ACADEMIC PRESS

Available at WWW.MATHEMATICSWEB.ORG POWERED BY SCIENCE DIRECT®

Journal of Algorithms 48 (2003) 407–428

Journal of Algorithms

www.elsevier.com/locate/jalgor

Dynamic TCP acknowledgment in the LogP model ☆

Jens S. Frederiksen,[a,1] Kim S. Larsen,[a,*,1] John Noga,[b] and Patchrawat Uthaisombut [c]

[a] *Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark*
[b] *Department of Computer Science, California State University, Northridge, CA 91330, USA*
[c] *Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA*

Received 10 May 2002

Abstract

When messages, which are to be sent point-to-point in a network, become available at irregular intervals, a decision must be made each time a new message becomes available as to whether it should be sent immediately or if it is better to wait for more messages and send them all together. Because of physical properties of the networks, a certain minimum amount of time must elapse in between the transmission of two packets. Thus, whereas waiting delays the transmission of the current data, sending immediately may delay the transmission of the next data to become available even more. We propose a new quality measure and derive optimal deterministic and randomized algorithms for this on-line problem.

Keywords: Dynamic TCP acknowledgment; Packet bundling; On-line algorithms; Competitive ratio; Flow-time; Deterministic; Randomized

☆ A preliminary version of this paper appeared as Jens S. Frederiksen, Kim S. Larsen, "Packet Bundling," Eighth Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci., Vol. 2368, Springer-Verlag, 2002, pp. 328–337.

* Corresponding author.
E-mail addresses: svalle@imada.sdu.dk (J.S. Frederiksen), kslarsen@imada.sdu.dk (K.S. Larsen), jnoga@csun.edu (J. Noga), utp@cs.pitt.edu (P. Uthaisombut).

[1] Supported in part by the Danish Natural Science Research Council (SNF) and in part by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

0196-6774/$ – see front matter
doi:10.1016/S0196-6774(03)00058-0

1. Introduction

We consider point-to-point transmission of data in a network. Transmission of data is in the form of packets, which contain some header information, such as the identification of the receiver and sender, followed by the actual data. For obvious reasons, data is also referred to as messages.

When messages to be sent become available a little at a time at irregular intervals, the question arises on the sending side whether to send a given message immediately or whether to wait for the next message to become available, such that they can be sent together. Sending the messages together is referred to as *Packet Bundling*.

The reason why this is at all an issue is because of physical properties of the networks which imply that after one packet has been sent, a certain minimum amount of time must elapse before the next packet may be sent. Thus, whereas waiting for more messages will certainly delay the transmission of the current message, sending immediately may delay the transmission of the next message to become available even more. In addition to reducing the overall transmission delay when bundling messages, we also reduce the bandwidth requirement of the sender, since overhead due to packet headers and network gap is reduced. The problem of making these decisions is referred to as the *Packet Bundling Problem*.

A very similar problem, the *Dynamic TCP Acknowledgment Problem*, was introduced in [5,6]. Since it usually does not make sense to delay the transmission of large messages, the focus for packet bundling is small messages, and acknowledgments to the receipt of packets are examples of such. The problem was studied as an on-line problem [3] with the cost function being the number of packets sent plus the sum of the latencies for each message. The latency of a message is the time from when the message is available until it is sent. In general for on-line problems, a flow-time cost measure [3] for a problem is a cost function defined as the sum of the lengths of time intervals over all requests from when each request was made until the treatment of it is completed. However, a cost function which contains the above as an important ingredient is also sometimes referred to as a flow-time cost measure.

Flow-time is used as a measure in many different contexts. With regards to competitive analysis, it has been established as a standard measure, most notably in scheduling (single as well as multiple machines) where it is used in, for example, [2,9,12,14,15]. As noted in [5,10], it also seems like an obvious first choice with regards to the present problem.

For the Dynamic TCP Acknowledgment Problem, an exact characterization of the optimal algorithms for the deterministic case can be found in [5]. For the randomized case, an exact characterization is given by the lower bound in [18] and the upper bound in [10]. The off-line version of the problem has also been considered, initiated by [5,6], and a linear-time algorithm has been obtained [17].

In this paper, we consider a different approach to investigating the Dynamic TCP Acknowledgment Problem: we prioritize choosing a model for distributed computing which incorporates the gap between messages necessitated by the physical properties of today's networks and, related to this decision, we also choose a different cost function.

One of the models of computation for distributed computing which seems to be accepted as a reasonable model by practitioners as well as theoreticians is the LogP model [4], which

is tractable from a theoretical point of view, but also realistic enough that good theoretical algorithms are likely to be good in practice as well; on many different platforms. The ingredients in the model are the latency L, the processor overhead in sending messages o, the gap imposed by the network between messages g, and the number of processors P. We refer to the original paper for a complete treatment, but give a short description of the model here.

The times when messages become available are beyond our control. Some application program simply hands over its message to the TCP protocol. When a message is sent by the TCP software, there is a CPU overhead in preparing the message. The message is then sent through the network, which takes time L, after which there is a CPU overhead at the receiving end to extract the message. In addition, there is a requirement that all messages in the network must be separated by a fixed time gap. Note that overhead and gap can overlap in the following sense: Right after a message is sent, while the CPU spends overhead time preparing the next message, the first message is traveling through the network, so after the next message has been prepared, there is already a gap of size equal to the overhead. Thus, if the overhead is larger than the gap, then the gap can be ignored, and if the overhead is much smaller than the gap, then the overhead can be ignored.

In our presentation, we will ignore the overhead. Thus, our results apply when the gap is significantly larger than the overhead. This decision has been made primarily to keep formulations of theorems cleaner and proofs simpler. However, there is support for that decision. In [4], the designers of the LogP model express hope that the overhead can be eliminated from the model at a later time, though they considered it premature (for all architectures; not selected architectures) at the time of their publication. They go on to say: "In future machines, we expect architectural innovations in the processor-network interface to significantly reduce the value of o with respect to g." In [13], the system analyzed to verify their results shows a gap value more than 50 times as large as the overhead. Whereas this system may not be typical, we believe it may be an important example since it resembles the scenario in Grid Computing.

The LogP model has been extended to improve the treatment of large messages [1] and the model is supported by experimental work, supplying methods for determining the concrete values of the parameters on a given system [13].

Comparing the theoretical work of [5,6,10] with the model assumptions of [4], the most noticeable difference is the lack of the gap parameter in [5,6,10]. In their work, packets are allowed to be sent arbitrarily small distances apart. This gives different results, because with decreasing intervals between messages, due to the latency contribution of each packet to the cost function, a good algorithm would send frequently. With small enough distances between messages which must be sent, the algorithm would wish to send more frequently than allowed by the gap we are enforcing.

We base our theoretical work on the LogP model, which means we respect the physical gap. At times when messages become available separated by very small time intervals, a flow-time based cost function, as in [5,10], would impose a very large penalty, if the decision is to delay transmission. This led us to consider another cost function: the sum of time intervals when at least one unsent message is available. In our opinion, this measure is just as natural and it has the significant advantage that good and bad algorithms can be distinguished.

Our results and the ones in [5,10] are largely incomparable. They supplement each other by providing separate results for applications where some physical gap is significant and applications where it is insignificant, or possibly not present at all.

In [6], a cost function similar to ours is used. Though the model used in [6] does not include a requirement concerning packet distances, their alternative cost function ($f_{\max}$) will probably more accurately predict the behavior of algorithms in a scenario where a physical gap must be respected than any of the flow-time based cost functions will.

Packet bundling has been considered experimentally in [16] for a concrete specialized application, namely real-time simulations. However, the conditions are quite different in that messages have different priorities. The goal is also different since there is an absolute tolerance for the delay of messages of different types.

We analyze natural families of deterministic and randomized algorithms for the problem and find the optimal algorithms for these families. Additionally, we show that these results could not have been obtained using the standard flow-time cost function. In our opinion, it is an interesting consequence that our step towards a possibly more realistic computational model as a basis for the analysis forces us away from this standard measure.

