundles, which are stored in the proxy's buffer. Whenever an outgoing link becomes free, we say the link presents a *transmission opportunity* to the proxy. The job of the proxy is to choose a bundle from its buffer and transmit the bundle over that link. The downlink scheduling problem asks the following question: assuming knowledge of the bus schedules, when and to which gateway should the proxy send each data bundle so that 1) the bandwidth is shared fairly and 2) given 1), the overall delay is minimized?

By assigning transmission opportunities to bundles, the proxy either explicitly or implicitly selects a gateway for each bundle and decides the order in which different users are served. Existing schemes tend to decouple these two tasks. What they usually do is select an outgoing link immediately after a bundle arrives, enter the bundle into a buffer associated with the chosen link, and apply a scheduling discipline such as FCFS or Round-Robin to each individual link to determine the order in which bundles are served. The best of such schemes to our knowledge is EDLQ [11], which, as we will show, while superficially appearing to work well, suffers from several inherent limitations which could lead to poor performance in some situations. We propose a novel scheme where token buckets are used to ensure fair allocation of bandwidth and where a scheduler repeatedly computes an optimal schedule for all admitted bundles as they arrive. The schedule computed by the scheduler minimizes the total delay of all admitted bundles.

The second part of the thesis shares some experience with building and testing the software for KioskNet. The success of KioskNet depends heavily on the robustness of the software and its ability to deliver high throughput during opportunistic connection periods. We subjected our software to stress tests and were able to identify and fix several performance bottlenecks. From this experience, we abstract some general principles common to software that deals with opportunistic communication.

1.1 Contributions

This thesis makes the following contributions:

1. A novel scheme for the downlink scheduling problem that overcomes the shortcomings of existing schemes.
2. Evaluation of the proposed scheme in both situations where buses follow their schedules precisely and situations where they do not.
3. Principles, abstracted from our experience with building and testing the software for KioskNet, that are common to software that deals with opportunistic communication

1.2 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2 we study the downlink scheduling problem. Chapter 3 shares some experience with building and testing the software for KioskNet. Related work is summarized in Chapter 4. Finally, Chapter 5 concludes the thesis and discusses some potential future research directions.

Chapter 2

The Downlink Scheduling Problem

In this chapter, we study an algorithmic problem that arises in the context of KioskNet, namely downlink scheduling, where the goal is to minimize delay while enforcing desired allocation of bandwidth. The following sections define the system model and objective in a semi-formal manner, provide some insights into this seemingly simple yet subtle and challenging problem by showing how some naïve approaches can lead to unnecessarily long delay, present our solution, and finally evaluate our solution using simulations.

2.1 System Model and Objective

In this section, we define the system model and objective of the downlink scheduling problem in a semi-formal manner.

2.1.1 System Model

Figure 2.1 shows the flow of data and the roles of various components of the system. Data arriving from external servers to the proxy is fragmented and encapsulated into fixed-length *bundles* and stored in the proxy's buffer. We use $arr(b)$ and $dst(b)$ to denote the arrival time and index of the destination kiosk of bundle b,

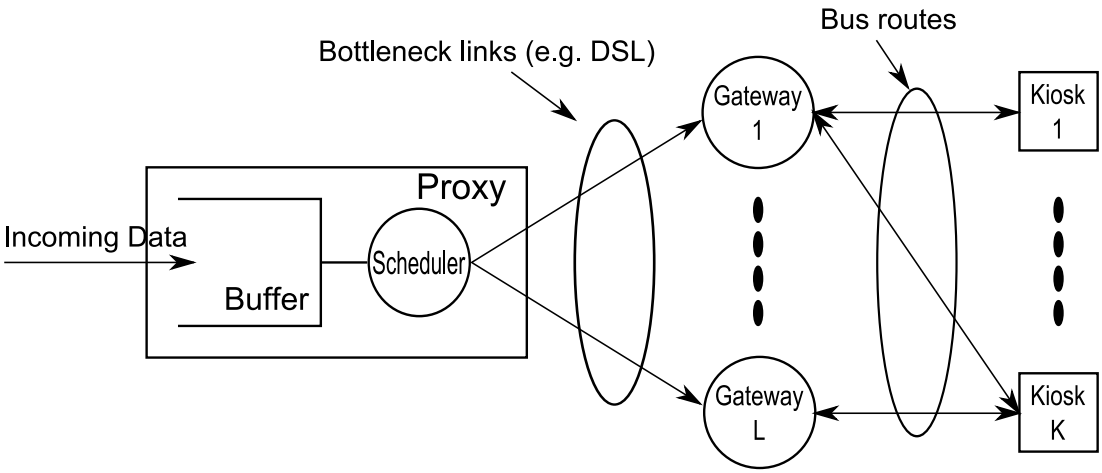


Figure 2.1: System model

respectively. The *delay* of a bundle is measured from the moment it arrives at the proxy to the moment it is delivered to the destination kiosk. A bundle is the unit of scheduling. There is a logical link between the proxy and each gateway — in reality, typically a TCP connection. The proxy is usually hosted in a data centre, provisioned with effectively unlimited inbound and outbound network bandwidth. The gateways, on the other hand, are usually connected to the Internet via DSL, which typically provides a data rate of around 100 Kbps. As a result, the bandwidth of individual links between the proxy and the gateways are limited by the capacity of the DSL subscription the gateways have. In our model, we assume that all proxy-gateway links have the same constant data rate r[1] and that the proxy may communicate with any number of gateways simultaneously. We also assume the latency between the proxy and the gateways is negligible. Because we assume bundles all have a fixed size, it takes a fixed amount of time to transfer a bundle from the proxy to any gateway.

A link is said to be *free* or *idle* when no data is being transmitted over that link, otherwise it is said to be *busy*. Whenever a link becomes free, the scheduler may either select a bundle from the buffer and send the bundle over that link or decide to leave the link idle. Data sent to a gateway is temporarily buffered at the

[1]Our solution only requires the rate of each individual proxy-gateway link to be constant over time, but does not require all the rates to be the same. Allowing the rates to be different only makes the discourse lengthy without offering additional insight.

gateway, waiting to be picked up by a bus that would take it to the destination kiosk.

The gateways and kiosks are connected by bus routes. *Bus schedules* define the start and end times of opportunistic connection windows that happen when a bus passes a gateway or a kiosk. We assume that the schedules of all buses are known to us and that the buses follow their schedules precisely. We further assume, as an approximation, that during an opportunistic connection window, the bandwidth of the wireless link between the two parties is infinite, and that therefore data transfer over wireless links finishes instantly.[2] Given these assumptions, by applying a modified version of Dijkstra's shortest path algorithm [11], we can tell exactly what the earliest possible delivery time of a bundle would be if it were to be sent to a given gateway at a given time. Essentially, the bus schedules define a function $D_{i,j}(t)$ for each kiosk-gateway pair $\langle k_i, g_j \rangle$, which is the delivery time of a bundle destined to kiosk k_i if it were to be sent to gateway g_j at time t. If there are no bus routes from gateway g_j to kiosk k_i, $D_{i,j}(t) = \infty$ for all t.

Note that $D_{i,j}(t)$ is necessarily monotonically increasing. For any two bundles b_1 and b_2 with $dst(b_1) = dst(b_2) = i$, if they are both sent to gateway g_j at time t_1 and t_2, respectively, with $t_1 < t_2$, then b_1 can always be delivered no later than b_2. On the other hand, $D_{i,j}(t)$ is not *strictly* increasing — that is, b_1 may be delivered at the same time as b_2, which is the case if between t_1 and t_2 no bus departs from gateway g_j for kiosk k_i. In fact, $D_{i,j}(t)$ is a step function, an example of which is given in Figure 2.2. Jumps in delivery time occur when buses leave the gateway in question.

2.1.2 Fairness Requirement

Given the unlimited inbound bandwidth of the proxy and the limited capacities of proxy-gateway links, traffic may arrive at the proxy at a higher rate than it can leave the proxy, which, coupled with our assumption that all wireless links have

[2]This assumption is not as strong as it might appear. Even if buses pass a gateway only once a day, it would take less than 3 minutes to transfer, using WiFi at 54 Mbps, the data that the gateway has accumulated at 100 Kbps over the course of a day. Even though the actual throughput of WiFi is well below the nominal rate (around 50% at 54 Mbps), given moderately frequent contact opportunities, the time it would take to transfer all pending data is negligible.

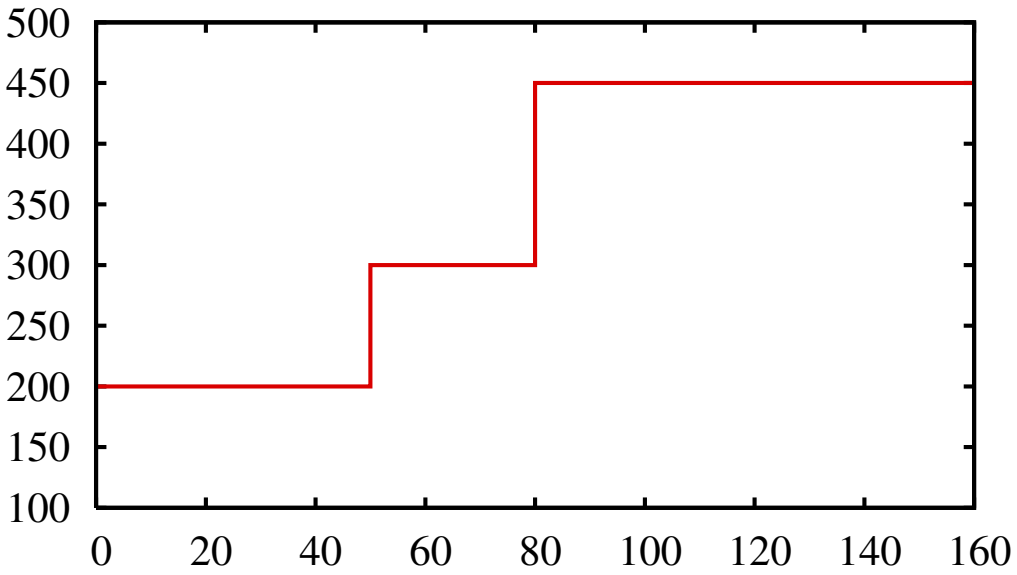


Figure 2.2: An example of step function $D_{i,j}(t)$. The x axis is the time when a bundle is sent to gateway g_j and the y axis the expected time when the bundle is delivered to kiosk k_i.

infinite bandwidth, indicates that the proxy-gateway links are the only bottlenecks of the system. Sharing of the bandwidth of the proxy-gateway links therefore must be regulated in some fashion.

But among whom should the bandwidth be allocated? It is argued in [5] that resources (or cost) should be allocated among economic entities. We consider each kiosk to be an economic entity to which a fair share of bandwidth is allocated. A kiosk owner pays the network provider a certain subscription fee which is directly related to the amount of bandwidth this kiosk gets allocated. The kiosk owner then charges end users who access the network through the kiosk. This provides economic incentives for kiosk owners to subscribe to an appropriate amount of bandwidth according to the sizes of their businesses. In the rest of the chapter we shall use the term "kiosk" interchangeably with the term "user".

During times of congestion, we require that each kiosk be guaranteed the amount of bandwidth allocated to it. A kiosk that downloads excessive amount of traffic that exceeds its allocated rate should not negatively impact the service received by

other kiosks. The measures of quality of service include delay and loss ratio[3].

2.1.3 Objective

The objective of the downlink scheduling problem is to minimize overall bundle delay across all kiosks subject to the fairness requirement described in Section 2.1.2.

2.2 Existing Techniques

Three schemes are proposed in [11] for routing in delay tolerant networks where information of precise future contact schedules is available, namely *Earliest Delivery (ED)*, *Earliest Delivery with Local Queuing (EDLQ)*, and *Earliest Delivery with All Queues (EDAQ)*. We summarize them here.

2.2.1 ED

All three schemes use a modified Dijkstra's algorithm, which can be applied to a graph with time-varying edge costs, to find shortest paths. The simplest of all three, ED, assumes zero queuing delay at any nodes. Using this algorithm, the proxy selects for an incoming bundle a gateway that will result in the earliest delivery, assuming that the bundle can be sent to the gateway immediately. In other words, if the proxy receives a bundle destined to kiosk k_i at time t, it chooses gateway g_{j^*} such that

$$j^* = \arg\min_j D_{i,j}(t).$$

This works well when queuing delays are negligible. In our case, however, the queuing delay at the proxy can be substantial due to the asymmetry in bandwidth of incoming and outgoing channels. The actual time the bundle can be sent to

[3]The reason why there may be losses is not because the buffer size is limited — the fact that nodes store data in their persistent storage means that there is effectively unlimited storage space. The real reason for losses is that if there is a sustained overload, dropping bundles is the only way to keep the system stable.

gateway g_{j^*} is $t + \Delta_{j^*}$, where Δ_{j^*} is queuing delay before the bundle can be sent to gateway g_{j^*}, and there may well be another gateway $g_{j^\diamond}$ such that

$$D_{i,j^\diamond}(t + \Delta_{j^\diamond}) < D_{i,j^*}(t + \Delta_{j^*}).$$

Then gateway $g_{j^\diamond}$ will be a better choice than gateway g_{j^*}.

2.2.2 EDLQ

EDLQ is superior to ED in that it considers local queuing delay. Compared to ED, it chooses gateway g_{j^*} such that

$$j^* = \arg\min_j D_{i,j}(t + \Delta_j).$$

Note that $D_{i,j}(t + \Delta_j)$ is an accurate estimate of the delivery time if the bundle is to be sent to gateway g_j. Therefore, EDLQ is able to find a best gateway for the incoming bundle, *given there is a way to compute Δ_j at the time the bundle arrives.* This is only possible when bundles for which the same gateway is chosen are served in FIFO order. To compute Δ_j then, we just need to see how many bundles are already in the queue for gateway g_j and multiply the number by $1/r$, the time it takes to transmit one bundle. In contrast, ED does not dictate the order in which bundles with the same gateway choice are served.

