Fair and Efficient Scheduling in Data Ferrying Networks

Shimin Guo, Srinivasan Keshav
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
{sguo,keshav}@cs.uwaterloo.ca

ABSTRACT

Data-ferrying disconnection-tolerant networks allow remote rural areas to access the Internet at very low cost, making them viable alternatives to more expensive access technologies such as DSL, CDMA, and dial-up. In such a network, an Internet-based proxy gathers data from the Internet and sends it to a set of edge nodes called "gateways", from which data 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. In this context, we pose the following question: assuming knowledge of ferry schedules, *when* and *to which gateway* should the proxy send each data bundle so that the overall delay is minimized and the bandwidth is shared fairly among competing kiosks? We show that a well-known schedule-aware routing scheme proposed in the literature, *i.e.*, EDLQ [11] is far from optimal. Moreover, EDLQ does not provide means to enforce bandwidth allocations. To remedy these problems, we employ a token bucket mechanism to decouple fairness and delay minimization concerns. We also describe a utility-maximizing scheduler based on the classical minimum-cost network flow problem, that finds optimal schedules. Through simulations, we show that our scheme performs at least as well as EDLQ in scenarios that favour EDLQ, yet achieves up to 40% reduction in delay in those that do not.

1. INTRODUCTION

Ferry-based networks provide a means of bringing extremely low-cost *non-interactive* Internet access to remote rural areas, where conventional access technologies such as DSL and CDMA are currently economically infeasible. Such networks trade off a lack of interactivity and increased delay for a substantial reduction in data transmission costs. An architecture for ferry-based networks is proposed in [9, 18], which we sketch next.

The system contains four major components: rural kiosks, ferries (or buses), Internet gateways, and a proxy server. End users send and receive data (in the form of fixed-length *bundles*) at rural kiosks. Buses serve as ferries, 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 towns or cities, have persistent Internet connections such as dial-up or DSL. Their job is to forward data to and from the proxy. A proxy is a well-provisioned machine on the Internet. It communicates on behalf of users with legacy servers such as web, FTP, and mail servers. In the uplink direction (*i.e.*, from kiosks to the Internet), it receives requests from users and initiates communication with legacy servers. In the downlink direction (*i.e.*, from the Internet to the kiosks), the proxy buffers data from legacy servers, forwarding the data to the gateways. Buses opportunistically pick up data from the gateways for eventual delivery to destination kiosks. Note that a kiosk may be reached by more than one gateway.

We study the operation of the proxy in the downlink direction. Here, the proxy acts as an application-layer switch, whose incoming and outgoing links are TCP connections to legacy servers and the gateways. Whenever an outgoing link to a gateway 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. In other words, the proxy continually assigns transmission opportunities to bundles with the goal of minimizing end-to-end delay and, during times of congestion, ensure fair allocation of bandwidth among kiosks.

By assigning transmission opportunities to bundles, the proxy selects a gateway for each bundle and decides the order in which different kiosks are served. Existing schemes tend to decouple these two tasks. Typically, they select an outgoing link (i.e gateway) immediately after a bundle arrives, placing the bundle into a buffer

*CoNEXT'*07, December 10-13, 2007, New York, NY, U.S.A.
Copyright 2007 ACM 978-1-59593-770-4/ 07/ 0012 ...$5.00.

associated with the chosen link. When a transmission opportunity arises on that link, they apply a scheduling discipline such as FCFS or Round-Robin to determine the order in which bundles destined to different kiosks (but sharing the same gateway) are served. To our knowledge, the best such scheme is EDLQ [11]. As we show later, EDLQ can be far from optimal.

We propose a novel scheme for downlink scheduling. We use a token bucket traffic regulator for each kiosk to enforce fair allocation of bandwidth during times of congestion. Moreover, every time new bundles leave the regulator, a utility-maximizing scheduler is invoked to compute a schedule for all unserved bundles. In other words, we associate with each bundle some utility, which captures the "value" gained from delivering it as a function of its delay. The scheduler computes a schedule that maximizes the total utility. We show that if we define the utility function in a certain way, the optimal scheduling problem can be formulated as a minimum cost network flow problem, for which efficient algorithms exist. We evaluate our scheme using simulations. We simulate a number of scenarios and the results show that our scheme performs at least as well as EDLQ in scenarios that favour EDLQ and significantly better in scenarios that do not.

