이더네트에 기반한 분산 실시간 시스템에서의 시계동기화 알고리즘

A Clock Synchronization Algorithm for Ethernet-based Distributed Systems

요 약

본 논문에서는 이더네트를 기반으로 하여 구축된 분산 실시간 시스템에 적합한 시계 동기화 알고리즘을 제시한다. 이더네트에서 메시지의 전송시간은 예측이 불가능하지만 방송 기능을 사용하면 이를 극복할 수 있다. N 개의 노드들이 시계 동기화에 참여한다고 할 때 시계들을 동기화시키기 위하여 N+2 개의 메시지가 필요한데, 이는 두 개의 포인팅 메시지와 N 개의 인터벌 메시지로 구성된다. 포인팅 메시지는 주노드에 의해 방송되며 단순히 메시지의 도착시간을 기록하기 위해 사용된다. 인터벌 메시지는 각 노드가 측정한 두 포인팅 메시지의 도착시간 간격을 포함하며, 인터벌 메시지를 통해 방송된 시간간격들은 시계를 동기화하는데 사용된다. 본 논문에서 제시하는 알고리즘에 의해 예측불가능한 메시지 전송시간에도 불구하고 시계들이 제한된 오류값 이내에서 동기화됨을 보인다.

Abstract

This paper presents an algorithm for clock synchronization in an Ethernet-based distributed computer system using $N+2$ broadcast messages, where N is the number of the nodes that participate in the clock synchronization process. In the proposed algorithm a synchronization round consists of two pointing messages and N interval messages. Two pointing messages broadcasted by the leader node are used by all nodes to record their arrival times, while interval messages by all nodes contain the time interval between the two pointing messages measured at each node. These intervals are then used to correct local clocks. This algorithm is shown to converge into the bounded clock value and tolerate arbitrary delays of messages.

1 Introduction

A distributed real-time system consists of a set of nodes that communicate with each other via real-time message passing over a local area network(LAN). Each node is an autonomous computer performing a specified function and has a local real-time clock whose rate may not be the

same as those in other nodes. In order to determine the time of an event, the duration between two events, and the ordering between events occurring at different nodes in distributed systems, it is necessary to provide applications with a common time reference of specified accuracy. Namely, the maximum difference between any two non-faulty clocks in the same network cluster should be less than a predetermined value. Thus, to periodically resynchronize the clocks in the systems is required. In this process, clock failure in a single node or in some nodes must not disable the clock synchronization process.

Many algorithms have been developed to synchronize clocks. They were built on diverse networks with specific characteristics, that is, for instance, point-to-point network and random delay network. In this paper we address the clock synchronization problem found on an Ethernet-based network. The Ethernet protocol, which has been adopted as one of the standards in local area networks, incorporates the CSMA/CD techniques[1]. However, although Ethernet implementations are widespread and simple, it does not guarantee real-time communications since transmission delays are not bounded. The delays depend on the amount of communication and computation occurring concurrently in the system: collisions may cause messages to be retransmitted many times, and waiting time for carrier sense can be severely long. Our clock synchronization algorithm is based on the broadcast facility and it provides the mechanism that is tolerant of arbitrary delays. By synchronized clocks these unpredictable communication elements in Ethernet can be eliminated. In short, the objective of this paper is to present a clock synchronization algorithm in order to support real-time communication using Ethernet.

This paper is organized as follows. We begin by introducing related works in Section 2. A real-time clock synchronization algorithm and its properties are given in Section 3. In Section 4, algorithm analysis is performed. Finally, the results are discussed in Section 5.

2 Related Works

A number of algorithms have been proposed for clock synchronization in distributed real-time computer systems. Some of them are shown in Table 1 where their network characteristics, number of messages, and use of broadcasting facility are compared.

Srikanth and Toueg[2] have developed an algorithm in which synchronization takes place in rounds, and the rounds start at multiples of a specified interval. Their algorithm is based on a

reliable, error-free, and fully connected point-to-point message system. Each process broadcasts resynchronization message and, after receiving prefixed number of resynchronization messages, accepts the resynchronization and resets its time to the correct multiple of the interval plus a constant conveniently chosen to avoid setting the time backward. This algorithm achieves optimal accuracy, in the sense that the accuracy of a clock is as good as that specified for the underlying hardware clock.

TEMPO[3,4,5] is a typical clock system with central control, which consists of a master and several slave nodes. A master time daemon periodically measures the time difference between the clock of the machine on which it is running and those of all other machines. The master computes the network time as the average of times provided by non-faulty clocks. Then it transmits the correction term to each slave node.

Arvind proposed a probabilistic algorithm[6] that involves the transmission of multiple synchronization messages and works by averaging the random variations in the end-to-end delays of the synchronization messages. This algorithm can be applied to Ethernet-based distributed computer systems, but needs many messages. Cristian's work[7] is another example of probabilistic algorithm.

Kopetz analyzed the clock synchronization in a loosely coupled distributed system[8], derived an upper bound on the synchronization accuracy and presented a VLSI clock synchronization of a fault-tolerant global time base.

