(**V.1.5**, 16/Nov/2022)

Study Guides on Computer Networks and Distributed Systems

Clock synchronization and NTP

Questionnaire

1. The Network Time Protocol (NTP) can be used to synchronize computer clocks. Explain why, even with this service, no bound is given for the resulting time difference between two clocks even right after synchronization.

   Computing the offset necessary to add to the requester clock (C_{Req}) which needs to be set to the responder clock (C_{Resp}) time entails determining $Delay_{Return}$ or the average of the return path delay over time. The approximation to $Delay_{Return}$ that we can compute is just $\frac{Rtt}{2}$, which, in reality can depart a lot from the actual $Delay_{Return}$ depending on the degree of asymmetry between the forward and the return paths and their respective delays, so the accuracy of this approximation can introduce a substantial error in the clock.

   Perfect synchronization would entail zero-Rtt which is impossible to achieve.

2. What transport does NTP use?

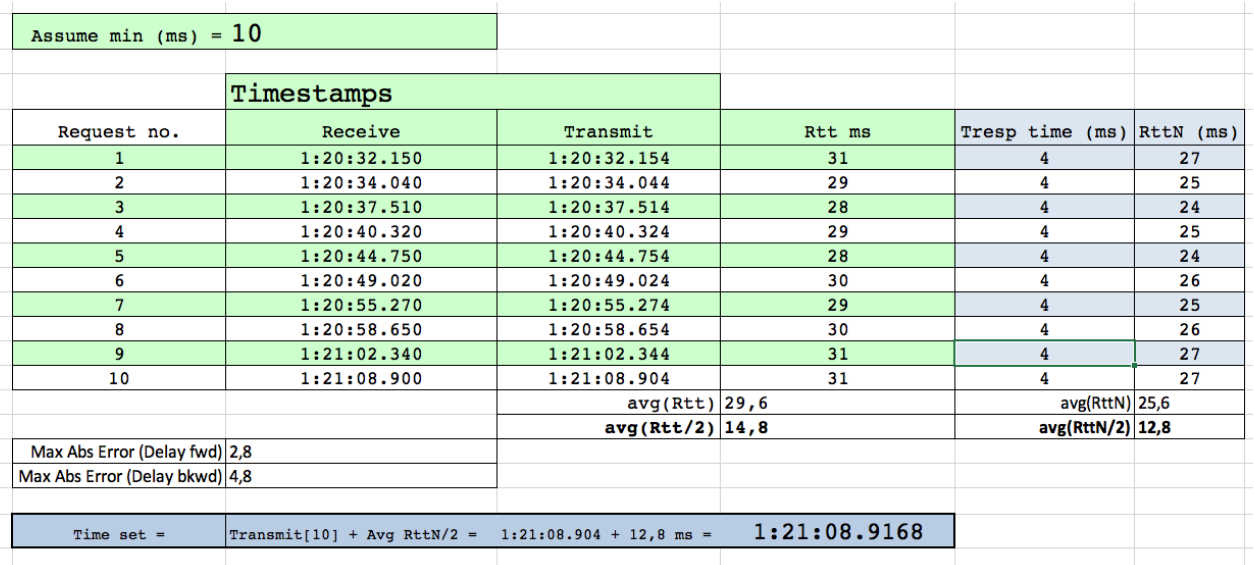
   NTP is documented in RFC 5905. The transport used by NTP is UDP

3. When synchronizing computer clocks, can a clock be set back? Explain what has to be done in order for a clock to acquire the same real time as another clock.

   Computer clocks, in general, must not be brought back, since that might cause the repetition of certain actions configured by system administrators. The correct action to take upon a clock that must be brought behind consists in reducing its speed so that in a certain period of real time it will be in sync with a remote clock (One that, for some reason, has a good reference time). By decelerating the clock, it will continue generating a sequence of growing time values, though with a reduced accuracy. This so-called *clock monotonicity* will avoid the foregoing undesirable effects altogether. POSIX function adjtime() will accelerate or decelerate the clock to gain or lose some time monotonically.

4. Solve the synchronization example included in the lecture slides

   The present example aims to illustrate the synchronization process of two hosts A and B. Host A is the time sync requester and B is the time sync responder. The requester host executes a *naïve* form of Cristian's algorithm in which it performs ten time requests to host B so it can calculate the average Rtt. The relevant timestamps are recorded in the spreadsheet below. The cells in green background represent the problem data that we are given and the gray-blue cells represent the calculations that we have to do in order to obtain the desired final result: the new time computer A must be set to so it be in sync with B's clock.



