2007－AL－110 (1)
2007／1／23

スライディングウィンドウを考慮した Dynamic TCP Acknowledgment 問題

古賀 久志
電気通信大学 大学院 情報システム学研究科

あらまし
TCP プロトコルにおける通信では受信者はパケットを受信すると到達確認 (acknowledgment, 以下 ack) を送信者に返して，送信が成功したことを知らせる．受信者は各到着パケットに 1 度ずつ ack を返すのではなく，複数の到着パケットに対して 1 度の ack でまとめて到達確認を済ますことが出来る．この機構を使うと ack 回数を減らせるが，逆に送信ホストによる到達確認が遅れ通信が遅滞するという欠点もある．Dynamic TCP acknowledgment 問題はこのトレードオフに対するオンライン最適化問題である． しかしながら、従来の Dynamic TCP Acknowledgement 問題の枠組では，受信者が ack をなかなか返さない場合に送信者が自主的に送信を抑えるスライディングウィンドウの機構を考慮していない．

そこで本研究では，スライディングウィンドウの機構を組み込んだ Dynamic TCP Acknowledgement 問題を定式化して，オンラインアルゴリズムの性能を競合比解析を用いて評価する．特にウィンドウサイズが固定値 W であると仮定して解析を行い，受信者が W を知らされていれば 2-competitive なオンラインアルゴリズムを構築できるのに対し，W を知らされていない場合には，従来の枠組での最適オンラインアルゴリズムを含むアルゴリズムクラスの競合比の下限値が送信者が送ろうとする単位時間辺りの最大パケット数に依存してしまうことを示す．

Dynamic TCP Acknowledgment with Sliding Window

Hisashi Koga
Graduate School of Information Systems
University of Electro-Communications

Abstract In TCP protocol, each packet arriving at the receiver must be acknowledged by the receiver in order to notify the sender that the transmission was successful. However, each packet need not be acknowledged individually. Instead, the receiver is allowed to acknowledge multiple packets with a single acknowledgement by postponing the acknowledgement. Though this mechanism is advantageous with respect to reduce the number of acknowledgements, delaying acknowledgements may add excessive latency to the TCP connection. Dooly et al. formulated this trade-off as *the dynamic TCP acknowledgement problem.* However, their framework does not consider the concept of *sliding window* that restricts the maximum number of packets that the sender can inject into the network without notified acknowledgements.

In this paper, we propose a new problem in which the sliding window is integrated into the dynamic TCP acknowledgement problem realistically. We evaluate the performance of on-line algorithms with competitive analysis, assuming that the window size is a constant integer $W \geq 1$. We show that there exists a 2-competitive deterministic on-line algorithm if on-line algorithms know the value of W beforehand. By contrast, if they do not know W, the lower bound of the competitive ratio for an algorithm class containing the optimal on-line algorithm for the original framework by Dooly et al. that is 2-competitive now depends on the maximum number of packets that the sender wishes to send per unit time.

1 Introduction

TCP (Transport Control Protocol) is the most well-known transport protocol in the Internet and adopted in many application protocols such as telnet, ftp and http. Thus, there is a strong need to grasp the behavior of TCP protocol both from theoretical and experimental sides. Among previous works that analyzed TCP theoretically, Dooly et al. [3] focused on the mechanism of TCP acknowledgement. Suppose that a sender S send packets to a receiver R using TCP. In TCP protocol, each packet arriving at R must be acknowledged by R in order to notify S that the transmission was successful. However, each packet need not be acknowledged individually. Instead, most TCP implementations employ a mechanism called "delayed ACK" which admits the receiver to acknowledge multiple packets with a single acknowledgement by postponing the acknowledgement. The delayed ACK mechanism contributes to reducing the overhead of the acknowledgements by decreasing the number of acknowledgements. On the other hand, it has the risk to add excessive latency to the TCP connection. Dooly et al.[3] formulated this trade-off as *the dynamic TCP acknowledgement problem.*

