On Statistically Estimated Optimistic Delivery in Wide-Area Total Order Protocols

José Mocito
Ana Respício
Luís Rodrigues

DI–FCUL TR–06–22

September 2006

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

On Statistically Estimated Optimistic Delivery in Wide-Area Total Order Protocols*

José Mocito Ana Respício Luís Rodrigues
University of Lisbon
{jmocito,respicio,ler}@di.fc.ul.pt

Abstract

Total order broadcast protocols have been successfully applied as the basis for the construction of many fault-tolerant distributed systems. Unfortunately, the implementation of such a primitive can be expensive both in terms of communication steps and of number of messages exchanged. To alleviate this problem, optimistic total order protocols have been proposed. This paper addresses the problem of offering optimistic total order in geographically wide-area systems. We present a protocol that outperforms previous work, by minimizing the average latency of the optimistic notification.

1. Introduction

Total order broadcast protocols have been successfully applied as the basis for the construction of many fault-tolerant distributed systems, from clock synchronization [8] to database replication [5, 2]. The purpose of such a protocol is to provide a communication primitive that allows processes to agree on the set of messages they deliver and also on their delivery order.

Unfortunately, the implementation of such a primitive can be expensive both in terms of communication steps and of number of messages exchanged. This problem is exacerbated in wide-area networks, where the performance of the algorithm may be limited by the presence of high-latency links. To alleviate this problem, optimistic total order protocols have been proposed [9, 10]. Such protocols provide to the application an early indication of the estimated definitive total order. The application can use this estimate to perform a number of actions optimistically, which are later committed when the final definitive order is established. The goal is to execute some application steps in parallel with the communication steps of the total order algorithm.

*This work has been partially supported by the project IST-STREP 004758, GORDA: Open Replication of Databases.

To the best of our knowledge, the protocol that is able to offer the smaller latency in the optimistic delivery notification in wide-area networks has been proposed by Sousa *et al.* in [9]. The protocol is an optimistic variant of the well known sequencer based total order algorithm.

In this paper we show that the algorithm used in [9] fails to offer the optimal average latency of optimistic deliveries. Departing from this observation we discuss how the optimal delays can be computed and, subsequently, propose an efficient heuristic to approximate the optimal result in an cost-effective manner. The resulting protocol is evaluated and compared with the original protocol.

This paper is structured as follows. In Section 2 we provide an overview of the optimistic total order protocol proposed in [9]. Section 3 highlights why the strategy followed in that protocol is suboptimal and discusses the optimal solution. An heuristic that approximates the optimal solution is presented in Section 4. An evaluation of the resulting algorithm is given in Section 5. Finally, in Section 6 we provide the concluding remarks.

2. Statistically Estimated Total Order

In this section we provide a brief overview of the optimistic total order presented in [9]. This protocol, that we will refer to as SETO (Statistically Estimated Total Order), was developed specifically targeting wide area networks and was inspired by the ideas originally presented in [6], where the spontaneous ordering properties of local area networks are used to provide optimistic deliveries in sequencer-based total order protocols.

2.1. Overview

The notion of optimistic total order was first proposed in the context of local-area broadcast networks [6]. In many of such networks, the spontaneous order of message reception is the same in all processes. Moreover, in sequencer-based total order protocols the total order is usually determined

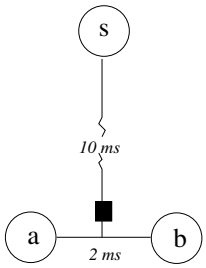


Figure 1. Local and wide-area links.

by the spontaneous order of message reception in the sequencer process. Based on these two observations a process may estimate the final total order of messages based on its local receiving order and, therefore, provide an optimistic delivery as soon as a message is received from the network.

Such approach is unfeasible in large-scale networks. The long latency in wide-area links causes different processes to receive the same message at different points in time. Consider the topology depicted in Figure 1. Assume that process a multicasts a message m_1 and that, at the same time, the sequencer s multicasts a message m_2. Clearly, the sequencer will receive $m_2 < m_1$, given that m_1 would require $12ms$ to reach the sequencer. On the other hand, process b will receive $m_1 < m_2$, as m_1 will take only $2ms$ to reach b while m_2 will require $12ms$. From this example, it should be obvious that the spontaneous total order provided by the network at b is not a good estimate of the observed order at the sequencer.

To address the problem above, [9] proposed to introduce artificial delays in the message reception to compensate for the differences in the network delays. It is easier to describe the intuition of the protocol by using a concrete example. Consider again the network of Figure 1. Assume also that we are able to provide to each process an estimate of the network topology and of the delays associated with each link. In this case, b could infer that message m_1 would take $10ms$ more to reach s than to reach b. By adding a delay of $10ms$ to all messages received from a, it would mimic the reception order of a's messages at s. A similar reasoning could be applied to messages from other processes.