Verissimo built clock synchronization mechanism[9] on xAMp[10], a reliable broadcast protocol. In this scheme, every node broadcasts a time-marker packet at each predefined time. For each round, the node which broadcasts the f_p-th packet collects replies which contain packet arrival time from other nodes and decides the adjustment value. f_p is tolerable number of faults.

In the distributed systems, the clock synchronization should be realized by the exchange of messages, so the synchronization algorithm should be able to adapt to the target network environment. In Ethernet, messages may be partially lost(a broadcast message may be received by some nodes while not by others), interfere one another(collide and are retransmitted, so the transmission time may be prolonged), and the message transmission time is not bounded. Thus it is desirable for clock synchronization algorithm on Ethernet to cope with arbitrary transmission delay, to have small number of messages, and enhance the possibility of succeeding in synchronization, let

Table 1. Comparison of clock synchronization algorithms

	[2]	[3]	[6]	[8]
broadcast	used	not used	may be used	used
network	point-point network	any type	random delay	bounded max delay
no. of mess	$\leq node$	$3 \times node$	$node \times rep$	$node$

Note: rep denotes the number of replication number of message

alone provide the precision of synchronization. Srikanth's work is not suitable for Ethernet as it is based on fully connected point-to-point network. TEMPO seems adaptable to Ethernet, however, it needs $3 \times N$ messages, and the arbitrary transmission delay induces correction error when master daemon reads clocks of other machines. The authors calculated error bound of TEMPO by assuming bounded maximum delay. Arvind's work also can be applied to Ethernet with reduced correction error, but needs many messages and mainly focuses on reading clock of other node(not global synchronization). As an example of distributed algorithm, Kopetz's work needs only N(the number of nodes) messages, however, it requires the bounded transmission time for all messages as it uses the local clock value as timing data at each synchronization point like TEMPO. As for Verissimo's *posteriori agreement*, while the idea of using time-marker packet is similar to ours, it requires reliable broadcast protocol where all messages can be broadcasted reliably and the order of messages should be preserved. In contrast, the algorithm we will propose in this paper can cope with arbitrary transmission delay as it uses time interval between message arrival times as timing data, needs $(N+2)$ messages as it uses broadcast facility of Ethernet, and enhances the possibility of succeeding in synchronizing clocks as it can tolerate up to $(N/2-1)$ partial loss of message.

3 Ethernet-Based Clock Synchronization

3.1 Assumptions

Our algorithm for Ethernet-based clock synchronization is developed on the basis of the following assumptions.

A1 When the leader node broadcasts a message, all receiving nodes experience the same amount of delay due to message collisions and waiting time to sense carrier. Any difference in the delay is caused only by propagation delay from the leader node to individual receiving nodes. As the type of network is restricted to single-segment Ethernet, there is no variable delay due

to repeater, bridge, or gateway.

A2 At each receiving node, the time required to check a message and to record the arrival time of message, i.e., system overhead, is bounded. Though system overhead cannot be disregarded, it is usually small and may be reduced.

A3 There is sufficient number of messages for synchronization.

A4 The hardware clocks are ρ-bounded, where ρ is the maximum drift rate of a local hardware clock.

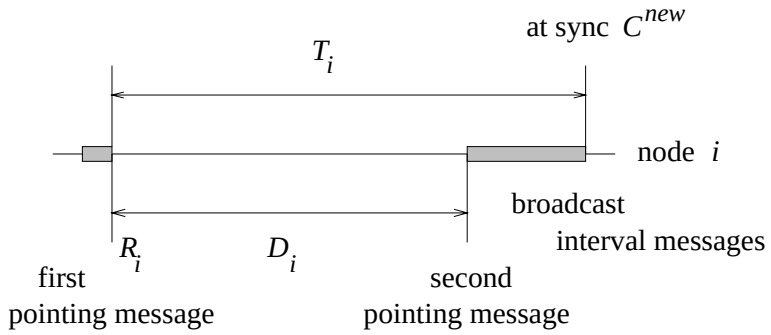
A5 The clocks do not go backward. After correction term is calculated, each node spreads the correction term over the next synchronization period and apply it continuously.

In addition, when the leader fails, an election algorithm that elects a new leader should be provided. It is assumed, however, that elections do not occur, as we are mainly concerned with clock synchronization.

3.2 Definition and Formulation

Of previous studies including those described in the preceding section, some use broadcasting facilities for the nodes to announce their times. Others use polling mechanism in which messages are interchanged by the initiative of a master. However, these schemes fail to completely exclude the uncertainty of transmission delay. In the case of Ethernet, this unbounded transmission delay is the most critical disadvantage for distributed real-time communication.

Taking advantage of the broadcasting capabilities of Ethernet, our algorithm is made independent of the message transmission delays and is able to keep clock deviation relatively small at a synchronization point of time. A synchronization round is composed of (1) two *pointing messages* and (2) N *interval messages*, where N is the number of nodes in the network. *Pointing messages* are messages broadcasted by the leader node. These messages are used by all nodes only to record the arrival time. The pointing messages can contain round specifications thus eliminating the intervention caused by the messages of other rounds. *Interval messages* are broadcasted by all nodes, and contain the time interval between the two pointing messages measured at each node. These intervals are then used to correct local clocks.