The Dynamic TCP acknowledgement problem is defined as follows: There is a sequence of n packets $\sigma = (p_1, p_2, \ldots, p_n)$ each of which reaches R in order. The arrival time of p_i at R is denoted by a_i. An acknowledgment algorithm operating in R divides σ into m subsequnces $\sigma_1, \sigma_2, \ldots, \sigma_m$, where each subsequence end corresponds to a single acknowledgment. All the packets contained in σ_j $(1 \leq j \leq m)$ are acknowledged together by the j-th acknowledgement. Let t_j be the time when the j-th acknowledgment is performed. To assure that all the packets should be acknowledged, it must hold that $m \geq 1$ and that $t_m \geq a_n$. In case a packet p is not acknowledged immediately, an extra latency arises. The purpose of the dynamic TCP acknowledgement problem is to minimize the sum of the cost for generating acknowledgements and the cost for the latency of acknowledgements by choosing the acknowledgement time sequence $(t_1, t_2, \cdots, t_m)$ adequately. Ordinarily, an acknowledgment time is decided in an on-line fashion without knowing the future packet arrivals. Dooly et al. proposes two kinds of objective functions that should be minimized. The first objective function f_{sum} combines the number of acknowledgements with the sum of delays for all the packets. f_{sum} is described as $m + \sum_{j=1}^{m} \sum_{p_i \in \sigma_j} |t_j - a_i|$. The second objective function f_{max} combines the number of acknowledgements with the sum of the maximum delays for a packet in each subsequence σ_j and defined as Eq. (1).

$$m + \sum_{j=1}^{m} \max_{p_i \in \sigma_j} |t_j - a_i|. \tag{1}$$

Dooly et al. evaluated on-line acknowledgement algorithms with competitive analysis [2] which compares the performance of an on-line algorithm with that of the optimal off-line algorithm OPT that knows the entire packet sequence σ in advance and can therefore achieve the minimum cost. Let $C_A(\sigma)$ be the value of the objective function after an acknowledgement algorithm A processes σ. An on-line acknowledgement algorithm A is called c-competitive if $C_A(\sigma) \leq c \cdot C_{opt}(\sigma) + b$ for any σ, where b is a constant independent of σ. As for f_{sum}, they presented a deterministic 2-competitive algorithm and proved that no deterministic on-line algorithm is better than 2-competitive. With respect to f_{max}, they also constructed a deterministic on-line algorithm which achieves the best competitive ratio of 2. This best on-line algorithm utilizes a timer. When p_i reaches R at time a_i, the timer is set to $a_i + 1$. If $a_{i+1} > a_i + 1$, the timer expires and the acknowledgement is performed at $a_i + 1$. Otherwise, the timer is updated to $a_{i+1} + 1$ at a_{i+1}, the arrival time of the next packet p_{i+1}. Note that this algorithm acknowledges just when the latency cost equals the cost for a single acknowledgement. Let us call this algorithm `WAIT(1)`, since it always waits for 1 unit time since the last packet arrival before performing an acknowledgement. Since this paper deals with f_{max} only, we abbreviate the dynamic TCP acknowledgement problem with f_{max} simply as DTCP.

We claim that DTCP abstracts the mechanism of TCP acknowledgement only partially, because it misses the concept of *sliding window* that plays a crucial role for congestion control in TCP. The sliding window functions in a TCP sender S and restricts the maximum number of packets that S can inject into the network without notified acknowledgements. See Figure 1 for example. The sliding window is depicted as a rectangle and divides the packet sequence into three subsequences such that its left end shows the last packet which has been already acknowledged, whereas its right end shows up to which unacknowledged packet S can send. Every time S is notified an acknowledgement, the sliding window slides rightward. In this way, the width of the sliding window defines the maximum number of packets that S can inject into the network without a receipt of an acknowledgement. The width of the sliding window is termed *window size*. The sliding window forces S to stop sending packets when an acknowledgement has not been returned from R for a long time, which may be a sign that R is overwhelmed against a large amount of incoming packets.

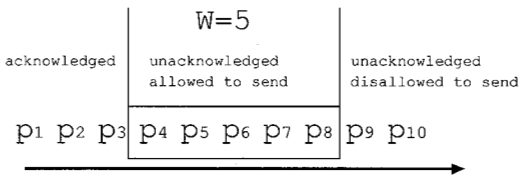

Figure 1: Sliding Window

Our primary contribution is to propose a new problem in which the sliding window is integrated into

DTCP realistically. Motivated by the fact that, in the standard TCP, R operates without knowing if S stops sending packets due to the sliding window, we examine how the power of on-line algorithms depends on whether R recognizes that S is awaiting. To simplify the analysis, we assume that the window size is a constant integer $W(\geq 1)$, though TCP protocol allows S to change it dynamically in practice. Namely, a packet p_i can never reach R before p_{i-W} is acknowledged by R $(i > W)$. Owing to this simplification, R has only to know the value of W so as to judge whether S is waiting or not. Our new problem is named as DTCPSW (DTCP with Sliding Window).