2.2. The SETO algorithm

We now provide a precise description of the algorithm (Figure 2) used in the original SETO protocol to estimate the delays that should be added to each message. The algorithm is the same that has been published in [9] and is depicted here for self-containment. We recall that the rationale behind the solution is to make every process equidistant in respect to the sequencer. This way, the spontaneous order observed in the sequencer will be the same as the optimistic

```
1:  Initialization:
2:      g ← 0 {Global sequence number}
3:      l ← 0 {Local sequence number}
4:      R ← ∅ {Messages received}
5:      S ← ∅ {Sequence numbers}
6:      O ← ∅ {Messages opt-delivered}
7:      F ← ∅ {Messages fnl-delivered}
8:      delay[1..n] ← 0
9:      r_delay[1..n] ← 0 {Delays requested to the sequencer}

10: procedure TO_multicast(m)
11:     R_multicast(DATA(m, max(delay[] − delay[seq]))

12: upon R_deliver(DATA(m, d)) do
13:     R ← R ∪ {(m, d, now + delay[m.sender])}

14: upon ∃(m, d, t) ∈ R : now ≥ t ∧ m ∉ O ∧ m ∉ F do
15:     opt_delivery(m)
16:     O ← O ∪ m
17:     if p = seq then
18:         g ← g + 1
19:         R_multicast(SEQ(m, g))
20:         r_delay[m.sender] ← d
21:         delay[p] ← max(r_delay[])

22: upon R_deliver(SEQ(m, s)) do
23:     S ← S ∪ {(m, s, now)}

24: upon ∃(m, d, o) ∈ R : (m, l + 1, t) ∈ S ∧ m ∉ F do
25:     fnl_delivery(m)
26:     if ∃(m′, d′, o′) ∈ R : (m′, l, t′) ∈ S then
27:         Δ ← (t − t′) − (o − o′)
28:         if Δ > 0 then
29:             adjust(m′.sender, m.sender, Δ)
30:         else
31:             adjust(m.sender, m′.sender, |Δ|)
32:     l ← l + 1
33:     F ← F ∪ {m}

34: procedure adjust(i, j, d)
35:     v ← (delay[i] × α) + (delay[i] − d) × (1 − α)
36:     if v ≥ 0 then
37:         delay[i] ← v
38:     else
39:         delay[i] ← 0
40:         delay[j] ← delay[j] + |v|
```

Figure 2. Original SETO algorithm.

order artificially induced in each process.

In simple terms, the algorithm provides a multicast primitive (line 10) that ensures regular total order delivery in a group of processes. When a message arrives from the network (line 12) it is queued for optimistic delivery (line 13) after a certain period of time, which was previously estimated to approximate the spontaneous order as seen by the sequencer process. When that time comes (line 14) the message is optimistically delivered (line 15). When this happens in the sequencer process, the sequence number to that message is also generated (line 18) and is multicast to every process (line 19). Upon reception of the sequence number (line 22) the message is authoritatively delivered to the application. This description is obviously simplistic, as it omits the parts that deals specifically with the estimation of the delays to apply to optimistic message deliveries. Those parts will now be described.

First we will deal with the estimation of the delays on all processes other than the sequencer. When messages are

received from the network their optimistic delivery time is recorded (line 13). Also, when the sequence numbers are received from the sequencer, the time is also recorded (line 23). During the authoritative delivery of messages both recorded times are used to compare the optimistic and authoritative order of the current a last delivered messages (line 27). This comparison then determines the adjustments that must be made to the artificial delays (lines 28-31).

The second part is the estimation of delays in the sequencer process. This procedure cannot be the same as used in other processes because the basic delay estimation mechanism uses the sequencer itself as a reference to compensate distance divergences between processes. The way the protocol copes with this problem is to make the sequencer delay its own messages by the maximum amount of time a message it sent takes to reach any other process. Consider the network depicted in Figure 3(a). The messages sent by the sequencer s would have to be delayed by $9ms$ which corresponds to the maximum time a message sent from s takes to reach any other process (in this case p_2). Note that, in the algorithm, instead of having the sequencer actively determine the maximum delay to any process, each process suggests a delay based on the delays it is applying to messages from other processes (line 11). The sequencer then chooses the suggestion that has the highest value (line 21).

In the next section we address the limitations of SETO and discuss how it can be improved.

