Report Number: WUCS-98-29

1998-01-01

TCP Dynamic Acknowledgment Delay: Theory and Practice

Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott

We study an on-line problem that is motivated by the networking problem of dynamically adjusting delays of acknowledgments in the Transmission Control Protocol (TCP). The theoretical problem we study is the following. There is a sequence of n packet arrival times A = and a look-ahead coefficient L. The goal is to partition A into k subsequences sigma1, sigma2, ...,sigmak (where a subsequence end is defined by an acknowledgment) that minimizes a linear combination of the cost for the number of acknowledgments sent and the cost for the additional latency introduced by delaying acknowledgments. At each arrival, an oracle... **Read complete abstract on page 2.**

TCP Dynamic Acknowledgment Delay: Theory and Practice

Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott

Complete Abstract:

We study an on-line problem that is motivated by the networking problem of dynamically adjusting delays of acknowledgments in the Transmission Control Protocol (TCP). The theoretical problem we study is the following. There is a sequence of n packet arrival times A = and a look-ahead coefficient L. The goal is to partition A into k subsequences sigma1, sigma2, ...,sigmak (where a subsequence end is defined by an acknowledgment) that minimizes a linear combination of the cost for the number of acknowledgments sent and the cost for the additional latency introduced by delaying acknowledgments. At each arrival, an oracle provides the algorithm with the times of the next L arrivals. For all the results of our paper, we describe how to incorporate other contraints to better match the true acknowledgment delay problem. We first define two different objective functions for measuring the cost of a solution, each with its own measure of latency cost. For each objective function we first given an O (nsquared)-time dynamic programming algorithm for optimally solbing the off-line problem. Then we describe an on-line algorithm that greedily acknowledges exactly when the cost for an acknowledgment is less than the latency cost incurred by not acknowledging. We show that for this algorithm there is a sequence of n packet arrivals for which it is Omega (squareroot(n))-competitive for the first objective function, 2-competitive for the second function for L = 0, and 1-competitive for the second function for L = 1. Next we present a second on-line algorithm which is a slight modification of the first that we prove is 2-competitive for both objective funcitons. Then for each objective function we give lower bounds on the competitive ration for any deterministic on-line algorithm. These results show that for each objective function, at least one of our algorithms is optimal. Finally, we give some initial empirical results using arrival sequences from real network traffic where we compare the two methods used in TCP for acknowledgment delay with our two on-line algorithms. In all cases we examine perforance with L = 0 and L = 1.

TCP Dynamic Acknowledgment Delay: Theory and Practice

Daniel R. Dooly, Sally A. Goldman and Stephen D. Scott

WUCS-98-29

December 1998

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

TCP Dynamic Acknowledgment Delay: Theory and Practice[*]

Daniel R. Dooly[†]
Dept. of Computer Science
Washington University
St. Louis, MO 63130-4899
drd1@cs.wustl.edu

Sally A. Goldman[†]
Dept. of Computer Science
Washington University
St. Louis, MO 63130-4899
sg@cs.wustl.edu

Stephen D. Scott[†]
Dept. of Computer Science and Engineering
University of Nebraska
Lincoln, NE 68588-0115
sscott@cse.unl.edu

WUCS-98-29

December 1998

Abstract

We study an on-line problem that is motivated by the networking problem of dynamically adjusting delays of acknowledgments in the Transmission Control Protocol (TCP). The theoretical problem we study is the following. There is a sequence of n packet arrival times $\mathcal{A} = \langle a_1, \ldots, a_n \rangle$ and a look-ahead coefficient L. The goal is to partition $\mathcal{A}$ into k subsequences $\sigma_1, \sigma_2, \ldots, \sigma_k$ (where a subsequence end is defined by an acknowledgment) that minimizes a linear combination of the cost for the number of acknowledgments sent and the cost for the additional latency introduced by delaying acknowledgments. At each arrival, an oracle provides the algorithm with the times of the next L arrivals. For all the results of our paper, we describe how to incorporate other constraints to better match the true acknowledgment delay problem.

We first define two different objective functions for measuring the cost of a solution, each with its own measure of latency cost. For each objective function we first give an

[*]An earlier version appears in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, 1998.

[†]Supported in part by NSF Grant CCR-9734940 and NSF NYI Grant CCR-9357707 with matching funds provided by Xerox PARC and WUTA.

$O(n^2)$-time dynamic programming algorithm for optimally solving the off-line problem. Then we describe an on-line algorithm that greedily acknowledges exactly when the cost for an acknowledgment is less than the latency cost incurred by not acknowledging. We show that for this algorithm there is a sequence of n packet arrivals for which it is $\Omega(\sqrt{n})$-competitive for the first objective function, 2-competitive for the second function for $L = 0$, and 1-competitive for the second function for $L = 1$. Next we present a second on-line algorithm which is a slight modification of the first that we prove is 2-competitive for both objective functions.

Then for each objective function we give lower bounds on the competitive ratio for any deterministic on-line algorithm. These results show that for each objective function, at least one of our algorithms is optimal.

Finally, we give some initial empirical results using arrival sequences from real network traffic where we compare the two methods used in TCP for acknowledgment delay with our two on-line algorithms. In all cases we examine performance with $L = 0$ and $L = 1$.

Keywords: Transmission Control Protocol (TCP), acknowledgment delay problem, on-line algorithms, competitive analysis, lookahead, Internet traffic simulations

1 Introduction

In this paper we study an on-line problem that is motivated by the networking problem of dynamically adjusting delays of acknowledgments in the Transmission Control Protocol (TCP) [15], a very popular protocol in use in the Internet. The specific problem we study is the following. There is a sequence of n packet arrival times $\mathcal{A} = \langle a_1, \ldots, a_n \rangle$. The goal is to partition $\mathcal{A}$ into k subsequences $\sigma_1, \sigma_2, \ldots, \sigma_k$ where a subsequence end is defined by an acknowledgment. We use σ_i to denote the set of arrivals in the partition and t_i to denote the time when the acknowledgment for σ_i is sent. To capture the fact that all arrivals must be acknowledged, we must have $k \geq 1$ and $t_k \geq a_n$, the time of the last arrival.

By delaying the acknowledgments for some packets we reduce the number of acknowledgments sent (and the associated costs) from n to k. However, delaying an acknowledgment adds to the latency costs[1], which we want to avoid since it can result in burstiness of TCP traffic, with potentially disastrous consequences [12]. Thus, one wants to optimally balance the costs of an acknowledgment with the costs associated with the increased latency. We model the tradeoff between the cost of an acknowledgment and the cost for the latency by using the objective function $\eta k + (1-\eta)\sum_{i=1}^{k} \nu_i$ where ν_i is a measure of the extra latency for the arrivals in subsequence σ_i. Section 3 describes our two measures of ν_i, which are appropriate for different circumstances. In an optimal solution t_i will always be when the last packet of σ_i arrives. The factor $\eta \in [0, 1]$ weights the relative importance of minimizing acknowledgments and minimizing latency, and would likely be set by a network administrator.

For the real networking problem, past traffic characteristics could be used to estimate

[1]In this paper we use latency to refer to the delay introduced by postponing the transmission of the acknowledgment message. There is additional latency introduced by the transmission of messages, but we do not affect this by delaying acknowledgments.

the time of the next arrival(s) which in turn could be used to decide when to send an acknowledgment, an idea endorsed by Clark [4]. But before pursuing this goal, we must first understand how useful it would be to develop learning algorithms to make a good prediction as to the next arrival time. Thus in this paper we study the performance of our acknowledgment delay algorithms when provided with an oracle that gives the exact time for the next L arrivals (we call L a *lookahead coefficient*). Namely, at the time of arrival a_i, the on-line algorithm is provided with the values for $a_{i+1}, \ldots, a_{i+L}$. For all $j > n$, we assume the oracle returns $a_j = \infty$.

This paper is organized as follows. The real problem of delaying TCP acknowledgments is more complex than our above description, so in Section 2 we discuss in more detail some of the other constraints of this problem. For each of the on-line algorithms that we present, we will first consider the simplified setting described above. However, we will then describe how each of our algorithms and their proofs can be extended to cover the other constraints required in a real TCP implementation. Additionally, each of our on-line algorithms runs in constant time per arrival, which is important given the large number of arrivals a TCP implementation must contend with in a short period of time.

Section 3 describes the two objective functions that we consider and the circumstances under which they are appropriate measures of performance. In Section 4, we describe the key differences between the on-line problem considered here and some classic on-line problems such as the rent-to-buy (or spin-block) problem [11, 9] and the snoopy caching problem [9]. Section 5 summarizes the results contained in this paper. All our results for our two objective functions appear in Sections 6 and 7. In each section, we present a dynamic programming algorithm for optimally solving the off-line version of the acknowledgment delay problem. We then describe algorithms $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ and give their competitive bounds for the cases of no lookahead and non-zero lookahead. We then present lower bounds on the competitive ratio. Finally, we close each section with initial empirical results using arrival sequences from real network traffic where we compare the two methods currently used in TCP for acknowledgment delay with our two on-line algorithms. In all cases we examine performance with $L = 0$ and $L = 1$. We conclude this paper in Section 8.

2 Motivation

Most TCP implementations used today employ some sort of acknowledgment delay mechanism. Delaying acknowledgments allows the TCP to acknowledge multiple incoming data segments with a single acknowledgment and sometimes to piggy-back the acknowledgment on an outgoing data segment [15, 12, 4]. By reducing the number of acknowledgments sent and received, we reduce bandwidth consumption and the overhead required to send and receive each acknowledgment (due to generating checksums, handling interrupts, etc.). So we want to minimize the number of acknowledgments sent, but not at the expense of adding excessive latency to the TCP connection.

Note that there is always space in a TCP packet header to indicate whether the current packet is also an acknowledgment and which packet(s) it acknowledges. Thus piggy-backing an acknowledgment on a packet costs us no extra bandwidth, since that space in the header would have been otherwise unused [15]. If an acknowledgment is not piggy-backed, then a new packet, to be used only as an acknowledgment (otherwise known as a "pure ack" [17, 12]) must be created. However, after creating the packet for the pure ack, there is no extra cost for acknowledging multiple packets with it.

TCP implementations, if they delay acknowledgments at all, use one of two approaches [15, 12]. Solaris TCP uses an interval timer of 50 ms. When a packet arrives, a timer is started that expires in 50 ms at which point an acknowledgment is sent for all currently unacknowledged packets. The BSD-derived TCPs use a 200 ms "heartbeat" timer. Here the timer fires every 200 ms independent of the arrivals of the packets. When the timer fires, all outstanding packets are acknowledged.

One of our research goals for the acknowledgment delay problem is to use prior traffic patterns to learn a rule for predicting when the next arrival will be. Clark [4] speculated that such a predictor could be very useful in dynamically adjusting the acknowledgment delay timer. However, having a very good prediction for the next arrival is only valuable if that information could help in deciding whether or not to delay the acknowledgment. Thus it is important to understand how having perfect knowledge of the next arrival time (i.e. a lookahead of $L = 1$) affects the performance that can be achieved. Similar arguments can be made for studying larger lookahead values.

To model the fact that different applications vary in the cost of an acknowledgment versus the cost of increased latency, we introduce an objective function, $f(k, \nu) = \eta k + (1 - \eta)\nu$, to measure the performance where k is the total number of acknowledgments, ν is the total extra latency[2], and η is a factor from $[0, 1]$ that weights the relative importance of minimizing acknowledgments and minimizing latency. An algorithm for this problem receives a sequence of arrival times $\mathcal{A} = \langle a_1, \ldots, a_n \rangle$ interspersed with a sequence of departure times $\mathcal{D} = \langle d_1, \ldots, d_m \rangle$ and schedules a sequence of acknowledgments that minimizes $f(k, \nu)$ for a given η. The following restrictions apply to the solution. (1) The arrival of a *rush* packet requires an immediate acknowledgment and a rush departure requires immediate transmission. A packet is rush if it falls into one of several categories such as packets that are used in the maintenance of a connection like a SYN or FIN [12, 7, 4]. (2) The maximum allowed delay of any acknowledgment and any departure is $\tau \leq 500$ ms [7]. (3) Each departure can only be combined with acknowledgments (i.e. two departures cannot be combined), and if a pending departure is a non-rush departure then we may choose to delay the departure if we believe that another packet is about to arrive (and hence we can piggyback the acknowledgment on the departing packet). However, we are not allowed to delay any departure d_i past the point when a later departure (d_j, $j > i$) is transmitted, i.e. the departures must be transmitted in the order of their *ready times* given in $\mathcal{D}$. We assume that the latency from delaying a non-rush departure is equivalent to the latency from delaying the acknowledgment of an arrival.

Modifying an algorithm to handle the first restriction (immediate acknowledgment of rush packets and immediate transmission of rush departures) is trivial. Because there is no choice as to whether or not to immediately acknowledge or transmit a rush packet (and hence end the subsequence), the rush packets simply partition the original problem into a set of subproblems for which our provided algorithms can be applied. Handling the other two

[2]The time units used for measuring latency would nominally be the same as those used in the system under consideration (for time stamping, etc.).

restrictions depends on the particular algorithm. In presenting our algorithms, we initially ignore the maximum delay restriction. We also assume that all departures are rush (requiring immediate transmission), so we can visualize $\mathcal{A}$ as being partitioned into subsequences, where each subsequence is delimited by a departure. Hence we can use these departure-free subsequences as inputs to our algorithms, so we introduce our algorithms assuming there is no maximum delay constraint and there are only rush departures. We later argue that our algorithms can be easily modified to accommodate the general setting and that our on-line algorithms' decisions as to when to send an acknowledgment (or departure) can be made using constant time per arrival.