In Section 2, we define the DTCPSW more formally. Section 3 presents the optimal off-line algorithm for DTCPSW whose computational complexity is $O(Wn)$. Because the optimal off-line algorithm is computable in $O(n)$ in DTCP, this result states that DTCPSW is more difficult than DTCP. Section 4 discusses the case in which R knows the value of W. In this case, we show that it is possible to construct a 2-competitive deterministic on-line algorithm by extending `WAIT(1)` naturally. Furthermore, this algorithm becomes the optimal. Thus, we can obtain a result comparable to DTCP for DTCPSW, if R knows W. Section 5 investigates the case in which R does not know W. We show that the on-line strategy that observes time intervals between two consecutive packets like `WAIT(1)` does not work efficiently. Concretely, we pick up an algorithm class `WAIT(`α`)` containing `WAIT(1)`, where α is arbitrary positive real. `wait(`α`)` is not competitive in the sense that its competitive ratio depends on the the maximum number of packets that the S wishes to send per unit time. In particular, it is proved that no on-line algorithm from the algorithm class `WAIT(`α`)` is better than $\frac{T}{W+\lfloor \frac{T}{W} \rfloor - 1}$-competitive when $T > W$, even if the number of packets S hopes to send per unit time never goes beyond T. Thus, Section 5 derives the result opposite to Section 4, which reflects the importance of the agreement between the sender and the receiver in communications. Section 6 is the conclusion.

1.1 Related Works

Recently, Albers and Bals [1] studied the dynamic TCP acknowledgement problem with a different object function $m + \max_{1 \leq i \leq m} \max_{p_i \in \sigma_j} |t_j - a_i|$. Karlin et al. [5] studied randomized on-line algorithms against oblivious on-line algorithms with f_{sum} and developed a randomized on-line algorithm that becomes $\frac{e}{e-1}$-competitive. In addition, they showed that all of the solutions to the ski-rental problem, the dynamic TCP acknowledgment problem and the bahncard problem [4] fall within a common framework.

2 Problem Statement

DTCPSW is formally defined as follows .We are given a sequence of packets $\sigma = (p_1, p_2, \ldots, p_n)$ that S shall send to R. DTCPSW differs from the original DTCP in that the next two sorts of times are associated with each packet p_i.

- Ready time r_i: the time when S prepares the transmission of p_i.
- Arrival time a_i^A: the time when the packet arrives at R and get eligible to the acknowledgment algorithm A in R. The superscript A indicates that the arrival time is influenced by the action of A as explained below.

S can send p_i at r_i unless impeded by the sliding window. On the other hand, the sliding window permits S to send p_i only after A acknowledges p_{i-W} if $i > W$. Let $\mathrm{ack}^A(p)$ be the time when A acknowledges a packet p. In DTCPSW, by assuming that the propagation delay between S and R equals 0, a_i is described by Eq. (2).

$$a_i^A = \max\{r_i, \mathrm{ack}^A(p_{i-W})\}. \tag{2}$$

Here, when $i - W \leq 0$, $\mathrm{ack}^A(p_{i-W})$ is defined to be $-\infty$. If A acknowledges p_{i-W} with the l-th acknowledgement at time t_l, Eq. (2) may be written as $a_i^A = \max\{r_i, t_l\}$. At t_l, immediately after A's l-th acknowledgement, a group of packets postponed by the sliding window are passed to A. We allow A to acknowledge them instantly by the $(l+1)$-th acknowledgement. In this case, it holds that $t_l = t_{l+1}$. Thus, A can make multiple acknowledgements at a given time.