Assume min (ms) = 10					
	Timestamps				
Request no.	Receive	Transmit	Rtt ms	Tresp time (ms)	RttN (ms)
1	1:20:32.150	1:20:32.154	31	4	27
2	1:20:34.040	1:20:34.044	29	4	25
3	1:20:37.510	1:20:37.514	28	4	24
4	1:20:40.320	1:20:40.324	29	4	25
5	1:20:44.750	1:20:44.754	28	4	24
6	1:20:49.020	1:20:49.024	30	4	26
7	1:20:55.270	1:20:55.274	29	4	25
8	1:20:58.650	1:20:58.654	30	4	26
9	1:21:02.340	1:21:02.344	31	4	27
10	1:21:08.900	1:21:08.904	31	4	27
		avg(Rtt)	29,6	avg(RttN)	25,6
		avg(Rtt/2)	**14,8**	**avg(RttN/2)**	**12,8**
Max Abs Error (Delay fwd)	2,8				
Max Abs Error (Delay bkwd)	4,8				
Time set =	Transmit[10] + Avg RttN/2 =	1:21:08.904 + 12,8 ms =	1:21:08.9168		

5. Does TCP offer some facility to allow a transmitter to compute the *network Rtt* (Rtt_N) of a connection, *i.e.,* the Rtt that doesn't include the IRQ time (T_{RESP}) consumed by the destination host servicing the request received from the client.

   TCP's timestamps can only be used to calculate the overall Rtt, which includes T_{RESP}.

6. Describe the kind of operations caused by the reception of a layer-2 frame. Specifically we wish to obtain a perspective about what happens right after the microprocessor deals with the hardware interrupt which initially reports the full deserialization of a new frame.

   This exercise consists of having you review the lab practices of Computer Networks.

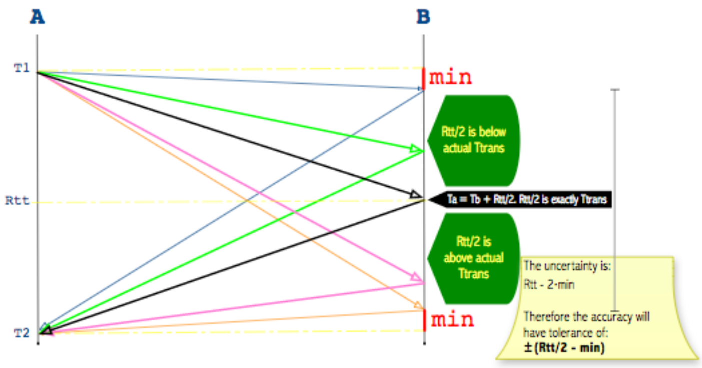
7. Explain the basic principles underlying the Christian's synchronization algorithm

8. Clock synchronization over the internet suffers from a lack of precision, can you explain the cause?

9. Tell several ways to have a host synchronized with a UTC source over the Internet

10. What is *clock drift?* What is its basic cause?

11. Give an explanation of the following diagram:



12. Study the solved exercise about Clock Synchro in paloalto.unileon.es/ds

13. Solve exercises **14.1** – **14.5** from Dollimore/Kindberg/Coulouris/etc (You can access the problem statements on the last page of the Physical Clocks ppt lecture presentation)

MORE EXERCISES ON NEXT PAGES

14. Solved exercise about conversion from the h:m:s time format to seconds and microseconds:

h:m:s conversion to μs for use in adjtime() (μs means microseconds)

$$\text{h to m} \rightarrow \text{h hour} \times \frac{60 \text{ min}}{\text{hour}}$$

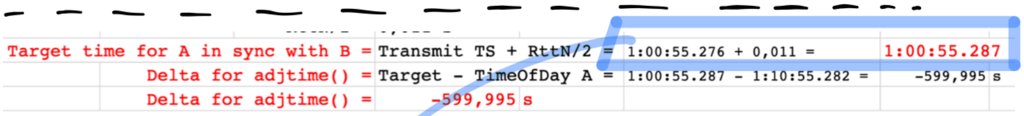
$$2 \text{ hour} = 2 \cancel{\text{hour}} \times \frac{60 \text{ min}}{\cancel{\text{hour}}} = 2 \times 60 \text{ min} = 120 \text{ min}$$

$$\text{m to s} \rightarrow \text{m min} \times \frac{60 \text{ sec}}{\text{min}}$$

$$3 \text{ min} = 3 \cancel{\text{min}} \times \frac{60 \text{ sec}}{\cancel{\text{min}}} = 3 \times 60 \text{ sec} = 180 \text{ sec}$$