2.2.3 EDAQ

EDAQ takes another step forward by considering not only local queuing delay, but all queuing delay along a path. Under our assumption that the bandwidth of all wireless links is infinite, however, queuing delay only occurs at the proxy. Therefore, EDLQ and EDAQ is equivalent in our system model.

2.3 What is Wrong with EDLQ

At a first sight, EDLQ appears to be a viable solution to our problem. It is able to accurately estimate delivery times and always chooses gateways that will result in the earliest delivery. When the load is high, it is able to respond to the increase

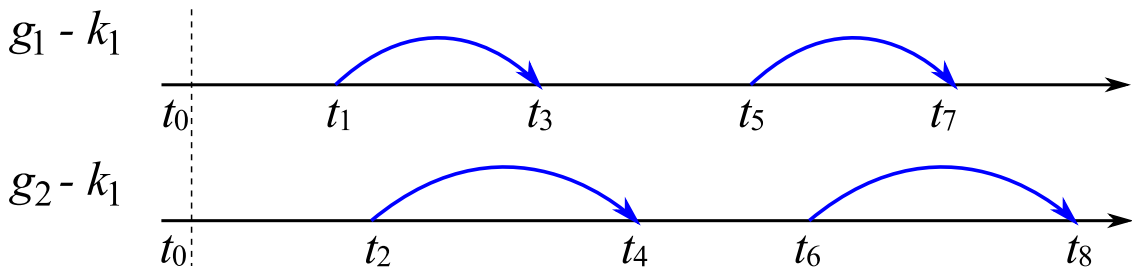


Figure 2.3: A scenario where EDLQ performs well. The top axis shows the schedule of buses from gateway g_1 to kiosk k_1 and the bottom one shows the schedule of buses from gateway g_2 to kiosk k_1. Each arc represents one bus trip, with the tail of the arc indicating the time when the bus leaves the gateway and the head indicating the time when the bus arrives at the kiosk. (Same graphical representation of bus schedules used in other figures)

in the local queue length by spreading the load across multiple links. Consider the scenario shown in Figure 2.3 which involves one kiosk and two gateways. Suppose initially the queues for both gateways are empty and that at time t_0, the proxy receives a batch of bundles destined to kiosk k_1. Using ED, the proxy will place all the bundles in the queue for gateway g_1. If not all bundles can be sent to gateway g_1 before t_1, those that are sent after t_1 can only be delivered at time t_7, while leaving gateway g_2 totally unused. On the other hand, if EDLQ is used, after putting a certain number of bundles in the queue for gateway g_1, the proxy will realize that any bundles put in g_1's queue afterwards would not be sent before t_1, and therefore will place the rest of the bundles in gateway g_2's queue, thereby achieving the shortest possible delay.

It may seem the problem is solved, but a closer study reveals three limitations of EDLQ which could lead to poor performance in some situations. The first — and most obvious — is that EDLQ cannot guarantee fair allocation of bandwidth. The fact that EDLQ relies on bundles being served in FIFO order allows a kiosk that requests an excessive amount of data to dominate the usage of bandwidth. One could argue that EDLQ is just a routing algorithm and is not charged with providing fairness in the first place, and that therefore it is not a problem with EDLQ itself. However, the point we are making here is that at least EDLQ alone is not sufficient to serve our purposes.

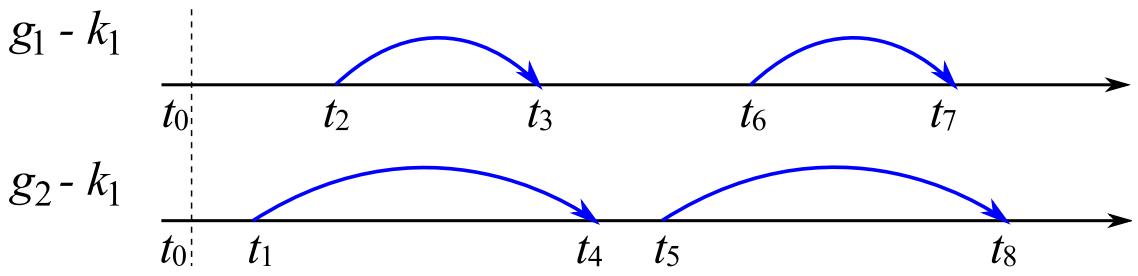


Figure 2.4: A scenario where EDLQ performs poorly due to its greedy nature.

The second problem with EDLQ stems from the greedy nature of EDLQ. Although as shown before it has the ability to switch to a secondary path when there is load on the primary path, sometimes such actions come too late. Consider the scenario shown in Figure 2.4 which is similar to the scenario shown in Figure 2.3. The only difference is that now the buses that go from gateway g_2 to kiosk k_1 leave g_2 earlier than they do in the previous scenario. Now let's suppose the proxy receives a batch of bundles at time t_0. It finds that there is just enough time to send all the bundles to gateway g_1 before t_2, so it puts all the bundles in the queue for g_1. Shortly after t_1, the proxy receives another batch. No bundle from the second batch can be sent to g_1 before t_2 because the bundles from the first batch already occupied all the slots before t_2. At this time, even if we send these bundles to gateway g_2, we cannot expect them to be delivered at t_4, since the bus that will arrive at kiosk k_1 at t_4 has already left. As a result, the delivery time of the second batch can only be t_7 or later. Had we sent some bundles from the first batch to g_2 — which delays their delivery slightly, from t_3 to t_4 — we would have saved some slots for the second batch, which could bring forward the delivery time of at least some bundles from the second batch from t_7 to t_3, a significant reduction in delay. The reason why EDLQ fails to be optimal in this case is because EDLQ chooses gateways greedily, with no regard to the fact that a path that is only slightly worse may disappear soon and the next path in line may be considerably worse.

Finally, the third problem with EDLQ lies with the its inability to reorder bundles. Consider the scenario shown in Figure 2.5. $t_0, t_1, \ldots, t_{10}$ are evenly spaced, with $t_{i+1} - t_i = \delta$. In this paragraph we shall use the term "batch" to refer to the set of bundles that would take time δ to be transferred from the proxy to a gateway. Suppose the proxy receives two batches for kiosk k_1 at time t_0, and one batch for kiosk k_i at time t_{i-1}, $i = 2, 3, 4$. Using EDLQ which serves bundles in the order of

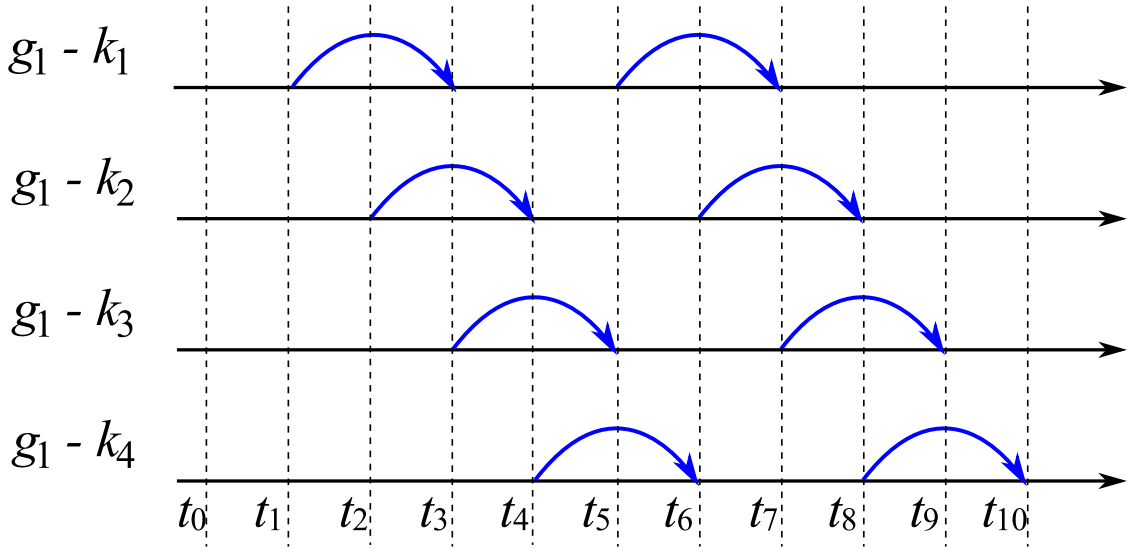


Figure 2.5: A scenario where EDLQ performs poorly due to its inability to reorder bundles.

arrival, the first batch for kiosk k_1 will be transmitted between t_0 and t_1, and the second batch between t_1 and t_2. The batch for kiosk k_i will be transmitted between t_i and t_{i+1}, $i = 2, 3, 4$. As a result, the first batch for kiosk k_1 will experience a delay of 3δ, while all the other batches will experience a delay of 7δ. The optimal scheduling in this case is to send the first batch for kiosk k_1 between t_0 and t_1, the batch for kiosk k_i between t_{i-1} and t_i, $i = 2, 3, 4$, and finally send the second batch for kiosk k_1 between t_4 and t_5. This way, all batches except the second batch for kiosk k_1 will experience a delay of 3δ, and the second batch for kiosk k_1 will experience a delay of 7δ. Compared to the optimal scheduling, EDLQ more than doubles the delay of three of the five batches. It is not hard to see that, by adding more kiosks, we can construct scenarios where EDLQ performs *arbitrarily* badly compared to the optimal scheduling.

2.4 Our Solution

In this section we present our utility-based approach to downlink scheduling in rural kiosk networks. To ensure fair bandwidth allocation, we use a token-bucket traffic regulator for each kiosk, where the token arriving rates reflect the allocated

bandwidth. We associate some utility with every bundle transmitted, which captures the "value" of sending a bundle based on the delay it will experience. Every time new bundles are admitted by the token-bucket regulators, the scheduler is invoked to compute a bundle transmission schedule[4] for all admitted bundles — which determines which bundle should be sent to which gateway at what time — that would maximize the total utility. The subsequent subsections describes our solution in more detail.

2.4.1 Token-Bucket Traffic Regulator

A token-bucket traffic regulator (or TB regulator for short)has two parameters: filling rate σ and depth d. A token is added to the bucket every $1/\sigma$ time units, unless the bucket already has d tokens — in which case the token is discarded. Traffic destined to each kiosk is subject to a separate TB regulator. When a bundle b destined to kiosk k_i arrives, if the bucket associated with k_i is not empty, one token is removed from the bucket and b becomes *eligible*. Eligible bundles are buffered in per-kiosk post-bucket queues. If the bucket is empty, b will be temporarily buffered in a pre-bucket queue. The pre-bucket queue has a fixed capacity. If the pre-bucket queue is full, new arriving bundles will be dropped. An illustration of a proxy with TB regulators is given in Figure 2.6.

The filling rate controls how fast bundles can pass the regulator in the long run. To meet the fairness requirement, we just need to set the filling rate of kiosk k_i's token bucket according to kiosk k_i's allocated rate. To be able to provide bandwidth guarantees at all times, the sum of rates allocated to all kiosks is not allowed to exceed the capacity of the system, *i.e.*, no over-booking is permitted, which means that bundles cannot become eligible faster than they can leave the proxy and once a bundle becomes eligible, it can be guaranteed to be delivered in finite time. A well-behaving kiosk, which does not request data faster than its allocated rate, will have all the requested data delivered to it. An ill-behaving kiosk, on the other hand, will eventually fill up its pre-bucket queue and experience losses.

The use of TB regulators decouples fairness and delay minimization. The scheduler only considers the set of eligible bundles and focuses solely on delay minimiza-

[4]This schedule should not be confused with bus schedules. The meaning of the term "schedule" should be clear from its context.

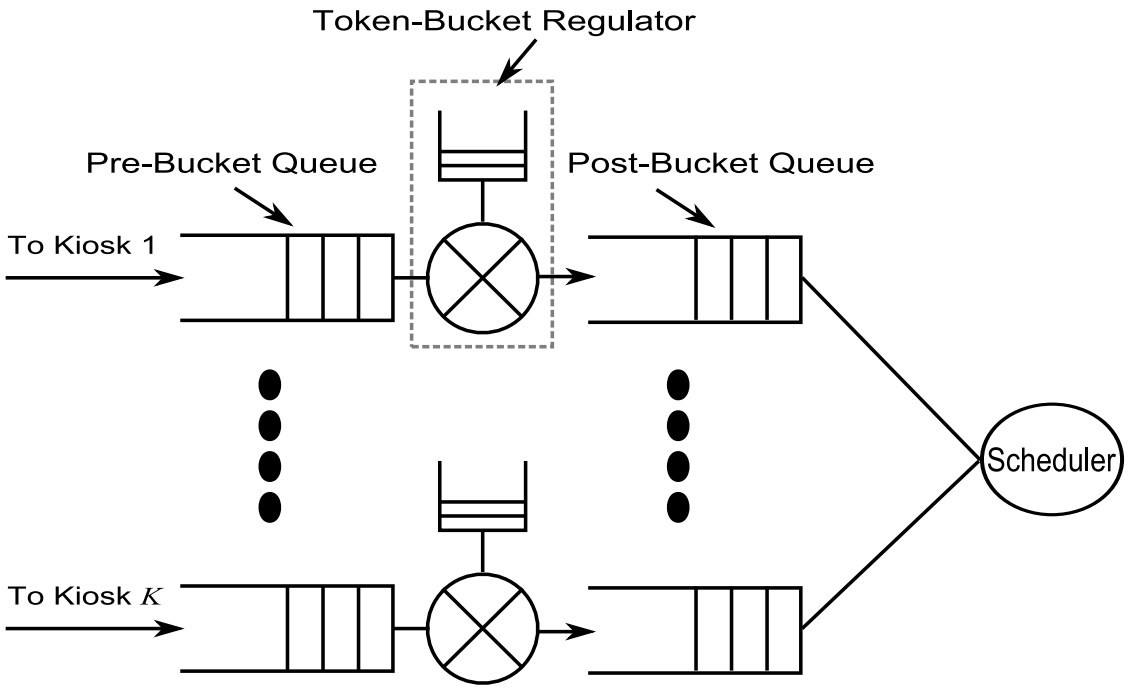


Figure 2.6: Token Bucket

tion, without any regard to fairness. It significantly simplifies the design of the scheduler as compared to one that has to concern itself with *both* fairness and delay minimization.