3. On the Latency of SETO

Let δ_i^j denote the latency of the optimistic delivery of messages from process p_i at process p_j. Figure 3(b) illustrates the values of δ_i^j for all processes, as the result of applying the original SETO algorithm to the topology of Figure 3(a). In this case, given that the sequencer (p_3) delays its own messages, we have $\delta_3^1 = 7$, $\delta_3^2 = 9$ and $\delta_3^3 = 9$. If all processes transmit at the same rate, we have an average message latency of $7.66ms$ as a result of this configuration. The question addressed in this paper is the following: is this set of optimistic delivery latencies optimal?

Figures 3(c) and 3(d) depict alternative sets of delivery latencies for the optimistic delivery that would also respect the total order. Again, considering that all processes send at the same rate, the average latency derived from configuration of Figure 3(c) is $9.66ms$ and that of Figure 3(d) is $7.00ms$. Thus, it is clear that SETO does not offer the optimal latency of optimistic delivery. Note that it is possible to find many other assignments that would also provide a basis for totally ordering the messages. In fact, any delay assignment that respects the following constraint can be used to establish a total order:

$$\delta_k^i - \delta_l^i = \delta_k^j - \delta_l^j \quad \forall i, j, k, l \in 1, ..., N \tag{1}$$

$\delta_k^i - \delta_l^i$ is the amount of time between the reception of messages from p_k and p_l at process p_i. This value may be negative – whenever p_i receives the message from p_l after the one from p_k. Equation (1) states that given a pair of processes p_k and p_l, $k, l = 1, ..., N$, the time interval between the reception of messages from these processes is always the same, independently of the receiver we consider.

Consider the permutation that defines the total order of messages $\prod =< \pi_1, ..., \pi_n >$, where $\pi_k \in \{1, ..N\}$ is the index of the process sending the message to be received in the k-th position. Another way to express equation (1), is the following:

$$\delta_{\pi_{1+k}}^i - \delta_{\pi_1}^i = \delta_{\pi_{1+k}}^j - \delta_{\pi_1}^j \forall i, j \in 1, ..., N, k \in 1, ..., N-1. \tag{2}$$

This equation expresses that the time interval between the reception of any message π_{k+1} and the reception of the first message in the total order π_1 is the same at all the processes. Obviously, in equation (2) the time differences under comparison are always positive, in opposition to those in equation (1). Clearly, all the assignments of Figure 3 respect the constraint above and are, therefore correct. Still, the same question remains: is the configuration from Figure 3(d) the optimal?

3.1. Optimal Assignment

In this paper we seek solutions that minimize the overall average latency (OAL) of all optimistic deliveries, denoted Δ_{avg}. Let r_i be the rate at which process p_i sends messages. Δ_{avg} is defined as:

$$\Delta_{\text{avg}} = \frac{\sum_{i,j=1}^{N} r_i \delta_i^j}{N \sum_{j=1}^{N} r_j} \tag{3}$$

Minimizing the overall average latency of all optimistic deliveries is equivalent to Minimize the Overall Latency (MOL) and this problem can be mathematically formulated in linear programming as:

MOL:

$$min \sum_{i,j=1}^{N} r_i \delta_i^j \tag{4}$$

s.t.:

$$\delta_i^1 - \delta_{i+1}^1 = \delta_i^j - \delta_{i+1}^j, \quad i = 1, ..., N-1, j = 2, ..., N \tag{5}$$

$$\delta_i^j \geq 0, \quad i, j = 1, \ldots, N. \tag{6}$$

Equation (4) states the objective function to be minimized: the weighted sum of all the latencies. Equation (5) ensures that the latencies under decision will give rise to a total order. It can be easily derived from equation (1) that,

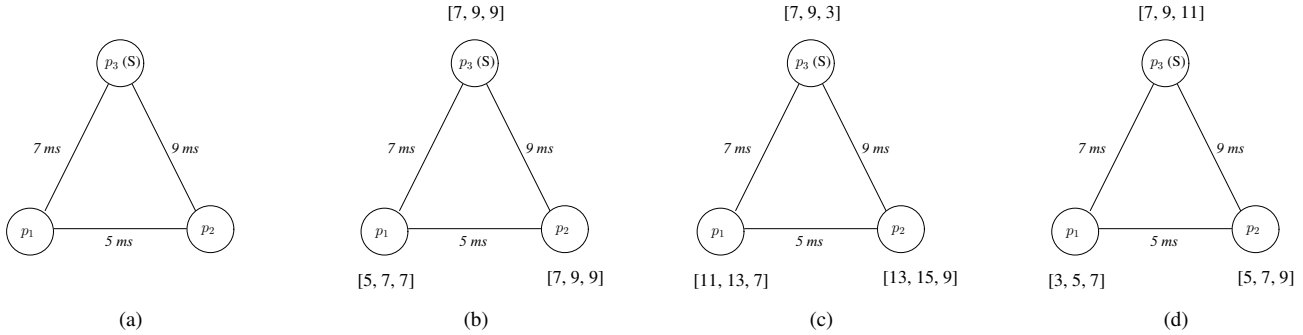


Figure 3. SETO latency and alternative configurations.

as we stated above, it assures total order. Finally, equation (6) ensures that all the latencies are non-negative.