A possible impediment to implementing our on-line algorithms is the need to perform more maintenance on timers in the operating system than we would in the schemes currently used in TCP implementations. This would normally be very expensive, but is mitigated by the work of Costello and Varghese [6] who reimplemented the BSD kernel to improve the overhead of maintaining timers from linear in the number of timers to constant time. This of course requires the use of the new kernel, but it shows that the requirement of extra timer maintenance need not be a severe problem.

3 Our Objective Functions

As discussed in the introduction, to model the networking problem of dynamically adjusting delays of acknowledgments in TCP, we selected an objective function (under two different ways of measuring latency) to trade off the cost per acknowledgment with the latency cost incurred due to the delay in acknowledging a packet. Clearly, the choice of the objective function affects the decision made by an optimal algorithm. In this section, we define our generic objective function that weights acknowledgment cost with the cost of adding latency. We then explore two definitions of latency cost for use in our generic objective function.

Our generic objective function is as follows:

$$f(k,\nu) = \eta k + (1-\eta)\sum_{i=1}^{k} \nu_i \tag{1}$$

where k is the number of acknowledgments sent, ν_i is a measure of the total extra latency for subsequence σ_i, and $\eta \in [0,1]$ is the factor that weights the cost of acknowledging versus the cost of waiting. Our two methods for measuring ν_i yield two specific objective functions, each of which is a special case of the following.

Definition 1

$$f_{lat} \doteq \eta k + (1-\eta)\sum_{i=1}^{k} lat_i(\cdot)$$

where $lat_i(\cdot)$ is any function that increases as an acknowledgment is further delayed.

We assume that lat_i is invertible, which is required by our algorithms to operate. It could be the case that lat_i is a *series* of functions $\langle lat_{i1}(\cdot), lat_{i2}(\cdot), \ldots, lat_{im}(\cdot)\rangle$ where $lat_{ij}(t)$ is the total latency accumulated from the first arrival of subsequence σ_i to time t when arrival

a_j is the most recent arrival added to σ_i. If this is the case then we require $lat_{ij}(t) < lat_{i\ell}(t)$ for all $\ell > j$ and $a_j, a_\ell \in \sigma_i$. This way latency for a given subsequence increases as more time passes and as more packets arrive. So f_{lat} not only includes the two functions we study in this paper, but any other reasonable[3] objective function of the form of Equation 1.

Suppose that each arrival $a_j \in \sigma_i$ was associated with a different computation being performed. Then in this case, the cost introduced to the entire system due to the extra latency would be the sum of the latency costs introduced for each of the computations. This motivates our first definition of lat_i, which is the sum of all the extra latencies of the packets in σ_i. This leads us to our first objective function f_{sum}.

Definition 2

$$f_{sum} \doteq \eta k + (1-\eta)\sum_{i=1}^{k}\sum_{a_j\in\sigma_i}(t_i - a_j),$$

where t_i is the time of acknowledgment i.

Note that f_{sum} is related to the average latency per packet over the entire sequence: simply divide the second term by n.

Now suppose instead that each arrival $a_j \in \sigma_i$ was associated with the *same* computation. In this case, the extra latency caused by delaying acknowledgments is best modeled by setting lat_i to be the maximum of all the extra latencies of the packets in σ_i. This leads us to our second objective function f_{max}.

Definition 3

$$f_{max} \doteq \eta k + (1-\eta)\sum_{i=1}^{k}\max_{a_j\in\sigma_i}(t_i - a_j),$$

where t_i is the time of acknowledgment i.

Of course, $\max_{a_j\in\sigma_i}(t_i - a_j)$ is always $t_i - a_{first_i}$ where a_{first_i} is the first arrival of σ_i.

Note that the only difference between the two objective functions is the cost per unit time of the extra latency for each σ_i. For f_{max}, this cost is linear in time with slope $(1-\eta)$. For f_{sum}, this cost is *piecewise-linear* in time with slope $(1-\eta)|\sigma_i|$, where $|\sigma_i|$ increases by 1 with each new arrival until an acknowledgment is sent.

4 Related Work

We use standard competitive analysis [14, 10, 3] to evaluate our algorithms. Specifically, let C_{opt} be the cost (for acknowledgments and latency) of an optimal solution and let C_A be the cost of the solution produced by on-line algorithm A. Then we say that A is α-competitive

[3]It seems unlikely that any reasonable objective function would remain constant or decrease when an acknowledgment is further delayed. This would be tantamount to rewarding an algorithm for indefinitely delaying an acknowledgment.

if for all sequences of arrivals, $\mathcal{C}_A \leq \alpha \mathcal{C}_{opt}$. We are not including an additive factor as is often done[4].

Our problem is a generalization of the rent-to-buy problem, which has also been called the spin-block problem [11, 9]. In the rent-to-buy problem the on-line algorithm is told that a given item can be rented at 1 dollar per day or bought for a cost of K dollars. The item will be needed for d days where d is *not known* to the on-line algorithm. At the beginning of each day the on-line decision is whether to buy the item or to rent it for another day. Clearly if $d \geq K$ the optimal solution is to buy the item at the start for a cost of K. Otherwise ($d < K$), the optimal solution is to rent for a cost of d. The rent-to-buy problem can be reduced to the acknowledgment delay problem (for either f_{sum} or f_{max}) as follows. Suppose that there are going to be only two arrivals, the first at time 0 and the second at time $a = d/(1-K)$. Let the cost for an acknowledgment $\eta = K$, the cost to buy. Furthermore, suppose that the cost of the solution does not include the final acknowledgment required at time a. Then the cost to buy at time t is equivalent to the cost for the acknowledgment delay solution in which an acknowledgment is sent at time $t/(1-K)$. Similarly, the cost to rent the entire d days is equivalent to the acknowledgment delay solution in which there is no acknowledgment before the second arrival.

The acknowledgment delay problem we study extends the rent-to-buy problem in three ways. First, we consider an arbitrary sequence of arrivals versus just two arrivals. Second, we consider when the on-line algorithm can have any constant lookahead factor L. Third, we consider the on-line problem in the presence of a maximum delay (i.e. maximum rent time) constraint. Additionally, while for the rent-to-buy problem the off-line problem is trivially solved, for the acknowledgment delay problem with objective function f_{sum}, finding an optimal solution in the off-line case is non-trivial, even when ignoring the maximum delay constraint.

Another well-studied on-line problem is the snoopy caching problem [9]. This problem is also a generalization of the rent-to-buy problem. However, the generalization there is orthogonal to what we consider here. Finally, there are some problems in which the on-line problem with lookahead has been studied. For example, Yeh et al. [18] study the on-line disk scheduling problem where one can look ahead at the next k variables that are to be read and from that knowledge one wants to pick the order in which to read the variables from the disk to minimize the seek start-up time. Also, Grove [8] presents on-line algorithms for the bin packing problem when the algorithm can look into the future up to some total cost, rather than a number of items into the future. While these papers allow the on-line algorithms to have some knowledge about the future, the problems themselves are very different from the acknowledgment delay problem.

Finally, Ben-David and Borodin [2] studied the problem of look ahead for the k-server problem. They showed that any finite look-ahead L could not improve the competitive ratio by simply repeating each request in the request sequence L times. While we obtain the same type of result for the our problem, a more involved lower bound is required. In their paper, Ben-David and Borodin also propose a different competitive measure to study issues like lookahead and memory usage. In particular they propose what they call a Max/Max

[4]Frequently an algorithm A is said to be α-competitive if $\mathcal{C}_A \leq \alpha \mathcal{C}_{opt} + c$ where c is a constant that is independent of $\mathcal{C}_{opt}$.

ratio which is the supremum over all sequence lengths ℓ of the amortized cost of the on-line algorithm divided by the amortized cost of the optimal algorithm.

5 Summary of Results

Section 6 gives our results for f_{sum} and Section 7 gives our results for f_{max}. In each section, we first give a dynamic programming algorithm for the off-line problem of finding an optimal partition when the algorithm knows $\langle a_1, \ldots, a_n \rangle$ at the time of the first arrival. The off-line algorithms work even with non-rush departures and a maximum delay constraint. We then consider the on-line problem.

In general, our on-line algorithms work as follows. After the jth arrival (at time a_j) an *alarm* is set for time $a_j + t$ (where we vary the method for selecting t). For $L = 0$ (i.e. no lookahead), we wait until the next arrival or the alarm, whichever comes first. If $a_{j+1} \leq a_j + t$ (i.e. the next arrival comes first) then we reset the alarm (possibly using a new value of t). Otherwise, at time $a_j + t$ (i.e. the alarm time) we acknowledge all outstanding arrivals, ending the current subsequence. For $L \geq 1$ we can use knowledge of a_{j+1} to potentially reduce the latency. Specifically, if $a_{j+1} > a_j + t$, we immediately send an acknowledgment at time a_j (versus unnecessarily waiting until $a_j + t$). Otherwise, we wait until arrival a_{j+1} and repeat. By changing the method for selecting t we get algorithms with very different behaviors.

Our two on-line algorithms, **greedy**$_{tot}$ and **greedy**$_{new}$, use different methods to select t. Algorithm **greedy**$_{tot}$ sets its alarm such that the cost for sending an acknowledgment is equal to the (total) latency cost of waiting time t until the alarm. By picking t in this way one would perform optimally (for lookahead $L = 1$) if the next arrival were the last one. We give an arrival sequence $\mathcal{A}$ of n arrivals for which **greedy**$_{tot}$ is $\Omega\left(\sqrt{n}\right)$-competitive (even for $L = 1$) under f_{sum}. Under f_{max}, we prove that **greedy**$_{tot}$ has a competitive ratio of 2 for $L = 0$ and a competitive ratio of 1 for $L = 1$ if τ is ignored (i.e. there is no maximum delay constraint). If τ is considered, then **greedy**$_{tot}$'s competitive ratio varies between 1 and 2 for $L = 1$ and between 2 and 3 for $L = 0$.

On-line algorithm **greedy**$_{new}$ selects t so that the cost of the (new) latency incurred from the last acknowledgment until the alarm is equal to the cost of one acknowledgment. This method of selecting t is similar to the algorithm of Karlin et al. [9, 10] for the rent-to-buy problem in which their on-line algorithm waits exactly long enough such that the cost of waiting ("renting") equals the cost of committing ("buying"). We prove that even when $L = 0$, **greedy**$_{new}$ is 2-competitive under f_{lat}, implying that it is 2-competitive under both f_{sum} and f_{max}. When $L = 1$, **greedy**$_{new}$'s competitive ratio does not change under f_{sum} and f_{max}, so in the worst case **greedy**$_{tot}$ is superior to **greedy**$_{new}$ under f_{max} and **greedy**$_{new}$ is superior to **greedy**$_{tot}$ under f_{sum}.

While it may seem that having $L \geq 1$ would enable an algorithm to obtain a better competitive ratio, we show that this is not the case under f_{sum}. Let $\mathcal{C}_{opt}$ be the cost of an optimal solution and let $\mathcal{C}_A$ be the cost of the solution produced by any deterministic on-line algorithm A. We prove that A with any constant lookahead L has competitive ratio $\mathcal{C}_A \geq 2\mathcal{C}_{opt} - c$, where c is a factor that can be made arbitrarily small with respect to $\mathcal{C}_{opt}$. Thus under f_{sum}, **greedy**$_{new}$ with $L = 0$ is the best possible in the worst case even for on-line algorithms that can use a constant lookahead. We also show that for $L = 0$, there is a worst-

Table 1: Summary of the results of this paper.

	f_{sum}		f_{max}	
	$L=0$	$L=1$	$L=0$	$L=1$
greedy$_{tot}$	$\Omega(\sqrt{n})$†	$\Omega(\sqrt{n})$†	2‡	1‡
greedy$_{new}$	2	2	2	2
Lower Bound	2†	2†	2†	1‡

†If τ is ignored.
‡If τ is ignored or $\tau < \eta/(1-\eta)$. For $\tau \geq \eta/(1-\eta)$, under f_{max} **greedy**$_{tot}$ with $L=1$ is $(1+\eta/(\tau(1-\eta)))$-competitive and **greedy**$_{tot}$ with $L=0$ is $(1+2\eta/(\tau(1-\eta)))$-competitive.

case lower bound of 2 on the competitive ratio under f_{max}. Our results are summarized in Table 1.

Although in the worst case having a lookahead of 1 does not help for f_{sum}, in practice it is likely to reduce the latency cost. Hence for each objective function we give some initial empirical results using arrival sequences from real network traffic where we compare the two methods used in TCP for acknowledgment delay with **greedy**$_{tot}$ and **greedy**$_{new}$ under both objective functions. In all cases we examine performance with $L=0$ and $L=1$.

6 Objective Function f_{sum}

Recall Definition 2 of Section 3, which defined f_{sum}:

$$f_{sum} \doteq \eta k + (1-\eta)\sum_{i=1}^{k}\sum_{a_j \in \sigma_i}(t_i - a_j),$$

where $\mathcal{A} = \langle a_1, \ldots, a_n \rangle$ is the sequence of packet arrivals, k is the number of acknowledgments, the acknowledgment for the packets in σ_i comes at t_i, and $\eta \in [0,1]$ weights the importance of minimizing acknowledgments with that of minimizing extra latency. We now present our algorithms for the acknowledgment delay problem under f_{sum}.