The limitation of TB regulators is that they are sometimes too conservative. There may be times when bundles are blocked by the regulators while the links sit idle. One work-around to this problem is using a large value for bucket depth. A deep bucket does not start throttling traffic until the traffic has been over limit for a sufficiently long time, so the likelihood of simultaneous occurrence of bundles being blocked and links sitting idle is smaller. Another solution is to bypass the TB regulators when the system is underloaded, and enable them only when there are already a certain number of eligible bundles. This way, we can ensure that there is always some work to keep the links busy.

2.4.2 Utility Model

Intuitively, the usefulness of a bundle depends on the delay it has experienced and the time at which it is delivered. We define a utility function $U(x, y)$ that captures

that usefulness of a bundle that has a delay of x and is delivered at time y. The function can be different for different kiosks to reflect their different preferences. We use $U_i(x, y)$ to denote the utility function of kiosk k_i and $W_{i,j}(n, t)$ to denote the utility of sending the nth bundle from kiosk k_i's post-bucket queue (whose arrival time we denote as $arr_i(n)$) at time t to gateway g_j. It is easy to see that

$$W_{i,j}(n, t) = U_i(D_{i,j}(t) - arr_i(n), D_{i,j}(t))$$

Recall that $D_{i,j}(t)$ is the delivery time of a bundle destined to kiosk k_i if it were to be sent to gateway g_j at time t.

This definition of utility functions is quite versatile at expressing various optimization objectives. However, it makes it a hard problem to compute a schedule that maximizes the total utility, because each single bundle must be treated individually. To make the optimization problem tractable, we define the utility function based on *remaining delay*, that is, the time remaining from the scheduling instant s until the delivery time of the bundle. Denoting the utility function of kiosk k_i as $U_i(x)$ where x is the remaining delay, we have

$$W^s_{i,j}(n, t) = U_i(D_{i,j}(t) - s)$$

where s is the time of the scheduling instant and the superscript of $W^s_{i,j}(n, t)$ signifies that it is a time-varying function. We can see that in this definition $W^s_{i,j}(n, t)$ no longer depends on n. In other words, all bundles belonging to the same kiosks are equivalent, hence significantly reduced search space. Since $W^s_{i,j}(n, t)$ no longer depends on n, we will write it as $W^s_{i,j}(t)$.

It is easy to see that if we define $U_i(x)$ to be simply $-x$, the opposite of the remaining delay, then a schedule that maximizes the total utility, therefore minimizing the total remaining delay, will minimize the total end-to-end delay for all eligible bundles. It should be noted that such a schedule is not necessarily one that eventually minimizes the total delay of *all* bundles, which would require knowledge of future traffic arrival. We refer to schedules that minimizes the total delay of all bundles as being *globally* optimal, and ones that minimizes the total delay of bundle that are eligible at the time they are computed as being *locally* optimal. It is impossible for an online algorithm to deterministically compute globally optimal schedules.

2.4.3 Scheduling

Every time new bundles become eligible, the scheduler computes a schedule that would maximize the total utility gained from sending all the bundles that are currently eligible.

We divide the time into slots and the length of each slot is the time it takes to transmit a bundle over a proxy-gateway link. We refer to the combination of time slot h (which finishes at t_h) and gateway g_j as an *transmission opportunity* p_{jh}. A schedule assigns transmission opportunities to bundles. We formulate the optimal scheduling problem as a *minimum-cost network-flow* problem [1]. We first describe a basic formulation, then show how we can reduce the input size using a more efficient formulation exploiting the fact that $D_{i,j}(t)$ is a step function, and finally discuss some subtle issues involved in making scheduling decisions.

A Basic Formulation

In a minimum-cost network-flow problem, there are some nodes with certain units of supply of goods, some nodes with certain units of demand, and some relay nodes. There are arcs connecting these nodes, each with a capacity and a unit cost. The goal is find a way to transport goods from supplying nodes to demanding nodes that incurs the least cost while honouring the capacities of all links. Many assignment problems can be formulated as network flow problems. Since scheduling is essentially assigning bundles to transmission opportunities, network flow is a natural tool to solve the optimal scheduling problem.

Formally, we create a directed bipartite graph $G = (\mathcal{N}_s + \mathcal{N}_d, \mathcal{A})$. For every kiosk k_i, we add a node s_i to $\mathcal{N}_s$ and associate with it a number $S(i)$ indicating the number of eligible bundles destined to kiosk k_i, which corresponds to the number of units of "supply" node s_i has. For each transmission opportunity p_{jh}, we add a node d_{jh} to $\mathcal{N}_d$, each of which "demands" one unit of supply. We create an arc from node $s_i \in \mathcal{N}_s$ to node $d_{jh} \in \mathcal{N}_d$ if a bundle from kiosk k_i may be sent to gateway g_j in time slot h. Each edge $(s_i, d_{jh}) \in \mathcal{A}$ has a capacity $w_{ijh} = 1$ and a cost $c_{ijh} = -W_{i,j}(t_h)$. The optimal scheduling problem can be stated as follows:

$$\text{Minimize} \quad z(x) = \sum_{(s_i, d_{jh}) \in \mathcal{A}} c_{ijh} x_{ijh}$$

subject to

$$\sum_{\{d_{jh}:(s_i,d_{jh})\in\mathcal{A}\}} x_{ijh} = S(i) \quad \text{for all } s_i \in \mathcal{N}_s,$$

$$\sum_{\{s_i:(s_i,d_{jh})\in\mathcal{A}\}} x_{ijh} \leq 1 \quad \text{for all } d_{jh} \in \mathcal{N}_d,$$

$$0 \leq x_{ijh} \leq 1 \quad \text{for all } (s_i, d_{jh}) \in \mathcal{A}.$$

where x is a mapping $f : \mathcal{A} \to \{0, 1\}$, with x_{ijh} indicating whether a bundle destined to kiosk k_i should be assigned to transmission opportunity p_{jh}.

The first constraint ensures all eligible bundles are assigned a slot and the second ensures that at most one bundle is assigned to any transmission opportunity. Although no constraints explicitly require x_{ijh} to be integral, it can be shown that as long as the capacities of all edges are integral, x_{ijh} will also be integral [1].

Many polynomial-time algorithms exist for solving minimum-cost network-flow problems. Let $n = |\mathcal{N}_s + \mathcal{N}_d|$ and $m = |\mathcal{A}|$, the best algorithm known solves the problem in $O((m \log n)(m + n \log n))$ [1].

A More Efficient Formulation

The formulation we just presented requires adding a node for each transmission opportunity. Since we must consider at least as many transmission opportunities as there are eligible bundles, the input size of an instance of the problem is proportional to the number of eligible bundles.

When we look at the created graph, we notice that nodes representing transmission opportunities offered by a given link can be divided into groups within which all nodes, except for representing transmission opportunities at different times, are completely indistinguishable from one another — they are connected to the same set of kiosk nodes by arcs with the same costs. This is due to the fact that $D_{i,j}(t)$ is a step function whose value changes only once in a while. If for $t \in [t_\hbar, t_{\hbar+l}]$, $D_{i,j}(t)$ remains the same for any given i, then the set of nodes $\{d_{jh} : h = \hbar, \ldots, \hbar + l\}$ are equivalent.

Nodes within the same group can be aggregated to form a new node, replacing the old nodes which represent individual transmission opportunities. The demand of the new node, as well as the capacities of arcs that point to the new node, equals

the number of transmission opportunities the new node represents. The costs of all arcs remain the same as before. Solving a minimum network flow problem on this new graph will also give us the solution to the optimal scheduling problem.[5]

Compared to the basic formulation, the input size for the same problem instance is dramatically reduced. Suppose on average $D_{i,j}(t)$ changes every 30 time slots. Using the more efficient formulation, both the number of nodes and the number arcs are reduced by almost a factor of 30.

Instantiating a Schedule Class

A solution returned from the first formulation tells us exactly which transmission opportunity should be granted to which kiosk. That is, however, not the case with the second formulation. Since in the second formulation we aggregate multiple transmission opportunities into one node, a solution returned only tells us, of each group of transmission opportunities, how many should be allocated to each kiosk, but not the exact allocation of each individual transmission opportunities. In fact, a solution does not correspond to a schedule, but rather a *class of schedules*. All schedules consistent with the solution belong to this class of schedules and have the same total utility.

Given a schedule class, the scheduler must pick a specific schedule to execute, which we refer to as "instantiating a schedule class". With the first formulation, the instantiation is implicit and is nothing but an artifact of the specific network flow solver used to solve the problem, which is out of our control. With the second formulation, we are given the opportunity to make more intelligent choices.

One may wonder, if all schedules belonging to the same schedule class have the same utility, why would one be better than another? The answer lies in the fact that we have to compute a new schedule every time new bundles become eligible. Depending on how we instantiate a schedule class, the next time new bundles come in, we may be facing different situations, of which some may be more desirable than others. For instance, consider the scenario shown in Figure 2.7. At time t_0, the scheduler is invoked with one eligible bundle for each kiosk. The bus going to kiosk k_1 is going to leave in 2 time slots and the bus going to kiosk k_2 is going to

[5]But not in exactly the same way as using the basic formulation. See the next subsection.

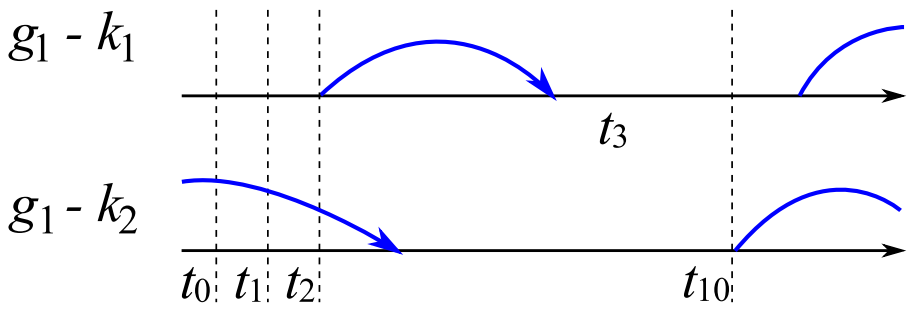


Figure 2.7: A scenario showing the effect of different ways of instantiating a schedule class. At t_0 each kiosk has one eligible bundle. At t_1, one more bundle becomes eligible for kiosk k_1.

leave in 10 time slots. Two groups of transmission opportunities can be formed with the first group containing the first two time slots and the second containing the next 8 time slots. The optimal solution, unsurprisingly, assigns each kiosk one time slot from the first group. Two possible ways to instantiate the schedule class are $S_1 = (\langle k_1, t_1\rangle, \langle k_2, t_2\rangle)$ and $S_2(\langle k_2, t_1\rangle, \langle k_1, t_2\rangle)$. One time step later, a bundle arrives for kiosk k_1. If at t_0 we executed schedule S_2, we would have 2 bundles for kiosk k_1 at t_1 — at least one bundle could not make the bus. If, alternatively, we executed schedule S_1 at t_0, both bundles for kiosk 1 would make the bus, so would the bundle for kiosk 2.

This example suggests that we should incorporate some notion of "urgency" into the decision of instantiating a schedule class. We now present a heuristic based on the notion of "urgency" which is used as a hint for the instantiation of a schedule class.

First, we need to understand what constitutes urgency. Clearly, if a bus is soon going to leave gateway g_j for kiosk k_i, we should schedule bundles for k_i to be transmitted to gateway g_j as early as possible so that if more bundles for kiosk k_i comes before the bus leaves they can still make the bus. So one factor that affects urgency is the time remaining before the next bus departure, and by the same reasoning, the one after that, and so on. What also affects the urgency is the cost of missing a bus, that is the increase in delay as a result of missing a bus. Clearly, the higher the cost, the greater the urgency.

Following the above analysis, we define the "urgency" with which kiosk k_i is in

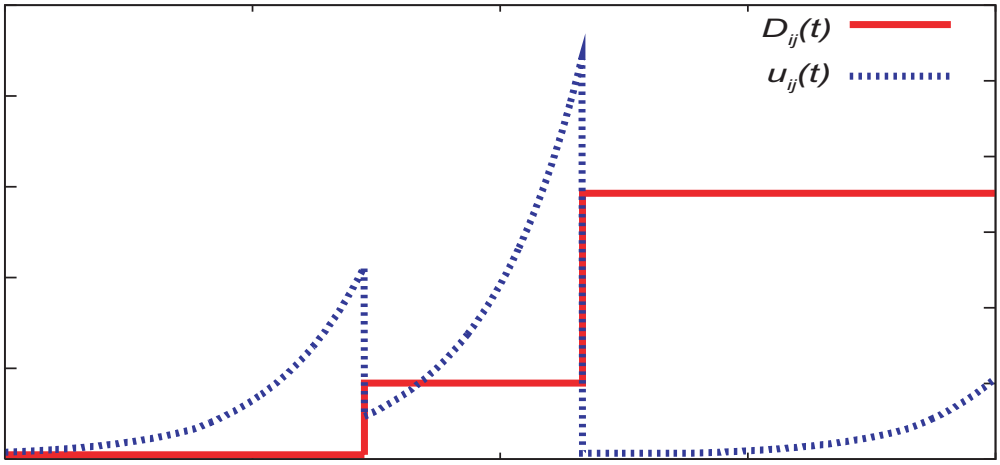


Figure 2.8: An example of urgency function $u_{i,j}(t)$ plotted on top of a delivery time function $D_{i,j}(t)$ from which it is derived.

need of getting bundles to gateway g_j at time t as

$$u_{i,j}(t) = \frac{\int_t^\infty (D_{i,j}(\tau) - D_{i,j}(t))e^{-\gamma(\tau-t)}d\tau}{\int_t^\infty e^{-\gamma(\tau-t)}d\tau} \tag{2.1}$$

As can be seen, $u_{i,j}(t)$ is defined as the weighted integral of the difference between the function $D_{i,j}(\tau)$ and constant $D_{i,j}(t)$ over time from t to infinity, where the weights decay exponentially with time. The γ in the index of the exponential term is called the *discount rate*, which controls the rate at which the weight decreases with time. What determine the value of $u_{i,j}(t)$ are 1) the times at which $D_{i,j}(t)$ increases, and 2) the amplitude of the increases. The more imminent and larger an increase is, the more it contributes to the urgency. Figure 2.8 illustrates an example urgency function plotted on top of a delivery time function from which it is derived.