T_i : period of synchronization round
D_i : time interval measured at node i
R_i : the time value when the first pointing message arrives at node i

$$D = \frac{1}{N}\sum_{i=1}^{N} D_i \quad , \qquad R = \frac{1}{N}\sum_{i=1}^{N} R_i$$

$$C^{new} = R + \left(\frac{D - D_i}{D_i} + 1\right) \cdot T_i = R + \frac{D}{D_i} \cdot T_i$$

Figure 1. Formulation of Ethernet-based clock synchronization

A round has three phases. (1) pointing phase, (2) broadcasting phase, and (3) correction phase. In *pointing phase* the leader broadcasts two pointing messages with a relatively long interval. Each node records the local clock values at the arrival of first and second pointing messages and measures the time interval using its own clock. The measured intervals, as expected, are different from node to node as a result of different clock drift rate of nodes. In the *broadcasting phase*, each node informs other nodes of its own measured interval via broadcasting interval message. In the third phase, i.e., *correction phase*, on receiving the measured values of other nodes, each node will calculate the new time. Figure 1 shows the basic idea of this procedure.

In Figure 1, D_i denotes the time interval between the arrivals of two pointing messages from the leader observed at node i. The filled area indicates arbitrary transmission time. The interval between arrival times of two pointing messages measured at each node should be the same if all clocks proceed at an identical rate regardless of collisions and retransmissions. That is, when measured at each node, the time intervals between two pointing messages, D_i's, are all the same in actual or absolute time; however, they are different in local time on account of the different rate of each node's physical clock. The interval values of each node are known to all nodes via

broadcasting interval messages. As a result, the mean value D or the agreed value by all nodes in this round can be calculated. The time value of each node before a synchronization round is more or less different, though it is bounded. Thus the *reference value* R_i, the time value when the first pointing message arrives at a node, is sent with D_i to the other nodes. The initial value of D_i and R_i is null at the start time of synchronization round. Finally each node calculates C^{new} using those data. In calculating C^{new}, we choose to average timing data(D_i's and R_i's) like TEMPO and Kopetz's work, as it can produce better result on message loss. Figure 2 describes this algorithm in detail. The use of time marker packet(i.e., pointing messages) is similar to *posteriori agreement* [9], but it is efficient for reliable broadcast protocol like xAMp. In the CSMA/CD-based network where transmission delay is not bounded, any algorithm cannot always synchronize clocks within a tolerable time. Thus it is desirable to enhance the possibility of tolerating the unbounded transmission delay. *Posteriori agreement* uses time marker packet, but it can proceed only after broadcast succeeds. The time marking phase may be prolonged to an intolerable time on Ethernet. In contrast, our algorithm can proceed without confirmation that all nodes receive the pointing messages, enhancing the possibility of synchronizing the clocks.

The proposed algorithm can efficiently(with small number of messages) synchronize clocks in Ethernet environment where the transmission delay is arbitrary and the broadcast message can be lost partially. It can cope with the arbitrary delay of message transmission as it is based on the interval between pointing messages as described above rather than uses local clock values and estimates transmission times or delays. This is due to the fact that the interval of message arrivals builds a time reference for the absolute time in spite of the variable transmission delay. As for fault tolerance, if a node fails, other nodes are not affected; they only detect the failure and ignore the failed node thereafter. Hence, the protocol has the fault tolerance capability. While broadcast facility can reduce the number of messages in the synchronization process, a broadcasted message may be partially lost, namely, some nodes receive while others do not(due to some reasons like sudden buffer full on a node). While complete loss of a message can be detected by the broadcaster, and retransmitted, partial receipt of message can hardly be detected without acknowledgement message. Our protocol can tolerate the partial receipt of message, as long as valid data exceeds $\frac{N}{2}$, thus enhances the possibility of synchronizing clocks. In addition, it is desirable for the synchronization process to reduce the number of messages to prevent a message from intervening other messages. As we use broadcast facility, the algorithm needs only $(N + 2)$ messages(much smaller than the