Let ω_i^j denote the network delay of messages from process p_i to process p_j and let x_i^j denote the delay these messages should suffer to accomplish the corresponding latency δ_i^j, $i, j = 1, ..., N$. Consequently, by rewriting δ_i^j as $\delta_i^j = \omega_i^j + x_i^j$, and considering x_i^j as the new decision variables, the MOL can be reformulated as:

$$min \sum_{i,j=1}^{N} r_i x_i^j \tag{7}$$

s.t.:

$$(\omega_i^1 + x_i^1) - (\omega_{i+1}^1 + x_{i+1}^1) = (\omega_i^j + x_i^j) - (\omega_{i+1}^j + x_{i+1}^j), \quad i = 1, ..., N-1, j = 2, ..., N \tag{8}$$

$$x_i^j \geq 0, \quad i, j = 1, \ldots, N. \tag{9}$$

In this new formulation, equation (7) states the objective function to be minimized: the weighted sum of the delays to impose to all the messages. Equation (8) ensure total order, while equation (9) guarantees for the non negativity of the delays.

This problem has N^2 decision variables and $(N-1)^2$ constraints of type (8). Instances of the MOL problem can be solved using a solver of Linear Programming models, such as ILOG CPLEX [1]. In our case, we used ILOG CPLEX 9.0 to obtain the optimal solutions of all the instances under testing. When applying the solver to the topology of Figure 3(a) we obtained the delays depicted in Figure 3(d) which correspond to the optimal solution.

4. Fast SETO

As noted in the previous section, it is possible to derive the optimal solution using a solver of linear programming models. Unfortunately, it may be unpractical to install and execute such solver in each protocol stack of every process of the distributed system. Therefore, in this section we provide a heuristic to calculate the delays to be applied to each incoming message, that approximates the optimal solution.

This section also provides a complete protocol that gathers the topology information required for the solver or the heuristic and provides an optimistic delivery that outperforms the original SETO algorithm described in Section 2.2.

4.1. Rationale for the Heuristic

Our heuristic works as follows. We incrementally build a network by adding one process at a time. The first process to be added defines the first message in the total order $\prod$, named π_1 (see Section 3). This first process has complete freedom to set the delay it imposes on its own messages (which may be zero).

Whenever another process is added to the network it tries to set itself as close as possible to π_1 in the total order $\prod$, subject to the restrictions from equations (2) and (9). Note that the later a process is inserted in the network, the larger the set of constraints it has to satisfy; therefore, it is likely that later processes may be required to add longer delays to their own messages.

A key point in the heuristic is the order by which processes are inserted in the network. To define this order, we use the following insight. Consider process p_k. When process p_k is added to the network, the delay that this process has to impose to its own messages x_k^k depends of the constraints imposed by processes p_1, ..., p_{k-1} previously inserted. Consider now the next process to be added to the system p_{k+1} and ω_k^{k+1} the latency of the link between p_k and p_{k+1}. If $x_k^k > \omega_k^{k+1}$, p_{k+1} will be forced to impose a delay to its own messages of at least $\omega_k^{k+1} - x_k^k$. This will happen if p_k is very close to p_{k+1} and is forced to impose a long delay to its own messages. This means that the closer a process is to other processes, the more likely it is to influence the delays imposed on the messages from those processes. Thus, processes that are closer to other processes should impose the minimum delays to their own messages.

Since processes that are inserted earlier in the network are more likely to impose smaller delays (as they have less constraints to satisfy), these processes should be the ones to be inserted first. The heuristic described in the next paragraphs, uses a precise metric to capture the fuzzy notion of "closeness" introduced here.

The paragraph above explains the negative impact of setting x_k^k such that $x_k^k > \omega_k^{k+1}$. It is interesting to note however, that it may not be always desirable to set $x_k^k = 0$. In fact, by setting $x_k^k = 0$, we are forcing p_{k+1} to set $x_{k+1}^{k+1} = \omega_k^{k+1}$. So, there is a trade-off between the delay process p_k imposes on its own messages and the delay that other processes will later have to impose on their own messages. In the heuristic below we also specify a concrete formula to capture this balance.

4.2. Optimal Delay Approximation

The specification of the heuristic is presented in Figure 4. The procedure works in a stepwise fashion, computing delays for one process at a time.