With urgency defined, when we instantiate a schedule class, if a group of transmission opportunities are allocated to multiple kiosks, we always serve the kiosk with the highest urgency. In other words, we assign transmission opportunity p_{jh} to kiosk k_i such that $u_{i,j}(t_h)$ is the highest among those that have not used up their allocated slots from the current group of transmission opportunities.

Work-Conserving vs. Non-Work-Conserving

Another issue not yet addressed is which set of transmission opportunities should be considered by the scheduler when it computes a schedule. Since an optimal schedule may assign all bundles to the same gateway, it seems that we should let the scheduler consider the next N time slots from each proxy-gateway link, or NL transmission opportunities where N is the number of eligible bundles and L is the number of gateways. The problem of doing so, however, is that the scheduler may choose to leave some links idle when there are eligible bundles waiting to be transmitted. In other words, it is not work-conserving. A non-work-conserving scheduler, like EDLQ, is prone to perform poorly in scenarios like the one shown in Figure 2.4.

So should the scheduler be work-conserving? The limitation of a work-conserving scheduler is also obvious. Such a scheduler may send a bundle to a gateway which will lead to a very late delivery while it could be delivered much earlier if we send it a little while later to another gateway which is currently busy.

Our solution to this dilemma is to use a work-conserving scheduler with a retransmission mechanism. When computing a schedule, we consider only the next $\lceil \frac{N}{L} \rceil$ time slots from each proxy-gateway link, so the resulting schedule is work-conserving. Then when we execute a schedule, after a bundle is sent, we do not delete the bundle immediately if it is not sent to the most desirable gateway, but store it in a secondary buffer and note the estimated delivery time. We keep the bundle in the secondary buffer as long as resending it to another gateway could lead to an earlier delivery. We resend bundles from the secondary buffer only when there are no eligible bundles left — so the links would otherwise be idle. In the absence of eligible bundles, we assign each transmission opportunity to a bundle from the secondary buffer so that the maximum reduction in delay can be achieved. Formally, we assign transmission opportunity p_{jh} to bundle b^* such that

$$b^* = \arg\max_{b \in B_s} (D(b) - D_{dst(b),j}(t_h))$$

where B_s is the set of bundles in the secondary buffer and $D(b)$ is the current estimated delivery time of bundle b. The receiver is expected to deal with duplicates.

We expect such a work-conserving scheduler with a retransmission mechanism to work well because when the load is light, most bundles that are not sent to the

most desirable gateway the first time will get a second chance and be delivered at the same time as with a non-work-conserving scheduler, and when the load is heavy, a work-conserving scheduler is the better choice to begin with.

2.5 Evaluation

In this section, we evaluate the performance of our proposed scheme using simulation. We show that our scheme does ensure fair allocation of bandwidth and compare our scheme with EDLQ in terms of delay.

2.5.1 Simulator and Simulation Setup

We developed a custom simulator which implements the model described in Section 2.1. Each simulation step corresponds to roughly one minute in reality. Each proxy-gateway link is capable of transmitting one bundle per step. For TB regulators, we use a generous bucket depth of 500. The maximum size of each pre-bucket queue is 200. For computing urgency, we use a discount rate of $\gamma = 0.5\%$. Each simulation is run for 43200 steps, or 30 days in reality. Each data point is obtained from running the simulation five times, and 95% confidence intervals are included. In all simulations involving EDLQ, TB regulators are used with EDLQ to ensure fair comparisons.

We use batched Poisson processes with a geometric batch size distribution as our traffic model . Such a process is characterized by mean inter-arrival time $1/\lambda$ and mean batch size b. The mean arrival rate can be computed as $b\lambda$. In all simulations we use a mean inter-arrival time of 20 steps and vary b to achieve desired arrival rates.

2.5.2 Microbenchmarks

The purpose of the microbenchmarks is to test the performance of our scheme in various scenarios that span as wide an area as possible in the parameter space. These scenarios are not meant to be realistic. While it is quite hard to parameterize the input space due to the enormous degrees of freedom with which bus

Load	Frequency	Transit Delay	Phase Difference
0.45	12	60	180°

Table 2.1: Parameters for the base case of Scenario 1. The first three columns apply to both kiosks.

schedules could vary, we identify four dimensions of the input space: load, phase difference between bus schedules, transit delay, and bus frequency. Load is the mean bundle arrival rate of a kiosk in number of bundles per step. The remaining three dimensions are about bus schedules. We consider a special class of bus schedules where for every gateway-kiosk pair $\langle g_j, k_i \rangle$, a bus leaves gateway g_j for kiosk k_i every $f_{i,j}$ steps, and each trip from gateway g_j to kiosk k_i takes $q_{i,j}$ steps. $1440/f_{i,j}$ is the number of buses going from gateway g_j to kiosk k_i in a day (recall that one simulation step corresponds to one minute in reality), which we call frequency. $q_{i,j}$ is the transit delay. If $f_{1,j} = f_{2,j}$, then the phase difference between these two schedules is defined as the difference of the departure time of a bus going to k_1 and the departure time of the next bus going to k_2 relative to $f_{1,j}$. A 180° phase difference means the difference in departure time is $\frac{1}{2} f_{1,j}$.

Single Gateway

In the first set of simulations we consider a simple scenario with one gateway, g_1, and two kiosks, k_1 and k_2. Since there is only one gateway, the gateway selection aspect of scheduling decisions is not tested in this scenario. Only the effect of service order is examined.

The parameters for the base case is shown in Table 2.1. We use a filling rate of 0.5 token per step for the TB regulator for both kiosks.

Every time, we vary the scenario from the base case along one of the four dimensions. Figure 2.9 shows the mean delay of the two kiosks when the load of kiosk k_2 varies from 0.1 to 2 using our scheme. As can be seen from the graph, after the load of kiosk k_2 exceeds its allocated rate, the delay of kiosk k_1 remains almost constant while the delay of kiosk k_2 increases significantly. Our simulation results also show that kiosk k_2 experiences no bundles loss while kiosk k_2 suffers from severe loss. The delay of kiosk k_2 finally levels off because the bundle dropping mechanism

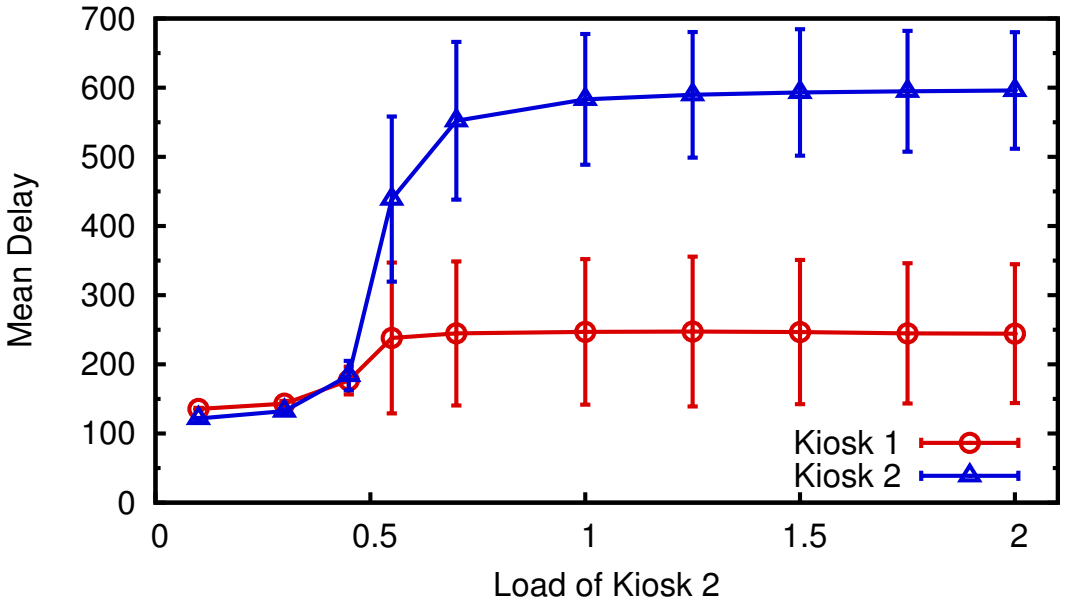


Figure 2.9: Delay using our scheme vs. load of kiosk k_2 in a single-gateway scenario

ensures the system is stable. This graph shows that the token-bucket mechanism does ensure fair allocation of bandwidth and protect well-behaving kiosks from being negatively impacted by ill-behaving kiosks.

Figure 2.10 shows the the percentage of reduction in delay using our scheme compared to EDLQ when the phase difference changes from 0 to 180°. When the phase difference is between 0 and 120°, one kiosk experiences slightly reduced delay and the other slightly increased delay, but overall the delay is reduced. when the phase difference is between 120° and 180°, both kiosks experience slightly reduced delay.

Figure 2.11 shows the effect of transit delay. We vary the transit delay of kiosk k_2 so that $q_{2,1}/q_{1,1}$ changes from 1 to 10. The reduction is not significant in this case, especially for kiosk k_2, whose delay become dominated by the large transit delay, which cannot be reduced by scheduling.

Finally, Figure 2.12 shows the effect of bus frequency. We reduce the frequency for kiosk k_2 so that the ratio of frequency between the two kiosks changes from 1 to 12. As the ratio increases, kiosk k_1 is able to enjoy more and more reduction in delay while kiosk k_2 enjoys less. This is because when a kiosk has infrequent buses,

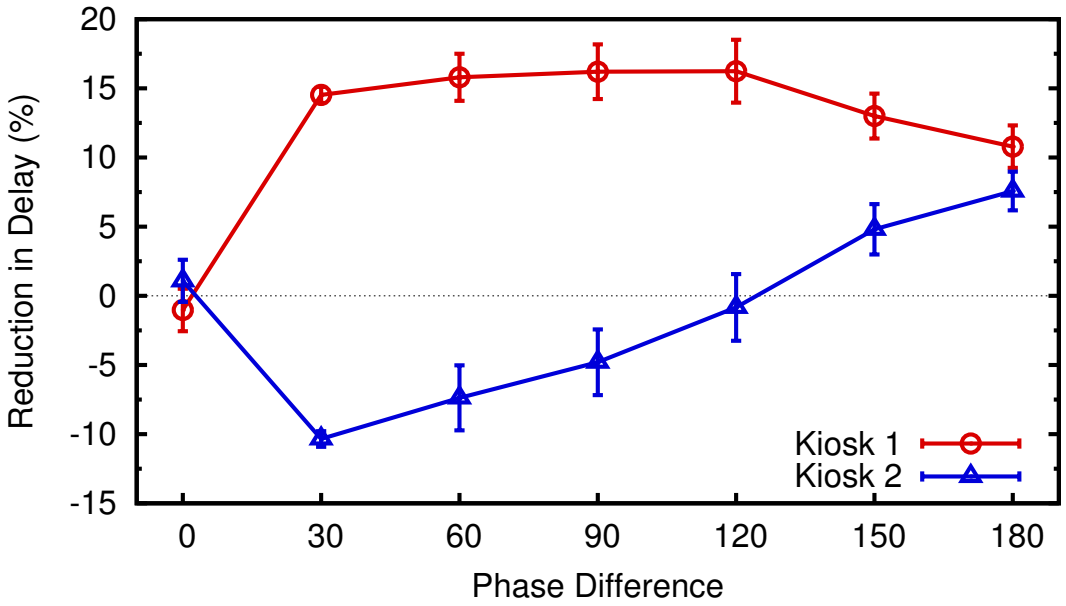


Figure 2.10: Reduction in delay vs. phase difference in a single-gateway scenario

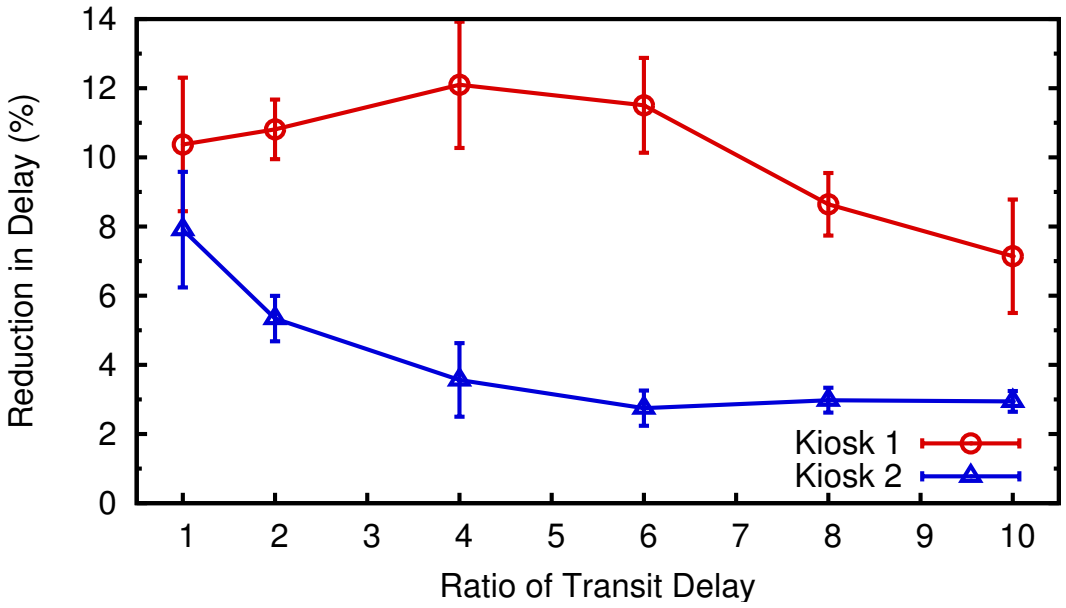


Figure 2.11: Reduction in delay vs. ratio of transit delay in a single-gateway scenario