above-mentioned algorithms). After all, as the proposed algorithm can cope with various problems in synchronizing clocks especially on Ethernet, it is suitable to Ethernet.

```
function clock_sync_at_leader()
begin
        pointing_phase_of_leader(R_mine, D_mine)
        broadcasting_phase(R_mine, D_mine, {D_i},{R_i})
        correction_phase(R_mine, D_mine, {D_i},{R_i})
end

function clock_sync()
begin
       pointing_phase(R_mine, D_mine)
       broadcasting_phase(R_mine, D_mine, {D_i},{R_i})
       correction_phase (R_mine, D_mine, {D_i},{R_i})
end

function pointing_phase_of_leader(R_mine, D_mine)
begin
       D_mine = NULL
       R_mine = NULL
       broadcast the first pointing message at t_1
       receive the first pointing message and record the time t_1 and R_mine
       sleep for the predefined interval
       broadcast the second pointing message at t_2
       receive the second pointing message and record the time t_2
        D_mine := t_2 - t_1
end

function pointing_phase(R_mine, D_mine)
begin
       D_mine = NULL
       R_mine = NULL
       receive message
       if (the first pointing message)
              record arrival time t_1 and R_mine
              receive message
              if (the second pointing message)
                     record arrival time t_2
                     D_mine := t_2 - t_1
                     return
       return                          /* illegal sequence of message */
end

function broadcasting_phase (R_mine, D_mine, {D_i},{R_i})
begin
        broadcast message (R_mine, D_mine)
        within T_time_out
              loop
                receive interval messages from other nodes and construct vector {R_i}, {D_i}
                if all messages are received
                   break
              endloop
        end within
        return                          /* transition by timeout */
end
```

```
function correction_phase (R_mine, D_mine, {D_i},{R_i}, R, D)
begin
        valid = remove_faulty_clock_&_calculate_R_D({R_i}, {D_i}, R, D)
        if (valid ≥ N/2)
                if (R_mine == NULL) R_mine = R
                t := current value of hardware clock
                if (D_mine == NULL)
                        C^new := R + D + (t − t_2)
                else
                        C^new := R + (t − R_mine) × D/D_mine
                set time with calculated value
        else                                /* insufficient data */
                do not update
        return
end

function remove_faulty_clock_&_R_D({R_i}, {D_i}, R, D)
begin
        for all ({R_i}, {D_i})
        remove D_i whose value is NULL
        remove (R_i, D_i) when D_i is out of bound from D
        endfor
        calculate R by averaging valid R_i's
        calculate D by averaging valid D_i's
return (the number of valid D_i's)
end
```

Figure 2. An algorithm for clock synchronization

3.3 Algorithm Description

Figure 2 describes the algorithm for our procedure and Figure 3 represents the state diagram of clock synchronization protocol. Generally, in the pointing phase, the leader node sends two pointing messages and other nodes measure the interval between two arrivals of the messages. In Ethernet, a packet may be lost due to transmission error or dropped at a heavy load. The leader also receives its pointing messages, so if the message is discarded by communication system, it rebroadcasts. Only the nodes that received both pointing messages can measure the interval, while others cannot. Now broadcasting phase begins. In the broadcasting phase, interval messages are exchanged among the nodes via broadcasts. The node which misses any pointing message should be made to broadcast null R_i and D_i after receiving the second pointing message, or an interval message for the first time from any other node. The broadcasting phase terminates when all interval messages are collected or timeout expires. Timeout prevents the overall process from being prolonged severely long. It is desirable that the broadcasting phase be as short as possible

because the clock deviation for this duration cannot be corrected precisely. Should the ratio to the duration of pointing phase be small, then the effect of imprecise correction for this phase will not be severe at all.

In the last phase, i.e., the correction phase, each node computes a new time with the collected data using the algorithm shown in Figure 2. Each node can drive two vectors, one for R_i's and the other for D_i's.

$$\begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ \vdots \\ R_n \end{pmatrix} \qquad \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_n \end{pmatrix}$$

Note that R_k and $D_k = 0$ when message from node k contains null data, is lost, or message delay had been so long that it didn't arrive within timeout. Each collected data should be examined to verify that it is valid. At node i, for instance, a clock would verify the validity of the interval message from node k by using

$$(1 - \rho)D_i \leq D_k \leq (1 + \rho)D_i$$

where ρ is the maximum drift rate of a local hardware clock(A4), so ρD_i is the clock deviation during interval D_i. Thus, each node extracts the elements which satisfy the above inequality from the vectors and from these calculates new time.

Figure 3 shows a state diagram of our algorithm. State 0 and 1 correspond to pointing phase. State 2 and 5 correspond to broadcasting phase, and state 3,4,6, and 7 correspond to correction phase. State 0 is the starting point of a synchronization round. Receving the first pointing message, a node(say i) waits for the second pointing message in state 1. After receiving the second pointing message at state 1, i broadcasts its interval message, goes to state 2, and waits for interval messages from other nodes. In case a node misses the first pointing message in state 0, or the second pointing message in state 1, the state of node changes to 5 (by an unexpected message). In state 5, the node i cannot calculate D_i and should broadcast its interval message with null D_i. It then waits for interval messages from other nodes in state 5. i changes its state to 3,4,6 and 7 (correction phase states) when it collects all interval messages or timeout expires. A node reaches state 3 when all