2. Packet bundling

Referring to the description of the LogP model from the introduction, since we are considering the situation from the perspective of a single processor, only the overhead and gap parameters are relevant, and as discussed in the introduction, we investigate the case where $o \ll g$, so that the overhead parameter o may be ignored. In the remainder of the paper, we normalize with respect to the gap g, and assume that it is one (the important fact is that it is different from zero).

With this in mind, we now state the problem in a form which is formal, but also convenient to work with.

Consider the problem of a person A wishing to send small messages to another person B. All messages have to be sent using the same, single messenger, who uses one time unit for each delivery, i.e., when the messenger has left with one or more messages (a packet), no messages can be sent for the next one time unit.

When A decides to send a message, A can either send it immediately (assuming that the messenger is in), or wait some time (probably less than one) to see whether other messages have to be sent, so that these messages can be sent together.

A possibly helpful analogy is to think of a hotel/airport shuttle scenario, assuming that the shuttle bus is large enough to carry any number of passengers. When the hotel manager decides to send the bus off, there is obviously a gap until it can be sent again, since it has to return to the hotel first.

The formal definition of the problem is as follows:

Definition 1. A *Packet* is a collection of messages. When a packet is sent, it contains all messages which have arrived since the time the last packet was sent (if it is the first packet, it contains all messages which have arrived up to the point in time where the packet is sent).

In the *Packet Bundling Problem* one is given a sequence $\sigma = \langle a_1, \ldots, a_n \rangle$ of message arrival times and is asked to give a sequence of packet times $p_1, \ldots, p_m$ at which packets of messages are sent. All messages are considered to be small, so that an unlimited number of messages can fit in a packet.

The set of messages sent in packet p_i is denoted $\tilde{p}_i$. For convenience, we identify a message with its arrival time, justifying the notation $a_j \in \tilde{p}_i$. If more than one message can arrive at the same time, $\tilde{p}_i$ can be thought of as a multi-set.

The packets should respect the following restrictions:

- All messages should be sent no earlier than their arrival time, i.e.,

$$\forall i \leqslant n \; \exists j \leqslant m: \quad a_i \in \tilde{p}_j \wedge a_i \leqslant p_j.$$

- All packet times should be at least one unit apart, i.e.,

$$\forall i < m: \quad p_{i+1} - p_i \geqslant 1.$$

- If a packet is sent at time p_i, then the set of messages contained in the packet are considered delivered at time $p_i + 1$.

The last two bullets capture the essence of the gap parameter in the LogP model.

The cost function measures the total time elapsed while there are messages which have arrived, but have not been delivered:

$$\sum_{i=1}^{m} \Big((p_i + 1) - \max\Big\{ (p_{i-1} + 1), \min_{a_j \in \tilde{p}_i} a_j \Big\} \Big)$$

where we define $p_0 = -\infty$.

Considering the cost function above, it is clear that if the formal definition should reflect our informal definition, then no more than $(p_i + 1) - \min_{a_j \in \tilde{p}_i} a_j$ should be paid for messages sent in p_i, i.e., we pay from when the first message in p_i arrived until the time $p_i + 1$ where we consider the messages delivered. However, this is sometimes too much, which can be seen as follows:

Assume that a_q is the last message sent in p_i. If a_{q+1} arrives between the times a_q and $p_i + 1$, then it is sent in p_{i+1}, and so both $(p_i + 1) - \min_{a_j \in \tilde{p}_i} a_j$ and $(p_{i+1} + 1) - \min_{a_j \in \tilde{p}_{i+1}} a_j$ include the interval from a_{q+1} to $p_i + 1$. To avoid this, the contribution from the last term should instead be $(p_{i+1} + 1) - (p_i + 1)$. Including the maximization in the cost function ensures that no interval is ever counted twice.

Equivalently, the cost function could be defined as the integral over a function which has the value one if there are one or more undelivered messages and zero otherwise.

For further discussions of alternative definitions of cost functions, see the concluding section. There, we will also mention some relations to earlier work which are difficult to discuss before the treatment of flow-time cost in Section 6.

The are two additional issues regarding the modeling of the problem which we discuss now. The first regards the assumption that an unbounded number of small messages can be bundled up into one packet. For the TCP protocol in particular, the acknowledgment of one packet automatically acknowledges all previous packets (packets are numbered

consecutively). Thus, a packet with many acknowledgments is actually no larger than a packet with just one. However, we would like our solutions to be applicable to small messages in general; not just TCP acknowledgments. Fortunately, in most cases, the maximum size of packets is fairly large (several kilobytes), which means that a very large number of message identifiers can actually be bundled, so we believe the assumption is reasonable; not in the least because the worst-case scenarios for our algorithms turn out to be sequences with few and scattered messages.

The other issue regards assumptions about how close together in time messages can become available. In the TCP acknowledgment scenario, and also in general, there is essentially no physical limit as to how close together messages can become available and become ready for delivery, since an application at any time can hand virtually any number of messages to the part of the operating system implementing the message protocol. Thus, we do not enforce any restrictions on arrival times.

We consider on-line algorithms [3]. The messages arrive over time, and the algorithm has to decide over time when to send a packet without any knowledge of the existence of future messages. For any algorithm, *ALG*, and input sequence, σ, we let $ALG(\sigma)$ denote the value of the cost function when *ALG* is run on σ.

The performance of deterministic algorithms is measured in comparison with the optimal off-line algorithm, *OPT*, using the standard competitive ratio [3,8,11,19]. *OPT* knows the entire input sequence, when it decides when to send each packet, and can hence achieve the minimum cost.

An algorithm *ALG* is (strictly) c-competitive for a constant c, if for all input sequences, σ, the following holds: $ALG(\sigma) \leqslant c \cdot OPT(\sigma)$. The infimum of all such values c is called the competitive ratio of *ALG*.

The performance of randomized algorithms is measured likewise, though using the expected cost, $E[ALG(\sigma)]$, instead.

3. The A_k & RA_Δ algorithm families

In this section, we first consider a family of deterministic on-line algorithms for the problem. Subsequently we consider a natural randomization of this family and in both cases we show tight results on the competitive ratio.

3.1. A_k-deterministic algorithms

The family of algorithms we consider is defined as follows:

A_k: When a message arrives, it is sent together with all messages (if any) arriving after this one at the earliest possible time after k time units.

Without loss of generality, we assume that if A_k decides to let the messenger leave at a certain point in time, and one or more messages arrive exactly when the messenger is about to leave, then the messenger leaves without these new messages. If this is a problem in a proof, then all messages arriving when the messenger leaves can be considered to arrive

$\epsilon \in o(1)$ time units later. Because of the infimum which is taken in the definition of the competitive ratio, this will generally not alter the result.

The algorithm family A_k is similar to the algorithms greedy$_{\text{new}}$ from [6] for varying η's (see Section 6), in that the first message to arrive determines when the next packet is sent.

The following theorem states the competitive ratio of this family of algorithms:

Theorem 2. *The competitive ratio of A_k is*:

$$\mathcal{R}(A_k) = \begin{cases} 1 + 1/(1+k), & \text{if } 0 \leqslant k < \hat{\varphi}, \\ 1 + k, & \text{if } \hat{\varphi} \leqslant k, \end{cases}$$

where $\varphi = (1+\sqrt{5})/2 \approx 1.618$ and $\hat{\varphi} = \varphi - 1$. The best ratio φ is achieved by $A_{\hat{\varphi}}$.

For a fixed algorithm, A_k, any input sequence for our problem can be divided into phases as follows: Each phase starts with the arrival of the first message after the previous phase has ended, and ends at the earliest possible time when there are no messages to deliver and the messenger is in. In the special case when the messenger returns at the exact same time a new message arrives (and no other messages are due for delivery), the phase ends, and the new message starts the next phase.