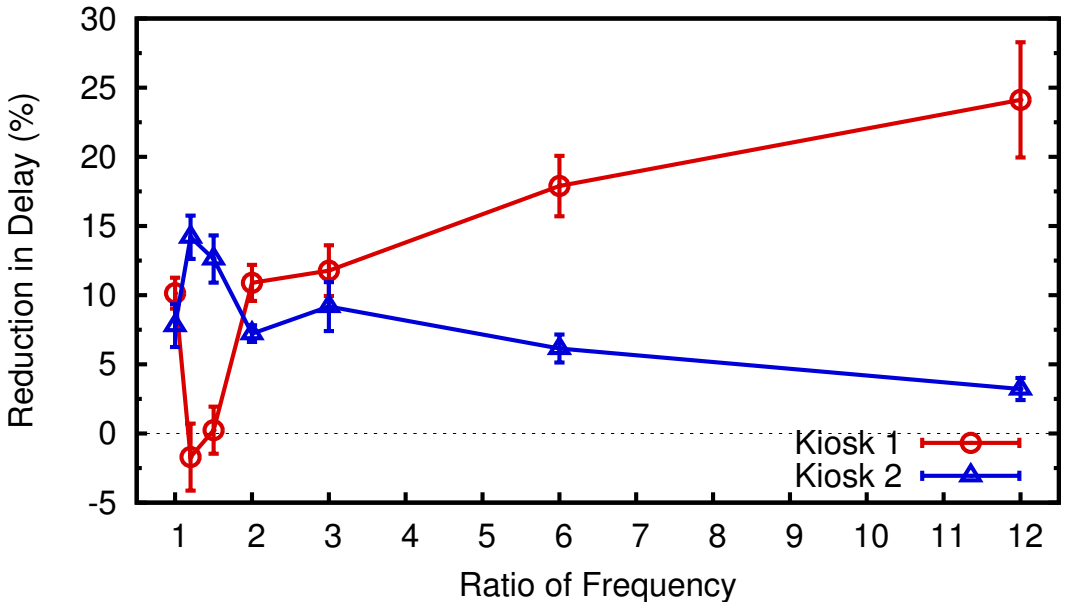


Figure 2.12: Reduction in delay vs. ratio of frequency in a single-gateway scenario

it is less likely to miss one so our scheduler can often schedule other kiosks that would otherwise miss their buses.

Multiple Gateways

Now let's consider scenarios with two gateways, g_1 and g_2, and two kiosks, k_1 and k_2. With two gateways, the gateway selection strategy will now play a role in determining delay. The token-bucket filling rate is set to 1 token per step for both kiosks.

Let's first consider a scenario where gateway g_1 is near to kiosk k_1 but far from kiosk k_2 and gateway g_2 is near to kiosk k_2 but far from kiosk k_1. The information about the bus schedules is shown in Table 2.2. Note that this is a favourable scenario for EDLQ because it is seldom necessary for any kiosk to use a secondary gateway.

We fix the load of kiosk k_1 at 0.9 bundle per step, and vary the load of kiosk k_2 from 0.1 to 2 bundles per step. The results are shown in Figure 2.13 and Figure 2.14. Again, we can see that kiosk k_1 is not affected by kiosk k_2 when the latter is

	Frequency	Transit Delay
$g_1 \rightarrow k_1$, $g_2 \rightarrow k_2$	10	60
$g_1 \rightarrow k_2$, $g_2 \rightarrow k_1$	4	150

Table 2.2: Bus schedule information. Each kiosk prefers a different gateway.

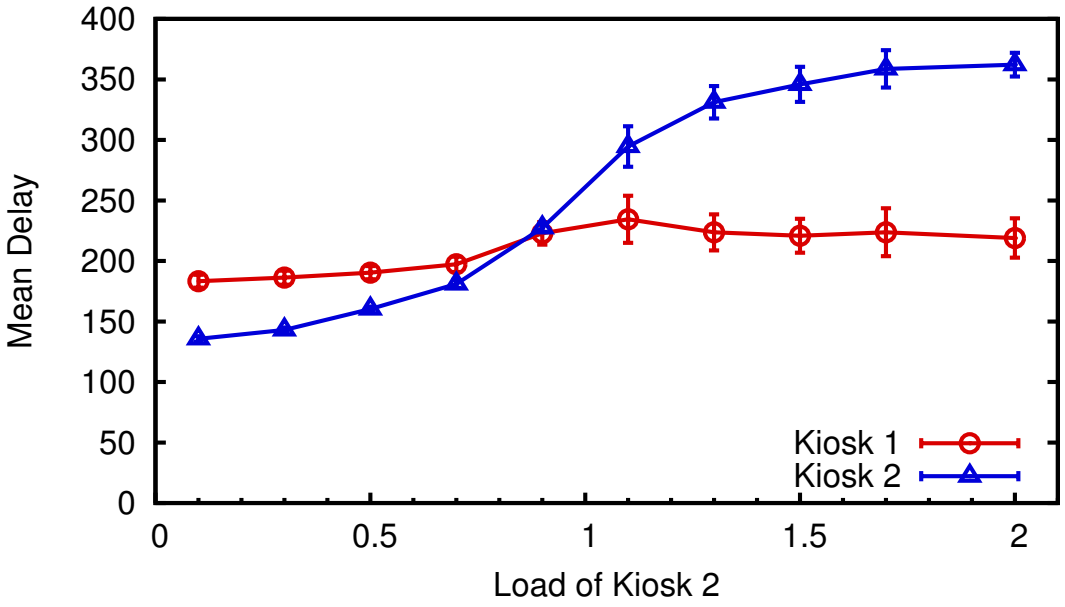


Figure 2.13: Delay using our scheme vs. load of kiosk k_2 in a scenario where each kiosk prefers a different gateway.

ill-behaving. Also we can see that our scheme is still slightly better than EDLQ even though this is a favourable scenario for EDLQ.

Next we consider a scenario where both kiosks prefer gateway g_1. The bus schedule information is shown in Table 2.3. This is a scenario where EDLQ tends to perform poorly because it only uses gateway g_2 when the queue for gateway g_1 grows too large, which may already be too late. Again, we fix the load of kiosk k_1 at 0.9 bundle per step, and vary the load of kiosk k_2 from 0.1 to 2 bundles per step. The results are shown in Figure 2.15 and Figure 2.16. As expected, our scheme offers more improvement over EDLQ than in the previous scenario. When the load of both kiosks is 0.9, both kiosks see a reduction in delay of about 25%.

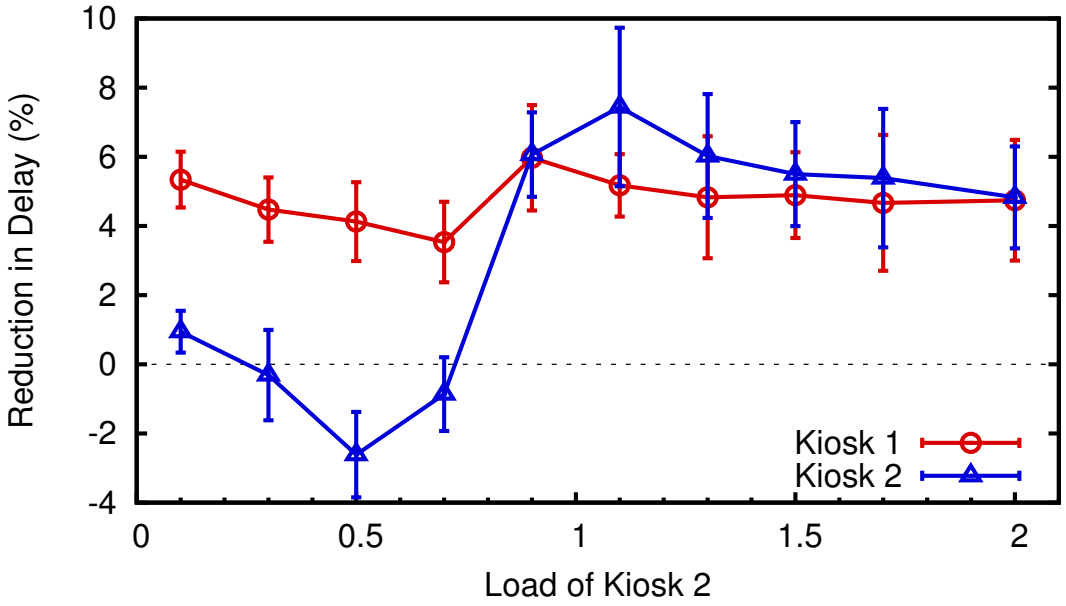


Figure 2.14: Reduction in delay vs. load of kiosk k_2 in a scenario where each kiosk prefers a different gateway

	Frequency	Transit Delay
$g_1 \to k_1$, $g_1 \to k_2$	10	60
$g_2 \to k_1$, $g_2 \to k_2$	4	150

Table 2.3: Bus schedule information. Both kiosks prefer the same gateway.

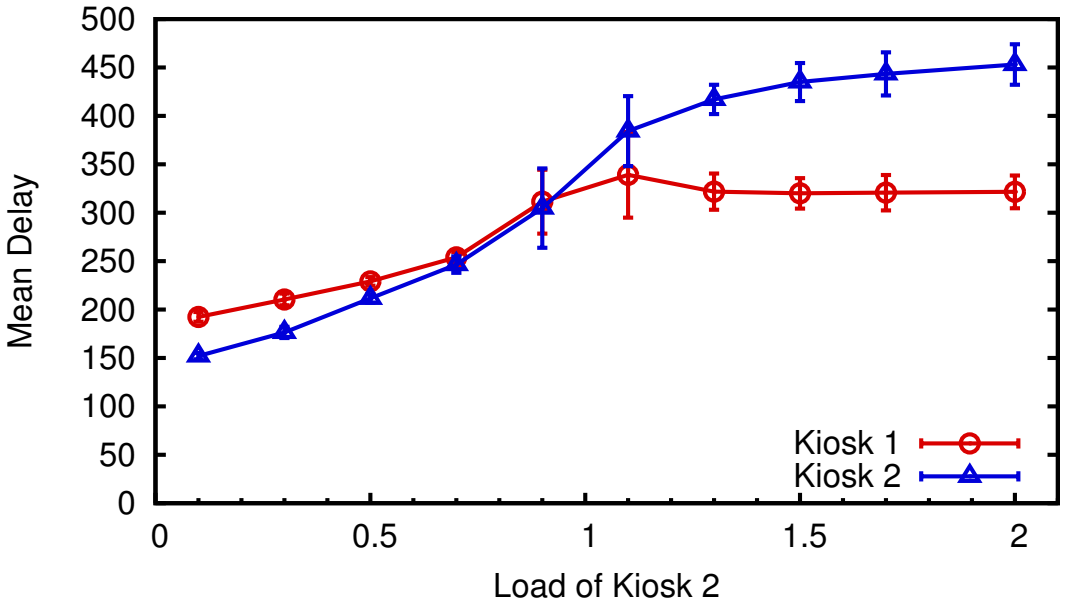


Figure 2.15: Delay using our scheme vs. load of kiosk k_2 in a scenario where both kiosks prefer the same gateway

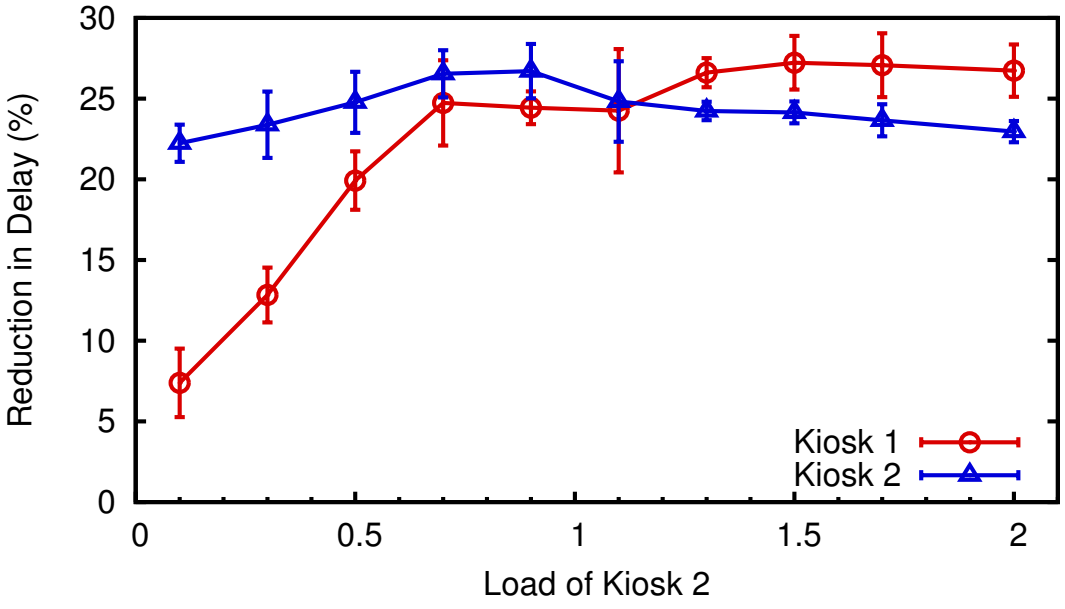


Figure 2.16: Reduction in delay vs. load of kiosk k_2 in a scenario where both kiosks prefer the same gateway

		From			
	IDT/MTD	Union	Yorkdale	York Mills	York U
To	Brampton	80/46	131/45	131/55	-
	Markham	53/38	-	-	72/35
	Milton	58/71	103/63	720/70	-
	NewMarket	48/60	58/54	111/80	90/55
	Oakville	48/35	206/70	1440/80	45/74
	Oshawa	58/54	25/105	25/95	85/76

Table 2.4: Mean inter-departure time (IDT) and mean transit delay (MTD) of each schedule in the GTA scenario.