6.1 An Off-Line Algorithm Under f_{sum}

In this section we consider the off-line problem in which $\mathcal{A} = \langle a_1, \ldots, a_n \rangle$ is known. The goal is to minimize f_{sum}. Note that t_i, the time of the acknowledgment ending σ_i, is exactly the time of the last arrival in σ_i. For such a partition the objective function is

$$\begin{aligned} f_{sum} &= \eta k + (1-\eta)\sum_{i=1}^{k}\sum_{a_j \in \sigma_i}(t_i - a_j) \\ &= \eta k + (1-\eta)\sum_{i=1}^{k}\left(|\sigma_i| t_i - \sum_{a_j \in \sigma_i} a_j\right) \end{aligned}$$

```
Off-Line-1
1  Initialize $M_{\min}[0] \leftarrow 0$
2  Initialize $M[1,1] \leftarrow 1 \cdot \eta + (1-\eta) \cdot a_1$
3  Initialize $M_{\min}[1] \leftarrow M[1,1]$
4  Initialize $M_{\mathrm{pt}}[1] \leftarrow 1$
5  For $i \in [2,n]$
6       $M_{\min}[i] \leftarrow \infty$
7       For $j \in [1,i]$
8            $M[i,j] \leftarrow 1 \cdot \eta + (1-\eta) \cdot j \cdot a_i + M_{\min}[i-j]$
9            If $M[i,j] < M_{\min}[i]$ then
10               $M_{\min}[i] \leftarrow M[i,j]$
11               $M_{\mathrm{pt}}[i] \leftarrow j$
```

Figure 1: An optimal off-line algorithm that runs under f_{sum}. Note that we can implement this algorithm without using the array M, but we retain it for clarity.

$$\begin{aligned} &= \eta k + (1-\eta)\left(\sum_{i=1}^{k} |\sigma_i| t_i - \sum_{i=1}^{k} \sum_{a_j \in \sigma_i} a_j\right) \\ &= \eta k + (1-\eta)\left(\sum_{i=1}^{k} |\sigma_i| t_i - \sum_{i=1}^{n} a_i\right). \end{aligned}$$

Since the last term is independent of the algorithms' actions it suffices to optimize $f'_{sum} = \eta k + (1-\eta)\sum_{i=1}^{k} |\sigma_i| t_i$. We use dynamic programming to obtain an $O(n^2)$-time, optimal off-line algorithm. Let $M[i,j]$ for $i, j \in [1,n]$ be the minimum cost solution (using f'_{sum}) for the subsequence $\langle a_1, \ldots, a_i \rangle$ (i.e. there is an acknowledgment just after a_i) when the second-to-last acknowledgment of this subsequence is just after a_{i-j}. If $i = j$ then the acknowledgment after a_i is the only one of this subsequence, and the acknowledgment (indicated by a $|$) is placed as follows: $\langle a_1, \ldots, a_i | \rangle$, with no acknowledgments between a_1 and a_i. Otherwise (if $j < i$) the acknowledgments are placed as follows: $\langle a_1, \ldots, a_{i-j}, | a_{i-j+1}, \ldots, a_i | \rangle$, possibly with acknowledgments between a_1 and a_{i-j}. To improve the algorithm's efficiency, we will maintain row i's minimum. Let $M_{\min}[i]$ be the minimum objective value of row i and let $M_{\mathrm{pt}}[i]$ be a value of j such that $M[i,j] = M_{\min}[i]$. The complete algorithm is shown in Figure 1.

The final solution consists of tracing back through the M_{pt} values and placing acknowledgments. The cost of this final (optimal) solution is $f'_{sum} = M_{\min}[n] = \min_{\ell \in [1,n]} M[n,\ell]$. We can then subtract $(1-\eta)\sum_{i=1}^{n} a_i$ from this quantity to get the true cost as measured by f_{sum}.

Theorem 4 *Algorithm* **Off-Line-1** *produces an optimal solution to the acknowledgment delay problem under f_{sum} when all departures are rush and there is no maximum delay constraint.*

Proof: It is well-known that if a problem possesses the *optimal substructure property*, then any dynamic programming algorithm that explores all subproblems is an optimal algorithm for that problem [5]. A problem exhibits the optimal substructure property if an optimal solution to the problem contains within it optimal solutions to subproblems. To see that the acknowledgment delay problem possesses the optimal substructure property, note that if an oracle tells us that the subsequence $\langle a_1, \ldots, a_i | \rangle$ has its second-to-last acknowledgment after arrival a_{i-j} in an optimal solution, then we merely need to optimize the subsequence $\langle a_1, \ldots, a_{i-j} | \rangle$ to obtain an optimal solution. It is also obvious from Figure 1 that our algorithm explores all possible subproblems. □

In the case when not all departures are rush (i.e. departures can be delayed), a simple extension to the above algorithm yields an optimal off-line algorithm. First we merge $\mathcal{A}$ and $\mathcal{D}$ into a single supersequence with $n+m$ entries. Now i and j are indexes into a $(n+m) \times (n+m)$ table. Since latency for departures counts the same as for arrivals, the only change to the algorithm is how many transmissions can occur in a subsequence. So simply change the update step of Line 8 from

$$M[i,j] \leftarrow 1 \cdot \eta + (1-\eta) \cdot j \cdot a_i + M_{\min}[i-j]$$

to

$$M[i,j] \leftarrow numdepartures(i, i-j) \cdot \eta + (1-\eta) \cdot j \cdot a_i + M_{\min}[i-j]$$

where $numdepartures(i, i-j)$ counts the number of rows between rows i and $i-j+1$ (inclusive) that correspond to departures in the supersequence, including the acknowledgment sent for row i if that row corresponds to an arrival and not a departure. Note that if all departures are rush, then we have no departure sequence and *numdepartures* is always 1, as indicated in Figure 1. We now make an observation about the departure sequence that we will use later. Note that this observation applies generically to f_{lat} (Definition 1), so it holds for both f_{sum} and f_{max}.

Observation 5 *No more than one departure will ever lie in any subsequence in an optimal solution under f_{lat}.*

Proof: By contradiction. Assume there exists an optimal solution in which multiple departures lie in the same subsequence. Then we could split that subsequence into multiple subsequences, decreasing latency without increasing transmission count. □

By Observation 5, in our off-line algorithm we could assign ∞ to all $M[i,j]$ for which $numdepartures(i, i-j) > 1$. We now further modify our algorithm to handle the maximum delay restriction. Namely, no arrival's acknowledgment or a departure may be delayed by more than τ. We enforce this constraint by altering the update step to assign ∞ to all $M[i,j]$ for which $x_i - y_{i-j+1} > \tau$ for $x, y \in \{a, d\}$. Finally, we note that by evaluating the function *lat* on Lines 2 and 8 of Figure 1, we get an algorithm that works under f_{lat}.

Corollary 6 *There is an $O(n^2)$ dynamic programming algorithm that produces an optimal solution to the acknowledgment delay problem under f_{lat} with non-rush departures and a maximum delay constraint.*

6.2 Algorithm $\mathbf{greedy}_{tot}$ Under f_{sum}

Our first deterministic on-line algorithm $\mathbf{greedy}_{tot}$ will balance the cost of acknowledging immediately with the cost of delaying the acknowledgment for some time $t_{\mathbf{greedy}_{tot},f_{sum}}$. First assume that all departures are rush and there is no limit on individual delays. Thus $\mathbf{greedy}_{tot}$ algorithm need only greedily partition $\mathcal{A}$. Let a_j be the time of the packet that just arrived. Let σ be the set of unacknowledged arrivals just after time a_j (this includes the packet corresponding to a_j, so $|\sigma| \geq 1$). If the algorithm waits some time $t_{\mathbf{greedy}_{tot},f_{sum}}$ (i.e. until time $a_j + t_{\mathbf{greedy}_{tot},f_{sum}}$) before sending an acknowledgment, then it will incur extra cost due to latency of $(1-\eta)\,|\sigma| t_{\mathbf{greedy}_{tot},f_{sum}}$ since each packet in σ will incur $t_{\mathbf{greedy}_{tot},f_{sum}}$ units of latency. Also, it has already incurred a cost of $(1-\eta)\sum_{a_i\in\sigma}(a_j - a_i)$ due to past latency. Acknowledging immediately incurs a cost of η for the acknowledgment plus the same cost due to past latency. Thus on each new arrival a_j, $\mathbf{greedy}_{tot}$ sets an alarm at $a_j + t_{\mathbf{greedy}_{tot},f_{sum}}$ for $t_{\mathbf{greedy}_{tot},f_{sum}}$ such that the cost for σ if an acknowledgment is not sent at a_j (namely, $(1-\eta)\,|\sigma| t_{\mathbf{greedy}_{tot},f_{sum}} + (1-\eta)\sum_{a_i\in\sigma}(a_j - a_i)$) is equal to the cost for σ if an acknowledgment is sent at a_j (namely, $\eta + (1-\eta)\sum_{a_i\in\sigma}(a_j - a_i)$). Solving for $t_{\mathbf{greedy}_{tot},f_{sum}}$ yields

$$t_{\mathbf{greedy}_{tot},f_{sum}} = \frac{\eta}{|\sigma|\,(1-\eta)}.$$

If no new arrival occurs before the alarm sounds, then an acknowledgment is sent when the alarm sounds. Otherwise, the old alarm is deleted and a new one is set at $a_{j+1} + \eta/\left(|\sigma'|\,(1-\eta)\right)$ for $\sigma' = \sigma \cup \{a_{j+1}\}$. If $\mathbf{greedy}_{tot}$ has access to an oracle[5] that provides a_{j+1} at time a_j (i.e. $L = 1$), then $\mathbf{greedy}_{tot}$ will send an acknowledgment at a_j if and only if $(a_{j+1} - a_j) > t_{\mathbf{greedy}_{tot},f_{sum}}$.

It is straightforward to extend this algorithm to accommodate non-rush departures and a maximum delay constraint of τ. On the first arrival a_j of a new subsequence, the algorithm sets a second alarm at $a_j + \tau$. This alarm is not updated at new arrivals within the same subsequence. So an acknowledgment is sent whenever either alarm sounds, while making sure that there is only one departure per subsequence (Observation 5) and that all departures are sent in order of their ready times. Also, the time complexity per arrival needed to update $t_{\mathbf{greedy}_{tot},f_{sum}}$ is constant.

Unfortunately, contrary to intuition, $\mathbf{greedy}_{tot}$ can perform quite poorly on f_{sum} if τ is ignored (i.e. there is no maximum delay constraint).

Lemma 7 *If we ignore τ, then under f_{sum} there exists a sequence of arrivals that forces $\mathbf{greedy}_{tot}$'s competitive ratio to be $\Omega\left(\sqrt{n}\right)$ for $L = 0$ and $L = 1$.*

Proof: We first state the proof for $L = 0$ and then argue why it holds for $L = 1$.

It will be convenient to focus on the latency costs between the arrivals. Let a_j and a_{j+1} be the times of two consecutive arrivals and consider a packet that had arrived by time a_j and had not yet been acknowledged just prior to time a_{j+1}. Let ℓ_j denote the latency cost for such a packet for the time period from a_j to a_{j+1}. In other words, $a_{j+1} = a_j + \ell_j/(1-\eta)$.

[5]Recall that the oracle returns $a_j = \infty$ for all $j > n$.

The arrival sequence for the lower bound is given by $\ell_j = \eta/j$ for $1 \leq j \leq n-1$. So $a_{j+1} = \frac{\eta}{1-\eta} H_j$ where H_j is the jth harmonic number (we let $a_1 = 0$). Note that $\mathbf{greedy}_{tot}$ will keep pushing the alarm out to the point of the next arrival. Thus it will never be the case that

$$\frac{\ell_j}{1-\eta} = a_{j+1} - a_j > t_{\mathbf{greedy}_{tot}, f_{sum}} = \frac{\eta}{j(1-\eta)} = \frac{\ell_j}{1-\eta},$$

so the algorithm will not send an acknowledgment until a_n. Thus there will be j arrivals that incur the latency cost of ℓ_j. By adding the acknowledgment cost to the latency cost we get that

$$C_{\mathbf{greedy}_{tot}} = \eta + \sum_{j=1}^{n-1} (j\ell_j) = \eta + \sum_{j=1}^{n-1} \left(j \cdot \frac{\eta}{j} \right) = \eta \cdot n.$$

An optimal algorithm must do at least as well as the following where $n = k(k+1)/2$. Let there be $k-1$ partitions of the form σ_i, where σ_i includes $a_{i(i+1)/2}$ through $a_{((i+1)(i+2)/2)-1}$ for $1 \leq i \leq k-1$. Finally, there is a final subsequence with a_n. Thus there are k acknowledgments. We now compute the latency cost for σ_i. Since there are $i+1$ arrivals in σ_i there are i latency costs associated with the subsequence. Further, the jth latency cost in the subsequence affects j arrivals since that is the number of arrivals that have not been acknowledged. Thus

$$\begin{aligned} \text{latency cost for } \sigma_i &= \sum_{j=1}^{i} j \cdot \ell_{i(i+1)/2+j-1} \\ &= \sum_{j=1}^{i} j \cdot \frac{\eta}{i(i+1)/2 + j - 1} \\ &\leq \eta \sum_{j=1}^{i} \frac{j}{i(i+1)/2} = \eta. \end{aligned}$$

Therefore $C_{opt} \leq \eta + \sum_{j=1}^{k-1} \eta = k\eta$. So $\frac{C_{\mathbf{greedy}_{tot}}}{C_{opt}} \geq \frac{\eta \cdot k(k+1)/2}{\eta \cdot k} = (k+1)/2 = \Omega\left(\sqrt{n}\right)$.

For $L = 1$, recall that $\mathbf{greedy}_{tot}$ acknowledges arrival a_j immediately if and only if $(a_{j+1} - a_j) > t_{\mathbf{greedy}_{tot}, f_{sum}}$. Since the adversary places a_{j+1} at exactly $a_j + t_{\mathbf{greedy}_{tot}, f_{sum}}$, $\mathbf{greedy}_{tot}$ still will not send any acknowledgment until the end of the sequence (arrival a_n)[6], so its behavior for $L = 1$ on this arrival sequence is identical to that for $L = 0$. □

This bound also holds if not all departures are rush since the adversary need not place any departures at all into the sequence. However, introducing τ into the problem causes difficulties in the above proof since both $\mathbf{greedy}_{tot}$ and an optimal solution must send an acknowledgment before $a_j + \tau$ for all $a_j \in \mathcal{A}$. But we conjecture that the ratio can still grow with some function of n for sufficiently large τ.