2. SYSTEM MODEL AND OBJECTIVE

2.1 System Model

Figure 1 shows the system model. Data arriving from legacy servers 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, respectively. The *delay* of a bundle is measured from the moment it arrives at the proxy to the moment it is delivered to its destination kiosk. There is a logical link between the proxy and each gateway — in reality, a TCP connection — which we assume has a constant rate. 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 provide a limited data rate of not more than 100 Kbps [16]. As a result, the bandwidth of individual links between the proxy and the gateways is limited by the capacity of the DSL subscription the gateways have. 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. 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. Whenever a link becomes free, the scheduler may either select a bundle from the buffer and sends the bundle over that link or decide to leave the link idle. Data sent to a gateway is temporarily buffered at the gateway, waiting to be picked up by a bus that would take it to the destination kiosk.

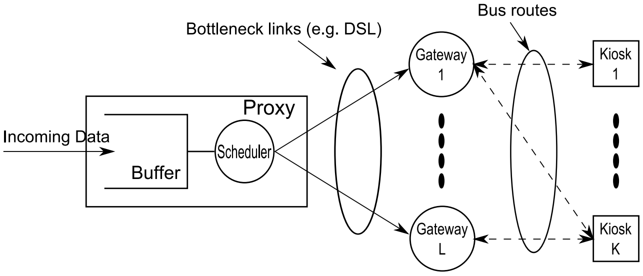

Figure 1: System model

The gateways and kiosks are connected by bus routes. *Bus schedules* define the start and end times of opportunistic connection windows when a bus passes by a gateway or a kiosk. We assume that the schedules of all buses are known and that the buses follow their schedules precisely (we will relax this assumption later). 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 allow us to 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 not any 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 non-decreasing. 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 this case, since we assume that wireless links are infinitely

[1] In reality, solution only requires the rate of each individual proxy-gateway link to be constant over time, but does not require all their rates to be the same. However, allowing the rates to be different only makes the notation more complex, but without offering additional insight.

[2] This assumption is not as strong as it might appear. [10] reports that a vehicle can transfer about 30 MB of data to a roadside access point during a 30-second connection window using 802.11g. At this rate, 1 GB of data can be transferred in less than 20 minutes, which would take one day at 100 Kbps, the typical rate of a proxy-gateway link.

fast, b_1 and b_2 will be guaranteed to be sent to the same bus and delivered at the same time. In fact, $D_{i,j}(t)$ is a step function, where jumps in delivery time occur when buses leave the gateway g_j for kiosk k_i.

2.2 Requirements

We now state the optimization criteria. First, note that given the unlimited inbound bandwidth of the proxy and the limited capacities of proxy-gateway links, traffic may arrive to the proxy at a higher rate than it can leave the proxy, which, coupled with our assumption that all wireless links have infinite bandwidth, indicates that the proxy-gateway links are the only bottlenecks of the system. The bandwidth of the proxy-gateway links therefore must be shared fairly.

But among whom should the bandwidth be allocated? We consider each kiosk as an economic entity for which a fair share of bandwidth is allocated. A kiosk owner would pay the network provider a certain subscription fee proportional 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 an appropriate amount of bandwidth according to the size of their business. In view of this allocation scheme, in the rest of the paper we shall use the term "kiosk" interchangeably with the term "user".

Returning to the issue of congestion, we require then, that when demands exceeds capacity, each kiosk be guaranteed to receive a certain amount of bandwidth that is allocated to it. As a secondary objective, the scheduler should minimize overall bundle delay across all kiosks. This requirement is more subtle than appears at first glance, because it requires the scheduler to choose gateways keeping in mind future bus arrivals as well as future bundle. Also note that the primary and secondary objectives are inherently in conflict: meeting a certain bandwidth requirement may cause excess delay. When the goals conflict, the primary goal is given priority. A more formal description of the scheduling objectives can be found in Section 5.2.

3. EDLQ

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)*. 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. We summarize only EDLQ here because EDAQ reduces to EDLQ in our system, and EDLQ has already been demonstrated to be considerably better than ED.