In the proof, we need the following two lemmas:

Lemma 3. *For a worst-case sequence for A_k, we can assume that the messenger carries only one message at a time.*

Proof. Consider a worst-case sequence, and assume that A_k has a $\tilde{p}_i$ of size larger than one. We can construct another sequence where $|\tilde{p}_i| = 1$ and the competitive ratio is no better.

Adjusting the sequence by removing messages which do not give rise to the minimum arrival time among the messages in its packet will not change A_k's behavior and will leave the competitive ratio unchanged, since *OPT* can send fewer messages without increasing its cost. □

Lemma 4. *There exists a worst-case sequence for A_k where, if any messages arrive when the messenger is out, they arrive exactly at the point in time where the messenger leaves or returns.*

Proof. We consider a worst-case sequence where a message arrives in the time interval, $(p_i, p_i + 1)$, when A_k's messenger is out, and neither is leaving nor returning, and we transform this sequence into an equally bad sequence, where one message fewer arrives in this interval.

The message which is being delivered in the time interval $[p_i, p_i + 1]$ must have arrived no later than time $p_i - k$ in order to be sent at time p_i. Now assume that a message arrives at time a_j, where $p_i < a_j < p_i + 1$. By Lemma 3, we may assume that this is the only message arriving in this time interval, since otherwise at least two messages would be carried the next time.

If $a_j \in (p_i, p_i + 1 - k)$ (which may be an empty interval, if $k \geqslant 1$), then A_k's messenger has to leave with the new message immediately after returning at time $p_i + 1$, and A_k's cost

is not influenced by exactly when in the interval $[p_i, p_i+1-k]$ the message arrives. If *OPT* sends the message together with previous messages, then *OPT*'s cost will be minimized for $a_j = p_i$. Otherwise a_j can be assumed to be $p_i + 1 - k$ as *OPT*'s cost cannot increase if this message arrives later. The time $p_i + 1 - k$ is still while A_k's messenger is out, but that will be dealt with in the next case.

If $a_j \in [p_i + 1 - k, p_i + 1)$, this message will be sent by A_k at time $a_j + k$. As the previous message arrived at some time before $a_j - 1$, *OPT* can be assumed not to send the new message together with the previous. Thus, if the new message is shifted together with all later messages by $p_i + 1 - a_j$ time units, then *OPT*'s cost will be the same, whereas A_k's cost does not decrease. Consequently, we may assume that $a_j = p_i + 1$. □

Proof of Theorem 2. Let us first consider the case when $k \leqslant 1$:

By Lemmas 3 and 4, a worst-case sequence can be assumed to consist of phases of the following form

$$\sigma_1 = \langle 0 \rangle \quad \text{or} \quad \sigma_n = \big\langle 0, k, k+1, k+2, \ldots, k+(n-2) \big\rangle$$

where n is the number of messages. We separate each phase from the next by more than two time units. By definition of A_k, this means that messages from different phases cannot interfere, so relative costs can be calculated separately for each phase.

A_k's cost is $A_k(\sigma_n) = k + n$, whereas *OPT*'s cost is

$$OPT(\sigma_n) = \begin{cases} k+n-1, & \text{if } n \neq 1, \\ 1, & \text{if } n = 1. \end{cases}$$

For $n = 1$, this gives a competitive ratio of $A_k(\sigma_1)/OPT(\sigma_1) = k + 1$, and for $n > 1$, it gives a competitive ratio of $A_k(\sigma_n)/OPT(\sigma_n) = (k+n)/(k+n-1)$, which is maximized for $n = 2$, where $A_k(\sigma_2)/OPT(\sigma_2) = (k+2)/(k+1) = 1 + 1/(1+k)$.

Comparing the two cases, we find that σ_2 is the worst possible for $k \leqslant \hat{\varphi}$, whereas σ_1 is worst for $\hat{\varphi} \leqslant k \leqslant 1$.

Let us then consider the case when $1 < k$. Again by Lemmas 3 and 4, a worst-case sequence can be assumed to consist of phases of the following form:

$$\sigma_n = \big\langle 0, k, 2k, \ldots, k(n-1) \big\rangle$$

where n is the number of messages. Each phase is separated from the next by more than $1 + k$ time units, so relative costs can be calculated separately for each phase.

A_k's cost is $A_k(\sigma_n) = 1 + kn$, whereas *OPT*'s cost is $OPT(\sigma_n) = n$. This gives a competitive ratio of $A_k(\sigma_n)/OPT(\sigma_n) = (1+kn)/n = 1/n + k$, which is maximized at $n = 1$, where $A_k(\sigma_1)/OPT(\sigma_1) = 1 + k$. □

3.2. RA_Δ-randomized algorithms

We now consider a natural randomization of the A_k family of on-line algorithms. Our deterministic algorithm family A_k chose a specific k and sent a message at the earliest possible time after k time units. RA_Δ chooses the interval it waits at random.

RA_Δ: Choose a k uniformly at random between 0 and Δ, and then run the corresponding A_k algorithm on the input sequence.

We only consider algorithms with $\Delta \leqslant 1$, since any algorithm, RA_Δ, with $\Delta > 1$ easily can be seen to have a competitive ratio larger than RA_1.

One could also consider other families of randomized algorithms. Instead of using a uniform distribution, we could have used an exponential distribution with parameter Δ varying from zero to infinity, or a cut-off exponential distribution described by the density function (as in [10]): $f(\delta) = e^{-\delta/\Delta}/(\Delta(1 - e^{-1}))$ for $\delta \in [0, \Delta]$, and zero otherwise. By careful examination, both of these are for any Δ easily shown to have a worse competitive ratio than the best member of the RA_Δ-family. Further, one could consider a randomized algorithm, where the time interval to wait is chosen uniformly by random at the arrival of the first message of each packet. In [7], this is shown to yield at best a $\frac{1}{2}\sqrt[3]{1/2} + 1 \approx 1.397$ competitive algorithm when the waiting time is chosen uniformly by random in $[0, \sqrt[3]{1/2}]$.

Without loss of generality, we will as with A_k assume that messages arriving exactly when the messenger leaves will not be delivered immediately. For $\Delta > 0$, this does not make any difference to the expected competitive ratio as k is chosen uniformly at random in the range $[0, \Delta]$. For $\Delta = 0$, RA_0 and A_0 behave identically, and we can as for A_0 consider all messages arriving when the messenger leaves, as arriving $o(1)$ time later without any difference.

The competitive ratio for RA_Δ is given by the following theorem.

Theorem 5. *The expected competitive ratio of RA_Δ is*

$$\overline{\mathcal{R}}(RA_\Delta) = \begin{cases} \dfrac{1}{2} + \dfrac{3}{2(\Delta+1)}, & \text{if } 0 \leqslant \Delta \leqslant \sqrt{3} - 1, \\ \dfrac{\Delta}{2} + 1, & \text{if } \sqrt{3} - 1 \leqslant \Delta \leqslant 1. \end{cases}$$

The best ratio $\sqrt{3}/2 + 1/2 \approx 1.366$ is achieved by $RA_{\sqrt{3}-1}$.

As for A_k, the theorem is shown by constructing a worst-case input sequence. For a fixed RA_Δ, any input sequence is divided into phases almost as before: Each phase starts with the arrival of the first message after the previous phase has ended, and ends exactly when, regardless of the random choice of k, there are definitely neither any messages waiting to be sent nor is the messenger out. In the event that a new message arrives at the exact same time as the messenger returns, and where no random choice of k would have made the messenger arrive later, the phase ends, and the new message starts the next phase. Due to linearity of expectation, it is enough to consider a worst-case phase.

Before showing the main theorem, we need the following lemmas:

Lemma 6. *Messages in a worst-case phase for RA_Δ are not further than one apart, i.e.,* $\forall i\colon a_{i+1} - a_i \leqslant 1$.