[6]If instead $\mathbf{greedy}_{tot}$ chooses to acknowledge a_j immediately if $a_{j+1} - a_j = t_{\mathbf{greedy}_{tot}, f_{sum}}$, then the adversary can place arrival a_{j+1} at $a_j + t_{\mathbf{greedy}_{tot}, f_{sum}} - O(1/n)$. This will reduce $\mathbf{greedy}_{tot}$'s competitive ratio by a constant amount, but not affect its asymptotic growth.

Notice that the crux of the proof of Lemma 7 is that $\mathbf{greedy}_{tot}$ always sets its alarm at a point where the new extra latency of the current subsequence would be exactly $\eta/(1-\eta)$, as indicated in the following observation.

Observation 8 *Under f_{lat}, $\mathbf{greedy}_{tot}$ always sets its alarm at a point where the new extra latency of the current subsequence would be exactly $\eta/(1-\eta)$.*

Proof: Let $lat_{ij}(t)$ be the amount of latency accumulated from the start of subsequence σ_i to time t when arrival a_j is the last arrival to be placed in σ_i. So for example, for f_{sum} we use $lat_{ij}(t) = \sum_{a_\ell \in \sigma_i}(t - a_\ell)$. Then at arrival a_j $\mathbf{greedy}_{tot}$ sets its alarm at $a_j + t_{\mathbf{greedy}_{tot}, f_{lat}}$ for $t_{\mathbf{greedy}_{tot}, f_{lat}}$ such that

$$(1-\eta)(lat_{ij}(a_j + t_{\mathbf{greedy}_{tot}, f_{lat}}) - lat_{ij}(a_j) + lat_{ij}(a_j)) = \eta + (1-\eta)lat_{ij}(a_j).$$

Solving for $t_{\mathbf{greedy}_{tot}, f_{lat}}$ yields

$$t_{\mathbf{greedy}_{tot}, f_{lat}} = lat_{ij}^{-1}\left(\frac{\eta}{1-\eta} + lat_{ij}(a_j)\right) - a_j.$$

So the new alarm is set at $a_j + t_{\mathbf{greedy}_{tot}, f_{lat}} = lat_{ij}^{-1}(\eta/(1-\eta) + lat_{ij}(a_j))$. In words, the new alarm is set to where the total latency of the subsequence is the sum of the current accumulated latency and $\eta/(1-\eta)$. □

Finally, note that it is unlikely that an application in a real network will exhibit the harmonic behavior of Lemma 7. Thus we do not expect this algorithm to really perform this poorly. The simulation results of Section 6.5 lend more insight into this algorithm's true performance.

6.3 Algorithm $\mathbf{greedy}_{new}$ Under f_{sum}

We now consider when the on-line algorithm waits exactly long enough such that the cost of the latency incurred exactly equals the cost of one acknowledgment, but we do not add the cost of the accumulated latency to the acknowledgment cost. Specifically, on each new arrival a_j, our algorithm $\mathbf{greedy}_{new}$ sets an alarm at $a_j + t_{\mathbf{greedy}_{new}, f_{sum}}$ for $t_{\mathbf{greedy}_{new}, f_{sum}}$ such that $(1-\eta)\,|\sigma| t_{\mathbf{greedy}_{new}, f_{sum}} + (1-\eta)\sum_{a_i \in \sigma}(a_j - a_i) = \eta$. Solving for $t_{\mathbf{greedy}_{new}, f_{sum}}$ yields

$$t_{\mathbf{greedy}_{new}, f_{sum}} = \frac{\eta}{|\sigma|\,(1-\eta)} - \frac{\sum_{a_i \in \sigma}(a_j - a_i)}{|\sigma|}. \tag{2}$$

Thus the difference between $t_{\mathbf{greedy}_{new}, f_{sum}}$ and $t_{\mathbf{greedy}_{tot}, f_{sum}}$ is that in Equation 2 we subtract the average latency of the packets in σ. We now argue that for a given subsequence, the alarm will always remain fixed or move back in time, but never move forward. (In fact, what we show is that this happens for $\mathbf{greedy}_{new}$ for any function f_{lat} as in Definition 1.) This prevents the adversary from blowing up the total latency of a subsequence, which gave us the $\Omega\left(\sqrt{n}\right)$ lower bound of the first algorithm (Lemma 7).

Observation 9 *Under f_{lat}, once $\mathbf{greedy}_{new}$ sets its alarm for a given subsequence, that alarm will not move into the future for the duration of that subsequence. That is, $\mathbf{greedy}_{new}$ does not move its alarm into the future until an acknowledgment is sent. Moreover, the alarm is set such that the final total latency of the subsequence is exactly $\eta/(1-\eta)$.*

Proof: As in the proof of Observation 8, let $lat_{ij}(t)$ be the amount of latency accumulated from the start of subsequence σ_i to time t when arrival a_j is the last arrival added to σ_i. So for example, for f_{sum} we use $lat_{ij}(t) = \sum_{a_\ell \in \sigma_i} (t - a_\ell)$. Then at arrival a_j **greedy**$_{new}$ sets its alarm at $a_j + t_{\mathbf{greedy}_{new}, f_{lat}}$ for $t_{\mathbf{greedy}_{new}, f_{lat}}$ such that

$$(1-\eta)(lat_{ij}(a_j + t_{\mathbf{greedy}_{new}, f_{lat}}) - lat_{ij}(a_j) + lat_{ij}(a_j)) = \eta.$$

Solving for $t_{\mathbf{greedy}_{new}, f_{lat}}$ yields

$$t_{\mathbf{greedy}_{new}, f_{lat}} = lat_{ij}^{-1}\left(\frac{\eta}{1-\eta}\right) - a_j.$$

So the new alarm is at $t_{\mathbf{greedy}_{new}, f_{lat}} + a_j = lat_{ij}^{-1}(\eta/(1-\eta))$. In words, the alarm is set at a point at which the amount of latency accumulated from the first arrival of σ_i to the alarm (given the current value of σ_i) is exactly $\eta/(1-\eta)$. Since $lat_{ij}(t) \le lat_{i\ell}(t)$ for all $\ell \ge j$ and $a_j, a_\ell \in \sigma_i$ (see Section 3), we get $lat_{ij}^{-1}(\eta/(1-\eta)) \le lat_{i,j-1}^{-1}(\eta/(1-\eta)$. Therefore, the alarm will not move into the future until an acknowledgment is sent. □

We now prove that **greedy**$_{new}$ is 2-competitive under f_{lat}.

Theorem 10 *Under f_{lat} and with no lookahead, $\mathcal{C}_{\mathbf{greedy}_{new}} \le 2\mathcal{C}_{opt}$.*

Proof: Assume **greedy**$_{new}$ sends k acknowledgments in $\mathcal{A}$, partitioning $\mathcal{A}$ into k subsequences, which we will think of as closed intervals[7]. Each interval i starts with the first packet of σ_i and ends with σ_i's acknowledgment at time t_i (see Figure 2). Note that there could be some time between t_i and and the first arrival of σ_{i+1}. We ignore these time periods because they do not add to the cost of **greedy**$_{new}$'s solution. Observation 9 implied that the extra latency in each interval is exactly $\eta/(1-\eta)$, so the cost due to extra latency for each interval is exactly η. Since each acknowledgment costs η, $\mathcal{C}_{\mathbf{greedy}_{new}} = 2k\eta$.

Let k^* be the number of acknowledgments in an optimal partition of $\mathcal{A}$. First we consider the case when $k \le k^*$. In this case, it immediately follows that $\mathcal{C}_{opt} \ge k^*\eta \ge k\eta$. Thus $\mathcal{C}_{\mathbf{greedy}_{new}}/\mathcal{C}_{opt} \le 2$.

We now consider the case when $k > k^*$. Since all partitions of $\mathcal{A}$ must send an acknowledgment at the last arrival, at most $k^* - 1$ of the optimal acknowledgments are distributed over the first $k-1$ intervals from **greedy**$_{new}$'s partition. Thus at least $(k-1)-(k^*-1) = k-k^*$ intervals have no acknowledgments from the optimal solution lying in them. Each of these $k-k^*$ intervals contributes to the optimal solution η cost due to latency since the optimal solution must place an acknowledgment in an interval to not be charged for its latency. So the cost of this optimal solution has a cost of $k^*\eta$ for its acknowledgments and a cost of at least $(k-k^*)\eta$ for its latency. So $\mathcal{C}_{opt} \ge k\eta$, which is at least half of **greedy**$_{new}$'s cost. □

The proof of Theorem 10 assumed $L = 0$, so clearly if $L \ge 1$ then the competitive ratio is no worse since **greedy**$_{new}$ sets its alarms independent of L. Algorithm **greedy**$_{new}$ handles τ and non-rush departures the same way that **greedy**$_{tot}$ does (Section 6.2) and remains 2-competitive even with $L = 0$.

[7]An acknowledgment sent at time t_i acknowledges all arrivals that come at that time. So the start of interval $i+1$ is at some time $> t_i$. Thus the intervals are disjoint.

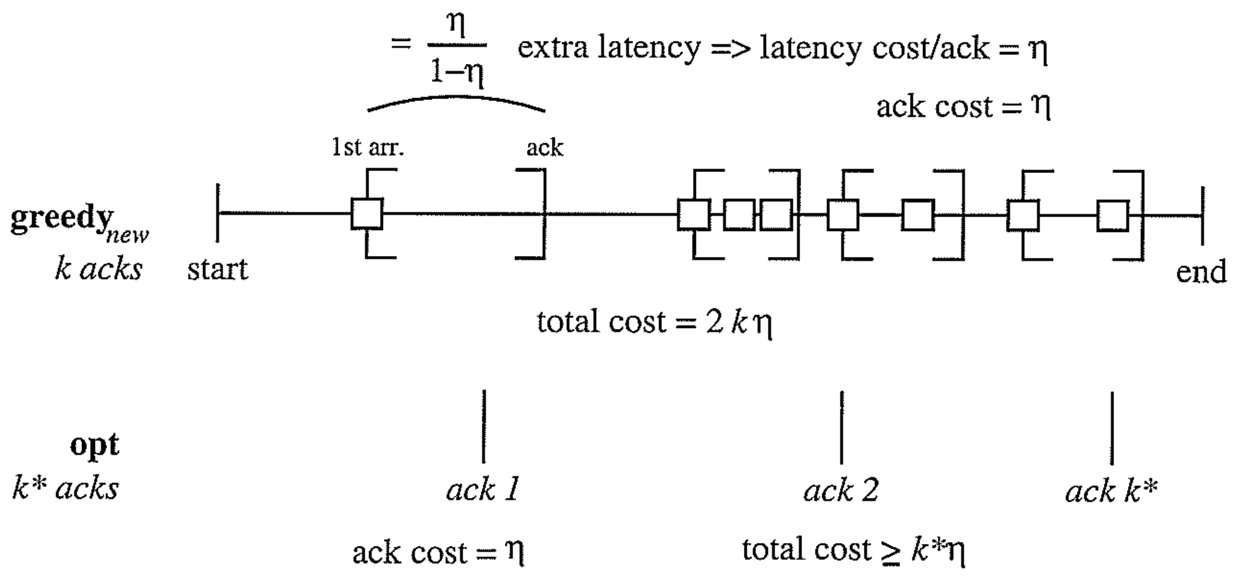

Figure 2: An example of a set of **greedy**$_{new}$'s intervals. Interval i starts with the first arrival of σ_i and ends with σ_i's acknowledgment at time t_i. According to Observation 9, the amount of extra latency per interval is exactly $\eta/(1-\eta)$, so **greedy**$_{new}$'s cost is exactly $2k\eta$. Also shown are the k^* acknowledgments from a hypothetical optimal solution.

Corollary 11 *Under f_{lat},* **greedy**$_{new}$ *with no lookahead remains 2-competitive in the presence of non-rush departures and a maximum delay constraint.*

Proof: Note that each of the first $k-1$ intervals created by **greedy**$_{new}$ was created because (1) cost due to total latency reached η, (2) the maximum delay τ for an individual arrival or departure was reached, or (3) more than one departure appeared in the interval. We will call these intervals *η-intervals*, *τ-intervals* and *d-intervals*, respectively, appearing k_η, k_τ, and k_d times. Again, interval i begins with the first arrival of σ_i and ends with an acknowledgment at time t_i. Note that each τ-interval must have an optimal acknowledgment lying in it since the optimal algorithm cannot allow any single delay to exceed τ. Also note that based on Observation 5, every d-interval contains an optimal acknowledgment. Finally, the kth interval (which can be of any type) must contain an optimal acknowledgment since all arrivals must be acknowledged. So at most $k^* - k_\tau - k_d$ η-intervals contain optimal acknowledgments, where again k^* is the number of acknowledgments in the optimal solution. Thus the optimal solution's cost is at least $(k - k^* + k_\tau + k_d)\,\eta + k^*\eta \geq k\eta$. But **greedy**$_{new}$'s solution is $\leq 2k\eta$ since each interval still must have $\leq \eta$ cost due to latency. $\square$

Finally, we note that **greedy**$_{new}$'s time complexity per arrival is constant under f_{sum}. To see this, recall the definition of $t_{\mathbf{greedy}_{new},f_{sum}}$ in Equation 2:

$$\begin{aligned} t_{\mathbf{greedy}_{new},f_{sum}} &= \frac{\eta}{|\sigma|\,(1-\eta)} - \frac{\sum_{a_i\in\sigma}(a_j - a_i)}{|\sigma|} \\ &= \frac{\eta}{|\sigma|\,(1-\eta)} - \frac{|\sigma|a_j - \sum_{a_i\in\sigma} a_i}{|\sigma|}. \end{aligned}$$

Maintaining a running sum of $\sum_{a_i \in \sigma} a_i$ yields the constant time cost required at each arrival.