First, let us introduce the notation used. A process in the group is denoted by p_i, where i is the process identifier. The *cost* metric used provides a measure of the distance each process has to all the remaining processes. Let ω_i^j denote the latency of the transmission delay between processes i and j, and r_i the rate at which process p_i sends messages. The cost c_i associated with process p_i is given by the following expression:

$$c_i = \frac{\sum_{j=1}^{N} \omega_i^j}{r_i}$$

The heuristic considers a group of N processes where each process holds complete information about all the transmission delays of all processes. This information can be expressed in a $N \times N$ matrix that we call Ω, as illustrated in Figure 5, where the rows are the processes and the columns are the transmission delays of messages from each of the corresponding processes, i.e., position $\Omega(i, j)$ holds the transmission delay ω_i^j of messages from process j to process i. Every process also maintains a similar matrix χ, holding the delays each process applies to messages from the others (including themselves) from which the final delay matrix Δ can be derived. We recall that the final delay δ_i^j is defined as $\delta_i^j = \omega_i^j + x_i^j$. Figure 5 illustrates the values of these matrices in the optimum configuration for the topology of Figure 3(a).

The heuristic proposed runs on an average of $O(N^2)$ steps (where N is the number of processes in P). The main cycle introduces a process at each iteration which is done exactly $N - 1$ times. Each of these iterations includes four cycles each of which executes $O(\sharp S)$ iterations, with $\sharp S$ going from 1 to N.

```
Initialization:
    P {processes in the communication group}
    N ← #P
    S ← ∅ {selected processes}
    for all i ∈ P do
        c_i ← (Σ_{j=1}^{N} ω_i^j) / r_i
        for all j ∈ P do
            x_i^j ← 0

procedure selectNextMin()
    next ← i : c_i = min{c_k, ∀k ∉ S}
    S ← S ∪ {next}
    return next

procedure firstDelay()
    iMin ← i : c_i = min{c_k, ∀k ∈ P}
    iNextMin ← i : c_i = min{c_k, ∀k ∈ P \ {iMin}}
    iMax ← i : c_i = max{c_k, ∀k ∈ P}
    α ← ω_{iMin}^{iNextMin} − (ω_{iMin}^{iNextMin} × (c_{iNextMin} − c_{iMin})) / (c_{iMax} − c_{iMin})
    return α

procedure FastSETOHeuristic()
    first ← selectNextMin()
    x_{first}^{first} ← firstDelay()
    ref ← first
    while (S ≠ P) do
        next ← selectNextMin()
        for all i ∈ S do
            z_i ← ω_{next}^i − δ_{ref}^i
        z_max ← max{z_i, i ∈ S}
        for all i ∈ S : i ≠ next do
            x_{next}^i ← δ_{ref}^i + z_max − ω_{next}^i
        shift^{next} ← 0
        for all i ∈ S do
            x_i^{next} ← δ_{ref}^{next} + δ_i^{ref} − δ_{ref}^{ref} − ω_i^{next}
            shift^{next} ← min(shift^{next}, x_i^{next})
        for all i ∈ S do
            x_i^{next} ← x_i^{next} + |shift^{next}|
        if z_max < 0 then
            ref = next
```

Figure 4. Heuristic.

We now illustrate its execution step by step for the topology of Figure 5. In this example, we assume that all processes transmit at the same rate. The algorithm is initiated by computing the costs for each process: $c_1 = 7 + 5 = 12$, $c_2 = 9 + 5 = 14$ and $c_3 = 7 + 9 = 16$. Thus, the first process to be added to the network is process p_1. Node p_1 has no constraint in the delay it may impose to its own messages. Therefore, it uses the formula depicted in Figure 4 (line 17) to set this value. In the case of the current example, the resulting value is 2.5. This concludes Step 1 of the algorithm. The value of the matrices after this step are depicted in Figure 6.

On Step 2, the next process with the lowest cost is process p_2. We first discover the delay x_2^2 that p_2 needs to impose on its own messages. Note that we want to minimize x_2^2 subject to the constraint $\delta_2^1 - \delta_1^1 = \delta_2^2 - \delta_1^2$. Given that a message sent by p_2 at time 0 is received at p_1 at time 5, we have $\delta_2^1 - \delta_1^1 = 5 - 2.5 = 2.5$. Also, given that a message sent by p_1 at time 0 is received at p_2 at time 5, to respect the constraint we need to set $x_2^2 = 7.5$. The value of the

$$\begin{bmatrix} 3 & 5 & 7 \\ 5 & 7 & 9 \\ 7 & 9 & 11 \end{bmatrix} = \begin{bmatrix} 0 & 5 & 7 \\ 5 & 0 & 9 \\ 7 & 9 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

Figure 5. Delay matrices ($\Delta = \Omega + \chi$)