The important point is that, whereas the ready time sequence $(r_1, r_2, \cdots, r_n)$ is the inherent input that has nothing to do with A, A affects the arrival time sequence $(a_1^A, a_2^A, \cdots, a_n^A)$. In the subsequence, the name of the acknowledgment algorithm is omitted from variables, when clear from the context. In addition, we equate the ready time sequence $(r_1, r_2, \cdots, r_n)$ with the input packet sequence $(p_1, p_2, \cdots, p_n)$

and denote it by σ. A meets the arrival time sequence only and serves it without viewing the ready time sequence. In DTCPSW, the latency of a packet p is naturally defined as the length of the time period between its ready time and the time when A acknowledges it. Again, A divides σ into m subsequences $\sigma_1, \sigma_2, \cdots \sigma_m$ whose ends correspond to acknowledgments. The purpose of DTCPSW is to minimize the objective function f_{max} shown in Eq. (3) that is the modification of Eq. (1).

$$f_{max} = m + \sum_{j=1}^{m} \max_{p_i \in \sigma_j} |t_j - r_i|. \tag{3}$$

In general, A cannot know the ready time sequence even after it finishes the processing of the arrival time sequence, though we dare to treat the case when A can estimate the ready time sequence correctly by knowing W in Section 4.

When $W = 1$, a trivial on-line algorithm that acknowledges every packet arrival instantly becomes 1-competitive for DTCPSW. Hence, we assume $W \geq 2$ in the remaining part of this paper.

DTCPSW opens up a new vista on the research of competitive analysis from the next reason: As far as we know, there is no previous research attempting to cope with the situation such that an on-line algorithm unconsciously changes the original input sequence which corresponds to the ready time sequence in our problem. We remark that it is not rare for an on-line algorithm not to figure out the original input sequence after changing it involuntarily. For example, consider a scenario in which there exist a couple of mice in some house and the inhabitant sees one of them by accident. If he chooses to get rid of the mouse on the spot, the number of mice cannot increase any more. On the contrary, if he lets the mouse escape, he will meet a lot of mice (i.e., the changed input sequence) in future and must have trouble exterminating all of them, but he will never become aware that there are only two mice at the beginning.

This section finishes by stating that it holds that $a_i = r_i$ for any i in the DTCP instead of Eq. (2).

3 Optimal Off-line Algorithm

This section presents the optimal off-line algorithm OPT for DTCPSW. OPT is given both the entire ready time sequence σ and W in advance before starting to process σ. We start with the next lemma that states the necessary condition of OPT.

Lemma 1 *Let B be an off-line algorithm. If S has ever defer sending a packet to R because of the sliding window in B's running, B is not the optimal.*

Proof: Let p_i be the first packet which S defers sending to R. Namely, $r_k = a_k$ for $1 \leq k \leq i-1$ and $r_i < a_i$. Since the sliding window does not impede the transmission of the first W packets, $i > W$. p_i becomes eligible to R at a_i, caused by some acknowledgement in the running of B. Consider the subsequence of σ that ends with this acknowledgement. Let this subsequence be σ_j. See Figure 2. Since $r_{i-1} < r_i$, we have $a_{i-1} < r_i$. Therefore, it is possible to reduce the latency for σ_j by acknowledging it at a_{i-1}. Since this modification does not influence the latency for other subsequences, B is not the optimal off-line algorithm. □.

From Lemma 1, OPT makes an acknowledgement, whenever the number of unacknowledged eligible packets increases to W. Therefore, OPT acknowledges at least once for every W packet arrivals. More importantly, the proof of Lemma 1 claims that $\forall i, a_i = r_i$ for any i in OPT's running.

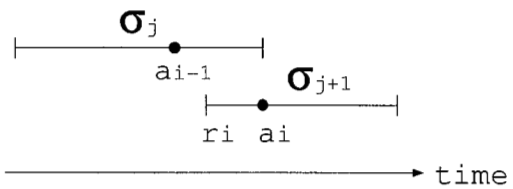

Figure 2: Off-line Algorithm B

for $1 \leq k \leq q$

$$\text{OPTCOST}[k] = \begin{cases} r_k^j - r_1^j + 1. & \text{if } k \leq W \\ \min_{1 \leq l \leq W} \text{OPTCOST}[k-l] + r_k^j - r_{k-l}^j + 1. & \text{if } k > W \end{cases}$$

$$\text{PREVACK}[k] = \begin{cases} 0 & \text{if } k \leq W \\ \text{argmin}_l \text{ OPTCOST}[k-l] + r_k^j - r_{k-l}^j + 1. & \text{if } k > W \end{cases}$$

Figure 3: Update of OPTCOST and PREVACK

When an acknowledgement algorithm A guarantees $a_i = r_i$ for any i, we can proceed the analysis of A by dividing the total incurred cost by A among each packet in the next manner. Dooly et al. [3] exploited the similar technique for DTCP.