Proof. Let a_i be the first message such that $a_{i+1} > a_i + 1$. As all messages before a_i are at most one apart, *OPT* can send the messages $a_1, \ldots, a_i$ at time a_i, so that it does not incur any cost between time $a_i + 1$ and a_{i+1}. This means that the arrival of message a_{i+1} (together with all other messages after message a_{i+1}) can be shifted to any point in time further ahead without increasing the cost of *OPT*.

Since we only consider $\Delta \leqslant 1$, RA_Δ will make message a_i leave with the messenger at time $a_i + 1$ at the latest. So, message a_{i+1} arrives either when the messenger is out or when the messenger has returned (and then no other messages will be waiting at that time). The cost of RA_Δ is maximal if the randomly chosen waiting time after a_{i+1} is not shared by time where the messenger is out, i.e., the cost is maximal if message a_{i+1} arrives after the messenger returns, and thereby the message is not in the same phase, but in the next. □

Lemma 7. *For a worst-case phase with messages $\sigma = \langle (a_1 = 0), \ldots, a_m \rangle$, the expected competitive ratio of RA_Δ is at most*

$$\overline{\mathcal{R}}(RA_\Delta) = \frac{E[RA_\Delta(\sigma)]}{OPT(\sigma)} \leqslant \frac{a_m + 2}{a_m + 1} = 1 + \frac{1}{a_m + 1}.$$

Proof. Follows directly from Lemma 6. □

The following lemma shows that a worst-case phase with $\sigma = \langle (a_1 = 0), \ldots, a_m \rangle$ and $a_m \leqslant 1$ can be assumed to contain at most two messages:

Lemma 8. *For any phase with messages $\sigma = \langle (a_1 = 0), \ldots, a_m \rangle$ and $a_m \leqslant 1$, the phase obtained by looking only at the first and the last message of σ, $\sigma' = \langle a_1, a_m \rangle$, has the same expected competitive ratio, i.e.,*

$$\frac{E[RA_\Delta(\sigma)]}{OPT(\sigma)} = \frac{E[RA_\Delta(\sigma')]}{OPT(\sigma')}.$$

Proof. Since $a_m \leqslant 1$, we have $OPT(\sigma) = OPT(\sigma') = a_m + 1$. For RA_Δ, we consider two cases. Let k be the (now fixed) value randomly chosen by RA_Δ. If $a_m < k$, then the cost of σ is the same as for σ'. If $a_m \geqslant k$, then message a_m is not sent until the messenger leaves the next time. This point of time is determined by the first message, a_i, with $a_i \geqslant k$. Since $a_i + k \leqslant a_m + k \leqslant k + 1$, this will be exactly at the messenger's next return. Leaving out the messages before message a_m (but after) a_1 does not change this, and the cost of σ' and σ is the same. □

Furthermore, as the next lemma shows, if $a_m \leqslant 1$, then in addition to assuming that $m \leqslant 2$, we can assume that $a_m \in \{0, \Delta\}$. Note that for $\Delta > 0$, due to the definition of our cost function, $a_2 = 0$, i.e., $\sigma = \langle 0, 0 \rangle$ gives the same expected competitive ratio as $\sigma = \langle 0 \rangle$. For $\Delta = 0$, the sequence $\sigma = \langle 0, 0 \rangle$ is the same as $\sigma = \langle 0, \Delta \rangle$.

Lemma 9. *A worst-case input sequence for RA_Δ with two messages, $\sigma = \langle (a_1 = 0), a_2 \rangle$, where $a_2 \leqslant 1$, can be assumed to have $a_2 \in \{0, \Delta\}$. This gives the following lower bounds, of which at least one is an upper bound for all input sequences $\sigma = \langle (a_1 = 0), \ldots, a_m \rangle$, where $a_m \leqslant 1$:*

When input is restricted to be of the form $\sigma = \langle 0 \rangle$, RA_Δhas an expected competitive ratio of

$$1 + \frac{\Delta}{2}.$$

When input is restricted to be of the form $\sigma = \langle 0, \Delta\rangle$, RA_Δ has an expected competitive ratio of

$$\frac{1}{2} + \frac{3}{2(\Delta+1)}.$$

Proof. For $\sigma = \langle 0\rangle$, the expected competitive ratio of RA_Δ is $E[RA_\Delta(\sigma)]/OPT(\sigma) = 1 + \Delta/2$.

For $\sigma = \langle 0, a_2\rangle$, we prove the result by case analysis depending on the values of a_2 and Δ. Let k be the (now fixed) value randomly chosen by RA_Δ. The cost of RA_Δ can then be described as follows:

$$c(a_2, k) = \begin{cases} k+1, & \text{if } a_2 \leqslant k, \\ k+2, & \text{if } k < a_2 \leqslant 1. \end{cases}$$

Note that the above holds with equality even in the second case since $a_2 \leqslant 1$ implies $a_2 + k \leqslant k + 1$. Thus, a_2 will be sent at time $k+1$.

The expected cost of RA_Δ is $E[RA_\Delta(\sigma)] = \frac{1}{\Delta}\int_0^\Delta c(a_2, k)\,\mathrm{d}k$, whereas $OPT(\sigma) = a_2 + 1$, giving us an expected competitive ratio of $c(a_2) = \frac{1}{\Delta(a_2+1)}\int_0^\Delta c(a_2, k)\,\mathrm{d}k$.

Let us first consider the case when $a_j \leqslant \Delta$. The expected competitive ratio is

$$c(a_2) = \frac{\int_0^{a_2}(k+2)\,\mathrm{d}k + \int_{a_2}^{\Delta}(k+1)\,\mathrm{d}k}{\Delta(1+a_2)} = \frac{\Delta^2/2 + \Delta + a_2}{\Delta(1+a_2)}.$$

By differentiation, we find that for $\Delta \leqslant \sqrt{3} - 1$, this is maximal in $a_2 = \Delta$, whereas for $\Delta \geqslant \sqrt{3} - 1$, this is maximal in $a_2 = 0$. For $\Delta = \sqrt{3} - 1$ and any $a_2 \in [0, \Delta]$, $c(a_2) = \frac{1}{2}(\sqrt{3}+1)$.

The second case is when $\Delta < a_2 \leqslant 1$. The expected competitive ratio is

$$c(a_2) = \frac{\int_0^{\Delta}(k+2)\,\mathrm{d}k}{\Delta(1+a_2)} = \frac{\Delta/2 + 2}{1+a_2}.$$

This is maximal for a_2 as small as possible, i.e., it is at most $c(\Delta) = 1/2 + 3/(2(1+\Delta))$. □

Lemma 10. *Let $\sigma = \langle (a_1 = 0), \ldots, a_m\rangle$ be any worst-case phase for RA_Δ with $1 < a_m \leqslant 1 + \Delta$, then*

$$\frac{E[RA_\Delta(\sigma)]}{OPT(\sigma)} \leqslant \frac{-2{a_m}^2 + 6a_m - 4 + 2a_m\Delta + \Delta^2 + 2\Delta}{2\Delta(a_m+1)}.$$

Proof. As before, let k be the (now fixed) value randomly chosen by RA_Δ. The worst-case cost of RA_Δ can in this case be described as follows:

$$c(a_m, k) \leqslant \begin{cases} a_m + k + 1, & \text{if } a_m < k+1, \\ k+3, & \text{if } k+1 \leqslant a_m. \end{cases}$$

This gives an expected worst-case cost for RA_Δ of at most

$$E\big[RA_\Delta(\sigma)\big] \leqslant \frac{1}{\Delta}\int_0^\Delta c(a_m,k)\,\mathrm{d}k = \frac{1}{\Delta}\left(\int_0^{a_m-1}(k+3)\,\mathrm{d}k + \int_{a_m-1}^{\Delta}(a_m+k+1)\,\mathrm{d}k\right)$$
$$= \frac{-2a_m^2+6a_m-4+2a_m\Delta+\Delta^2+2\Delta}{2\Delta}$$

whereas $OPT(\sigma) = a_m + 1$, since σ is a worst-case phase. □

Proof of Theorem 5. Lemma 9 gives the ratio of the cost functions of RA_Δ and OPT on some selected sequences. Thus, the expected competitive ratio of RA_Δ is at least as high as those ratios. However, the ratios are also best possible for phases $\sigma = \langle (a_1 = 0), \ldots, a_m\rangle$ with $a_m \leqslant 1$. By Lemma 7, if $a_m > 1$, the ratio of the cost functions of RA_Δ and OPT on a worst-case input sequence is less than $3/2$.