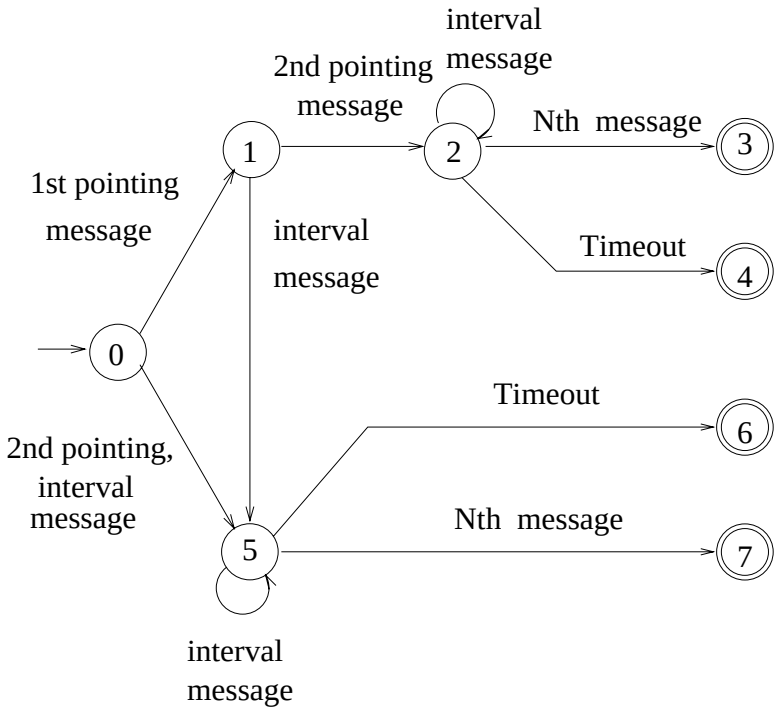


Figure 3. State diagram of clock synchronization protocol

the messages are received, and it can calculate C^{new} with D_{mine}, D,R_{mine}, and R. When a node receives two pointing messages, but misses any interval message, it changes to state 4 by timeout. In this state, it calculates C^{new} with D_{mine}, D,R_{mine}, and R, however, D and R may be different from other nodes. A node goes to state 6 when it misses at least one pointing message and any interval message. If a node has experienced the transition along the path $0 \rightarrow 1 \rightarrow 5 \rightarrow 6$, namely, it has received the first pointing message, its R_{mine} is valid. Otherwise, R_{mine} is set to R. In this case D_{mine} has not been measured so C^{new} is calculated as $R + D + (t - t_2)$. When a pointing message is missed but all interval messages have been received, a node reaches state 7. C^{new} is calculated as in state 6, but D and R have less error. In correction phase, if there is less than $\frac{N}{2}$ valid messages, the synchronization algorithm cannot guarantee calculating the new time within a bounded value. As we have assumed that required number of messages for synchronization is the same as that of other algorithms in A3, the node should ignore the synchronization of this round, namely, does not update its clock, and goes to the next round.

3.4 Main Properties

• Convergence

As shown in Theorem 4 and 5, if above $\frac{N}{2}$ interval messages contain valid data and are received by all nodes, the clocks do not diverge and synchronize to a common value with bounded error.

• Tolerance for the arbitrary delay of messages

As the pointing messages play a role of only marking the arrival time of the packet, arbitrary transmission time of the packet does not matter at all. Furthermore, the arbitrary delay of interval messages can be tolerated as long as it is not too long, i.e., in timeout period.

• Startup

When a network is set up, the leader will notify the other nodes of its time value. When a node is added to or detached from the network, it should notify all the other nodes, and other nodes must update the number of needed messages to synchronize.

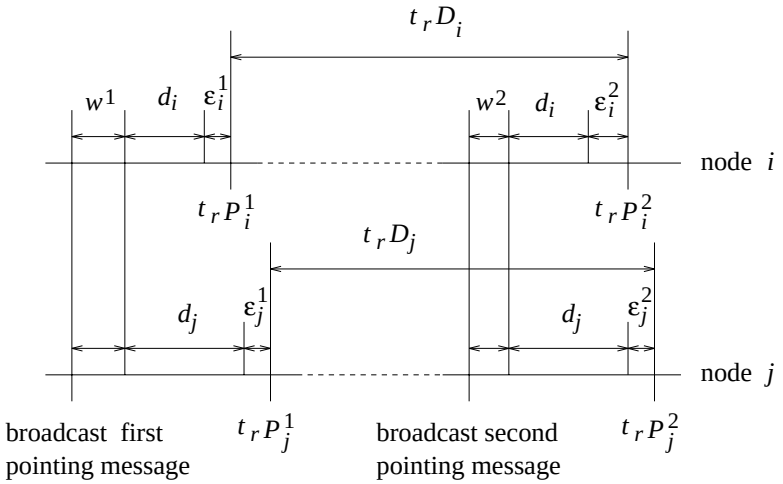
• External synchronization

The term of external clock deviation is calculated in Lemma 2. When the deviation goes too far, it can be corrected in the initiative of leader node. The leader node can contact with external clocks, which are leaders of other network or external time providers. Thus the leader knows how fast or slow the time of its network goes. If we make the pointing messages include this difference with external clocks, local clocks can use the information to correct that term in their synchronization rounds.

4 Analysis of the Algorithm

In this section we provide a detailed analysis of our algorithm and establish its properties. *Local Time* t_l is defined to be the time of the local clock in a node and *Real Time* t_r is the universal time.

As mentioned in Section 3.3, the leader node broadcasts two pointing messages. One is sent at the beginning of the synchronization round and the other is sent at the end of it. By $t_l P_i^1$ and $t_l P_i^2$, we refer to the arrival time value of the first and the second pointing message, respectively, measured by local clock of node i. $t_r P_i^1$ and $t_r P_i^2$ are regarded as the real time value of the arrivals of these pointing messages.



w : The arbitrary delay for the leader node to hold the transmission media

d_i : The pure transmission delay through media from the leader to node i. This is constant.