6.4 A Lower Bound Under f_{sum}

We now prove that under f_{sum}, any deterministic on-line algorithm A with any constant lookahead L has a competitive ratio $\mathcal{C}_A \geq 2\,\mathcal{C}_{opt} - c$ where c can be made arbitrarily small with respect to $\mathcal{C}_{opt}$. In other words, $\mathcal{C}_A$ can be made arbitrarily close to twice that of optimal. Thus not only does **greedy**$_{new}$ perform as well as possible under f_{sum}, but one could not even obtain improvements (in the worst case) by increasing the lookahead by any constant amount.

Theorem 12 *Let A be any deterministic on-line algorithm for the acknowledgment delay problem with constant lookahead L. Then if we ignore the maximum delay constraint, under f_{sum} there exists an arrival sequence such that $\mathcal{C}_A \geq 2\,\mathcal{C}_{opt} - c$ where c can be made arbitrarily small with respect to $\mathcal{C}_{opt}$.*

Proof: The basic idea is that the adversary would like to have an arrival immediately after each acknowledgment by the on-line algorithm A (which knows the times for the next L arrivals).

We use a *burst* to denote a set of B arrivals arbitrarily close together where $B \gg L$. (We can make B arbitrarily larger than L.) Similarly, we use a *blip* to denote a set of L arrivals that occur arbitrarily close together. For ease of exposition we assume that all the arrivals in a burst or blip arrive at the same time. Let ϵ be a small constant, and let $t_\epsilon = \frac{\epsilon}{(1-\eta)B}$. The adversary begins with a burst at time 0 and a blip at time t_ϵ. There will be two possible scenarios at time t_ϵ. Either there was really just a blip (i.e. L arrivals) at time t_ϵ, or there was really a burst (i.e. B arrivals) at time t_ϵ and A has just seen the first L arrivals of the burst. If A chooses to send an acknowledgment at time 0 then the adversary places a burst at time t_ϵ. If A does not choose to send an acknowledgment at time 0 then the adversary leaves just a blip at time t_ϵ and places a new blip at time $2t_\epsilon$. In general, at time t the on-line algorithm A sees a blip at time $t + t_\epsilon$. If A acknowledges at time t then the adversary places a burst at time $t + t_\epsilon$. If A does not acknowledge at time t then the adversary leaves just a blip at time $t + t_\epsilon$ and places the next blip at time $t + 2t_\epsilon$ (see Figure 3). This continues until either A sends an acknowledgment or the latency cost for the current subsequence is η. In the latter case, we can force A to place an acknowledgment (or be worse than 2-competitive) by having the next blip far enough away that the latency cost would be prohibitive. Thus, without loss of generality, we assume that the latency cost of a subsequence is never larger than η. That is, the time between bursts is at most $\eta/(B(1-\eta))$.

We now compute a lower bound for the cost of the on-line algorithm's solution. Let $\sigma_1, \ldots, \sigma_s$ be the subsequences defined by A's solution. Let t_i be the time from the burst to the acknowledgment in σ_i. The cost of A's solution is at least the cost of the acknowledgments and the latency due to the bursts. Thus

$$\mathcal{C}_A \geq \left(\eta(s+1) + \sum_{i=1}^{s} t_i(1-\eta)B \right) = Z.$$

We now upperbound the cost of the optimal solution by averaging the costs of solution S_1 that acknowledges after the odd-numbered bursts, and the solution S_2 that acknowledges

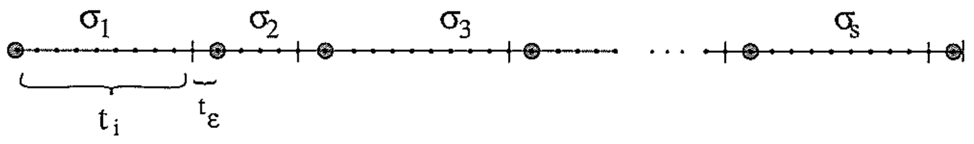

Figure 3: The arrival sequence for our lower bound. The subsequences $\sigma_1, \ldots, \sigma_i$ shown are those from the on-line algorithm A. Each vertical line represents an acknowledgment by A, each large gray circle is a burst (i.e. B arrivals for some large B) and each small black circle is a blip (i.e. L arrivals). At time t, A sees that L arrivals will come at time $t + t_\epsilon$ but does not know if this will be a blip or a burst.

after the even-numbered bursts. Both S_1 and S_2 must acknowledge after the final arrival. Thus there are $s + 2$ acknowledgments among both S_1 and S_2. Also, notice that each of the odd-numbered bursts is immediately acknowledged in S_1 and each even-numbered burst is immediately acknowledged in S_2. If S_1 (respectively, S_2) acknowledges immediately after a burst then S_1 (respectively, S_2) incurs no latency cost due to that burst. Thus for each subsequence σ_i, the burst latency incurred (by exactly the one of S_1 and S_2 that does not acknowledge immediately) is $t_i + t_\epsilon$ since there is an extra delay of t_ϵ between A's acknowledgment and the acknowledgment of S_1 (or S_2). Letting ℓ_{blips} be the latency costs incurred for all the blips, we get that

$$\begin{aligned}
\mathcal{C}_{opt} &\leq \frac{1}{2}\left(\eta(s+2) + \sum_{i=1}^{s}((t_i + t_\epsilon)(1-\eta)B) + \ell_{blips}\right) \\
&= \frac{\eta(s+1)}{2} + \frac{\eta}{2} + \frac{1}{2}\sum_{i=1}^{s} t_i(1-\eta)B + \frac{st_\epsilon(1-\eta)B + \ell_{blips}}{2} \\
&= \frac{Z}{2} + \frac{\eta}{2} + \frac{s\epsilon + \ell_{blips}}{2}.
\end{aligned}$$

We now argue that $c = \eta/2 + (s\epsilon + \ell_{blips})/2$ can be made arbitrarily small with respect to $\mathcal{C}_{opt}$. First, since the number of acknowledgments in the solution that led to our upperbound for $\mathcal{C}_{opt}$ was only one larger than that from $\mathcal{C}_A$, by increasing s we can make $\eta/2$ arbitrarily small with respect to $\mathcal{C}_{opt}$. Suppose that such an s has been selected and thus its value is now fixed.

We now consider the extra latency cost from c. These come from two sources. There is a cost of $c_1 = s\epsilon/2$ from the extra latency for the burst at each subsequence, and there is a cost of $c_2 = \ell_{blips}/2$ from the latency costs for the blips. We now argue that both of these quantities are independent of B and thus by making B sufficiently large, we can make $c_1 + c_2$ arbitrarily small with respect to $\mathcal{C}_{opt}$. Clearly c_1 is independent of B and thus can be made arbitrarily small with respect to $\mathcal{C}_{opt}$ by making B large. Since the latency cost for each subsequence is at most η and thus the duration is at most $\eta/(B(1-\eta))$, the number of blips in σ_i is at most $\left(\frac{\eta}{B(1-\eta)}\right)/t_\epsilon = \eta/\epsilon$, which is a constant and thus independent of B. So the stated result follows. □

6.5 Simulation Results Under f_{sum}

We ran preliminary simulations to empirically evaluate the benefits of lookahead and to contrast our algorithms' performances with those of the algorithms currently used by TCP implementations (see Section 2 for a brief description of these algorithms). The data for our simulations came from a trace of two hours of all wide-area TCP traffic between the Lawrence Berkeley Laboratory and the rest of the world [13]. This trace (LBL-TCP-3) is available at `http://ita.ee.lbl.gov/`. While the simulations are based on real data, for our initial simulations we simplified the traces. Specifically, we ignored τ and all departures, so our algorithms only processed one long arrival sequence with no limits on individual acknowledgment delays. We also filtered out connections that did not fit our definition of a "normal" TCP connection, e.g. connections that were reset before shutting down normally. Finally, we note that changing the delays of acknowledgments of packets can change the times of subsequent arrivals, so the arrival sequence that each algorithm processed may not be consistent with the sequence it would have seen if it were the algorithm actually used to delay the acknowledgments.

In our plots (Figures 4 and 5), the solid lines correspond to $\mathbf{greedy}_{new}$ and the lines with long dashes correspond to $\mathbf{greedy}_{tot}$. The performance of the algorithm using the interval timer is indicated by the lines with short dashes and the dotted lines represent the performance of the heartbeat timer-based algorithm. Each algorithm's upper line shows its performance for $L = 0$ and its lower line shows its performance for $L = 1$. In each plot, the vertical axis gives the average ratio of the cost of each algorithm's solution to the cost of an optimal solution. The horizontal axis represents how many seconds of extra latency is as expensive as a single acknowledgment, so a value of S on the horizontal axis corresponds to an acknowledgment cost of $\eta = S/(S+1)$. Our simulations are for $S \in \{0.001, 0.01, 0.02, 0.03, 0.05, 0.08, 0.1, 0.2, 0.3, 0.5, 0.8, 1.0, 1.5, 2.0, 3.0\}$.

Figure 4 shows each algorithm's average performance on 185 `telnet` connections from the `telnet` client to the `telnet` server (results were similar for the connections from the server to the client). Notice that the timer-based algorithms without lookahead were superior to ours with $L = 0$ for a particular value of η, but as we moved away from this value, our algorithms without lookahead were superior to the timer-based algorithms even with lookahead. Our algorithms with lookahead were always the best. We saw similar results for the following other types of connections: NNTP (Usenet), SMTP (e-mail), HTTP (WWW) from the server to the client, `gopher` from the server to the client, FTP, and FTP-data (connections opened in an FTP session to transfer files). All the connections of these types tended to involve transmission of a large number of packets (typically at least 10–20, frequently > 100).

Figure 5 shows each algorithm's average performance on 237 `finger` connections from the `finger` client to the `finger` server (results were similar for the connections from the server to the client). Only one line is visible for each of $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ because those algorithms' lines for $L = 1$ are indistinguishable from the horizontal axis. Notice that the timer-based algorithms without lookahead were superior to ours for $L = 0$ for all values of $\eta \geq 0.23$, and this trend does not seem inclined to change. But our algorithms with lookahead were still always the best. We saw similar results for the following other types of connections: HTTP from the client to the server and `gopher` from the client to the server. All the connections of these types tended to involve transmission of a small number

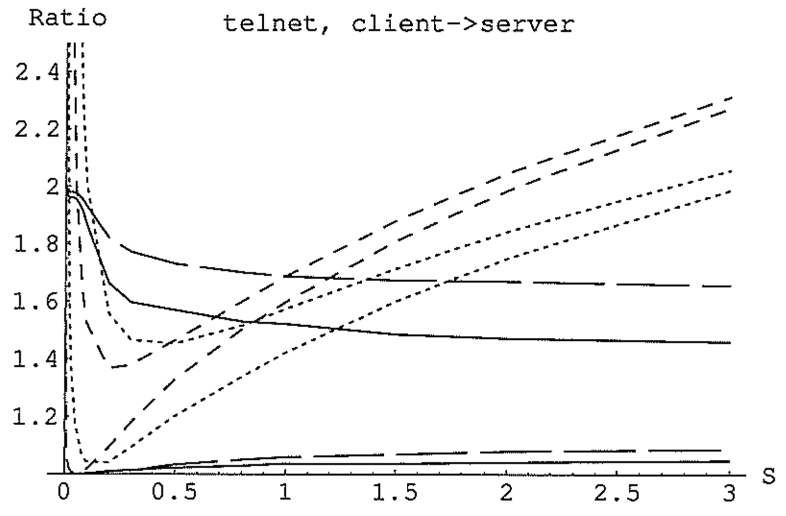

Figure 4: Simulation results under f_{sum} for 185 `telnet` connections from the `telnet` client to the `telnet` server. These results are typical for connections involving transmission of a large number of packets (typically at least 10–20, frequently > 100). The solid lines correspond to $\mathbf{greedy}_{new}$ and the lines with long dashes correspond to $\mathbf{greedy}_{tot}$. The performance of the algorithm using the interval timer is indicated by the lines with short dashes and the dotted lines represent the performance of the heartbeat timer-based algorithm. Each algorithm's upper line shows its performance for $L = 0$ and its lower line shows its performance for $L = 1$. The vertical axis gives the average ratio of the cost of each algorithm's solution to the cost of an optimal solution, and the horizontal axis represents how many seconds of extra latency is as expensive as a single acknowledgment.

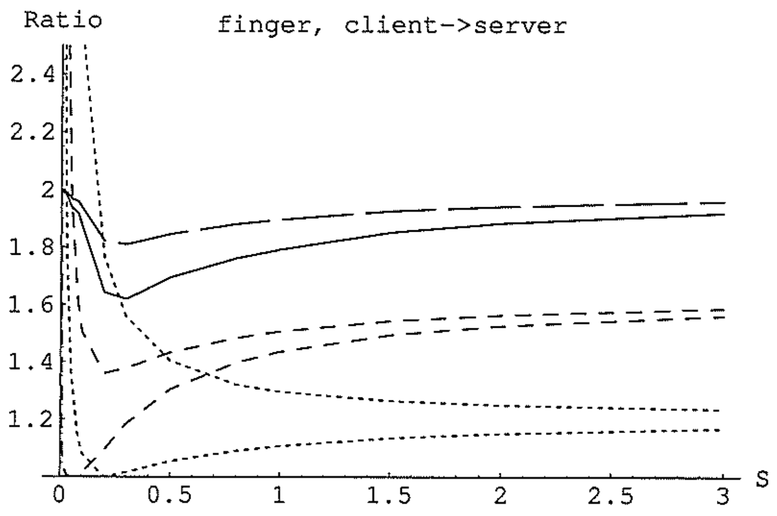

Figure 5: Simulation results under f_{sum} for 237 `finger` connections from the `finger` client to the `finger` server. These results are typical for connections involving transmission of a small number of packets (typically ≈ 2). The solid lines correspond to $\mathbf{greedy}_{new}$ and the lines with long dashes correspond to $\mathbf{greedy}_{tot}$. The performance of the algorithm using the interval timer is indicated by the lines with short dashes and the dotted lines represent the performance of the heartbeat timer-based algorithm. Each algorithm's upper line shows its performance for $L = 0$ and its lower line shows its performance for $L = 1$. (Only one line is visible for each of $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ because those algorithms' lines for $L = 1$ are indistinguishable from the horizontal axis.) The vertical axis gives the average ratio of the cost of each algorithm's solution to the cost of an optimal solution, and the horizontal axis represents how many seconds of extra latency is as expensive as a single acknowledgment.

of packets (typically ≈ 2).