For $\Delta \leqslant 1/2$, Lemma 9 is enough, since the input sequence $\langle 0, \Delta\rangle$ gives rise to an expected ratio of $1/2 + 3/(2(\Delta+1)) \geqslant 3/2$.

For $\Delta > 1/2$, by Lemma 7, any input sequence with $a_m > 1+\Delta$ gives rise to an expected ratio of at most $((1+\Delta)+2)/((1+\Delta)+1)$. Lemma 10 gives a similar bound on the expected ratio for $a_m \in (1, 1+\Delta]$. It can easily be shown that

$$\max\left\{\frac{\Delta+3}{\Delta+2}, \frac{-2a_m^2+6a_m-4+2a_m\Delta+\Delta^2+2\Delta}{2\Delta(a_m+1)}\right\}$$
$$\leqslant \max\left\{1+\frac{\Delta}{2}, \frac{6\Delta^2+4\Delta+1}{4\Delta(\Delta+1)}\right\}.$$

Thus, either $\langle 0\rangle$ or $\langle 0, \Delta\rangle$ is a worst-case input sequence in this case. Now, Lemma 9 gives the result. □

4. The B_k & RB_m algorithm families

We will now study the problem from another angle. In the previous section we studied both a deterministic and a randomized family of on-line algorithms. For the deterministic family, no randomness is needed, whereas for the randomized family, essentially infinitely many random bits are necessary to choose a number uniformly between 0 and Δ. In this section we consider the cases in between. The algorithm is allowed to choose a random value only once and to do so uniformly at random from a set of m values.

Let $\langle a_1, a_2, \ldots, a_n\rangle$ be the sequence of arrival times of messages. The sequence can be broken into phases where a phase contains a maximal subsequence of message arrivals for which consecutive messages are at most one time unit apart. In other words, the first phase starts with the first packet. A phase continues until there are consecutive revealed messages a_i and a_{i+1} which are more than one time unit apart. Message a_i is then the last message in that phase and a_{i+1} is the first packet in the next phase. Note that the definition of phases used in this section is different from the definition used in the previous section.

Let u_p and v_p be the arrival time of the first and the last messages in phase p, respectively. For a given schedule, let s_p be the time the messenger leaves to service the first

message in phase p, and let w_p be the time the messenger comes back from servicing the last message in phase p. To simplify the discussion, we define a dummy variable $w_0 = a_1$.

Let F be the total cost of an algorithm and let F_p be the cost charged to phase p. For any $p \geqslant 1$, any waiting time in the interval $[w_{p-1}, w_p]$ will be charged to phase p. Since the entire schedule is covered, then $F = \sum_p F_p$.

Lemma 11. *For any* $p \geqslant 1$, $F_p = w_p - \max\{w_{p-1}, u_p\}$.

Proof. Suppose $p \geqslant 1$ is fixed. Since consecutive messages in phase p are at most one time unit apart, at any time in the interval $[u_p, w_p]$, either there is a pending message, or the messenger is out. Now consider the time interval $[w_{p-1}, w_p]$. If $u_p < w_{p-1}$, then the entire interval $[w_{p-1}, w_p]$ is charged to phase p. If $w_{p-1} \leqslant u_p$, then during the time interval $[w_{p-1}, u_p]$ the messenger is in and there are no pending messages. Thus, there is no charge to phase p during the time interval $[w_{p-1}, u_p]$. Only the interval $[u_p, w_p]$ is charged to phase p. In any case, the charge is $w_p - \max\{w_{p-1}, u_p\}$. □

We define the deterministic on-line algorithm B_k for the packet bundling problem as follows:

B_k: When the first message of a phase arrives, the algorithm will wait for k time units. After this waiting period, the algorithm sends out any messages in the phase as soon as possible.

The algorithm B_k and the algorithm A_k defined in the previous section are different. To illustrate this, consider the input sequence $\sigma = \langle 0, k-\epsilon, 1+k-\epsilon, 1+k+\epsilon \rangle$.

Upon the arrival of the first message, A_k will wait for k time units. It sends out the first and the second messages with the messenger at time k. The messenger will return at time $1+k$ just after the third message arrives. A_k will wait for k time units after the arrival of the third message. It sends out the third and the fourth messages with the messenger at time $1+2k-\epsilon$. The cost of A_k is $2+2k-\epsilon$.

Now consider the schedule produced by B_k. Since the messages of σ arrive no more than one time unit apart, they all belong to the same phase. Upon the arrival of the first message, B_k will wait for k time units. It sends out the first and the second messages with the messenger at time k. The third message arrives at time $1+k-\epsilon$ just before the messenger returns. Since the third message belongs to the same phase, when the messenger returns at time $1+k$, it will be sent out again immediately. This is different from A_k. The fourth message arrives at time $1+k+\epsilon$. The fourth message has to wait for the messenger to return at time $2+k$ before it can be serviced. The cost of B_k is $3+k$, which is worse than A_k.

In contrast, consider the same example without the last message. The cost of A_k is still $2+2k-\epsilon$, but the cost of B_k is now $2+k$.

Let F_p^k be the cost of algorithm B_k in phase p.

Lemma 12. *For* $0 < k \leqslant 1$ *and for any phase* p,

$$F_p^k \leqslant \begin{cases} k+1+j, & \text{if } x < k, \\ k+2+j, & \text{if } x \geqslant k, \end{cases}$$

where $j = \lfloor v_p - u_p \rfloor$ *and* $x = (v_p - u_p) - j$.

Proof. Suppose k is fixed and suppose we consider any phase p. The messenger can leave to service the first message in phase p when k time units after the first arrival in the current phase have elapsed and the messenger has returned from servicing the last message in the previous phase. Thus, $s_p = \max\{u_p + k, w_{p-1}\}$.

From the definition of B_k, beginning at time s_p the messenger will leave every 1 time unit to service the messages in the phase. It leaves to service the last message at time $s_p + \lfloor v_p - s_p \rfloor + 1$, and will come back 1 time unit later. Thus, $w_p = s_p + \lfloor v_p - s_p \rfloor + 2$.

Let $j = \lfloor v_p - u_p \rfloor$ and $x = (v_p - u_p) - j$. Note that $0 \leqslant x < 1$ and $j + x = v_p - u_p$. Now we compute F_p^k.

$$\begin{aligned} F_p^k &= w_p - \max\{u_p, w_{p-1}\} \\ &= \max\{u_p + k, w_{p-1}\} + \lfloor v_p - s_p \rfloor + 2 - \max\{u_p, w_{p-1}\} \\ &\leqslant k + \lfloor v_p - u_p - k \rfloor + 2 = k + \lfloor j + x - k \rfloor + 2 \\ &= \begin{cases} k+j+2, & \text{if } x \geqslant k, \\ k+j+1, & \text{if } x < k. \end{cases} \qquad \square \end{aligned}$$

We define the randomized on-line algorithm RB_m for the packet bundling problem as follows:

RB_m: RB_m is a random distribution over the class of algorithm B_k. First, RB_m picks a value for an internal parameter t. Then, RB_m behaves like B_t. In particular, the value for t is randomly chosen to be $t_i(m)$ with probability $1/m$ for $i = 1, \ldots, m$ where

$$t_1(m) = \frac{\sqrt{3 + 2/m} - 1}{m+1}, \qquad t_i(m) = i \cdot t_1(m), \quad \text{for } i = 2, \ldots, m.$$

Since RB_m behaves like B_t, it does the following. When the first message of a phase arrives, RB_m will wait for t time units before this message and later messages (if any) are sent. After that if there are more messages in the phase, the messages are sent out as soon as possible.