- When A acknowledges at a_i, p_i is charged a cost of 1 which corresponds to the cost for a single acknowledgement.
- Suppose p_i is not the final packet of σ. When A does not acknowledge at a_i, p_i is charged a cost of $a_{i+1} - a_i$ which corresponds to the extra latency cost incurred when A does not acknowledge at a_i.

Note that no reasonable acknowledgement algorithm acknowledges at a halfway time between a_i and a_{i+1}. It is easy to verify that the sum of the costs assigned to all the packets equals the total cost that A pays.

Lemma 2 *If $r_{i+1} - r_i > 1$. OPT acknowledges at r_i.*

Proof: Recall that $\forall i, a_i = r_i$. If OPT does not acknowledge at r_i, p_i is charged a latency cost that equals $r_{i+1} - r_i$. Else if OPT acknowledges at a_i, the latency cost decreases by $r_{i+1} - r_i$, while the acknowledgement cost increases by 1. Thus, if $r_{i+1} - r_i > 1$, OPT must acknowledge at r_i to achieve the optimality. □

Finally, we can describe OPT.

Algorithm OPT: By obeying Lemma 2, OPT first checks if $r_{i+1} - r_i > 1$ for all values of i, while scanning $\sigma = (r_1, r_2, r_3, \cdots r_n)$ from its head. When $r_{i+1} - r_i > 1$, OPT decides to put an acknowledgement at r_i. As the result, σ is cut into subsequences $\sigma_1', \sigma_2', \cdots, \sigma_{m'}'$. Within each of these subsequences, the gap between the ready times of any two adjacent packets is at most 1.

Consider one of these subsequence, say $\sigma_j' = (r_1^j, r_2^j, \cdots, r_q^j)$ whose length is q. Here r_k^j is the k-th ready time in σ_j' $(1 \leq k \leq q)$. OPT computes where to put acknowledgements in serving σ_j' with dynamic programming. OPT prepares two integer arrays OPTCOST and PREVACK. The element OPTCOST[k] $(1 \leq k \leq q)$ is used to store the optimal minimum cost to serve the prefix of σ_j', i.e., $(r_1^j, r_2^j, \cdots, r_k^j)$. PREVACK[$k$] remembers the location of the second to last acknowledgment in the optimal solution for $(r_1^j, r_2^j, \cdots, r_k^j)$. OPTCOST and PREVACK are updated according to the procedure in Figure 3.

When $k \leq W$, the optimal solution for processing the prefix of σ_j' has only to acknowledge once at the end. When $k > W$, it must decide where to put the second-to-last acknowledgement, since OPT must acknowledge at least once for every W packets. This information is saved into PREVACK[k]. In the end, OPTCOST[q] holds the cost for processing σ_j' optimally. The times for acknowledgements are acquired by tracing back the array PREVACK.

Since it takes an $O(W)$ time to perform the min operation and the min operation is invoked for each packet, the computational complexity of OPT becomes $O(Wn)$ in total. Because the optimal off-line algorithm is computable in $O(n)$ in DTCP, the above fact shows that DTCPSW is more difficult than DTCP.

4 Known Window Size

This section evaluates the performance of on-line algorithms when they are told a value of W beforehand. We show that a natural extension of `WAIT(1)` named as `WAITSW(1)` becomes 2-competitive in this case. Remarkably `WAITSW(1)` accomplishes the same competitive ratio in DTCPSW as the competitive ratio `WAIT(1)` in DTCP.

Algorithm WAITSW(1): Like `WAIT(1)`, `WAITSW(1)` observes a packet interval. At a_i $(i \geq 1)$, the timer is set to a_i+1. In case $a_{i+1} > a_i+1$, the timer expires before the next packet arrival and `WAITSW(1)` acknowledges at $a_i + 1$. Otherwise the timer is updated at a_{i+1} to $a_{i+1} + 1$. In addition, instantly the number of unacknowledged eligible packets is increases to W, they are immediately acknowledged.

Since the number of unacknowledged eligible packets does not exceed W, the sliding window never obstructs S from sending packets in `WAITSW(1)` . Hence, $\forall i, a_i = r_i$ in `WAITSW(1)` running in the same way as *OPT* in Section 3.