Let Δ_{j^*} be the queuing delay before a bundle can be sent to gateway g_{j^*}. EDLQ chooses the 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 only 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.

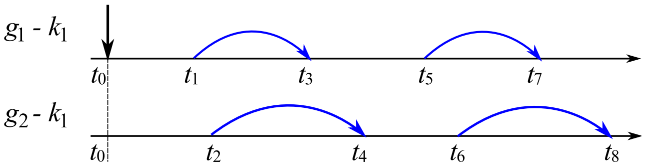

Figure 2: A scenario where EDLQ performs well. The top axis shows the schedule of buses from gateway g_1 to kiosk k_1 and the bottom one shows the schedule of buses from gateway g_2 to kiosk k_1. Each arc represents one bus trip, with the tail of the arc indicating the time when the bus leaves the gateway and the head indicating the time when the bus arrives at the kiosk. Downward arrows represent batched bundle arrivals at the proxy. (The same graphical representation of bus schedules used in other figures)

4. 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 correctly responds to the increase in the local queue length by spreading the load across multiple links, as shown in Figure 2.

However, closer study reveals three limitations. 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 on each proxy-gateway link 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.

The second problem with EDLQ stems from its greedy

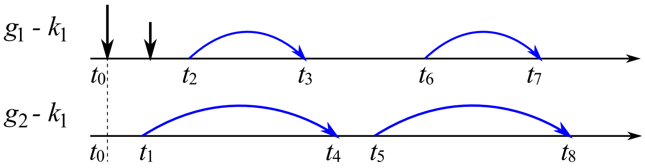

Figure 3: A scenario where EDLQ performs poorly due to its greedy nature.

nature. Although, as shown before, it has the ability to switch to a secondary path when there is already enough load on the primary path, sometimes such actions come too late. Consider the scenario shown in Figure 3, which looks similar to Figure 2 but now the buses from gateway g_2 to kiosk k_1 leave g_2 earlier than they do in Figure 2. 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 it 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 4. $t_0, t_1, \ldots, t_{10}$ are evenly spaced, with $t_{i+1} - t_i = \delta$. In this paragraph we shall use the term “batch” to refer to the amount of bundles that would take 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 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* worse than optimal scheduling.

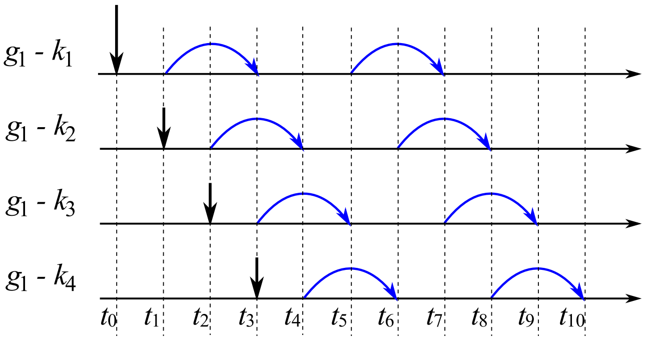

Figure 4: A scenario where EDLQ performs poorly due to its inability to reorder bundles.

5. OUR SOLUTION

We now present our utility-based approach to downlink scheduling in data ferrying 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*[3] 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.

5.1 Token-Bucket Traffic Regulator

As discussed earlier, we allow each kiosk to receive a bandwidth allocation at the proxy. 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. These allocations are managed by a standard token-bucket (TB) traffic regulator with parameters of filling rate σ and token bucket depth d. Bundles leaving the regulator are deemed *eligible* for subsequent scheduling and are buffered in per-kiosk queues. If the token bucket is empty, bundles are temporarily buffered in a pre-bucket queue with a fixed ca-

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

pacity. If the pre-bucket queue is full, newly arriving bundles are dropped.

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

On the other hand, the limitation of TB regulators is that are non-work-conserving. Thus, there may be times when bundles are blocked by the regulators while the links sit idle. An alternative is to use a work-conserving fair allocator, such as WFQ, to allocate bundles. In our work, for simplicity, we only study TB regulators.