Note that the m in RB_m denotes the number of random choices available to the algorithm, and not as for RA_Δ, where Δ denotes which interval the algorithm should choose its random values among.

Theorem 13. *For any* $m \geqslant 1$, *the competitive ratio of* RB_m *is at most*

$$\overline{\mathcal{R}}(RB_m) \leqslant \frac{1}{2}\left(\sqrt{3 + \frac{2}{m}} + 1\right).$$

Proof. Suppose m is fixed and suppose we consider a worst case phase σ_p. For brevity, we use t_i to represent $t_i(m)$. Also, we let $t_0 = 0$ and $t_{m+1} = 1$.

Let $j = \lfloor v_p - u_p \rfloor$ and $x = (v_p - u_p) - j$. Note that $0 \leqslant x < 1$ and $j + x = v_p - u_p$. Let q be the integer such that $t_q \leqslant x < t_{q+1}$.

From the definition of t_i and x, it is the case that $0 \leqslant q \leqslant m$. By Lemma 12, the cost of RB_m can be bounded from above by

$$\begin{aligned} RB_m(\sigma_p) = E_i\big[F_p^{t_i}\big] = \frac{1}{m}\sum_{i=1}^{m} F_p^{t_i} &\leqslant \frac{1}{m}\left(\sum_{i=1}^{q}(t_i + 2 + j) + \sum_{i=q+1}^{m}(t_i + 1 + j)\right) \\ &= \frac{1}{m}\left(\sum_{i=1}^{q}(i \cdot t_1 + 2 + j) + \sum_{i=q+1}^{m}(i \cdot t_1 + 1 + j)\right) \\ &= \frac{1}{m}\left(\sum_{i=1}^{m}(i \cdot t_1 + 1 + j) + \sum_{i=1}^{q} 1\right) \\ &= \frac{1}{m}\left(m(1+j) + t_1 \cdot \frac{m(m+1)}{2} + q\right) = 1 + j + t_1 \cdot \frac{m+1}{2} + \frac{q}{m}. \end{aligned}$$

The optimal schedule is to wait for the last message in the phase and send all messages in one packet at time v_p. Thus, the optimal cost is

$$OPT(\sigma_p) = 1 + (v_p - u_p) = 1 + j + x \geqslant 1 + j + t_q = 1 + j + q \cdot t_1.$$

Next, we find an upper bound for the ratio $RB_m(\sigma_p)/OPT(\sigma_p)$.

$$\begin{aligned} \frac{RB_m(\sigma_p)}{OPT(\sigma_p)} &\leqslant \left(1 + j + t_1 \cdot \frac{m+1}{2} + \frac{q}{m}\right)\left(\frac{1}{1 + j + q \cdot t_1}\right) \\ &\leqslant \left(1 + t_1 \cdot \frac{m+1}{2} + \frac{q}{m}\right)\left(\frac{1}{1 + q \cdot t_1}\right) \\ &= \left(\frac{m+1}{2}\frac{\sqrt{3 + 2/m} - 1}{m+1} + 1 + \frac{q}{m}\right)\left(\frac{1}{1 + q(\sqrt{3+2/m} - 1)/(m+1)}\right) \\ &= \left(\frac{1}{2}\left(\sqrt{3 + \frac{2}{m}} + 1\right) + \frac{q}{m}\right)\left(\frac{1}{1 + \frac{q}{m+1}(\sqrt{3+2/m} - 1)}\right) \\ &= \left(\frac{1}{2}\left(\sqrt{3 + \frac{2}{m}} + 1\right) + \frac{q}{m+1}\left(\sqrt{3 + \frac{2}{m}} - 1\right)\frac{1}{2}\left(\sqrt{3 + \frac{2}{m}} + 1\right)\right) \\ &\quad \times \left(\frac{1}{1 + \frac{q}{m+1}(\sqrt{3+2/m} - 1)}\right) \\ &= \frac{1}{2}\left(\sqrt{3 + \frac{2}{m}} + 1\right). \qquad \square \end{aligned}$$

If RB_m has access to b random bits, it can choose among 2^b random choices.

Corollary 14. *For $b \geqslant 0$, if RB_m has access to b random bits, then*

$$\overline{\mathcal{R}}(RB_m) \leqslant \frac{1}{2}\left(\sqrt{3 + 2^{1-b}} + 1\right).$$

Proof. Immediate from Theorem 13 and by setting $m = 2^b$. □

If RB_m has does not have access to any random bits at all, it reduces to a deterministic algorithm.

Corollary 15. *If RB_m does not have access any random bits, it reduces to a deterministic algorithm, and*

$$\mathcal{R}(RB_0) \leqslant \frac{\sqrt{5}+1}{2} \approx 1.618.$$

Proof. Immediate from Corollary 14 by setting $b = 0$. □

Corollary 16. *If RB_m has access to 1 random bit, then $t_1 = 1/3$, $t_2 = 2/3$, and*

$$\overline{\mathcal{R}}(RB_1) \leqslant 3/2.$$

Proof. Immediate from Corollary 14 by setting $b = 1$. □

If b tends to infinity, RB_m will become a randomized algorithm that chooses a waiting time from a uniform distribution in the range $[0, \sqrt{3}-1] \approx [0, 0.732]$.

Corollary 17. *If RB_m has access to an unlimited number of random bits, it becomes a randomized algorithm that chooses a waiting time from a uniform distribution in the range* $[0, \sqrt{3}-1] \approx [0, 0.732]$. *Furthermore,*

$$\overline{\mathcal{R}}(RB_\infty) \leqslant \frac{\sqrt{3}+1}{2} \approx 1.366.$$

Proof. Immediate from Corollary 14 by letting b tend to infinity. □

5. Lower bounds

Finally, we show that among the algorithms we have considered, we find both optimal deterministic and optimal randomized algorithms.

First, $A_{\hat{\varphi}}$ and consequently by Lemma 15, RB_0 are shown to be optimal deterministic algorithms.

Theorem 18. *Let ALG be any deterministic algorithm for the* Packet Bundling Problem. *Then $\mathcal{R}(ALG) \geqslant \mathcal{R}(A_{\hat{\varphi}}) = \varphi$.*

Proof. We show how to construct an input sequence for *ALG*, where it has a competitive ratio larger than or equal to φ.

The input will be given in a number of phases, each consisting of either one or two messages. Between each phase, there is a time interval large enough so that neither *ALG*

nor $A_{\hat{\varphi}}$ at the end of the interval has any messages to deliver, nor are they at the moment delivering any messages.

Let us first consider phase σ_i, and let the first message in this phase arrive at time a_{i_1}. Let k_i be the length of the time interval *ALG* waits before it sends the message. Referring to Theorem 2, we know for A_{k_i} whether a worst-case phase for A_{k_i} has one or two messages. If it has two, another message is set to arrive at time $a_{i_2} = a_{i_1} + k_i$; if not, the phase ends.

Referring again to Theorem 2, the following holds for phase σ_i:

$$ALG(\sigma_i) \geqslant A_{k_i}(\sigma_i) \geqslant A_{\hat{\varphi}}(\sigma_i) \geqslant \varphi OPT(\sigma_i).$$

The first inequality holds for the following reason. Recall that we only consider sequences of up to two messages. If a sequence contains only one message, since *ALG* is deterministic, it waits some fixed time k_i before it sends the message. So, clearly, if σ_i has length one, $ALG(\sigma_i) = A_{k_i}(\sigma_i)$. For $k_i \leqslant \hat{\varphi}$, a worst-case phase consists of two messages. Again, by definition of A_{k_i}, *ALG* and A_{k_i} will send the first message at the same time k_i. However, whereas A_{k_i} will send the second message immediately after having sent the first (at time $k_i + 1$), we cannot be sure that *ALG* does too. For a sequence of length two, delaying sending the second message can of course only increase the cost. Thus, the first inequality holds for sequences of length at most two.