Step 1

$$\begin{bmatrix} 2.5 & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} = \begin{bmatrix} 0.0 & 5.0 & 7.0 \\ 5.0 & 0.0 & 9.0 \\ 7.0 & 9.0 & 0.0 \end{bmatrix} + \begin{bmatrix} 2.5 & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Step 2

$$\begin{bmatrix} 2.5 & 5.0 & ? \\ 5.0 & 7.5 & ? \\ ? & ? & ? \end{bmatrix} = \begin{bmatrix} 0.0 & 5.0 & 7.0 \\ 5.0 & 0.0 & 9.0 \\ 7.0 & 9.0 & 0.0 \end{bmatrix} + \begin{bmatrix} 2.5 & 0.0 & ? \\ 0.0 & 7.5 & ? \\ ? & ? & ? \end{bmatrix}$$

Step 3

$$\begin{bmatrix} 2.5 & 5.0 & 7.0 \\ 5.0 & 7.5 & 9.5 \\ 7.0 & 9.5 & 11.5 \end{bmatrix} = \begin{bmatrix} 0.0 & 5.0 & 7.0 \\ 5.0 & 0.0 & 9.0 \\ 7.0 & 9.0 & 0.0 \end{bmatrix} + \begin{bmatrix} 2.5 & 0.0 & 0.0 \\ 0.0 & 7.5 & 0.5 \\ 0.0 & 0.5 & 11.5 \end{bmatrix}$$

Figure 6. Execution of the heuristic.

matrices after Step 2 are depicted in Figure 6.

In the final step, we need to add the last process p_3. Note that the value of x_3^3 needs to be minimized having now two constraints to satisfy. When looking at the reception order at process p_1, it is possible to see that messages from p_3 have to be delivered $4.5ms$ after messages from p_1. When looking at the reception order at process p_2, it is possible to see that messages from p_3 have to be delivered $4ms$ after messages from p_1. The limiting factor is therefore the interval at p_1, that forces p_3 to set $x_3^3 = 11.5$, resulting in the final matrices depicted in Figure 6.

Note that, in this case, the final average cost for the configuration derived from our heuristic is $\Delta_{\text{avg}} = 7.16$ which is within 3% of the optimum computed by the solver. In the evaluation we present results for other networks as well.

4.3. Algorithm

We now present an augment version of the SETO algorithm, that we have named Fast SETO, that can work with the heuristic or by calling a solver to obtain the minimum optimistic latency. The complete Fast SETO algorithm is specified in Figure 7. The algorithm works in four steps. In the first step, every process collects round-trip delays from all the other processes and estimates the corresponding transmission delays. This step is omitted in the algorithm specification for clarity sake, given there are multiple ways of collecting round-trip estimations and that procedure is orthogonal to the main algorithm. For instance, if TCP is used as the underlying transport protocol, the round-trip estimation could be extracted from the TCP implementation

```
1: Initialization:
2:     P ← p_1, ..., p_n {Process group}
3:     delay[1..n] ← 0 {Delay applied to messages}
4:     tdelay[1..n] ← 0 {Transmission delays to the process}
5:     c_t delay[1..n][1..n] ← 0 {Complete transmission delay matrix}

6: procedure computeDelays()
7:     {computes delays using the solver or the heuristic}

8: upon R_deliver(DELAY(new_delay[]) do
9:     if sender = seq then
10:        c_t delay[new_delay.sender] = new_delay
11:    else
12:        delay = new_delay

13: procedure updateDelays()
14:    R_unicast(seq,DELAY(tdelay))

15: upon allDelaysGathered() do
16:    c_delay ← computeDelays()
17:    for all p_i ∈ P do
18:        R_unicast(p_i,DELAY(c_delay[p_i])

19: procedure TO_multicast(m)
20:    R_multicast(DATA(m))

21: upon R_deliver(DATA(m) do
22:    R ← R ∪ {(m, now + delay[m.sender])}

23: upon ∃(m, d, t, md) ∈ R : now ≥ t ∧ m ∉ O ∧ m ∉ F do
24:    opt_delivery(m)
25:    O ← O ∪ m
26:    if p = seq then
27:        g ← g + 1
28:        R_multicast(SEQ(m, g))

29: upon R_deliver(SEQ(m, s)) do
30:    S ← S ∪ {(m, s, now)}

31: upon ∃(m, d, o) ∈ R : (m, l + 1, t) ∈ S ∧ m ∉ F do
32:    fnl_delivery(m)
33:    l ← l + 1
34:    F ← F ∪ {m}
```

Figure 7. Fast SETO algorithm.

without any extra cost. In the second step, all processes send the gathered delay information to a specific process. This process's identity can be easily derived from the group membership; for instance, it can be the process with the smallest identifier. The process gathers all the delay information and computes the optimal delay when a solver is available, or approximates the optimal solution using the heuristic described in Section 4.2 when the use of a solver is impractical. Finally, it sends to each process in the group the corresponding line in the delay matrix, which holds the delays that must be enforced by that specific process.