When η is large and input sequences are long, our algorithms typically work better than the timer-based ones because our algorithms are allowed to set alarms farther into the future while the timer-based algorithms always use constants in setting their alarms, so they cannot take advantage of cheap latency like ours can. Rather, they must send more expensive acknowledgments. This explanation is consistent with the observation that the two curves for an algorithm tend to converge as η increases, since increasing η makes latency cheaper, diminishing the gain from looking ahead. When η is very small, then latency is very expensive and the constants used by the timer-based algorithms cause them to send acknowledgments much too late, even if $L = 1$. This gives an inherent advantage to our algorithms since the amount they wait is related to the cost of waiting. This phenomenon is independent of the length of the input sequence, as evidenced in Figures 4 and 5.

When sequences consist of only a few arrivals and η is not extremely small, then very few acknowledgments are required. Thus all the algorithms will probably send the same number of acknowledgments and the difference in performance between the algorithms will be the cost of the extra latency incurred. As latency becomes cheaper, $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ set their alarms farther into the future while the timer-based algorithms set theirs the same distance into the future regardless of η. Thus $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ set their alarms much farther into the future than the timer-based algorithms do. Since probably the only difference between the algorithms' performances on short sequences is latency cost, it is not surprising that $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ with $L = 0$ lose their advantage over the timer-based algorithms on short sequences for large η. Thus it seems reasonable that short input sequences favor the timer-based algorithms unless $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ have lookahead, in which case they are optimal or nearly optimal. Unfortunately, short input sequences do not facilitate inference of upcoming arrival times, so any learning algorithm used to predict arrival times might not do well. A possible exception is a learner that remembers the inter-arrival times for the last connection of the same type, e.g. if the current connection is a `finger` connection, then the learner can look back to the inter-arrival times it learned from the previous `finger` connection.

Finally, we note two observations concerning our algorithms. First, the cost of our algorithms' solutions were always at most twice optimal (even with $L = 0$), which is encouraging since we know by Lemma 7 that under f_{sum} there exists an input sequence forcing $\mathbf{greedy}_{tot}$ to output a solution that is $\Omega\left(\sqrt{n}\right)$ of optimal. Second, our algorithms with $L = 1$ were always at least as good (and typically much better than) any other algorithms we tested. This implies that a good learning system to predict packet inter-arrival times would be very useful to any algorithm that delays acknowledgments, including the timer-based algorithms currently in use by many TCP implementations.

7 Objective Function f_{max}

Recall Definition 3 of Section 3, which defined f_{max}:

$$f_{max} \doteq \eta k + (1 - \eta) \sum_{i=1}^{k} \max_{a_j \in \sigma_i} (t_i - a_j),$$

```
Off-Line-2
    Initialize $M_{\min}[0] \leftarrow 0$
    Initialize $M[1,1] \leftarrow 1 \cdot \eta$
    Initialize $M_{\min}[1] \leftarrow M[1,1]$
    Initialize $M_{\mathrm{pt}}[1] \leftarrow 1$
    For $i \in [2, n]$
        $M_{\min}[i] \leftarrow \infty$
        For $j \in [1, i]$
            $M[i,j] \leftarrow 1 \cdot \eta + (1-\eta) \cdot (a_i - a_{i-j+1}) + M_{\min}[i-j]$
            If $M[i,j] < M_{\min}[i]$ then
                $M_{\min}[i] \leftarrow M[i,j]$
                $M_{\mathrm{pt}}[i] \leftarrow j$
```

Figure 6: An optimal off-line algorithm that runs under f_{max}.

where $\mathcal{A} = \langle a_1, \ldots, a_n \rangle$ is the sequence of packet arrivals, k is the number of acknowledgments, the acknowledgment for the packets in σ_i comes at t_i, and $\eta \in [0,1]$ weights the importance of minimizing acknowledgments with that of minimizing extra latency. We now present our algorithms for the acknowledgment delay problem under f_{max}.

7.1 An Off-Line Algorithm Under f_{max}

To optimally solve the acknowledgment delay problem under f_{max}, we can run **greedy**$_{tot}$ with lookahead 1, which will yield an optimal solution if there is no maximum delay constraint or if $\tau < \eta/(1-\eta)$ (see Section 7.2). This would run in $O(n)$ time and also work in the presence of non-rush departures. But if there exists a maximum delay constraint of $\tau \geq \eta/(1-\eta)$, then **greedy**$_{tot}$ with $L = 1$ might not work. Thus we make a simple adaptation to the dynamic programming solution of Section 6.1. Specifically, we change Lines 2 and 8 of Figure 1 to evaluate the function *lat* that corresponds to f_{max}. The complete algorithm is shown in Figure 6. We can also extend this algorithm to handle non-rush departures and the maximum delay constraint the same way we extended the algorithm of Section 6.1. So by Corollary 6 we have an $O(n^2)$ time algorithm for the acknowledgment delay problem under f_{max} in the presence of non-rush departures and a maximum delay constraint.

7.2 Algorithm greedy$_{tot}$ Under f_{max}

As in Section 6.2, first assume that all departures are rush and there is no limit on individual delays. Let a_j be the time of the packet that just arrived. Let σ be the set of unacknowledged arrivals just after time a_j (this includes the packet corresponding to a_j, so $|\sigma| \geq 1$). If the algorithm waits some time $t_{\mathbf{greedy}_{tot}, f_{max}}$ (i.e. until time $a_j + t_{\mathbf{greedy}_{tot}, f_{max}}$) before sending an acknowledgment, then it will incur extra cost due to latency of $(1-\eta)\, t_{\mathbf{greedy}_{tot}, f_{max}}$. Also, it has already incurred a cost of $(1-\eta)(a_j - a_{first_i})$ due to past latency, where a_{first_i} is the first arrival of σ_i. Acknowledging immediately incurs a cost of η for the acknowledgment plus the same cost due to past latency. Thus on each new arrival a_j, **greedy**$_{tot}$ sets an alarm at $a_j + t_{\mathbf{greedy}_{tot}, f_{max}}$ for $t_{\mathbf{greedy}_{tot}, f_{max}}$ such that the cost for σ if an acknowledgment is not sent at a_j (namely, $(1-\eta)\, t_{\mathbf{greedy}_{tot}, f_{max}} + (1-\eta)(a_j - a_{first_i})$) is equal to the cost for σ if an acknowledgment is sent at a_j (namely, $\eta + (1-\eta)(a_j - a_{first_i})$). Solving for $t_{\mathbf{greedy}_{tot}, f_{max}}$ yields

$$t_{\mathbf{greedy}_{tot}, f_{max}} = \frac{\eta}{(1-\eta)}.$$

If no new arrival occurs before the alarm sounds, then an acknowledgment is sent when the alarm sounds. Otherwise, the old alarm is deleted and a new one is set at $a_{j+1} + \eta/(1-\eta)$. If $L = 1$, then **greedy**$_{tot}$ will send an acknowledgment at exactly a_j if and only if $(a_{j+1} - a_j) > t_{\mathbf{greedy}_{tot}, f_{max}}$.

As with **greedy**$_{tot}$ under f_{sum}, we can extend this algorithm to accommodate non-rush departures and a maximum delay constraint of τ. On the first arrival a_j of a new subsequence, the algorithm sets a second alarm at $a_j + \tau$. This alarm is not updated at new arrivals within the same subsequence. So an acknowledgment is sent whenever either alarm sounds, again making sure that there is only one departure per subsequence (Observation 5) and that all departures are sent in order of their ready times. In addition to the alarms provoking an acknowledgment transmission, this algorithm also immediately sends a departure if it encounters a second departure within the same subsequence (i.e. the transmission occurs at the second departure's ready time), which is a condition in addition to those for acknowledging under f_{sum}. This is motivated by the following observation.

Observation 13 *In an optimal solution under f_{max}, no departure will be scheduled past the next departure's ready time.*

Proof: By contradiction. Assume there exists an optimal solution S in which departure d_j is scheduled past d_{j+1}'s ready time. Since S is optimal, then d_j's transmission time must coincide with some arrival a_i. We can slide d_j's departure time back to the last arrival between the ready times of d_j and d_{j+1} (or back to d_j's ready time if no such arrivals exist), and let d_{j+1} acknowledge a_i in addition to the other arrivals it acknowledges. This decreases total latency under f_{max} since latency cost increases linearly with time independent of the number of unacknowledged arrivals and unsent departures. Also, since both d_j and d_{j+1} must be transmitted anyway, the number transmissions does not increase. Thus the total cost decreases, contradicting the optimality of S. □

Finally, we note that the time complexity per arrival needed to update $t_{\mathbf{greedy}_{tot}, f_{sum}}$ is constant.

Despite **greedy**$_{tot}$'s poor performance under f_{sum}, under f_{max} it does quite well. Before proving upper bounds on **greedy**$_{tot}$'s competitive ratio, we start with a lemma about the structure of an optimal solution.

Lemma 14 *Under f_{max}, there exists an optimal solution S that places an acknowledgment at a_j if and only if $(a_{j+1} - a_j)(1-\eta) > \eta$.*

Proof: Consider a subsequence $\sigma_\ell = \langle a_i, a_{i+1}, \ldots, a_m \rangle$ which is a subsequence in S. The total cost for σ_ℓ is exactly $(a_m - a_i)(1 - \eta) + \eta$ (the first term is for the latency, the second for the acknowledgment). We can divide this cost among the arrivals in σ_ℓ by associating with $a_j \in \{a_i, \ldots, a_{m-1}\}$ a latency cost of $(a_{j+1} - a_j)(1 - \eta)$ and charge a_m the cost η for the acknowledgment. This accounts for all of the cost for σ_ℓ, so no additional cost is associated with any arrivals in σ_ℓ.

Now consider the case when $(a_{j+1} - a_j)(1 - \eta) > \eta$. If S did not send an acknowledgment at a_j, then a_j was charged $> \eta$ for the latency between a_j and a_{j+1}. But if S did send an acknowledgment at a_j, then a_j was charged exactly η. Thus it is better to send an acknowledgment at a_j than it is to wait. For the case when $(a_{j+1} - a_j)(1 - \eta) \leq \eta$, the cost of sending an acknowledgment at a_j is at least as much as the cost of waiting. □

Now we prove that **greedy**$_{tot}$ is 2-competitive under f_{max}. Combined with Theorem 23 (Section 7.4), this means that **greedy**$_{tot}$ is an optimal on-line algorithm under f_{max} when $L = 0$.

Theorem 15 *Under f_{max},* **greedy**$_{tot}$ *is 2-competitive with lookahead $L = 0$.*

Proof: In this proof we use S and the same cost accounting strategy used in the proof of Lemma 14. First note that from Observation 8 we know that **greedy**$_{tot}$ avoids incurring more than $\eta/(1 - \eta)$ units of extra latency (for a latency cost of η) between any two consecutive arrivals a_j and a_{j+1}. We now consider the two possible cases for this pair of arrivals.

If $a_{j+1} - a_j \leq \eta/(1 - \eta)$, then by Observation 8 and Lemma 14, **greedy**$_{tot}$ does exactly the same thing that S does, namely not send an acknowledgment at a_j. If instead $a_{j+1} - a_j > \eta/(1 - \eta)$, then S sends an acknowledgment at a_j (by Lemma 14), charging η to a_j. But **greedy**$_{tot}$ waits until $a_j + \eta/(1 - \eta)$ and then sends an acknowledgment, charging 2η to a_j (η for the latency and η for the acknowledgment). Note that no matter which case applies, **greedy**$_{tot}$ is synchronized with S, i.e. at any arrival a_j, **greedy**$_{tot}$'s solution always has the same set of unacknowledged arrivals as S. Thus the cost attributed to any arrival in **greedy**$_{tot}$'s solution is at most twice that of the same arrival in S. Therefore $C_{\mathbf{greedy}_{tot}} \leq 2C_{opt}$. □

Obviously **greedy**$_{tot}$ is better suited for f_{max} than f_{sum}. But in fact, Theorem 15 is only part of the story. In the proof of Theorem 15, the only difference between the costs incurred by **greedy**$_{tot}$ and S came from their behaviors when $a_{j+1} - a_j > \eta/(1 - \eta)$. Namely, **greedy**$_{tot}$ waited until $a_j + \eta/(1 - \eta)$ before sending an acknowledgment while S sent its acknowledgment at a_j. But if **greedy**$_{tot}$ knew at time a_j that the next arrival was too far into the future, it would have sent an acknowledgment at a_j just as S did. Thus we get the following corollary.

Corollary 16 *Under f_{max},* **greedy**$_{tot}$ *is 1-competitive with lookahead $L = 1$.*

Note that in Theorem 15 and Corollary 16, we ignore τ and assume that all departures are rush. By Observations 13 and 5, non-rush departures do not alter **greedy**$_{tot}$'s competitive ratios for $L = 0$ and $L = 1$ since **greedy**$_{tot}$ is still synchronized with S. However, adding τ can change the situation if it is sufficiently large. We start by exploring when adding τ does not change the bounds on the competitive ratios. If $\tau < \eta/(1 - \eta)$, then (since **greedy**$_{tot}$ and optimal solution S are synchronized without τ) if **greedy**$_{tot}$ must send an acknowledgment due to τ, then so must S. So the algorithms remain synchronized. Thus **greedy**$_{tot}$ with $L = 1$ remains optimal and **greedy**$_{tot}$ with $L = 0$ still only pays for extra latency, but this latency still does not exceed $\eta/(1-\eta)$, just as in the proof of Theorem 15. This yields the following corollary.