Thus, for the entire input sequence, we have $ALG(\sigma) \geqslant \varphi OPT(\sigma)$. □

Next, we show a lower bound for any randomized on-line algorithm. As a consequence of this, both $RA_{\sqrt{3}-1}$ and RB_∞ are optimal randomized algorithms.

Although the details of the following proof are somewhat technical, the idea is simple. A message becomes available at time 0. Whenever the behavior of the on-line algorithm in the expected case deviates from that of RB_∞ in a way that reduces its cost on the current sequence by any significant amount, another message becomes available.

Theorem 19. *No randomized on-line algorithm can be better than* $(\sqrt{3}+1)/2$ *competitive.*

Proof. Fix a randomized on-line algorithm *ALG*. We construct a sequence of message arrivals for which the ratio of costs is arbitrarily close to $(1+\sqrt{3})/2$.

Fix $N \gg 1$ and let $\epsilon = (\sqrt{3}-1)/N$. Let a message arrive at time 0. For each $0 < i \leqslant N$ we decide whether a message will arrive at time $i\epsilon$ depending on the behavior of *ALG* prior to time $i\epsilon$. Note that this behavior cannot depend upon whether messages arrive at time $i\epsilon$ or later.

In order to describe the criteria for making the decision at time $i\epsilon$, we need to introduce a technical concept. For $0 \leqslant t \leqslant i\epsilon$, let $p(t)$ be the (partial) probability density function describing the probability density that *ALG* sends a packet at time t. In other words, $P(t) = \int_0^t p(x)\,\mathrm{d}x$ is the probability that *ALG* has sent a packet by time t. Without loss of generality, we can assume that $p(t)$ is continuous (if it is not, there is another algorithm with cost arbitrarily close for which $p(t)$ is continuous).

Call a time t *sufficiently heavy* if for all $s < t$, the inequality $P(t) - P(s) > (t-s) \times (\sqrt{3}+1)/2$ holds. A message arrives at time $i\epsilon$ if there is a time $t \in ((i-1)\epsilon, i\epsilon]$ which is sufficiently heavy.

Let T be the time that the last message arrives and let T' be the supremum of all times which are sufficiently heavy. Note that $0 \leqslant T \leqslant \sqrt{3}-1$ and $T - T' \leqslant \epsilon$. We now claim that

$$P(T') - P(t) \geqslant (T' - t)\big(\sqrt{3}+1\big)/2 \quad \text{for all } t < T'$$

and

$$P(t) - P(T') \leqslant (t - T')\big(\sqrt{3}+1\big)/2 \quad \text{for all } t > T'.$$

The first inequality follows immediately from the definition of T' and properties of limits. To see that the second inequality is true, let $s > T'$ be a time that maximizes $P(t) - P(T') - (t - T')(\sqrt{3}+1)/2$. If this maximum was greater than 0 then s would be sufficiently heavy.

The optimal cost on the sequence is $1 + T$.

Now consider the on-line cost. Suppose the on-line algorithm chooses to send the messenger out at time x, where x is drawn from the probability density function $p(t)$. If $x \leqslant T$, the on-line cost is $2 + x$ because the messenger will come back at time $1 + x$, go out again with the message that arrives at time T, and come back again at time $2 + x$. If $x > T$, the on-line cost is $1 + x$ because no messages arrive after time x, and the messenger will be back at time $1 + x$. Since no messages will arrive after time $\sqrt{3} - 1$, we can assume that $p(t) = 0$ for $t \geqslant (\sqrt{3} - 1) + \epsilon \approx 0.732 + \epsilon$. Thus, the expected on-line cost is

$$\int_0^T (2 + t)\, p(t)\, \mathrm{d}t + \int_T^{\sqrt{3}-1+\epsilon} (1 + t)\, p(t)\, \mathrm{d}t. \tag{1}$$

To more easily estimate the on-line cost, we allow the on-line algorithm to modify its probability density function $p(t)$ subject to the restriction that T' remains the supremum of all times which are sufficiently heavy. From the cost function (1), $2 + t_2 \geqslant 2 > 1 + t_1$ for $t_2 \in [0, T]$ and $t_1 \in (T, \sqrt{3} - 1 + \epsilon]$, to minimize the expected on-line cost, the on-line algorithm will place as much probability mass in $p(t)$ after time T as possible. Furthermore, the mass remaining prior to time T will be moved as early as possible while keeping T' sufficiently heavy. Similarly, the mass occurring after time T will be moved as early as possible while keeping no $t > T$ sufficiently heavy and while still keeping it after time T.

The on-line algorithm will modify its probability density function $p(t)$ so that it is arbitrarily close to the following probability density function $q_T(t)$, which is arbitrarily close to the uniform density distribution in the interval $[0, \sqrt{3} - 1]$.

$$q_T(t) = \begin{cases} \big(\big(\sqrt{3}+1\big)/2\big)^+, & \text{if } t \in [0, T], \\ \big(\big(\sqrt{3}+1\big)/2\big)^-, & \text{if } t \in \big(T, \sqrt{3}-1\big]. \end{cases}$$

It can be verified that, with respect to $q_T(t)$, time T is sufficiently heavy and, any time $t \in (T, \sqrt{3} - 1]$ is not sufficiently heavy. A simple calculation of

$$\int_0^T \frac{(2 + t)(1 + \sqrt{3})}{2}\, \mathrm{d}t + \int_T^{\sqrt{3}-1} \frac{(1 + t)(1 + \sqrt{3})}{2}\, \mathrm{d}t$$

shows that the on-line cost is arbitrarily close to $(1 + T)(1 + \sqrt{3})/2$, and the result follows. □

6. Flow-time cost

The most standard cost function for problems of this nature is flow-time cost, which is also used in [5,6,10]. The flow-time cost for some algorithm is the sum of the waiting times or latencies of all messages. For one message, the latency is the length of the time interval from when it arrives until either the packet time of its packet is reached or it has been delivered (dependent on the definition used).

As stated in [5,6,10], a possible definition of this is the following:

$$\eta m + (1 - \eta) \sum_{i=1}^{m} \sum_{a_j \in \tilde{p}_i} (p_i - a_j).$$

As before, $\eta \in [0, 1]$ denotes the relative weight of the cost of an acknowledgment and the message latency. We will not consider the case when $\eta = 1$, i.e., the case when cost is only paid, if a packet is sent; the choice of any competitive algorithm will then be never to send any packets.

Though we are considering the same cost function in this section as has been considered in earlier work, the problem itself is different since packet times are required to be at least one time unit apart. When trying to use this cost function for our problem, we are unable to distinguish between on-line algorithms, as no on-line algorithm is competitive, i.e., for any on-line algorithm, it is possible to find input sequences giving arbitrarily large competitive ratios.

Theorem 20. *For the* Packet Bundling Problem *using the flow-time cost function with* $\eta \in [0, 1)$, *no deterministic on-line algorithm is competitive.*

Proof. Let *ALG* be any deterministic on-line algorithm. First give the on-line algorithm a message arriving at time 0. The algorithm sends this message in a packet at some time, k. Just after the messenger has left, give s messages all arriving at time k. The cost of *ALG* is then at least $2\eta + (1 - \eta)(k + s)$, whereas *OPT* can send all messages at time k such that its cost is at most $\eta + (1 - \eta)k$.