Theorem 1 `WAITSW(1)` *is 2-competitive.*

Proof: In both of OPT and `WAITSW(1)` , we have $a_i = r_i$ for any i. Furthermore, both algorithms perform an acknowledgement before a_{i+1} when $a_{i+1} > a_i + 1$. These two facts allow us to analyze `WAITSW(1)` by decomposing the entire input sequence σ into subsequences $\sigma_1', \sigma_2', \cdots, \sigma_{m'}'$ and treating them separately. In a single subsequence, the gap between the ready times of any two adjacent packets is less than 1.

Hence, in the rest of this proof, it is sufficient to consider ready time sequences $\sigma = (r_1, r_2, \cdots, r_n)$ for which $r_{i+1} < r_i+1$ for any $i < n-1$. Since $(r_1, r_2, \cdots, r_n) = (a_1, a_2, \cdots, a_n)$, we have $a_{i+1} < a_i+1$ for $i \leq n-1$. Therefore, `WAITSW(1)` acknowledges when W unacknowledged packets are accumulated at R except the last acknowledgement. Suppose that `WAITSW(1)` divides σ into subsequences $\sigma_1, \sigma_2, \cdots, \sigma_m$ that are delimited by an acknowledgement. $C_A(\sigma_j)$ is defined as the sum of the costs assigned to the packets in σ_j, where the cost assigned to a packet is derived by dividing the total incurred cost among packets based on the cost-division scheme mentioned in Section 3. Given a subsequence $\sigma_j = (a_i, a_{i+1}, a_{i+2}, \ldots a_{i+l})$, we need to consider two cases depending on whether $j = m$ or not.
Case 1: Suppose that $j = m$. In this case, `WAITSW(1)` acknowledges after a 1 time passes since the time of the last packet arrival a_n. Namely, `WAITSW(1)` acknowledges at a_n+1. Thus, we have $C_{\texttt{WAITSW(1)}}(\sigma_m) = 1 + (a_n + 1 - a_i) = a_n - a_i + 2$. On the other hand, $C_{opt}(\sigma_m) \geq a_n - a_i + 1$. Thus,

$$\frac{C_{\texttt{WAITSW(1)}}(\sigma_m)}{C_{opt}(\sigma_m)} \leq \frac{a_n - a_i + 2}{a_n - a_i + 1} \leq \frac{2}{1} = 2. \tag{4}$$

Case 2: Suppose that $j < m$. In this case, $C_{\texttt{WAITSW(1)}}(\sigma_j) = a_{i+W} - a_i + 1$ because `WAITSW(1)` acknowledges at a_{i+W}. Since σ_j consists of W packets, $C_{opt}(\sigma_j) \geq 1$. It also holds that $C_{opt}(\sigma_j) \geq a_{i+W} - a_i$, as $a_{i+W} - a_i$ is the optimal cost for serving σ_j with ignoring the constraint imposed by the sliding window. Then, the next inequality holds:

$$C_{\texttt{WAITSW(1)}}(\sigma_j) = a_{i+W} - a_i + 1 = 2 * \frac{a_{i+W} - a_i + 1}{2} \leq 2\max\{a_{i+W} - a_i, 1\} \leq 2C_{opt}(\sigma_j). \tag{5}$$

Equations (4) and (5) together complete the proof. □

The proof of the lower bound for DTCP in [3] constructs a bad packet sequence for an arbitrary on-line algorithm such that at most two packets appear in a single subsequence in the running of the on-line algorithm and OPT both. Hence, this lower bound is also valid for DTCPSW when $W \geq 2$.

Theorem 2 *Let A be any deterministic on-line algorithm for DTCPSW. Then, there exists a ready time sequence σ such that $C_A(\sigma) \geq 2C_{opt}(\sigma) - \epsilon$, where ϵ can be made arbitrarily small with respect to $C_{opt}(\sigma)$.*

From Theorem 2, `WAITSW(1)` turns out to be the optimal.