5.2 Utility Model

Consider a scheduling decision made at time t_s. We assume that the usefulness of a bundle to kiosk k_i is captured by a utility function $U_i(x)$ where x is the bundle's delay from t_s to its time of delivery. Let $W_{i,j}(t)$ denote the utility of sending a bundle from kiosk k_i's post-bucket queue at time t to gateway g_j. 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. It is easy to see that

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

Although this formulation requires all bundles to the same kiosk to share the same utility, we can easily transform a single kiosk into multiple virtual kiosks, each corresponding to a different utility function, so there is no loss of generality.

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 currently 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.

5.3 Scheduling

Each 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 where the length of a 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 ends at t_h) and gateway g_j as a *transmission opportunity* p_{jh}. A schedule assigns transmission opportunities to bundles. We formulate the optimal scheduling problem as a *minimum cost network flow* problem [2]. 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.

5.3.1 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 under the capacity constraints.

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 of 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 (all units of supply are removed) and the second ensures that at most one bundle is assigned to any transmission opportunity (demanding nodes receive no more than what they demand). 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 [2]. Note that, because we are considering a potentially infinite time horizon (we use as many future slots as we need to), there is always a feasible solution.

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 knows so far solves the problem in $O((m \log n)(m + n \log n))$ [2].

5.3.2 A More Efficient Formulation

The formulation presented in the previous subsection 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.

However, 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.[4]

Compared to the basic formulation, the input size for the same problem instance is dramatically reduced. For instance, 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.

5.3.3 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, some of which may be more desirable than others. The key insight is that, all other things being equal, we should send more 'urgent' bundles first, where a bundle is considered 'urgent' if its bus is leaving soon. By doing so, if new bundles arrive for a soon-to-depart bus, they can still be accommodated. Otherwise, when new bundles arrive, there may be too many that need to leave on soon-to-depart bus, so that some of them will miss the bus. That is, 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.

As we have seen, 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. Another factor affecting 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.

This motivates the definition of "urgency" with which kiosk k_i is in need of getting bundles to gateway g_j at time t as

$$u_{i,j}(t) = \frac{\displaystyle\int_t^{\infty} (D_{i,j}(\tau) - D_{i,j}(t)) e^{-\gamma(\tau - t)} d\tau}{\displaystyle\int_t^{\infty} e^{-\gamma(\tau - t)} d\tau} \qquad (1)$$

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.

With urgency defined, when we instantiate a schedule

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

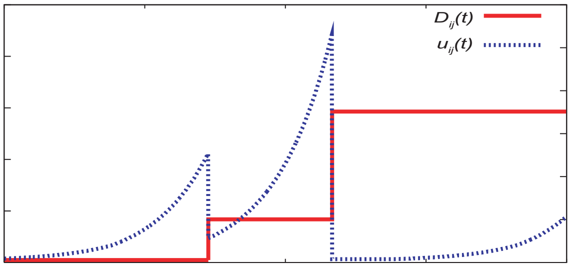

Figure 5: An example of urgency function $u_{i,j}(t)$ plotted on top of a delivery time function $D_{i,j}(t)$ from which it is derived.

class, 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.

5.3.4 *Work-Conserving vs. Non-Work-Conserving*

One 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 a utility-maximizing 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. Note that any non-work-conserving scheduler, like EDLQ, is prone to perform poorly in scenarios like the one shown in Figure 3.

So should the scheduler be work-conserving? Unfortunately, a naïve work-conserving scheduler may, to prevent a link from being idle, send a bundle to a 'bad' gateway that will delay a bundle a great deal, instead of waiting a little while later to send the bundle to another 'good' 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 guaranteed to be work-conserving. However, this can result in bundles being allocated to 'bad' gateways. To counteract this problem, 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 keep track of its estimated delivery time. We keep the bundle in the secondary buffer as long as resending it to another (currently busy) gateway could lead to an earlier delivery. We resend bundles from the secondary buffer when there are no eligible bundles left, so that the links would otherwise be idle. In this situation, we assign transmission opportunities to bundles in the secondary buffer that maximally reduce their delay. 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.

Such a work-conserving scheduler with a retransmission mechanism works 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. Note that the only eligible bundles are retransmitted, so this does not affect fairness.

6. 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.