Our algorithm clearly differs from the original SETO algorithm by requiring complete knowledge of the transmission delays between all processes, which translates into the exchange of distance vectors between all nodes. The original SETO requires no such information, making use of only local clock values to determine the artificial delays. However, our proposal significantly improves the overall average latency of the system, as will be shown in the next section.

5. Evaluation

In this section we evaluate our proposed algorithm against the optimal delay assignment and the original SETO algorithm. The evaluation tests were performed in a simulated environment that consists of a network topology, transmission rates associated with each node and three models that describe the three algorithms at stake: optimal assignment, original SETO and Fast SETO using the heuristic. The network topologies used were generated with BRITE [3]. The tests were performed in networks with 30 nodes (10 nodes for the last evaluation test) that where randomly placed in a topological space.

5.1. Network Plane Size

We first compare the performance of the optimal assignment, original SETO and Fast SETO when the network plane size is changed. BRITE allows for the definition of the plane size by specifying the dimension of one side. In the experiments performed we made this side vary between 1000 and 5000 units. For each space dimension 20 network topologies were generated, and the results shown are average values of the observations on those networks. Also, in the original SETO algorithm a randomly selected sequencer was used.

The results are depicted in Figure 8. The explanation for these results lies in the way the original SETO algorithm determines the delay imposed by the sequencer to its own messages, which is equal to the longest link that reaches the sequencer. This value then conditions the adjustments in the remaining nodes and produces the observed results. The results also show the improvements obtained by Fast SETO in regard to the original algorithm and also its proximity to the optimal solution. In the experiments, the original SETO algorithm was, on average, 66% to 114% worst than the optimal assignment. Fast SETO with the heuristic was, on average, 16% to 33% worst than the optimal assignment, which shows the significant gains obtained by using our proposed algorithm.

5.2. Process Transmission Rates

We now compare the performance of the optimal assignment, original SETO and Fast SETO when the transmission rates of the processes in the network are changed. This time we set the topological space to a constant value. Each experiment consisted in generating 20 different topologies where all nodes exhibited an average transmission rate of 300 messages per second, with a predefined variance. The standard deviation of the transmission rates for each experiment was made variable between 0% to 100%. As in the previous

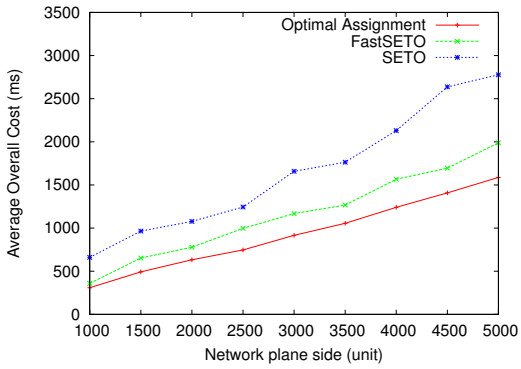
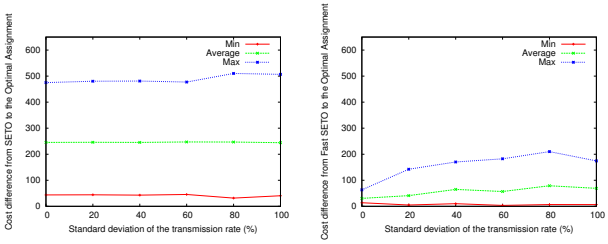


Figure 8. Network diameter.



(a) Original SETO (b) Fast SETO

Figure 9. Relative rates.

experiment, the sequencer for the original SETO algorithm experiments was randomly selected.

Figures 9(a) and 9(b) hold the results for both the experiments comparing the original SETO with the optimal assignment and Fast SETO with also the optimal assignment, respectively. Each figure presents the results as differences from each algorithm to the optimal assignment. The three lines presented in each figure are the minimum, maximum and average values observed from all the 20 topologies for each standard deviation value.

As expected, the average overall cost of the Fast SETO algorithm suffers less variation than the original SETO algorithm. The reason for this is that Fast SETO takes into account the transmission rates when computing the artificial delays. The original SETO algorithm makes no use of this information, which makes its results more dependent of the specific topology where it is executing.