Corollary 17 *Under f_{max}, when including $\tau < \eta/(1-\eta)$ and non-rush departures,* **greedy**$_{tot}$ *is 2-competitive for $L = 0$ and 1-competitive for $L = 1$.*

When $\tau \geq \eta/(1-\eta)$, the proof of Theorem 15 does not apply. But we can still show a bound on the competitive ratio for **greedy**$_{tot}$.

Theorem 18 *Under f_{max}, with lookahead $L = 1$ and for any $\tau \geq \eta/(1-\eta)$,* **greedy**$_{tot}$[8] *has a competitive ratio of $\leq 1 + \frac{\eta}{\tau(1-\eta)} \leq 2$. For $L = 0$, the competitive ratio is $\leq 1 + \frac{2\eta}{\tau(1-\eta)} \leq 3$.*

Proof: We prove the theorem for $L = 1$ and then state how to adapt it for $L = 0$.

We divide the acknowledgments into those sent because the of the τ requirement (τ-acks) and the rest (regular acks)[9]. We partition the solution created by **greedy**$_{tot}$ into groups where each group consists of any number of subsequences ended by τ-acks followed by a subsequence ended by a regular ack. (So each regular ack subsequence is the last subsequence in the group.) Let ℓ be the latency cost (for $L = 1$) for the subsequence that ends with the regular ack.

We show in Lemma 19 below that there exists an optimal solution S that places an acknowledgment corresponding to each regular ack of **greedy**$_{tot}$. Thus we can treat each group as the first one. Let a τ-interval be any subsequence that ends with a τ-ack. We show in Lemma 20 that the latency cost for each τ-interval in S is at least $(\tau - \eta/(1-\eta))(1-\eta)$. Since $\tau \geq \eta/(1-\eta)$, this quantity is non-negative. Also, we note that S must place an acknowledgment in each τ-interval (otherwise it would violate the maximum delay constraint). Thus if there are q τ-acks[10], then S pays $q\eta$ for the τ-acks, η for the regular ack, at least $q(\tau - \eta/(1-\eta))(1-\eta)$ for the τ-interval latency, and ℓ for the regular ack latency (see Lemma 21). So the cost for S is at least

$$(q+1)\eta + q(1-\eta)\left(\tau - \frac{\eta}{1-\eta}\right) + \ell = \ell + \eta + q\tau(1-\eta).$$

Also, **greedy**$_{tot}$'s cost (for $L = 1$) is at most

$$(q+1)\eta + q\tau(1-\eta) + \ell = q\eta + \ell + \eta + q\tau(1-\eta). \tag{3}$$

Hence the competitive ratio of **greedy**$_{tot}$ is at most

$$1 + \frac{q\eta}{\ell + \eta + q\tau(1-\eta)} \leq 1 + \frac{\eta}{\tau(1-\eta)}$$

[8] *For this result we require that* **greedy**$_{tot}$ *is modified so that it sends an acknowledgment if the next arrival is at the alarm time.*

[9] If the acknowledgment qualifies as both a τ- and a regular ack, we call it a regular ack.

[10] For this proof we assume that $q > 0$ for some group. Otherwise τ is irrelevant and **greedy**$_{tot}$ remains synchronized with S, implying that the bounds of Corollary 17 hold.

since ℓ and η are non-negative.

The only difference between $L = 1$ and $L = 0$ is that for $L = 0$, **greedy**$_{tot}$ incurs more latency. Specifically, it pays $\tau(1-\eta)$ for each τ subsequence and at most $\ell + (1-\eta)\eta/(1-\eta) = \ell + \eta$ for the subsequence ending with a regular ack. Equation 3 has already accounted for the latency cost for the τ subsequences, so we merely add η to the latency cost for $L = 1$ to correct for $L = 0$. So the competitive ratio for $L = 0$ is at most

$$1 + \frac{(q+1)\eta}{\ell + \eta + q\tau(1-\eta)} \leq 1 + \frac{q\eta + \eta}{q\tau(1-\eta)} \leq 1 + \frac{2\eta}{\tau(1-\eta)}.$$

□

We now prove the technical lemmas required to complete the proof of Theorem 18.

Lemma 19 *Let everything be as in the proof of Theorem 18. Then there exists an optimal solution S that places an acknowledgment corresponding to each regular ack of* **greedy**$_{tot}$.

Proof: Let a_r be the arrival that **greedy**$_{tot}$ sent its regular ack at, so a_r is at the end of the regular ack subsequence at the end of a group. Since **greedy**$_{tot}$ sent a regular ack at a_r, it must be the case that $a_{r+1} - a_r > \eta/(1-\eta)$. So by Lemma 14, S must also place an acknowledgment at a_r. □

Lemma 20 *Let everything be as in the proof of Theorem 18. Then the latency cost for each τ-interval in S is at least $(\tau - \eta/(1-\eta))(1-\eta)$.*

Proof: By the definition of **greedy**$_{tot}$ the time between two consecutive arrivals in a τ-interval is $< \eta/(1-\eta)$, and the time between the last arrival of the τ-interval and the next arrival is $< \eta/(1-\eta)$ (otherwise a regular ack would have been sent). See Figure 7 for an example. Thus the optimal solution cannot benefit by having more than one acknowledgment in each τ-interval (by Lemma 14), but it must place at least one acknowledgment in the τ-interval so it does not violate the maximum delay constraint. Since S places exactly one acknowledgment in the τ-interval, it only saves $\leq \eta/(1-\eta)$ in latency for the entire interval and must pay for the remaining latency. The remaining latency is at least $\tau - \eta/(1-\eta)$, so the latency cost for each τ-interval in S is at least $(\tau - \eta/(1-\eta))(1-\eta)$. □

Lemma 21 *Let everything be as in the proof of Theorem 18. Then the latency cost for S for the regular ack subsequence is at least ℓ.*

Proof: By Lemma 19, S places an acknowledgment at the last arrival of the regular ack subsequence. Since each two consecutive arrivals in the regular ack subsequence are separated by $< \eta/(1-\eta)$, by Lemma 14 S will not place an acknowledgment elsewhere in that subsequence. Thus S incurs as much latency as **greedy**$_{tot}$ does for $L = 1$, so S's latency cost for the regular ack subsequence is ℓ. □

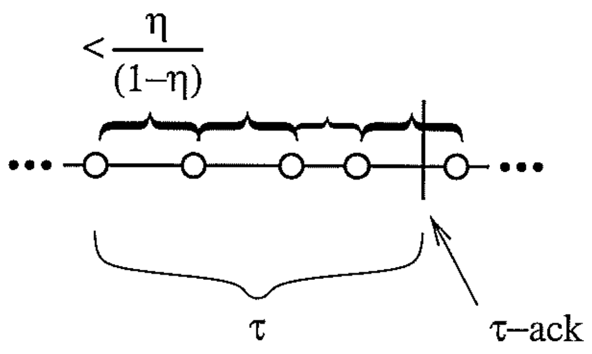

Figure 7: An example of a τ-interval. The τ-interval in the figure is the set of four arrivals to the left of the τ-ack. The distance between any two consecutive arrivals (as indicated by curly braces) is $< \eta/(1-\eta)$, otherwise a regular ack would have been sent. S places exactly one acknowledgment in the τ-interval, so it pays for all but at most $\eta/(1-\eta)$ of the τ total latency between the first arrival and the acknowledgment.

7.3 Algorithm greedy$_{new}$ Under f_{max}

We now describe how **greedy**$_{new}$ behaves under f_{max}. Specifically, on each new arrival $a_j \in \sigma_i$, **greedy**$_{new}$ sets an alarm at $a_j + t_{\mathbf{greedy}_{new}, f_{max}}$ for $t_{\mathbf{greedy}_{new}, f_{max}}$ such that

$$(1-\eta)\, t_{\mathbf{greedy}_{new}, f_{max}} + (1-\eta)\,(a_j - a_{first_i}) = \eta.$$

Solving for $t_{\mathbf{greedy}_{new}, f_{max}}$ yields

$$t_{\mathbf{greedy}_{new}, f_{max}} = \frac{\eta}{(1-\eta)} - (a_j - a_{first_i}), \tag{4}$$

which can be computed in constant time per arrival.

Since f_{max} is a special case of f_{lat}, Observation 9 holds for it as well. We also get from Corollary 11 that **greedy**$_{new}$ with $L = 0$ is 2-competitive under f_{max} in the presence of non-rush departures and a maximum delay constraint. According to Theorem 23, for $L = 0$ this is the best any on-line algorithm can do in the worst case.

While **greedy**$_{new}$ is optimal under the general-purpose objective function f_{lat} for no lookahead, **greedy**$_{tot}$ has an advantage over it under f_{max} for $L = 1$. Corollary 16 and Theorem 18 state that **greedy**$_{tot}$ is better than 2-competitive with $L = 1$, but as we show below, **greedy**$_{new}$ with $L = 1$ still cannot beat a competitive ratio of 2.

Observation 22 *Under f_{max}, even with $L = 1$, there exists an arrival sequence such that $C_{\mathbf{greedy}_{new}} \geq 2\,C_{opt} - c$ where c can be made arbitrarily small with respect to C_{opt}.*

Proof: First recall Observation 9 that says under f_{lat}, **greedy**$_{new}$ acknowledges if and only if the total latency of a subsequence reaches $\eta/(1-\eta)$. The adversary exploits this property by defining an input arrival sequence consisting of m groups, where each group contains

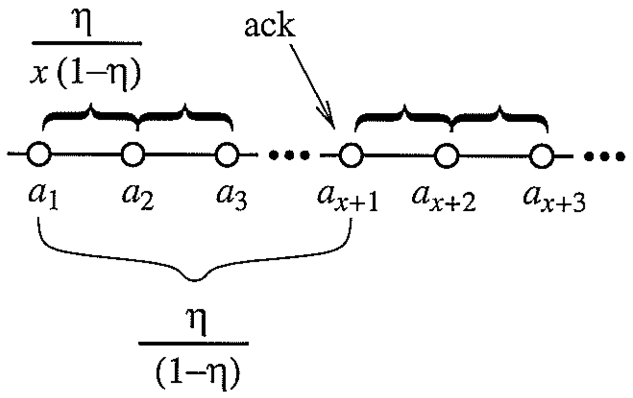

Figure 8: An example of an arrival sequence forcing **greedy**$_{new}$ to be 2-competitive under f_{max}. Arrivals a_1 through a_{x+1} comprise a group of $x+1$ arrivals where a group is ended by an acknowledgment from **greedy**$_{new}$ (the next group is a_{x+2} through a_{2x+2}). Two consecutive arrivals are separated by $\eta/(x(1-\eta))$ for a total latency of $\eta/(1-\eta)$ for each group.

$x+1$ arrivals, each separated by time $\eta/(x(1-\eta))$. (See Figure 8 for an example.) Since **greedy**$_{new}$ sends an acknowledgment at the end of each group, its total cost is

$$C_{\mathbf{greedy}_{new}} = m\eta + m(1-\eta)\frac{\eta}{1-\eta} = 2m\eta.$$

An optimal solution is at least as good as the strategy to send just one acknowledgment at a_n for $n = m(x+1)$. This strategy pays η for the single acknowledgment plus the latency costs within the groups plus the latency cost between the groups. This yields a cost of

$$\eta + m(1-\eta)\frac{\eta}{1-\eta} + (m-1)(1-\eta)\left(\frac{\eta}{x(1-\eta)}\right) = C_{\mathbf{greedy}_{new}}/2 + \eta + \eta(m-1)/x.$$

By making m arbitrarily large, the adversary can make the second term arbitrarily small with respect to $C_{\mathbf{greedy}_{new}}$. After fixing m, the adversary can make the last term arbitrarily small with respect to $C_{\mathbf{greedy}_{new}}$ by making x arbitrarily large. $\square$

7.4 A Lower Bound Under f_{max}

We now give some lower bounds for the acknowledgment delay problem under f_{max}. First recall that for $L = 1$, Corollary 16 says that **greedy**$_{tot}$ is 1-competitive if we ignore τ or if $\tau < \eta/(1-\eta)$, and Theorem 18 says that **greedy**$_{tot}$ is $\left(1 + \frac{\eta}{\tau(1-\eta)}\right)$-competitive for finite $\tau \geq \eta/(1-\eta)$. We do not know if the latter bound is tight (i.e. can we be 1-competitive in the presence of finite $\tau \geq \eta/(1-\eta)$?), but obviously no on-line algorithm can be better than 1-competitive. So here we focus on the more interesting problem of lowerbounding the case for $L = 0$ and study the general objective function f_{lat}.

First recall Definition 1 that defines f_{lat} as any objective function for which the latency cost increases with the amount of time that an acknowledgment is delayed. By using the standard adversary strategy that places the next arrival in the sequence immediately after the on-line algorithm acknowledges, we can prove that any deterministic on-line algorithm without lookahead is no better than 2-competitive under f_{lat}. This means that for $L = 0$ both **greedy**$_{tot}$ and **greedy**$_{new}$ are optimal under f_{max}, but it also means that **greedy**$_{new}$ is optimal for $L = 0$ under f_{lat}. Of course, **greedy**$_{tot}$ is not optimal under f_{lat} because Lemma 7 tells us that **greedy**$_{tot}$ can be as bad as $\Omega(\sqrt{n})$-competitive under f_{sum}, which is a special case of f_{lat}.