6.1 Simulator and Simulation Setup

We developed a custom simulator which implements the model described in Section 2. Source code for the simulator can be found at [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 of real time. Each data point is obtained from running the simulation five times, and 95% confidence intervals are included. We use $U_i(x) = -x$ as the utility function for our algorithm. 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.

6.2 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 input parameter

Load	Frequency	Transit Delay	Phase Difference
0.45	12	60	180°

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

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 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 $\frac{1}{2} f_{1,j}$.

6.2.1 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 1. We use a filling rate of 0.5 token per step for the TB regulator for both kiosks.

In subsequent simulations, we vary the scenario from the base case along one of the four dimensions. Figure 6 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 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 7 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.

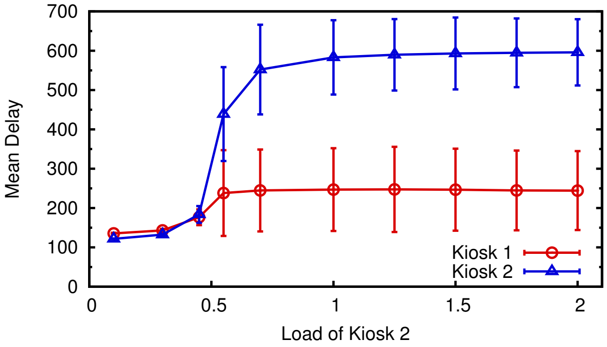

Figure 6: Delay using our scheme vs. load of kiosk k_2 in a single-gateway scenario

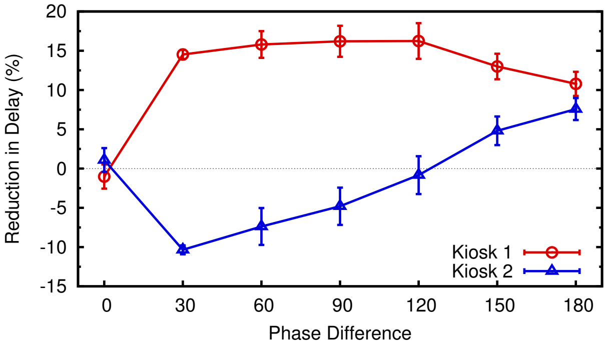

Figure 7: Reduction in delay vs. phase difference in a single-gateway scenario

Figure 8 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 9 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, it is less likely to miss one so our scheduler can often schedule other kiosks that would otherwise miss their buses.

6.2.2 Multiple Gateways

Now let's consider scenarios with two gateways, g_1

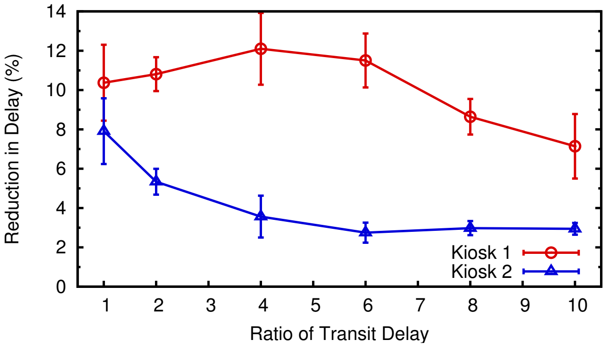

Figure 8: Reduction in delay vs. ratio of transit delay in a single-gateway scenario

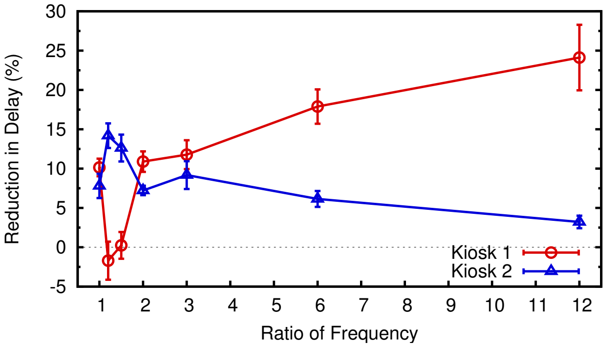

Figure 9: Reduction in delay vs. ratio of frequency in a single-gateway scenario

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. 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 10 and Figure 11. Again, we can see that kiosk k_1 is not affected by kiosk k_2 when the latter is ill-behaving. Also we can see that our scheme is still slightly better than EDLQ even though this is a favourable scenario for EDLQ.