Summary

The simulation results of microbenchmarks confirm the effectiveness of token buckets as a means to ensuring fair allocation of bandwidth and protecting well-behaving users from ill-behaving ones. It is also shown that our scheme outperforms EDLQ in all scenarios in terms of overall delay. While the margin is slim in scenarios that favour EDLQ, it is prominent in ones that do not.

2.5.3 A More Realistic Scenario

We now consider a more realistic scenario. Figure 2.17 shows part of the public transportation system in the Greater Toronto Area (GTA) . While the GTA, being a modernized metropolitan area, is certainly not in need of a data ferrying network to access the Internet, the bus routes connecting Toronto and its surrounding towns may resemble those found in developing regions. We select four locations in the city of Toronto to place our (imaginary) gateway nodes at, and serve six (imaginary) kiosks in surrounding towns. The information about bus schedules is taken from the website of the Greater Toronto Transit Authority[6]. These schedules, as is commonly observed, have more frequent departures during peak hours, less frequent departures during the rest of the day, and no departure during late night hours. Table 2.4 shows the mean inter-departure time and the mean transit delay of each schedule.

[6]http://www.gotransit.com/

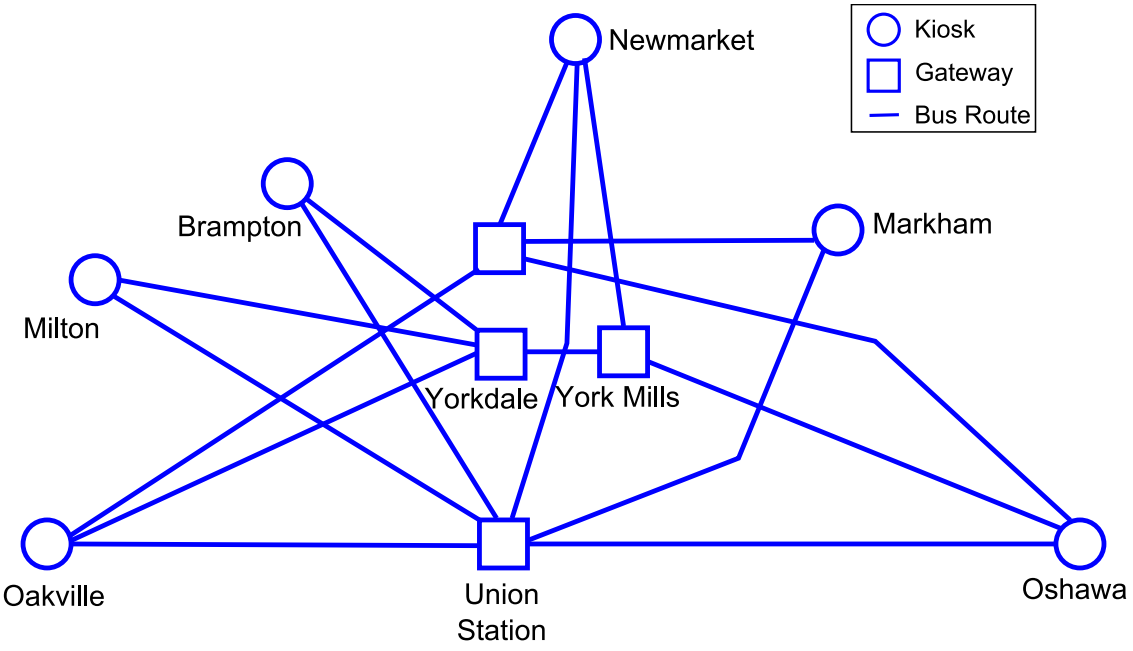


Figure 2.17: Selected part of the public transportation system of the Greater Toronto Area. Information about bus routes is taken from the published schedules of the Greater Toronto Transit Authority.

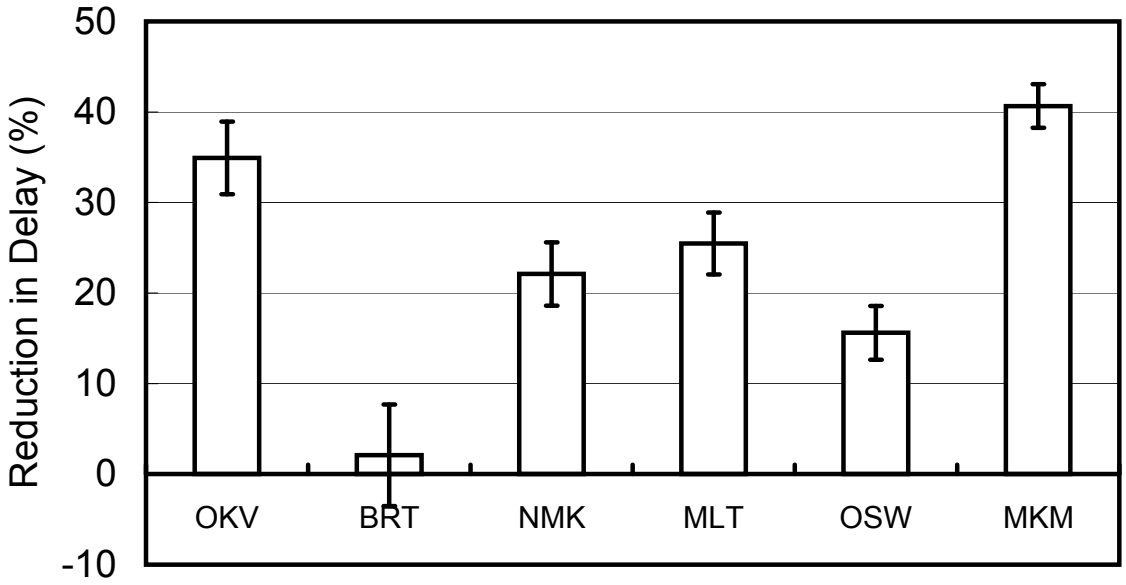


Figure 2.18: Reduction in delay in the GTA scenario

We set the token-bucket filling rate to 0.65 bundle per step for all six kiosks. All kiosks have a load of 0.6 bundle per step. The results about reduction in delay using our scheme compared to EDLQ are shown in Figure 2.18. We can see that the kiosk that benefits the most from our scheme enjoys a reduction in delay of about 40%. Four of the six kiosks see a reduction of over 20% and none of the kiosks experiences longer delay in a statistically significant sense. Compared to the simple scenarios in microbenchmarks, we see more improvement in this more complex scenario. We believe that our scheme offers more advantage in scenarios with complex topologies where there is more opportunity for optimization.

2.6 Impact of Imprecise Schedules

So far we have assumed that buses follow their publicized schedules precisely. However, this is never the case in reality. The punctuality of buses is subject to a number of factors that are random in nature, such as traffic conditions, weather conditions, and the conditions of the buses themselves. In this section we evaluate through simulation how this randomness affect the performance of our scheme.

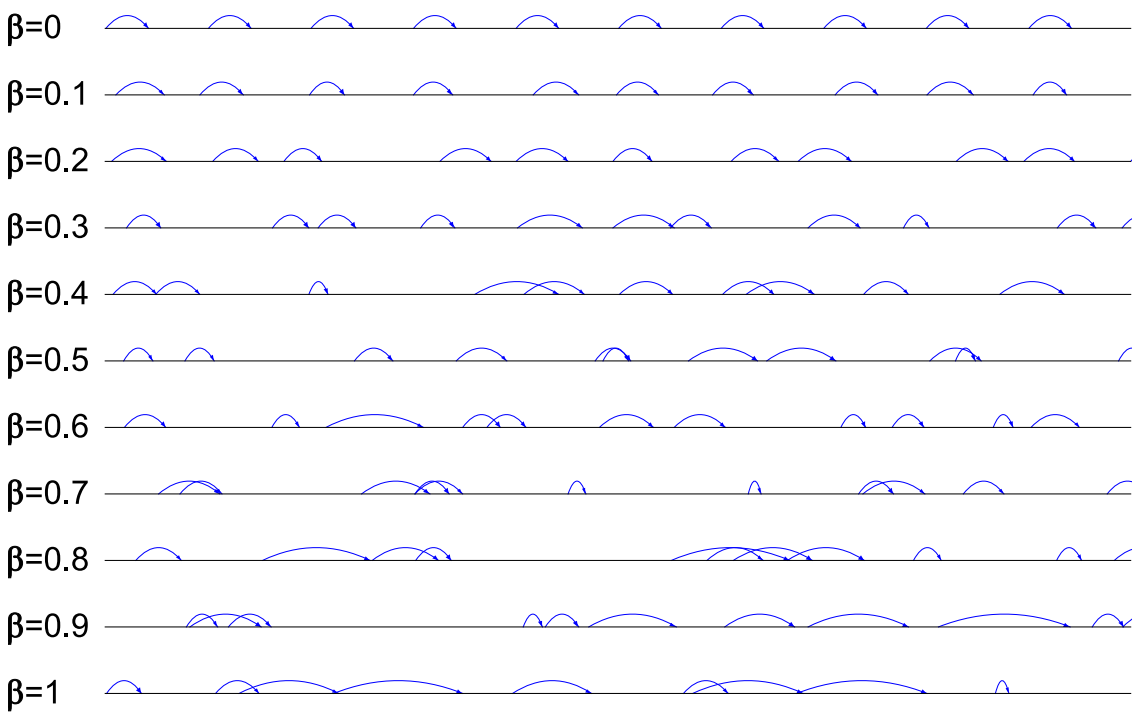
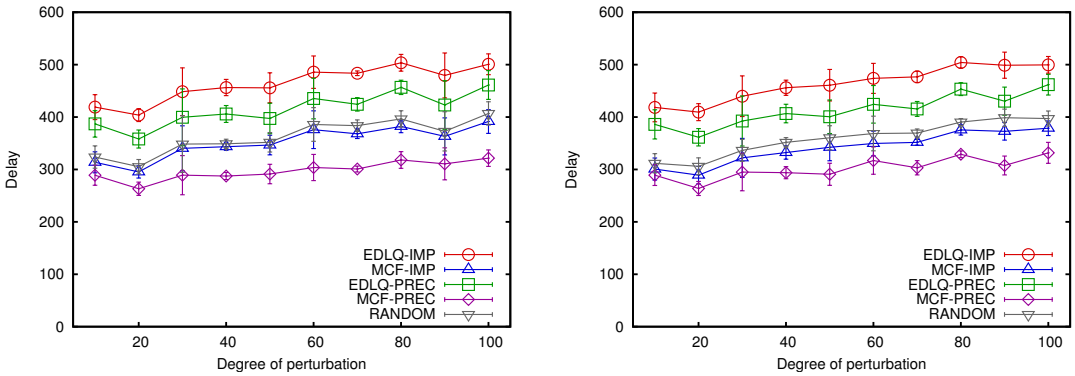


Figure 2.19: Effect of increasing degree of perturbation.

We are concerned with two types of randomness — randomness in departure times and randomness in transit delays. The first captures the fact that buses may leave earlier or later than scheduled, and the second the fact that it may take a bus a longer or shorter time than expected to reach the destination. In our simulations, we perturb the departure times and transit delays in the following manner. Given a schedule, we first compute the mean inter-departure time, denoted as $\overline{T}_d$. For each trip with scheduled departure time t, the actual departure time is drawn from a Gaussian distribution with a mean of t and a standard deviation of $\beta\overline{T}_d$, where β is called the *degree of perturbation*. Similarly, for each trip with expected transit delay q, the actual transit delay is drawn from a Gaussian distribution with a mean of q and a standard deviation of βq.[7]

We simulate again the multiple-gateway scenario in the microbenchmarks, except that this time the actual bus departure times and transit delays are randomized in the described above. Figure 2.19 is a visualization of the effect of increasing de-

[7]For transit delay, if the number turns out to be negative, we discard the number and draw another.



(a) Kiosk 1 (b) Kiosk 2

Figure 2.20: Impact of Imprecise Schedules — The Two-Gateway Two-Kiosk Scenario

gree perturbation to the schedule from gateway g_1 to kiosk k_1. Each line in Figure 2.19 represents the actual timing of bus trips in a typical day. One can see that as the degree of perturbation increases from 0, the actual departure times and transit delays deviate more and more from the schedule. However, from around $\beta = 50\%$, the actual timing is almost completely different from the schedule. Figure 2.20 shows the results. Scheme names that end with "-IMP" represent schemes that are fed with the original schedules, and those that end with "-PREC" represent schemes that are fed with the actual timing of bus trips. For comparison, we also included a random scheme which assigns bundles to gateways randomly, provided the bundle can be delivered via the randomly chosen gateway. Note the random scheme is work-conserving. From the figure, we can make four observations:

1. Unsurprisingly, for the same base scheme, schemes that are fed with precise timing information perform better than those fed with imprecise timing information.

2. The difference between *-IMP and *-PREC at first increases with the degree of perturbation, and remains almost constant beyond around $\beta = 20\%$.