Since $1 - \eta > 0$, we can by the choice of s get an arbitrarily large competitive ratio, and consequently the algorithm is not competitive. □

The result also holds true for randomized on-line algorithms, although the proof is more complicated:

Theorem 21. *For the* Packet Bundling Problem *using the flow-time cost function with* $\eta \in [0, 1)$, *no randomized on-line algorithm is competitive.*

Proof. Let *ALG* be any randomized on-line algorithm. We again show how to construct an input sequence with an arbitrarily large competitive ratio.

The first part of the input sequence is $s > 0$ messages all arriving at time 0. Let $p(s)$ be the probability that *ALG* decides to send a packet before time $1/2$. This packet may or may not contain all s messages. We now have two cases depending on $p(s)$.

If $p(s) < 1/2$ for all s, then the input sequence only contains these s messages. The expected ratio between the cost of *ALG* and *OPT* is then at least

$$\frac{E[ALG(\sigma)]}{OPT(\sigma)} \geqslant \frac{(1-p(s))(\eta+(1-\eta)s/2)}{\eta} > \frac{2\eta+(1-\eta)s}{4\eta}.$$

Since $1-\eta > 0$, we can get an arbitrarily large ratio by choosing s large enough.

If $p(s) \geqslant 1/2$ for some s, then $t \gg s$ more messages arrive at time $1/2$. The ratio between *ALG* and *OPT* is in this case at least

$$\frac{E[ALG(\sigma)]}{OPT(\sigma)} \geqslant \frac{p(s)(2\eta+(1-\eta)t/2)}{\eta+(1-\eta)s/2} \geqslant \frac{4\eta+(1-\eta)t}{4\eta+2(1-\eta)s}.$$

Since $1-\eta > 0$, we can by choosing t large enough get an arbitrarily large ratio. □

Thus, no on-line deterministic or randomized algorithm is competitive using the flow-time cost function defined earlier.

7. Concluding remarks

We have considered a new cost function instead of the cost function which is almost a standard in theoretical analysis of this type of problems, namely flow-time. With the new cost function, algorithms can be distinguished effectively, whereas using flow-time, this is not possible while respecting the LogP model assumptions. The behavior of the optimal off-line algorithm can be a little peculiar, however. If we consider sequences where n messages arrive less than one unit apart, nothing in our cost function encourages the optimal off-line algorithm to send any messages until the nth message has arrived.

While the behavior of an off-line optimal algorithm is secondary to the ability of the total set-up to distinguish between good and bad on-line algorithms, our results are robust enough that the behavior of *OPT* could be altered. Assume that we change the cost function such that when a message has been waiting for one time unit (or equivalently, has not been delivered two units after it became available), a strictly higher penalty is imposed. This will encourage a different behavior, where messages are sent earlier. However, *OPT* can still send all messages with the same cost. It will send at time t_n immediately after the nth message has arrived (as before), but it could also send at all times in the set $\{t_n - i \mid i \in \mathbb{N},\ t_n - i \geqslant 0\}$.

It is of course also possible to consider entirely different cost functions. A cost function should be reasonable in the sense that it should be a value which it would be good to minimize. A cost function should also be useful in the sense that it should make it possible to distinguish between algorithms. However, these requirements do not lead us to a canonical choice of cost function.

The choice as to when a packet is considered delivered is somewhat arbitrary. We have chosen to consider a packet sent at time p_i delivered at time $p_i + 1$ (that is time p_i plus the

gap time, since we have normalized with respect to the gap), the informal reason being that this is the time when we are allowed to send again. Other choices give similar results. For instance, if we consider a packet delivered at the time it is sent, a flow-time cost measure will also be unable to distinguish between our algorithms. In this case, all the algorithms become non-competitive, i.e., they do not have constant competitive ratios. Also adding acknowledgment costs, i.e., a constant cost for each packet sent, gives rise to similar results. For our presentation, we have chosen what we believe is the simplest cost function which give useful results, under the constraint that packet times must be at least one unit apart.

Continuing the discussion of our cost function following Definition 1, the only thing that distinguishes our cost function from $f_{\max}$ with $\eta = 1/2$ in [6] is the included maximization. This is because the constant one in $(p_i + 1)$, which for us reflects when a packet is considered delivered, will take the role of the acknowledgment cost used in [6]. The more important difference, however, is the difference in problem formulation regarding the decision as to whether or not packets may be sent any $\epsilon > 0$ apart. This decision seems to be so fundamental that despite similarities in cost functions, our results and the results in [6] are incomparable.

Finally, our algorithms can in principle be built into any operating system, though the ease with which this can be done depends on the exact design of the operating system in question, in particular on the availability of an extra timer to support interrupts from our algorithm.

Acknowledgments

We thank Brian Vinter for drawing our attention to the Packet Bundling Problem and for initial discussions regarding the cost function. We also thank the anonymous referees for constructive comments and suggestions that improved the presentation of the paper.

References

[1] A. Alexandrov, M.F. Ionescu, K.E. Schauser, C. Scheiman, LogGP: Incorporating long messages into the LogP model for parallel computation, J. Parallel Distrib. Comput. 44 (1) (1997) 71–79.
[2] B. Awerbuch, Y. Azar, S. Leonardi, O. Regev, Minimizing the flow time without migration, in: Proceedings of the 31st Annual ACM Symposium on the Theory of Computing, 1999, pp. 198–205.
[3] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge Univ. Press, 1998.
[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, LogP: Towards a realistic model of parallel computation, in: Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1993, pp. 1–12.
[5] D.R. Dooly, S.A. Goldman, S.D. Scott, TCP dynamic acknowledgment delay: Theory and practice, in: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM Press, 1998, pp. 389–398.
[6] D.R. Dooly, S.A. Goldman, S.D. Scott, On-line analysis of the TCP acknowledgment delay problem, J. ACM 48 (2) (2001) 243–273.
[7] J.S. Frederiksen, K.S. Larsen, Packet bundling, Technical Report 9, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, 2002.
[8] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell Systems Tech. J. 45 (1966) 1563–1581.

[9] B. Kalyanasundaram, K.R. Pruhs, Minimizing flow time nonclairvoyantly, in: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, 1997, pp. 345–352.
[10] A.R. Karlin, C. Kenyon, D. Randall, Dynamic TCP acknowledgement and other stories about $e/(e-1)$, in: Proceedings of the 33th Annual ACM Symposium on Theory of Computing, ACM Press, 2001, pp. 502–509.
[11] A.R. Karlin, M.S. Manasse, L. Rudolph, D.D. Sleator, Competitive snoopy caching, Algorithmica 3 (1988) 79–119.
[12] H. Kellerer, T. Tautenhahn, G.J. Woeginger, Approximability and nonapproximability results for minimizing total flow time on a single machine, SIAM J. Comput. 28 (4) (1999) 1155–1166.
[13] T. Kielmann, H.E. Bal, K. Verstoep, Fast measurement of LogP parameters for message passing platforms, in: Proceedings of the Workshop on Run-Time Systems for Parallel Programming, 2000, pp. 1176–1183, Satelite workshop of the International Parallel and Distributed Processing Symposium.
[14] S. Leonardi, D. Raz, Approximating total flow time on parallel machines, in: Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 1997, pp. 110–119.
[15] J.Y.-T. Leung, G.H. Young, Minimizing schedule length subject to minimum flow time, SIAM J. Comput. 18 (2) (1989) 314–326.
[16] L.A.H. Liang, W. Cai, B.-S. Lee, S.J. Turner, Performance analysis of packet bundling techniques in DIS, in: Proceedings of the 3rd International Workshop on Distributed Interactive Simulation and Real-Time Applications, IEEE Comput. Society, 1998.
[17] J. Noga, S.S. Seiden, G.J. Woeginger, A faster off-line algorithm for the TCP acknowledgement problem, Inform. Process. Lett. 81 (2) (2002) 71–73.
[18] S.S. Seiden, A guessing game and randomized online algorithms, in: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, ACM Press, 2000, pp. 592–601.
[19] D.D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, Commun. ACM 28 (2) (1985) 202–208.