ϵ_i : The system overhead at node i. (The time interval for which the system reads the message)

Figure 4. The timing diagram of a synchronization round

LEMMA 1 (*pointing error*) *Let* $t_r D_i = t_r P_i^2 - t_r P_i^1$ *and* $t_r D = \frac{1}{N}\sum_{i=1}^{N} t_r D_i$, *where* N *is the number of nodes. Then node* i *can measure time interval with a bounded error.*

PROOF $t_r D_i$ and $t_r D_j$ can be measured with a bounded and small error that is the maximum intranode variation of system overhead, ϵ. This is shown in Figure 4.

Specifically,

$$t_r P_i^1 - (w^1 + d_i + \epsilon_i^1) = t_r P_j^1 - (w^1 + d_j + \epsilon_j^1) \quad (1)$$

$$t_r P_i^2 - (w^2 + d_i + \epsilon_i^2) = t_r P_j^2 - (w^2 + d_j + \epsilon_j^2) \quad (2)$$

By subtracting (2) from (1),

$$t_r P_i^2 - t_r P_i^1 - (\epsilon_i^2 - \epsilon_i^1) = t_r P_j^2 - t_r P_j^1 - (\epsilon_j^2 - \epsilon_j^1)$$

$$\begin{aligned} t_r D_i - t_r D_j &= (t_r P_i^2 - t_r P_i^1) - (t_r P_j^2 - t_r P_j^1) \\ &= (\epsilon_i^2 - \epsilon_i^1) - (\epsilon_j^2 - \epsilon_j^1) \end{aligned}$$

Let $\Delta\epsilon_{max}$ be $\max(\epsilon_i^2 - \epsilon_i^1)$ for all i, then

$$|t_rD_i - t_rD_j| < 2\Delta\epsilon_{max} \quad \text{for } i \neq j \text{ and}$$

$$|t_rD_i - t_rD| < \Delta\epsilon_{max} \quad \text{for } all\ i$$

Here $\Delta\epsilon_{max}$ is very small and can be approximated to about zero by assumption A2 where there is minimal variance in the time of reading of messages from the network when our algorithm is implemented in kernel. Thus each node can measure its clock deviation of the synchronization round with small and bounded error. □

LEMMA 2 (*closeness to real time*) *Let* $t_lD_i = t_lP_i^2 - t_lP_i^1$ *and* $t_lD = \frac{1}{N}\sum_{i=1}^{N} t_lD_i$. *Then* t_lD *approximates to* t_rD *for some large* N.

PROOF Let $t_lP_i^1 = t_rP_i^1 + \delta_i^1$ and $t_lP_i^2 = t_rP_i^2 + \delta_i^2$, where δ_i^k is the difference between the local clock at node i and the real time when the k-th pointing message arrives.

$$\begin{aligned} t_lD_i &= t_lP_i^2 - t_lP_i^1 \\ &= (t_rP_i^2 - t_rP_i^1) + (\delta_i^2 - \delta_i^1) \\ &= t_rD_i + (\delta_i^2 - \delta_i^1) \end{aligned}$$

$$\begin{aligned} tlD &= \frac{1}{N}\sum_{i=1}^{N} t_lD_i \\ &= t_rD + \frac{1}{N}\sum_{i=1}^{N}(\delta_i^2 - \delta_i^1) \end{aligned} \tag{3}$$

In (3), $|\delta_i^2 - \delta_i^1|$ is bounded to ρT, where T is the time interval of a synchronization round and ρ is the maximum clock drift rate. Thus theoretically $|t_lD - t_rD| < \rho T$ follows.
But in reality $\frac{1}{N}\sum_{i=1}^{N}(\delta_i^2 - \delta_i^1)$ converges to zero for some large N because some of $(\delta_i^2 - \delta_i^1)$ are positive and others are negative. Lemma follows. □

LEMMA 3 (*calculating error*) *The difference in the time value* C_i *calculated at each node is bounded. Specifically for* $i \neq j$

$$|C_i - C_j| < \Delta C_{max}, \text{ where } \Delta C_{max} = \frac{4\rho(T-D)}{1-\rho^2} + \Delta T\frac{1+\rho}{1-\rho}$$

PROOF When each node calculates $R + \frac{D}{D_i}T_i$, the maximum internode difference occurs between two nodes, where one goes fastest in pointing phase but slowest in broadcasting phase, and the other goes reversely.