5.3. Sequencer Position

The final evaluation compared the three algorithms: original SETO, Fast SETO and optimal assignment, in regard to the sequencer position in the original SETO algorithm. This "position" refers to the identifier of the node that performs the sequencer role. For the experiments we used a network of 10 nodes with transmission rates uniformly distributed by all nodes and varying from 0 to 100

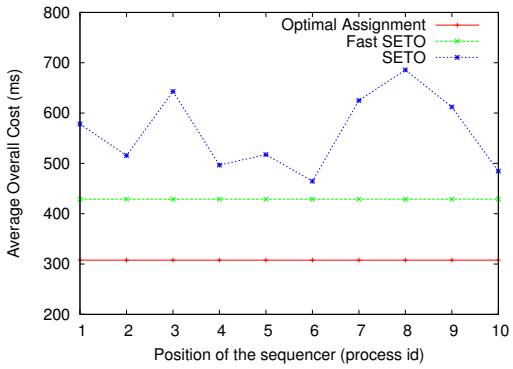


Figure 10. Sequencer's position.

messages per second. For each sequencer position 20 network topologies where generated. The same 20 network topologies where used in all tests for the different positions, and the average overall cost of the 20 observations was used to produce the results presented in Figure 10.

The lines that represent the optimal assignment and the Fast SETO algorithm are obviously straight lines, because both algorithms results are not dependent of the sequencer position. As for the original SETO line, it is quite irregular and varies from little below $500ms$ to almost $700ms$, and always stays above the other algorithms lines. This clearly shows how the results of this algorithm depend on the sequencer position for a given network topology.

For this, the first step of the heuristic we propose for the Fast SETO algorithm may be quite useful for improving the results of the original SETO algorithm by helping to choose the best sequencer location.

6. Conclusion

In this paper we have studied the problem of minimizing the latency of optimistic delivery in total order protocols for wide-area networks. The contributions of the paper are the following. Firstly, we have shown that previous solutions do not provide the optimal solution. We have formalized the problem of finding the optimum with a linear programming model and have shown how to find the optimal solution when a solver is available. Secondly, we have proposed an heuristic that allows to approximate the optimal solution; the heuristic can be used whenever is impractical to have a solver available at every process. Finally we have compared the performance of the optimal solution, as well as the results from our heuristic, with the results from the SETO protocol of [9] (to the best of our knowledge, the previous best results for optimistic delivery in wide-area networks). We show that SETO performs considerably worse than the optimal assignment and that our heuristic, for all the networks tested, is within 16%-33% (as opposed to 66%-114% for the original SETO algorithm) of the optimum and is more immune to variations in the topology.

This paper has not addressed the accuracy of optimistic protocols. Experimental results regarding this issue can be found in [9, 7]. These results show that an optimistic approach like the one described in this paper is useful when executed in stable networks. To deal with network instability, we have also proposed a protocol that allows the switching between total order algorithm implementations when necessary [4].

Acknowledgments

The authors are grateful to the anonymous reviewers for their insightful comments on earlier versions of this paper.

References

[1] ILOG CPLEX. http://www.ilog.com/products/cplex/.

[2] B. Kemme, F. Pedone, G. Alonso, A. Schiper, and M. Wiesmann. Using optimistic atomic broadcast in transaction processing systems. *IEEE Trans. on Knowledge and Data Engineering*, 15(4):1018–1032, Jul/Aug 2003.

[3] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *Proc. of the 9th Int. Symp. in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 346, Washington, DC, USA, 2001. IEEE CS.

[4] J. Mocito and L. Rodrigues. Run-time switching between total order algorithms. In *Proceedings of the Euro-Par 2006*, pages 582–591, Dresden, Germany, August 2006. Springer-Verlag.

[5] F. Pedone, R. Guerraoui, and A. Schiper. Exploiting atomic broadcast in replicated databases. In *Proc. of the 4th International Euro-Par Conference on Parallel Processing*, pages 513–520, London, UK, 1998. Springer-Verlag.

[6] F. Pedone and A. Schiper. Optimistic atomic broadcast. In *Proc. of the 12th International Symposium on Distributed Computing*, 1998.

[7] L. Rodrigues, J. Mocito, and N. Carvalho. From spontaneous total order to uniform total order: different degrees of optimistic delivery. In *Proc. of the 2006 ACM Symp. on Applied Computing*, volume 1, pages 723–727, Dijon, France, April 2006. ACM.

[8] L. Rodrigues, P. Veríssimo, and A. Casimiro. Using atomic broadcast to implement *a posteriori* agreement for clock synchronization. In *Proc. of the 12th IEEE Symp. on Reliable Distributed Systems*, pages 115–124, Princeton, New Jersey, Oct. 1993. IEEE.

[9] A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *Proc. of the 21st IEEE Symp. on Reliable Distributed Systems*, pages 190–199. IEEE CS, Oct. 2002.

[10] P. Vicente and L. Rodrigues. An indulgent uniform total order algorithm with optimistic delivery. In *Proc. of the 21st IEEE Symp. on Reliable Distributed Systems*, pages 92–101, Osaka, Japan, Oct. 2002.