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

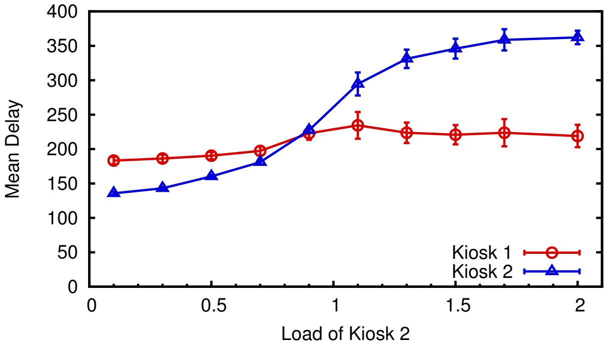

Figure 10: Delay using our scheme vs. load of kiosk k_2 in a scenario where each kiosk prefers a different gateway.

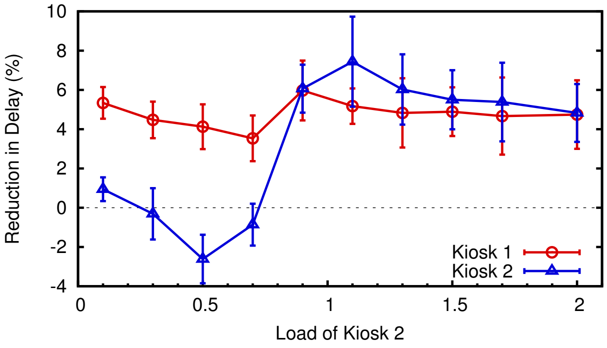

Figure 11: Reduction in delay vs. load of kiosk k_2 in a scenario where each kiosk prefers a different gateway

Next we consider a scenario where both kiosks prefer gateway g_1. The bus schedule information is shown in Table 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 12 and Figure 13. As expected, our scheme offers more improvement over EDLQ than in the previ-

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

Table 3: Bus schedule information. Both kiosks prefer the same gateway.

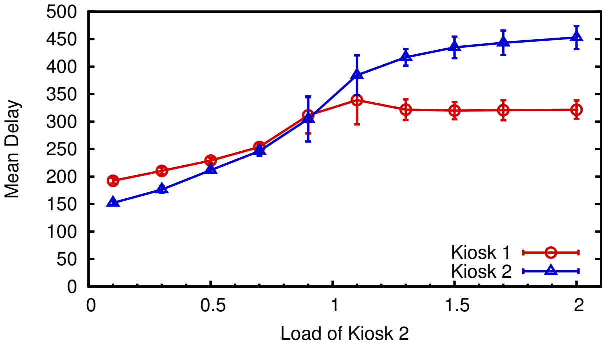

Figure 12: Delay using our scheme vs. load of kiosk k_2 in a scenario where both kiosks prefer the same gateway

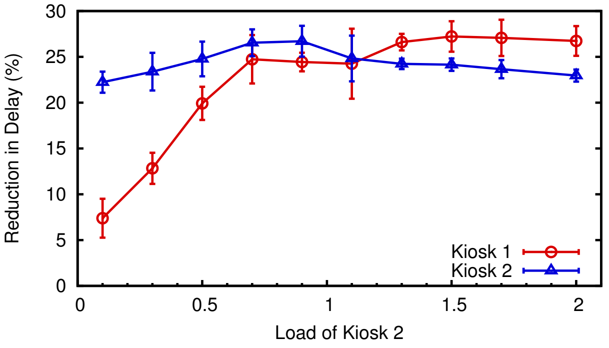

Figure 13: Reduction in delay vs. load of kiosk k_2 in a scenario where both kiosks prefer the same gateway

ous scenario. When the load of both kiosks is 0.9, both kiosks see a reduction in delay of about 25%.

6.3 A More Realistic Scenario

We now consider a more realistic scenario. Figure 14 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[5]. The detailed information about departure and arrival times is omitted due to space constraints, but can be found in [8].