The first clock computes as follows

$$\frac{D}{D(1+\rho)}\{D \cdot (1+\rho) + (T-D) \cdot (1-\rho)\} \tag{4}$$

and the other computes as follows

$$\frac{D}{D(1-\rho)}\{D \cdot (1-\rho) + (T-D) \cdot (1+\rho)\} \tag{5}$$

Subtracting (4) by (5), we can get the maximum difference between nodes, namely

$$(T-D)\frac{4\rho}{1-\rho^2}$$

Furthermore, in case the farthest node from the leader goes fastest in the broadcasting phase, (5) is rewritten as

$$\frac{D}{D(1-\rho)}\{D \cdot (1-\rho) + (T+\Delta T-D) \cdot (1+\rho)\} \tag{6}$$

where ΔT is the maximum difference in the propagation delay from the leader node. Subtracting (4) by (6), we can get the maximum difference between nodes, namely

$$(T-D)\frac{4\rho}{1-\rho^2} + \Delta T\frac{1-\rho}{1+\rho}$$

□

THEOREM 4 (*the effect of message loss*) *Clock deviation function does not diverge if there are above* $\frac{N}{2}$ *good interval messages.*

PROOF If there are k messages lost, a node computes with $(N-k)$ good messages, namely,

$$\frac{1}{N-k}\left(\sum_{i=1}^{N-k} R_i + \sum_{i=1}^{N-k} D_i\right)$$

At worst case, an interval message is received by one node and not received by another node. Let

f be the node whose message is lost at node i. Then the reference time and the measured interval at node f are lost in this round. So node i should compute without information on node f. Let $F(i)$ be the error term of round i, δ be the sum of maximum clock drift in a round and pointing error (namely, $\delta = 2\rho \mathrm{T} + \epsilon_{max}$), and R' be the sum of good R's excluding R_f.
The difference between computing with and without R_f is

$$\begin{aligned} \frac{R'+R_f}{N-k+1} - \frac{R'}{N-k} &\leq \frac{R'+R+F(i-1)}{N-k+1} - \frac{R'}{N-k} \\ &\leq \frac{R'+F(i-1)}{N-k} - \frac{R'}{N-k} \\ &= \frac{F(i-1)}{N-k} \end{aligned} \tag{7}$$

So a loss of R_f have an effect of at most $\frac{F(i-1)}{N-k}$. As is the case for loss of D_i, let D' be the sum of good D's excluding D_f.

$$\begin{aligned} \frac{D'+D_f}{N-k+1} - \frac{D'}{N-k} &\leq \frac{D'+D+\delta}{N-k+1} - \frac{D'}{N-k} \\ &= \frac{D'+\delta}{N-k} - \frac{D'}{N-k} \\ &= \frac{\delta}{N-k} \quad (= M_{loss}) \end{aligned} \tag{8}$$

By summation of (7) and (8), we get

$$F(i) = \frac{k}{N-k} F(i-1) + \frac{k}{N-k}\delta$$

In recurrence formula (7) the common ratio is $\frac{k}{N-k}$ and for this process to converge,

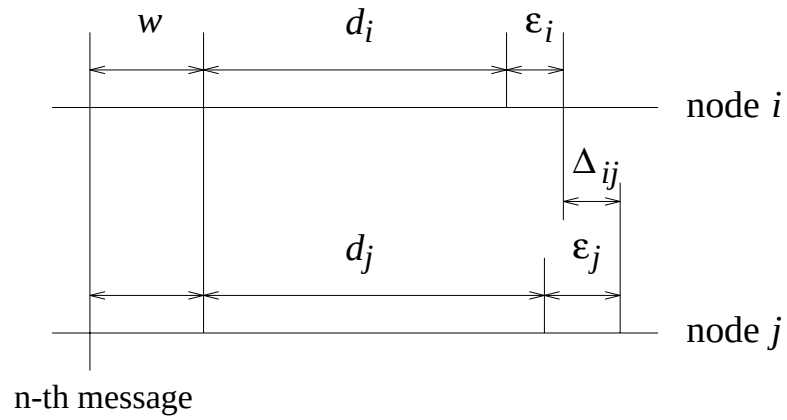
$$k < \frac{N}{2} \qquad \square$$

THEOREM 5 *The maximum deviation between two different clocks at synchronization point is bounded. That is, for nodes i and j*

$$|t_l C_i - t_l C_j| < \Delta_{max},\ where\ \Delta_{max} = \Delta C_{max} + U\rho + M_{loss}$$

$$U\ is\ MAX(\tau + \Delta\epsilon_{max},\ timeout - \sum d_i)$$

PROOF Each node updates its clock to the C^{new} at synchronization point. So the deviation between the clock of node i and that of j is due to the difference in C^{new} and the difference in time when



$$\Delta_{ij} = |(d_i - d_j) + (\epsilon_i - \epsilon_j)| \leq |d_i - d_j| + |\epsilon_i - \epsilon_j| = \Delta d_{max} + \Delta \epsilon_{max}$$

Figure 5. Arrival of n-th interval message

the clock is updated to the C^{new} as shown in Figure 5. U is the difference of updating time, and caused by the arrival time of n-th interval message and timeout. Δd_{max} is the maximum difference in time of the pure transmission delay through transmission media from leader to node. Thus it is at most differentiated by the end-to-end propagation delay, τ. $\Delta \epsilon_{max}$ is defined in Lemma 1, and M_{loss} in Theorem 4. ΔC_{ij} is the difference between C_i and C_j and this is bounded to ΔC_{max}, calculating error.

$$|t_l C_i - t_l C_j| = \Delta C_{ij} + \Delta_{ij}\rho + M_{loss} < \Delta C_{max} + U\rho + M_{loss} \quad \square$$

COROLLARY 6 (*the effect of pointing message loss*) *When a node misses a pointing message, the clock deviation increases by* ρT.