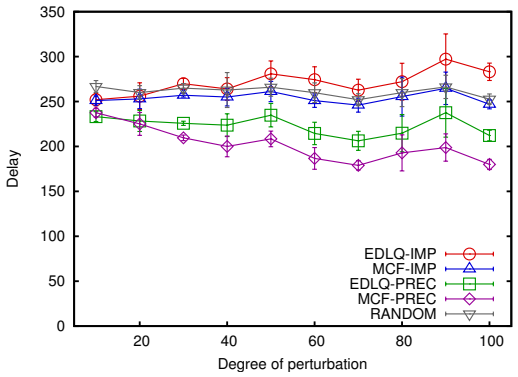
3. MCF performs better than EDLQ, even when MCF is fed with imprecise timing information and EDLQ is fed with precise timing information.

4. There is no discernable difference in the performance of MCF-IMP and RAN-

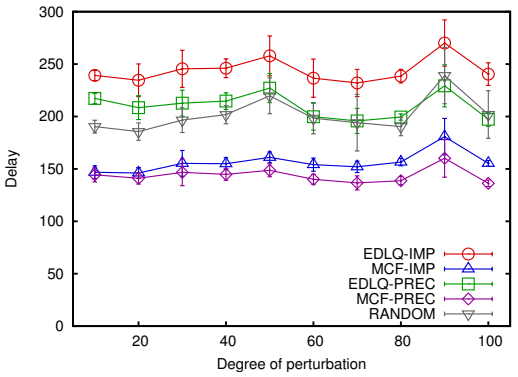
DOM. This indicates that in this scenario the gain from using MCF over EDLQ is mostly attributable to the work-conserving nature of MCF. MCF still outperforms RANDOM when fed with precise timing information.

We then simulate the GTA scenario with perturbed timing. The results are shown in Figure 2.21. The results are in large part consistent with the previous case. Except for Brampton, where EDLQ-PREC performs better than MCF-IMP, for all other kiosks MCF outperforms EDLQ, regardless of the preciseness of timing information. RANDOM in this scenario performs quite badly for some kiosks, indicating that even when there is substantial randomness in the timing of bus trips, schedule-aware schemes such as MCF and EDLQ are still superior to schedule-oblivious ones.

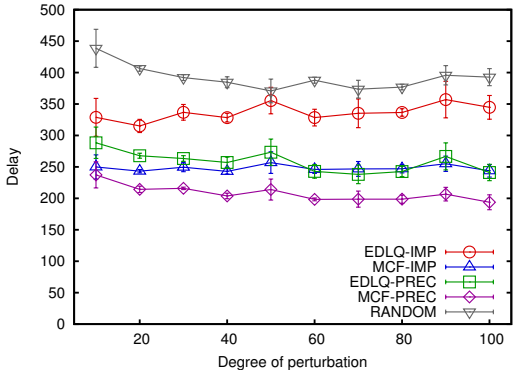
These initial results suggest that the impact of imprecise schedules is bounded and that our scheme compares favourably with EDLQ or schedule-oblivious schemes even when fed with imprecise timing information.



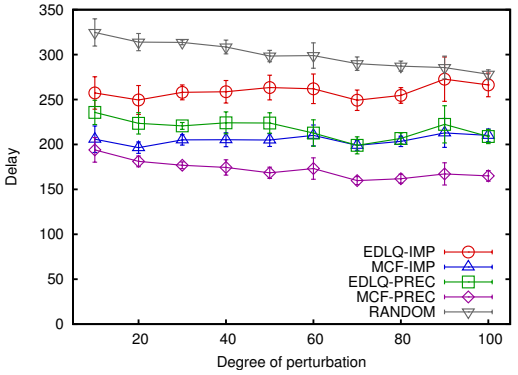
(a) Brampton



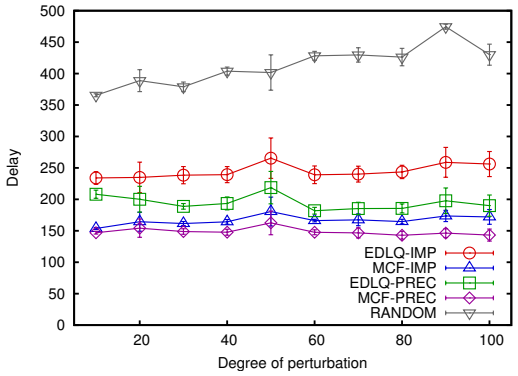
(b) Markham



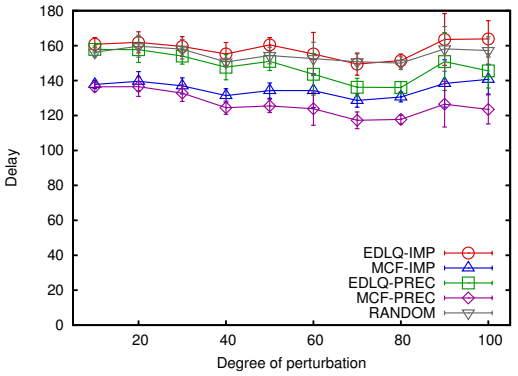
(c) Milton



(d) Newmarket



(e) Oakville



(f) Oshawa

Figure 2.21: Impact of Imprecise Schedules — The GTA Scenario

Chapter 3

Implementation

In this chapter we focus on the practical issues involved in the implementation of the KioskNet architecture. The implementation of the KioskNet system is based on the DTNRG DTN reference implementation [1]. We extended the the DTN reference implementation to add flooding and support user mobility. We subjected our software to long-term stress tests, and identified and fixed several performance bottlenecks. In the following sections, we describe the overall software architecture, the testing environment, and the stress tests we conducted. We discuss problems revealed by the stress test and abstract some principles for designing software for opportunistic communication.

3.1 Software Architecture Overview

KioskNet software runs on proxies, gateways, ferries, kiosk controllers, and cell phones, as shown in Figure 3.1. The DTNRG DTN reference implementation (shown in the figure as "DTN") runs on kiosks, ferries, and gateways. DTN is ultimately responsible for detecting peer nodes when they are in range, transferring data opportunistically, and routing data from the source to the destination. TCA Admin is an extension to DTN which implements flooding and location management. A *region* is identified by a region ID and each kiosk or gateway belongs to one region. Messages are flooded within a region. When a user account is created,

[1] Can be downloaded from http://www.dtnrg.org/wiki/Code

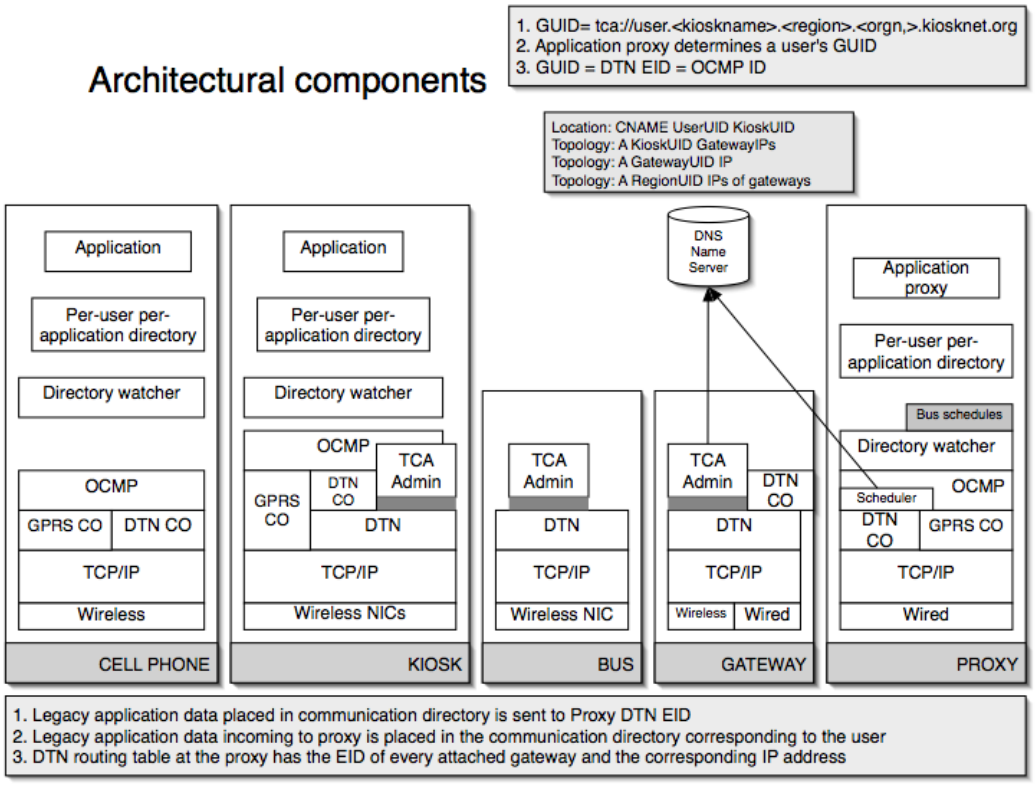


Figure 3.1: Software components

a central DNS name server is updated with the user ID and the ID of his/her home kiosk. The DNS name server also maintains information about which region each kiosk belongs to and the gateways in each region. When a gateway receives a message destined to a user in another region, it queries the DNS name server for a list of gateways in that region and chooses one to send the message to. The gateway in the other region that receives the message can then flood the message in its own region.

The *Opportunistic Connection Management Protocol (OCMP)* is session-layer protocol designed to work with multiple types of connections, such as GPRS, dial-up, VSAT, and DTN, and decide how to most efficiently use them to meet user-specified requirements. It runs on cell phones, kiosks and proxies. Each type of connection is modelled as a *Connection Object (CO)* which implements a common interface used by OCMP. OCMP allows KioskNet users to access legacy services on

the Internet. The OCMP on the proxy handles all communication with those legacy services on behalf of disconnected users. The OCMP software on the proxy can interact with the DTN software on multiple gateways via DTN COs. The scheduling algorithm presented in the previous chapter is implemented as a scheduler module in OCMP to control the use of these COs. OCMP also provides an easy-to-use directory API to support development of delay-tolerant applications.

3.2 Stress Tests

We performed long-term stress tests on our software for two purposes: 1) to find out the maximum throughput that can be achieved by our system, or how much overhead the software introduces, and 2) whether the performance degrades after the software has been running for an extended period of time. Through the stress tests, not only were we able to identify and fix several performance bottlenecks, but we also learned some lessons that should be regarded as principles for designing software for opportunistic communication.

3.2.1 Testbed Setup

The testbed setup is shown in Figure 3.2. The testbed consists of four nodes, which represent a kiosk, a bus, a gateway, and a proxy, respectively. All the nodes except the proxy are Soekris [2] boxes, while the proxy is a server-class machine. Soekris boxes are low-power single-board computers, widely used as mobile wireless routers in wireless projects. The model we use is Soekris net4801, which has an AMD 266 MHz processor and 256 MB of RAM. We equip each Soekris box with an Atheros WiFi interface card and a Toshiba 40 GB hard drive. The gateway is connected to the proxy via a wired network. The kiosk, bus, and gateway each have a WiFi interface operating in the 802.11a mode. The kiosk and the gateway run as access points on different channels using different essid's. The bus runs as a 802.11 client. We periodically flip the essid of the bus to emulate a real bus moving between the kiosk and the gateway. In the tests, we let the bus remain associated with the kiosk or the gateway for one minute, and associated with the other node for one minute, with five seconds of gap in between, and repeat the process indefinitely.

[2] http://www.soekris.com/

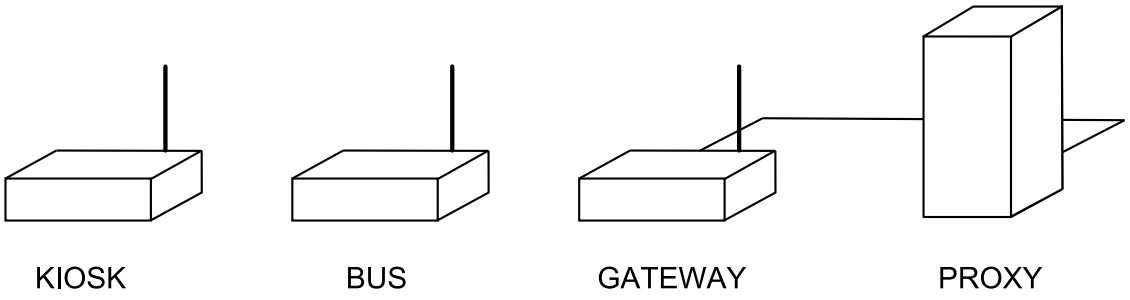


Figure 3.2: Testbed setup

Our testing application generates traffic between the kiosk and the proxy in both directions. It injects bundles into the network every minute, and each time injects a specified number of bundles. Previous study shows that bundle size has a great impact on the performance of the DTN reference implementation [16] and that larger bundle sizes generally lead to better throughput. Therefore, we use a bundle size of 50 KB, the maximum bundle size supported by the DTN API as of the time of the writing of this thesis. We request delivery receipt for every bundle. The testing application keeps track of every outstanding bundle that is sent but for which the delivery receipt has not been received. A test terminates when the number of outstanding bundles reaches a certain point, which indicates that our system is not able to handle the current load.