5 Unknown Window Size

The previous section demonstrates that `WAIT(1)` can be easily extended for supporting DTCPSW without deteriorating the competitiveness, if the window size is given. This section deals with the case when the window size is unknown to on-line algorithms. We show that the competitive ratios for a class of on-line algorithms `WAIT(`α`)` containing `WAIT(1)` depend on the maximum number of packets S wishes to inject into the network per unit time and does not become a constant. Like `WAIT(1)`, on-line algorithms in `WAIT(`α`)` measure the length of the time interval between two consecutive packet arrivals. The formal description of `WAIT(`α`)` is given below.
Algorithm `WAIT(`α`)` **:** Let α is an arbitrary positive real value. At a_i $(i \geq 1)$, the timer is set to $a_i + \alpha$. In case $a_{i+1} > a_i + \alpha$, `WAIT(`α`)` acknowledges at $a_i + \alpha$. Otherwise the timer is updated at a_{i+1} to $a_{i+1} + \alpha$.

Definition 1 (peak rate) *For a TCP connection, if at most T ready times lie between time intervals $[t, t+1)$ for any t, the peak rate of the TCP connection is said to be T.*

We derive the lower bound on the competitiveness of on-line algorithms WAIT(α) when the peak rate of the TCP connection is T, which is stated in Theorem 3. In the proof, an adversary constructs some ready time sequence that annoys on-line algorithms.

Theorem 3 *For any α, WAIT(α) is not better than $\frac{T}{W+\lfloor\frac{T}{W}\rfloor-1}$-competitive if $T > W$.*

Proof: Let $k = \lfloor\frac{T}{W}\rfloor$. We assume that $T > W$ so that k may be strictly greater than 0. First consider a ready time sequence σ' such that kW ready times appear every unit time until time $I-1$ where I is a positive integer. Namely,

$$r_{(i-1)kW+1} = r_{(i-1)kW+2} = \cdots r_{ikW} = i-1$$

for $1 \leq i \leq I$. OPT processes σ by acknowledging k times at each time i for $1 \leq i \leq I$ without incurring no delay. Here, a single acknowledgement acknowledges W packets at once. Thus, $C_{opt}(\sigma') = kI$.

On the other hand, WAIT(α) must waits for α for each clump of W packets before acknowledging it. Thus, the sum of the latency cost for the first kW packets whose ready times are all 0 becomes

$$\sum_{i=1}^{k}(j\alpha - 0) = \sum_{j=1}^{k} j\alpha = \frac{k(k+1)}{2}\alpha.$$

Next, since $a_{kW+1} = \max\{r_{kW+1}, a_{kW-W+1}\} = \max\{1, k\alpha\}$ for p_{kW+1}, p_{kW+1} is acknowledged at time $\max\{1, k\alpha\} + \alpha$. In this way, the sum of the latency costs for packets from p_{kW+1} to p_{2kW} becomes $\sum_{j=1}^{k}(\max\{0, k\alpha - 1\} + j\alpha) = k\max\{0, k\alpha - 1\} + \frac{k(k+1)}{2}\alpha$. In general, the sum of the latency costs for the kW packets from $p_{(i-1)kW+1}$ to p_{ikW} becomes

$$\sum_{j=1}^{k}(\max\{0, (i-1)(k\alpha-1)\} + j\alpha) = k\max\{0, (i-1)(k\alpha-1)\} + \frac{k(k+1)}{2}\alpha. \tag{6}$$

By summing up the right-hand term of (6) for $1 \leq i \leq I$ and adding the acknowledgement cost of kI, we have

$$C_{\texttt{WAIT}(\alpha)}(\sigma') = kI + k\max\{0, \frac{I(I-1)}{2}(k\alpha-1)\} + \frac{Ik(k+1)}{2}\alpha. \tag{7}$$

The competitive ratio is obtained by dividing $C_{\texttt{WAIT}(\alpha)}(\sigma')$ by $C_{opt}(\sigma') = kI$ as shown in Eq. (8).

$$1 + \max\{0, \frac{(I-1)(k\alpha-1)}{2}\} + \frac{(k+1)}{2}\alpha. \tag{8}$$

Since I may be chosen to be an arbitrary large integer, in order for WAIT(α) to be competitive, $\alpha \leq \frac{1}{k}$.