We set the token bucket filling rate to 0.65 bundle

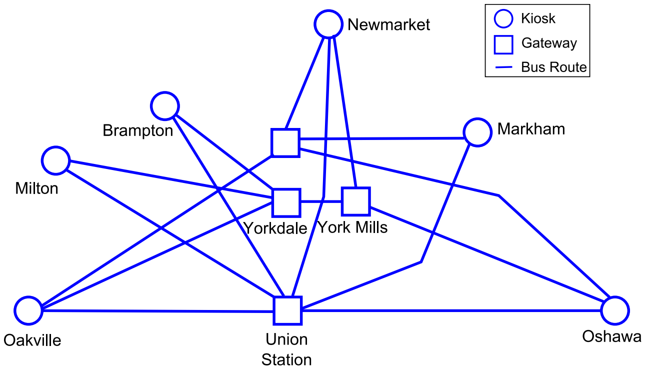

Figure 14: Selected part of the public transportation system of the Greater Toronto Area. Information about bus routes is taken from the published schedules of the Greater Toronto Transit Authority.

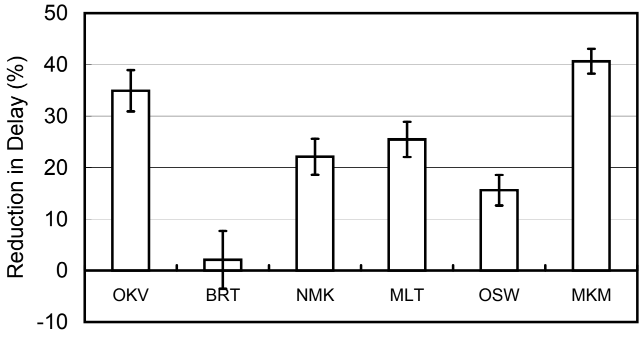

Figure 15: Reduction in delay in the GTA scenario

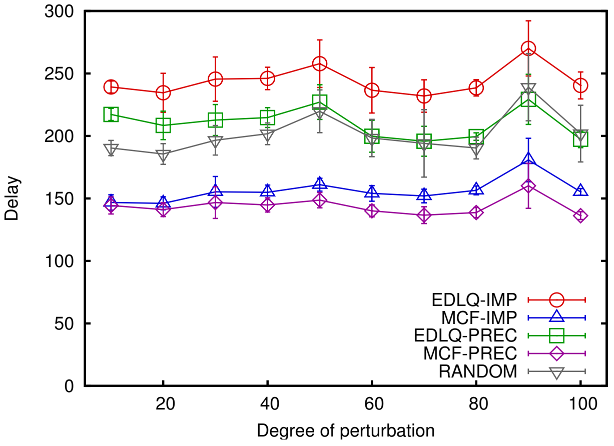

Figure 16: Markham in the GTA scenario with perturbed bus timing

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

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 15. 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.

6.4 Impact of Imprecise Schedules

We are aware that in reality buses never follow their schedules precisely. We did simulations where the actual timing of bus trips is randomly perturbed from their schedules and have some preliminary results about the impact of imprecise schedules on our algorithm. Due to space constraints we can only show some sample results here. More detail can be found in [8].

In Figure 16, 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 pulls a bundle from a randomly chosen per-kiosk queue whenever a proxy-gateway link becomes free, provided the chosen bundle can be delivered via the gateway in question. Note that the random scheme is work-conserving. Unsurprisingly, the gap between MCF-PREC and MCF-IMP widens as the degree of perturbation increases. However, it remain almost constant at the right end of the graph. This observation is consistent across all our other simulation results. It suggests that the impact of imprecise schedules on our algorithm is bounded.

7. RELATED WORK

Fair sharing of bandwidth in packet-switching networks has been well studied in the context of traditional networks. The well known max-min fairness criterion, is closely approximated by a number of packetized scheduling algorithms [20]. A fair scheduling algorithm that operates on multiple links [5] has also been recently proposed. Our work is different in two main aspects. First, our notion of fairness is defined on a longer time scale. While classical 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.