Theorem 23 *Let A be any deterministic on-line algorithm for the acknowledgment delay problem with $L = 0$. Then under f_{lat} there exists an arrival sequence such that $C_A \geq 2\,C_{opt} - c$ where c can be made arbitrarily small with respect to C_{opt}.*

Proof: The proof is a simplified version of the proof of Theorem 12. In this case, the adversary places arrival a_{j+1} immediately[11] after A acknowledges a_j. Thus A sends an acknowledgment for every packet. Define ℓ_j to be the latency accumulated between arrival a_j and a_j's acknowledgment. Then for a sequence of n arrivals, the cost of A's solution is

$$C_A = \eta\, n + (1-\eta)\sum_{j=1}^{n} \ell_j.$$

We now upperbound the cost of the optimal solution by averaging the costs of solution S_1 that acknowledges immediately after the odd-numbered arrivals and solution S_2 that acknowledges immediately after the even-numbered arrivals. Since both solutions must acknowledge all packets, there are a total of $n+1$ acknowledgments between the two solutions. Also, between the two solutions each interval of latency ℓ_j is incurred exactly once, except ℓ_n, which neither of them incurs. Thus the average of S_1's cost and S_2's cost is

$$\frac{1}{2}\left((n+1)\,\eta + (1-\eta)\sum_{j=1}^{n-1} \ell_j\right) = C_A/2 + \eta/2 - \ell_n/2.$$

By making n arbitrarily large, the adversary can make $\eta/2$ arbitrary small with respect to C_A. Since the cost of S_1 or S_2 is at least as good as their average, C_A can be made arbitrarily close to $2\,C_{opt}$. □

7.5 Simulation Results Under f_{max}

We ran simulation results under f_{max} similar to those we ran under f_{sum} (Section 6.5), using the same data and applying the same methodology. Overall, we found the results under f_{max} similar to those under f_{sum} in general form, except that under f_{max}, the fixed timer-based algorithms fared much worse for long connections. Also, empirically we saw that **greedy**$_{tot}$ performed at least as well as (and often better than) **greedy**$_{new}$, where under f_{sum} the opposite was true. This is not surprising since in terms of loss bounds, **greedy**$_{tot}$ is at least as good as **greedy**$_{new}$, and in particular, **greedy**$_{tot}$ is 1-competitive for $L = 1$.

[11] While a_{j+1} will not coincide with a_j's acknowledgment, they can be arbitrarily close. So for ease of exposition we will assume that they are coincident.

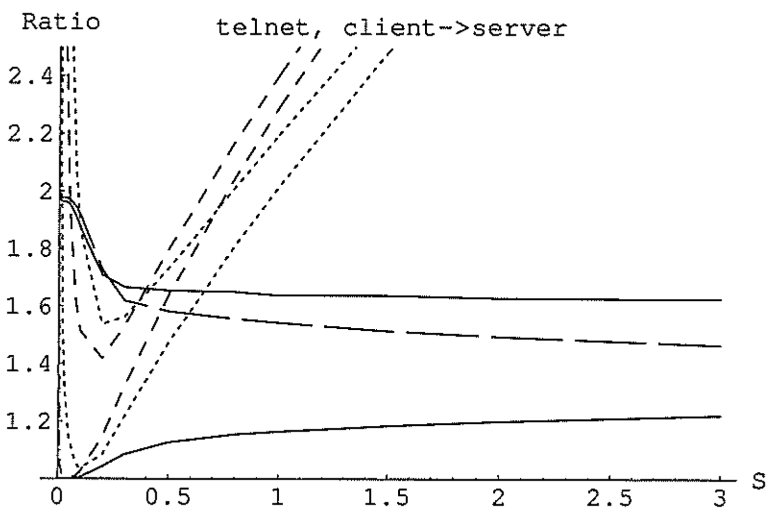

Figure 9: Simulation results under f_{max} for 185 `telnet` connections from the `telnet` client to the `telnet` server. These results are typical for connections involving transmission of a large number of packets (typically at least 10–20, frequently > 100). The solid lines correspond to $\mathbf{greedy}_{new}$ and the lines with long dashes correspond to $\mathbf{greedy}_{tot}$. The performance of the algorithm using the interval timer is indicated by the lines with short dashes and the dotted lines represent the performance of the heartbeat timer-based algorithm. Each algorithm's upper line shows its performance for $L = 0$ and its lower line shows its performance for $L = 1$. (Only one line is visible for $\mathbf{greedy}_{tot}$ because its line for $L = 1$ is optimal so it is indistinguishable from the horizontal axis.) The vertical axis gives the average ratio of the cost of each algorithm's solution to the cost of an optimal solution, and the horizontal axis represents how many seconds of extra latency is as expensive as a single acknowledgment.

In our plots (Figures 9 and 10), the solid lines correspond to $\mathbf{greedy}_{new}$ and the lines with long dashes correspond to $\mathbf{greedy}_{tot}$. The performance of the algorithm using the interval timer is indicated by the lines with short dashes and the dotted lines represent the performance of the heartbeat timer-based algorithm. Each algorithm's upper line shows its performance for $L = 0$ and its lower line shows its performance for $L = 1$. In each plot, the vertical axis gives the average ratio of the cost of each algorithm's solution to the cost of an optimal solution. The horizontal axis represents how many seconds of extra latency is as expensive as a single acknowledgment, so a value of S on the horizontal axis corresponds to an acknowledgment cost of $\eta = S/(S+1)$. Our simulations are for $S \in \{0.001, 0.01, 0.02, 0.03, 0.05, 0.08, 0.1, 0.2, 0.3, 0.5, 0.8, 1.0, 1.5, 2.0, 3.0\}$.

Figure 9 shows each algorithm's average performance on 185 `telnet` connections from the `telnet` client to the `telnet` server (results were similar for the connections from the server to the client). Since $\mathbf{greedy}_{tot}$ with $L = 1$ is optimal, its curve is indistinguishable

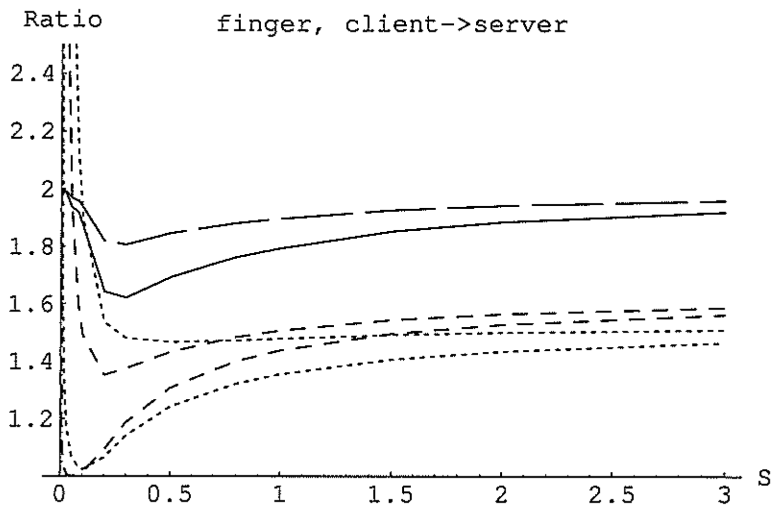

Figure 10: Simulation results under f_{max} for 237 `finger` connections from the `finger` client to the `finger` server. These results are typical for connections involving transmission of a small number of packets (typically ≈ 2). The solid lines correspond to **greedy**$_{new}$ and the lines with long dashes correspond to **greedy**$_{tot}$. The performance of the algorithm using the interval timer is indicated by the lines with short dashes and the dotted lines represent the performance of the heartbeat timer-based algorithm. Each algorithm's upper line shows its performance for $L = 0$ and its lower line shows its performance for $L = 1$. (Only one line is visible for each of **greedy**$_{tot}$ and **greedy**$_{new}$ because those algorithms' lines for $L = 1$ are indistinguishable from the horizontal axis.) The vertical axis gives the average ratio of the cost of each algorithm's solution to the cost of an optimal solution, and the horizontal axis represents how many seconds of extra latency is as expensive as a single acknowledgment.

from the horizontal axis. Notice that the timer-based algorithms without lookahead were superior to our algorithms with $L = 0$ for a particular value of η, but as we moved away from this value, our algorithms without lookahead were superior to the timer-based algorithms even with lookahead (note that under f_{max}, this advantage is more pronounced than under f_{sum}). Our algorithms with lookahead were always the best. We saw similar results for the following other types of connections: NNTP (Usenet), SMTP (e-mail), HTTP (WWW) from the server to the client, `gopher` from the server to the client, FTP, and FTP-data. All the connections of these types tended to involve transmission of a large number of packets (typically at least 10–20, frequently > 100).

Figure 10 shows each algorithm's average performance on 237 `finger` connections from the `finger` client to the `finger` server (results were similar for the connections from the server to the client). Only one line is visible for each of $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ because those algorithms' lines for $L = 1$ are indistinguishable from the horizontal axis. Notice that the timer-based algorithms without lookahead were superior to ours for $L = 0$ for sufficiently large η, and this trend does not seem inclined to change. But our algorithms with lookahead were still always the best. We saw similar results for the following other types of connections: HTTP from the client to the server and `gopher` from the client to the server. All the connections of these types tended to involve transmission of a small number of packets (typically ≈ 2).

In summary, our general observations concerning how $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ compare to the timer-based algorithms are the same as for f_{sum} in Section 6.5. Our explanations for these observations are also the same. However, how $\mathbf{greedy}_{tot}$ performs relative to $\mathbf{greedy}_{new}$ under f_{max} is a different story. For $L = 0$, $\mathbf{greedy}_{tot}$ was noticeably better than $\mathbf{greedy}_{new}$ for `telnet`, FTP-data, and NNTP. $\mathbf{greedy}_{new}$ outperformed $\mathbf{greedy}_{tot}$ for `gopher` from the server to the client, on SMTP for $S \geq 1$, and for all the short connection types. For all other connections, $\mathbf{greedy}_{tot}$ and $\mathbf{greedy}_{new}$ performed comparably. For $L = 1$, naturally $\mathbf{greedy}_{tot}$ produces an optimal solution, so the only question is how well does $\mathbf{greedy}_{new}$ fare with $L = 1$. For all short connections, $\mathbf{greedy}_{new}$'s performance was similar to that under f_{sum}: it did so well that its curve is indistinguishable from the horizontal axis. For long connections, its performance was mixed. Its performance for `telnet`, NNTP, FTP-data from client to server, and FTP from client to server was similar to that in Figure 9. For all other long connections it fared much better: its performance curve was much closer to the horizontal axis.

Again, our algorithms' outstanding performance for $L = 1$ makes a good learning system desirable.

8 Concluding Remarks

In this paper we presented two on-line algorithms for the acknowledgment delay problem, and proved several bounds on their performance when measured by two objective functions that we defined. We then gave initial simulation results that indicate that our algorithms perform well empirically under our objective functions for sufficiently long TCP connections. They also show that the ability to predict the future could be very valuable in enhancing the performance of our algorithms.

Avenues of future work include exploring randomized algorithms for this problem and developing a model that takes packet sizes into account. Another research direction is to examine other objective functions such as the average latency per packet per subsequence (which is a special case of f_{lat}) or some completely different way of measuring network performance such as throughput or how our algorithms affect burstiness of transmissions. (We chose our objective functions in part due to the fact that our on-line algorithms can directly control how their actions influence the functions' values, unlike e.g. throughput.) We also plan on performing more extensive simulations, running our algorithms on more traces and including departures and the τ constraint. After developing good learners to act as lookahead oracles, we plan to simulate our algorithms in a network simulator such as ns [1]. This is necessary to assess how our algorithms affect other aspects of TCP, including round-trip time estimates and packet retransmissions [15].

Finally, we can explore the applicability of our model and algorithms to delaying acknowledgments in other contexts, such as in the data link layer over noisy channels [16].

Acknowledgments

The authors thank George Varghese, Girish P. Chandranmenon, Sally Floyd, Jonathan Turner, and Ellen Zegura for their helpful discussions. We also thank Dan Blandford for his careful reading of a draft of this paper, the STOC committee members for their comments, and Thomas Erlebach for identifying a bug in an earlier draft of this paper. Stephen Scott performed this work at Washington University.

References

[1] UCB/LBNL network simulator - ns, 1997. `http://www-mash.cs.berkeley.edu/ns/`.

[2] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.

[3] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *Communications of the ACM*, 39(4):745–763, 1992.

[4] D. Clark. Window and acknowledgment strategy in TCP. Internet Request for Comments 813, July 1982. `http://www.cis.ohio-state.edu/htbin/rfc/rfc813.html`.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[6] A. Costello and G. Varghese. Redesigning the BSD callout and timer facilities. Technical Report WUCS-95-23, Washington University in St. Louis, November 1995. `http://www.cs.wustl.edu/cs/techreports/1995/wucs-95-23.ps.gz`.

[7] R. Braden (Editor). Requirements for Internet hosts—communication layers. Internet Request for Comments 1122, October 1989.
`http://www.cis.ohio-state.edu/htbin/rfc/rfc1122.html`.

[8] E. F. Grove. Online bin packing with lookahead. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–436, 1995.

[9] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990.

[10] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[11] P. Krishnan, Philip M. Long, and Jeffrey Scott Vitter. Learning to make rent-to-buy decisions with systems applications. In *Proc. 12th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1995.

[12] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics.* PhD thesis, University of California, Berkeley, April 1997.

[13] V. Paxson and S. Floyd. Wide-area traffic: The failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[14] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[15] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols.* Addision-Wesley Publishing Company, 1994.

[16] A. S. Tanenbaum. *Computer Networks.* Prentice Hall, third edition, 1996.

[17] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume 2: The Implementation.* Addison-Wesley Publishing Company, 1995.

[18] Tzuoo-Hawn Yeh, Cheng-Ming Kuo, Chin-Laung Lei, and Hsu-Chun Yen. Competitive analysis of on-line disk scheduling. In *Proceedings of the 7th International Symposium on Algorithms and Computation*, pages 356–365, 1996.