Next, consider another ready time sequence σ" whose length equals W such that $r_1 = a_1 = 0$ and r_{i+1} becomes immediately after WAIT(α) acknowledges the i-th packet at $a_i + \alpha$. From this condition, it holds that $r_j = (j-1)\alpha$. Since $W < T$, this ready time sequence does not break the condition that the peak rate of the TCP connection is lower than T. *opt* serves σ" by acknowledging only once after r_w. Thus, since the latency of *opt* equals $(W-1)\alpha$, $C_{opt}(\sigma") = 1 + (W-1)\alpha$. WAIT($\alpha$) must pay a latency cost of α and one acknowledgement cost for each packet. $C_{\texttt{WAIT}(\sigma)}(\sigma") = W(1+\alpha)$. The cost ratio between WAIT(α) and *opt* becomes

$$\frac{W(1+\alpha)}{1+(W-1)\alpha}. \tag{9}$$

From (8) and (9), the lower bound for WAIT(α) becomes $\max\{1 + \frac{(k+1)}{2}\alpha, \frac{W(1+\alpha)}{1+(W-1)\alpha}\} \geq \frac{W(1+\alpha)}{1+(W-1)\alpha}$ for $\alpha \leq \frac{1}{k}$.

For $W \geq 2$, $\frac{W(1+\alpha)}{1+(W-1)\alpha}$ monotonically decreases with respect to α and takes the minimum value when $\alpha = \frac{1}{k}$. The minimum value becomes

$$\frac{W(k+1)}{W+k-1} = \frac{W(\lfloor\frac{T}{W}\rfloor+1)}{W+\lfloor\frac{T}{W}\rfloor-1} \geq \frac{T}{W+\lfloor\frac{T}{W}\rfloor-1},$$

which finishes the proof. □

Intuitively, the proof is interpreted as follows: An on-line algorithm has to wait for a shorter time before acknowledging when W is unknown than when W is known so as to cope with the possible delay of a packet arrival caused by the sliding window. However, this action deteriorates the performance, when S is not hindered by the sliding window. The peak rate has strong relationship with the maximum delay of the packet arrival.

Regarding to the case when $T < W$, Eq.(9) presents the lower bound of `WAIT(`α`)` for $\alpha > \frac{1}{T}$. As $\alpha \leq 1$ and (9) decreases monotonically for $W \geq 2$, the lower bound of the competitive ratio becomes 2 by substituting 1 for α in (9).

6 Conclusion

This paper analyzes the dynamic TCP acknowledgement problem with f_{max} on the realistic condition that the sender cannot keep on sending packets beyond the boundary of the sliding window. This new problem is named DTCPSW. Motivated by the fact that the receiver does not notice that the sender stops sending while waiting for the next acknowledgment in the standard TCP, we evaluate how the performance of on-line acknowledgement algorithms is influenced according to whether they can recognize whether the sender is waiting or not. In particular, under the assumption that the window size is a constant W, we investigate the difference between the two cases: (Case I) on-line algorithms know W beforehand, and (Case II) on-line algorithms does not know W. In the first case, we succeeded in developing the optimal 2-competitive deterministic on-line algorithm `WAITSW(1)` that is extended from the optimal on-line algorithm `WAIT(1)` for the previous framework of the dynamic TCP acknowledgement problem. On the contrary, in the second case, we showed that a class of on-line algorithms `WAIT(`α`)` that contain `WAIT(1)` is not better than $\frac{T}{W+\lfloor \frac{T}{W} \rfloor - 1}$-competitive if $T > W$, where T is the maximum number of packets that the sender hopes to inject into network per unit time. Thus, the competitive ratio is determined by the traffic rate of the TCP connection in the second case. We suppose that these results clarify the importance of the agreement between the sender and the receiver in communications.

Our problem is also interesting in terms of competitive analysis, because, it deals with the situation in which an on-line algorithm involuntarily changes the original input and processes the modified input without understanding that it is modified from the original input for the first time as far as we know.

Our future work is to derive the general lower bound of the competitiveness and to develop efficient on-line algorithms in the case when W is unknown. It is also important to study the case that the window size may alter dynamically.

Acknowledgements

This work is supported by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B), 17700054, 2006.

References

[1] S. Albers and H. Bals. Dynamic TCP acknowledgment: Penalizing long delays. In *Proceedings of 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 47–55, 2003.

[2] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[3] D. R. Dooly, S. A. Goldman, and S. D. Scott. On-line analysis of the TCP acknowledgment delay problem. *Journal of the ACM*, 48(2):243–273, 2001.

[4] R. Fleischer. On the bahncard problem. *Theoretical Computer Science*, 268:161–174, 2001.

[5] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic TCP acknowledgment and other stories about $\frac{e}{(e-1)}$. *Algorithmica*, 36(3):209–224, 2003.