Opportunistic scheduling has been studied in the context of wireless data networks, 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 on utility modeling is inspired by research in Time/Utility Function (TUF) based scheduling [13] [17] [19] [7] 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. Other work considers 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 [7], 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) [22] and DieselNet [6]. 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 [21] [23]. DieselNet is a large-scale testbed of delay-tolerant networking consisting of 40 buses in Amherst, Massachusetts. They study problems including routing [6] [3], security, and power management [4]. 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 data ferrying network is a relatively easy problem.

8. CONCLUSIONS

We study the downlink scheduling problem in data-ferrying networks where the goals are fair allocation of bandwidth and end-to-end delay minimization. We show that EDLQ is inadequate because it does not provide bandwidth control, is greedy, and does not allow bundle reordering. We use token buckets for bandwidth control, decoupling the rate allocation and delay minimization optimization criteria. We achieve delay minimization with a utlity-maximizing scheduler that also allows us to overcome the problems faced by EDLQ. We show how to reduce the problem size, and then use urgency-based schedule selection to design an even better solution. Finally, by carefully balancing work-conserving and non-work-conserving aspects in our design, we are able to get the best of both worlds. We believe that the use of use of utility maximizing schedulers, and retransmission buffers to combine work- and non-work-conserving scheduling are both novel, and readily applicable to other systems. 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%.

9. REFERENCES

[1] Anonymous access to kiosknet code. *http://tinyurl.com/2z48op*, 2007.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

[3] A. Balasubramanian, B. N. Levine, and A. Venkataramani. DTN Routing as a Resource Allocation Problem. In *Proc. ACM SIGCOMM*, 2007.

[4] N. Banerjee, M. D. Corner, and B. N. Levine. An Energy-Efficient Architecture for DTN Throwboxes. In *Proceedings of IEEE INFOCOM*, May 2007.

[5] J. M. Blanquer and B. Özden. Fair queuing for aggregated multiple links. In *Proc. ACM SIGCOMM*, 2001.

[6] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *Proc. IEEE INFOCOM*, April 2006.

[7] H. Cho, H. Wu, B. Ravindran, and E. D. Jensen. On multiprocessor utility accrual real-time scheduling with statistical timing assurances. In *EUC*, pages 274–286, 2006.

[8] S. Guo. Algorithms and design principles for rural kiosk networks. Master's thesis, University of Waterloo, 2007.

[9] S. Guo, H. Falaki, E. Oliver, S. U. Rahman, A. Seth, M. Zaharia, and S. Keshav. Very low-cost internet access using kiosknet. *Proc. ACM SIGCOMM CCR*, October 2007.

[10] D. Hadaller, S. Keshav, T. Brecht, and S. Agarwal. Vehicular opportunistic communication under the microscope. In *Proc. MobiSys*, 2007.

[11] S. Jain, K. R. Fall, and R. K. Patra. Routing in a delay tolerant network. In *Proc. ACM SIGCOMM*, 2004.

[12] S. S. Kulkarni and C. Rosenberg. Opportunistic scheduling for wireless systems with multiple interfaces and multiple constraints. In *Proc. MSWIM '03*, 2003.

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

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

[15] Y. Liu and E. Knightly. Opportunistic fair scheduling over multiple wireless channels. In *Proc. IEEE INFOCOM 2003*.

[16] A. Pentland, R. Fletcher, and A. Hasson. A road to universal broadband connectivity. *Proc. 2nd International Conference on Open Collaborative Design for Sustainable Innovation*, 2002.

[17] B. Ravindran, E. D. Jensen, and P. Li. On recent advances in time/utility function real-time scheduling and resource management. In *Proc. ISORC*, pages 55–60, 2005.

[18] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *Proc. ACM MOBICOM*, 2006.

[19] J. Wang and B. Ravindran. Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis. *IEEE Trans. Parallel Distrib. Syst.*, 15(2):119–133, 2004.

[20] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proc. of the IEEE*, pages 1374–1396, 1995.

[21] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proc. MobiHoc*, 2004.

[22] W. Zhao and M. H. Ammar. Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks. In *Proc. FTDCS*, 2003.

[23] W. Zhao, M. H. Ammar, and E. W. Zegura. Controlling the mobility of multiple data transport ferries in a delay-tolerant network. In *Proc. IEEE INFOCOM*, 2005.