3.2.2 Lessons Learned from the Stress Tests

When we first started the stress tests, we soon found that the DTN reference implementation was not always able to establish a connection between two peers when they discovered each other. The DTN reference implementation is a multi-threaded, event-driven program, where a main thread executes an event loop and various other threads do actual work such as sending and receiving data and detecting neighbours, which also post events to the event queue when something happens that needs attention. The log output shows that the main thread did receive events telling it that a neighbour was discovered, but the times at which such events were processed were significantly later than the times at which those event occurred. Not surprisingly, by the time these events were processed, the peer had already gone. The log output also reveals that the main thread was busy processing backlogged

events, most of which non-urgent and data-related, such as bundle received events, when neighbour discovery events occurred. Clearly, non-urgent data-related events were interfering with time-sensitive link control events. We worked around the problem by posting control events at the head of the event queue, rather than the default action — posting at the tail. Although this worked quite well — the DTN software is now able to reliably establish connections between peers when they are in range, ideally we should have separate event queues for data-related events and control-related events. This leads us to the first principle for designing software that deals with opportunistic communication: *cleanly separate the data plane and the control plane, so that changes in the environment are responded to with the least possible delay.*

The performance of the modified software, however, was still not good. It could not even sustain a load of 10 bundles per minute, or 66.7 Kbps. We confirmed that the DTN software was able to establish a connection when two peers are in range. We ran system profiling software and found that during an opportunistic connection window, the NIC was only transmitting data for a fraction of the duration of the window, and the CPU utilization was 100%, of which an overwhelming majority is spent running user-level code. Finally we found the following code snippet in the router module of the DTN reference implementation.

```
for (iter = pending_bundles_->begin();
     iter != pending_bundles_->end();
     ++iter)
{
    fwd_to_matching(*iter, next_hop);
}
```

This code is part of the event handler for link open events, executed directly by the main thread. The `for` loop iterates through the list of backlogged bundles, and tries to send each bundle over the link that has just become open, by calling `fwd_to_matching(*iter, next_hop)`, where `next_hop` represents the link in question. `fwd_to_matching()` checks if the bundle should be sent over this link and if the link is busy. If all looks good it puts the bundle to a queue associated with the link, from which another thread (the sending thread) pulls bundles out and push them to the NIC. The queue has a limited capacity, the default being 10

bundles. When the queue gets filled, the status of the link will immediately become busy, and consequently subsequent calls to `fwd_to_matching()` will fail. As the sending thread pulls bundles out from the queue, it posts an event requesting to change the status of the link from busy to open. Now the cause of the problem becomes clear: after the main thread calls `fwd_to_matching()` successfully for the first few bundles, the links becomes busy, but the main thread continues to execute the loop. As the same time, the sending thread clears the queue and requests to change the status of the link by posting an event. The main thread cannot process this event until it iterates through the entire bundle list, which, when there is a large backlog, not only takes significant amount of time on a under-powered processor like the ones used by Soekris boxes, but also wastes time doing nothing useful, which explains the low utilization of connectivity and high user-level CPU utilization. Clearly, this is another violation of the first principle, where data operations (sending bundles) interfere with control operations (link status management). Furthermore, this case also demonstrates the detrimental effect of having an event handler that takes too much time to execute, hence the second principle: *in an event-driven program, event handlers should never block the main thread regardless of the load; if an event could potentially take a long time to process, the main thread should delegate the processing to another thread*. While in our case it is preferable to delegate the processing of link open events to another thread, as a quick fix, we simple break the loop when we find the link is busy. Now the code looks like

```
for (iter = pending_bundles_->begin();
     iter != pending_bundles_->end();
     ++iter)
{
    if (next_hop->isbusy())
        break;
    fwd_to_matching(*iter, next_hop);
}
```

This change has been included in the latest release of the DTN reference implementation.

Further stress tests showed there was still a significant amount of CPU time spent running user-level code when bundle transfer between two peers took place.

We profiled CPU usage using oprofile and were able to identify two pieces of code as the main culprits. The first piece of code performs duplicate detection. Upon the receipt of a bundle, the DTN software iterates through its bundle list to see if the same bundle has been received before. The second piece of code is in our extension to the DTN reference implementation which deals with death certificates. A death certificate is a special type of bundle which is generated when a bundle is delivered to its destination. The death certificate is then flooded over the network and whoever receives the death certificate can safely remove the delivered bundle from its storage, if it has it. Recall that we use flooding, where, without death certificates, the only way to discard bundles is through expiration. The second piece of code checks the list of bundles against a death certificate. What is common in both pieces of code is that both involve operations whose running time is $O(n)$, where n is the number of bundles in a node's storage. Duplicate detection and death certificates have the potential benefit of reducing storage requirement and network traffic, but in order to realize such benefit, we must design efficient algorithms and data structures that scale well, especially with the low-powered processors we currently have, otherwise the high computational cost could easily defeat the purpose of having these measures in place. This leads us to the third principle: *use efficient algorithms and data structures that scale well with load to keep computational overhead low.*

Chapter 4

Related Work

Fair sharing of bandwidth in packet-switching networks has been well studied in the context of traditional low-delay networks. The most popular notion of fairness in the literature is called max-min fairness, which is what can be achieved with Generalized Processor Sharing (GPS). Since GPS is unimplementable in reality, a number of packetized scheduling algorithms have been proposed which in general all try to mimic GPS as closely as possible. For a survey of these scheduling algorithms see [21] and references therein. A fair scheduling algorithm that operates on multiple links[4] is also proposed after the survey paper is published. Our work is different from theirs in two aspects. First, our notion of fairness is defined on a longer time scale. While their scheduling disciplines try to achieve max-min allocation of bandwidth at time intervals as short as possible, we focus on long-term fairness which is exactly what token-bucket regulator can provide. We are willing to allocate a disproportionately large portion of bandwidth to some users at one time and compensate for other users at another, provided the bandwidth is allocated fairly in the long run. We do not lose anything by giving up short-term fairness because bundles sent to the gateways early will have to wait for their buses to come anyway. Second, besides ensuring fair allocation of bandwidth, our scheme also tries to minimize end-to-end delay. The addition of this second objective makes it a much harder problem if the scheduler has to fulfill both objectives. In our scheme, we offload the task of ensuring fairness to token-bucket regulators and let the scheduler focus exclusively on delay minimization.

Opportunistic scheduling has been studied in the context of wireless data net-

works, where base stations can exploit temporal fluctuation of link quality to maximize aggregate throughput[12][14][15]. What is common in our and their work is that we both trade strict adherence to max-min fairness for improvement in other performance metrics. However, delay minimization and throughput maximization require fundamentally different approaches. Therefore techniques developed in their area cannot be applied in our system.

Our work is inspired by research in Time/Utility Function (TUF) based scheduling [13][18][20][9] in the area of real time scheduling. TUFs are a generalization of the hard real time constraints. Instead of specifying a hard deadline for each job, a TUF specifies the utility resulting from the completion of a job as a function of arbitrary shape of its completion time. In our system, the utility resulting from sending a bundle is determined by the time at which it is sent and the gateway to which it is sent. They consider a richer set of objectives than just utility maximization, such as providing bounds on the probability with which jobs are completed before their critical times [9], and additional constraints, such as interdependencies between jobs and mutually exclusive accesses to non-CPU resources[13].

Other projects that use ferries to physically transport data in challenged environments include Message Ferrying (MF) [24] and DieselNet [7]. MF considers mobile ad hoc networks where there may not always be a single- or multiple-hop route between some or all node pairs due to node mobility, short radio range, physical obstacles, or so forth. MF deploys a set of mobile nodes called *message ferries* which move along certain routes and can be used to deliver messages in a store-carry-forward fashion. The controlling of the trajectory of mobile nodes is a major concern in the MF scheme [22][25]. DieselNet is a large-scale testbed of delay-tolerant networking consisting of 40 buses in Amherst, Massachusetts. They study problems including routing [7][2], security [6], and power management [3]. The main differences between our work and theirs are 1) while traffic is between peers in their systems, it is always either uplink or downlink in our system; and 2) we assume that the bus schedules are known to us so routing inside the kiosk network is a relatively easy problem.

Chapter 5

Conclusions and Future Work

In this thesis we study the downlink scheduling problem in rural kiosk networks and report some experience with building and testing the software for KioskNet. For the downlink scheduling problem, we analyze why EDLQ, an existing shortest path based routing algorithm which assumes FIFO service order, fails to meet the goals. Our proposed solution consists of two parts: token-bucket regulators and a utility-based scheduler. The former ensures fair allocation of bandwidth and the latter repeatedly computes a utility-maximizing bundle transmission schedule for all bundles admitted by the former. We formulate the optimal scheduling problem as a minimum-cost network-flow problem for which efficient algorithms exist. We describe a technique based on the notion of urgency for choosing a schedule from a set of schedules that are equivalent in terms of utility. We discuss the pros and cons of work-conserving and non-work-conserving scheduler and propose to use a work-conserving scheduler with a retransmission mechanism to get the best from both. Simulation results show that, compared to EDLQ, our scheme reduces overall delay, and, most of the time, reduces delay for all users. The amount of reduction in delay for a single user can be sometimes up to 40%. Our initial results regarding the impact of imprecise schedules indicate that the proposed scheme is robust against randomness in the timing of bus trips.

From our experience with building and testing the software for KioskNet, we abstract three principles for designing software for opportunistic communication: 1) *cleanly separate control and data planes*, 2) *never block the main thread in a event-driven program*, and 3) *use efficient algorithms and data structures that scale*

well with load.

5.1 Future Work

In the future, we plan to work on the following problems:

1. Investigating the impact of imprecise schedules in more depth. Our evaluation with imprecise schedules is preliminary. The model we use to generate randomness in the timing of bus trips is not necessarily realistic. To have a more realistic model, we need to understand the nature of the randomness in bus movement.

2. Application-layer delay minimization. In this thesis our goal is to minimize overall bundle delay, which is different from application-layer delay, which is determined by the delay of the last received bundle of an application-layer data unit. Minimizing bundle delay does not necessarily minimizes application-layer delay because of out-of-order delivery as a result of load balancing across multiple gateways, known as *link striping*. Minimizing application-layer delay is a more meaningful goal since that is the delay perceived by end users.

3. Deploying the system in the field. We would like to deploy a real system in the field and run experiments on the real system to evaluate the performance of our scheme. A deployed system would also allow us to collect traffic traces which could help us understand traffic patterns in such networks. As stated earlier, our current scheduling algorithm is not globally optimal because it does not consider future traffic. With an understanding of the traffic arrival patterns, it may be possible to improve our algorithm by taking into account prediction of future traffic.

List of References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993. 17, 18

[2] Aruna Balasubramanian, Brian Neil Levine, and Arun Venkataramani. DTN routing as a resource allocation problem. In *Proc. ACM SIGCOMM*, August 2007. 46

[3] Nilanjan Banerjee, Mark D. Corner, and Brian Neil Levine. An energy-efficient architecture for DTN throwboxes. In *Proc. IEEE INFOCOM*, Anchorage, Alaska, May 2007. 46

[4] Josep M. Blanquer and Banu Özden. Fair queuing for aggregated multiple links. In *Proc. ACM SIGCOMM*, pages 189–197, 2001. 45

[5] Bob Briscoe. Flow rate fairness: Dismantling a religion. *SIGCOMM Comput. Commun. Rev.*, 37(2):63–74, 2007. 8

[6] John Burgess, George Bissias, Mark D. Corner, and Brian Neil Levine. Surviving attacks on disruption-tolerant networks without authentication. In *Proc. ACM MobiHoc*, Montreal, Quebec, Canada, September 2007. 46

[7] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. MaxProp: routing for vehicle-based disruption-tolerant networks. In *Proc. IEEE INFOCOM*, April 2006. 46

[8] V. Cerf, S. Burleigh, A. Hooke, L.Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-tolerant networking architecture. IETF RFC 4838, April 2007. 1

[9] Hyeonjoong Cho, Haisang Wu, Binoy Ravindran, and E. Douglas Jensen. On multiprocessor utility accrual real-time scheduling with statistical timing assurances. In *Proc. EUC*, pages 274–286, 2006. 46

[10] Alan J. Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. ACM SIGCOMM*, pages 1–12, 1989.

[11] Sushant Jain, Kevin R. Fall, and Rabin K. Patra. Routing in a delay tolerant network. In *Proc. ACM SIGCOMM*, pages 145–158, 2004. iii, 3, 7, 9

[12] S. S. Kulkarni and C. Rosenberg. Opportunistic scheduling for wireless systems with multiple interfaces and multiple constraints. In *Proc. ACM Modeling Analysis and Simulation of Wireless and Mobile Systems*. 46

[13] Peng Li, Haisang Wu, Binoy Ravindran, and E. Douglas Jensen. A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints. *IEEE Trans. Computers*, 55(4):454–469, 2006. 46

[14] X. Liu, E. K. P. Chong, and N. B. Shroff. A framework for opportunistic scheduling in wireless networks. *Computer Networks*, 41(4):451–474, March 2003. 46

[15] Y. Liu and E. Knightly. Opportunistic fair scheduling over multiple wireless channels. In *Proc. INFOCOM*. 46

[16] Earl Oliver and Hossein Falaki. Performance evaluation and analysis of delay tolerant networking. In *Proc. MobiEval*, pages 1–6, New York, NY, USA, 2007. ACM Press. 41

[17] Alex (Sandy) Pentland, Richard Fletcher, and Amir Hasson. Daknet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83, 2004. 1

[18] Binoy Ravindran, E. Douglas Jensen, and Peng Li. On recent advances in time/utility function real-time scheduling and resource management. In *Proc. IEEE ISORC*, pages 55–60, 2005. 46

[19] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *MobiCom '06:*

Proceedings of the 12th annual international conference on Mobile computing and networking, pages 334–345, New York, NY, USA, 2006. ACM Press. 1

[20] Jinggang Wang and Binoy Ravindran. Time-utility function-driven switched ethernet: packet scheduling algorithm, implementation, and feasibility analysis. *IEEE Trans. Parallel Distrib. Syst.*, 15(2):119–133, 2004. 46

[21] Hui Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proc. IEEE*, pages 1374–1396, 1995. 45

[22] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proc. ACM MobiHoc*, pages 187–198, New York, NY, USA, 2004. ACM Press. 46

[23] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. Multicasting in delay-tolerant networks: Semantic models and routing algorithms. In *Proc. ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN)*, pages 268–275, New York, NY, USA, 2005. ACM Press.

[24] Wenrui Zhao and Mostafa H. Ammar. Message ferrying: proactive routing in highly-partitioned wireless ad hoc networks. In *Proc. IEEE FTDCS*, pages 308–314, 2003. 46

[25] Wenrui Zhao, Mostafa H. Ammar, and Ellen W. Zegura. Controlling the mobility of multiple data transport ferries in a delay-tolerant network. In *Proc. IEEE INFOCOM*, pages 1407–1418, 2005. 46