PROOF If a node does not receive one or two(all) pointing messages, the node should compute the new time without this lost D_i and R_i, that is, $R + DT$. Though D_i is not reflected in correction, the term averaged R corrects the error in the previous round, so the clock deviation cannot diverge. The added error term is ρT, the clock drift in T period. This error term ρT is added only once even though a pointing message has continuously missed. $\square$

As stated in Section 3.2, our algorithm is more adaptable in the Ethernet environment. In this

subsection, we analyze the synchronization possibilities of our algorithm. Additional parameters are defined as follows;

t: the possibility that a broadcasting packet arrives at each node within the predetermined bounded time.

b: the possibility that a node receives the arriving packet.

B: the possibility that a broadcasting packet is received by all nodes

THEOREM 7 The possibility of synchronizing clocks within a bounded time using our algorithm, O, is

$$\sum_{i=\frac{N}{2}}^{N} t^2 \ {}_NC_i \ b^{2i} \ (1-b^2)^{N-i} \ (\sum_{j=\frac{N}{2}}^{i} \ {}_NC_i \ B^j \ (1-B)^{i-j}), \ where \ B = tb^N$$

PROOF Our algorithm can synchronize clocks when $\frac{N}{2}$ or more interval messages contain valid data. The possibility that exactly i nodes receive two pointing messages, O_i, is

$$O_i = t^2 \ {}_NC_i \ b^{2i} \ (1-b^2)^{N-i}$$

Among i nodes, $\frac{N}{2}$ or more node should succeed in broadcasting interval messages.

$$O_j = \sum_{j=\frac{N}{2}}^{i} \ {}_iC_j \ b^j (1-b)^{i-j}$$

So the possibility of synchronizing clocks with our algorithm is as follows.

$$O = t^2 \ {}_NC_i \ b^{2i} \ (1-b^2)^{N-i} (\sum_{j=\frac{N}{2}}^{i} \ {}_iC_j \ b^j (1-b)^{i-j})$$

□

5 Conclusion

We have described a clock synchronization algorithm we can use in Ethernet-based networks. Here we mainly focused on the synchronization on arbitrary delay networks. By using intervals of two messages, each of which may experience arbitrary delays, our algorithm is made independent of message transmission delays. Furthermore since an Ethernet broadcasting facility is used, the algorithm requires relatively small number of messages, and the number of messages to be exchanged increases linearly with the addition of new machines. This synchronization process thus will not jeopardize the overall performance of the system. Upper bounds on the accuracy of algorithm have been derived, from the pointing error, transmission delay, the number of tolerated faults, and the duration of the synchronization interval. We also showed that clocks will not diverge above a certain bounded value, even in case of pointing message loss. Synchronized clocks can eliminate packet collisions and waiting time to sense the carrier in an Ethernet-based system, by making a node transmit a message only in the predefined time as in TDMA-implemented Ethernet [11, 12]. As transmission time then becomes predictable, real-time communication can be implemented on Ethernet.

참고 문헌

[1] *Carrier Sense Multiple Access with Collision Detection(CSMA/CD): Access Method and Physical layer Specification. ANSI/IEEE Standard 802.3–1985.* IEEE, 1985.

[2] T.K. Srikanth and Sam Toueg, "Optimal clock synchronization," *Journal of ACM*, Vol.34, No.3, pp. 626–645, Jul. 1987.

[3] R. Gusella and S. Zatti, " TEMPO-Time services for the Berkeley Local Network," *Report No. UCB/CSD 83/163*, University of California, Berkeley, Dec. 1983.

[4] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD," *Report No. UCB/CSD 87/337*, University of California, Berkeley, Jan. 1987.

[5] R. Gusella and S. Zatti, "The Berkeley UNIX 4.3 BSD time synchronization protocol: protocol specification," *Report No. UCB/CSD 85/250*, University of California, Berkeley, Jun. 1985.

[6] K. Arvind, "A new probabilistic algorithm for clock synchronization," *Proc. Real-Time Systems Synposium*, pp. 330–339, Santa Monica, CA, 1989.

[7] F. Cristian, "A probabilistic approach to distributed clock synchronization," *Proc. of the Ninth International Conference on Distributed Comuting Systems*, May. 1989.

[8] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Transactions on Computers*, Vol.C–36, No.8, Aug. 1987.

[9] L. Rodrigues, P. Verissimo and A. Casimiro, "Using atomic broadcast to implement a posteriori agreement for clock synchronization," *Proc. the 12th Symposium on Reliable Distributed Systems*, pp.115-124, 1993.

[10] L. Rodrigues and P. Verissimo, "xAMp: A multi-primitive group communications service," *Proc. the 11th Symposium on Reliable Distributed Systems*, Octobor, 1992.

[11] Hermann Kopetz and W. Merker, "The Architecture of MARS," *Proc. 15-th Fault-Tolerant Computing Symposium*, pp.274-279, June 1985.

[12] Junghoon Lee and Heonshik Shin, "A variable bandwidth allocation scheme for Ethernet-based real-time communication," *Proc. the First International Workshop on Real-Time Computing Systems and Applications*, pp.28-32, December 1994.