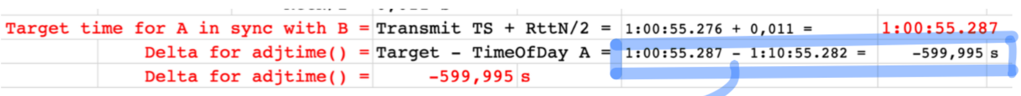
$$\text{s to } \mu\text{s (microseconds)} \rightarrow \text{s sec} \times \frac{1000000 \, \mu s}{1 \text{sec}}$$

$$18 \text{ sec} = 18 \cancel{\text{sec}} \times \frac{1000000 \, \mu s}{1 \cancel{\text{sec}}} = 18 \times 10^6 \mu s = 18000000 \, \mu s$$



Target time for A in sync with B =	Transmit TS + RttN/2 =	1:00:55.276 + 0,011 =	1:00:55.287
Delta for adjtime() =	Target - TimeOfDay A =	1:00:55.287 - 1:10:55.282 =	-599,995 s
Delta for adjtime() =	-599,995 s		

$$\underbrace{1}_{h}{:}\underbrace{00}_{m}{:}\underbrace{55}_{s},\underbrace{276000}_{\mu s} \rightarrow \begin{array}{r} 1{:}00{:}55{,}276 \\ +\quad 0{,}011 \\ \hline 1{:}00{:}55{,}287 \end{array} \text{ seconds}$$



Target time for A in sync with B =	Transmit TS + RttN/2 =	1:00:55.276 + 0,011 =	1:00:55.287
Delta for adjtime() =	Target - TimeOfDay A =	1:00:55.287 - 1:10:55.282 =	-599,995 s
Delta for adjtime() =	-599,995 s		

Subtraction

$$\begin{array}{rl} & 1{:}00{:}55{,}287 \ (M) \\ - & 1{:}10{:}55{,}282 \ (S) \\ \hline \end{array}$$

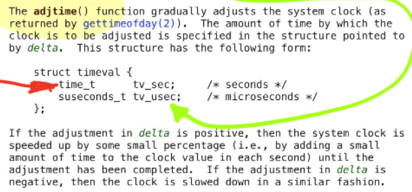
Since S>M we compute S-M and change the sign of the result

$$\begin{array}{r} 1:10:55,282 \\ -\ 1:00:55,287 \\ \hline 0:10:00,095 \end{array}$$

→ Result is $-0:10:00,095$ sec.

The actual argument sent on to adjtime() must be expressed as seconds and µs (microseconds), in the present case we have:

$$-0:\underline{10}:00,095\text{ sec} = -\Big(\underbrace{0\times 60\times 60+\underline{10}\times 60+00}_{\text{seconds}} + \underbrace{0,095000}_{\mu s}\Big)$$



The **adjtime**() function gradually adjusts the system clock (as returned by gettimeofday(2)). The amount of time by which the clock is to be adjusted is specified in the structure pointed to by *delta*. This structure has the following form:

```
struct timeval {
    time_t      tv_sec;     /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

If the adjustment in *delta* is positive, then the system clock is speeded up by some small percentage (i.e., by adding a small amount of time to the clock value in each second) until the adjustment has been completed. If the adjustment in *delta* is negative, then the clock is slowed down in a similar fashion.

NOTES ABOUT LOGICAL CLOCKS (On next page)

LAMPORT'S CLOCKS (Scalar logical clocks)

DS 29th. OCT. 2022

- LC = Logical clock
- Time is the set $\mathbb{N}$ represented by variable C_i
- C_i variable represents the LC of process P_i
- events happen in processes ($e_i, e_j, \ldots$)
- $e_i \to e_j$: The "$\to$" relation means "happened before". The timestamp of e_i is smaller than the timestamp of e_j.
- In logical clocks this property must be true:

  if $e_i \to e_j$, then $C(e_i) < C(e_j)$

  (consistency, monotonicity property)
- event e_i:
  - Send message
  - Receive message
  - Internal

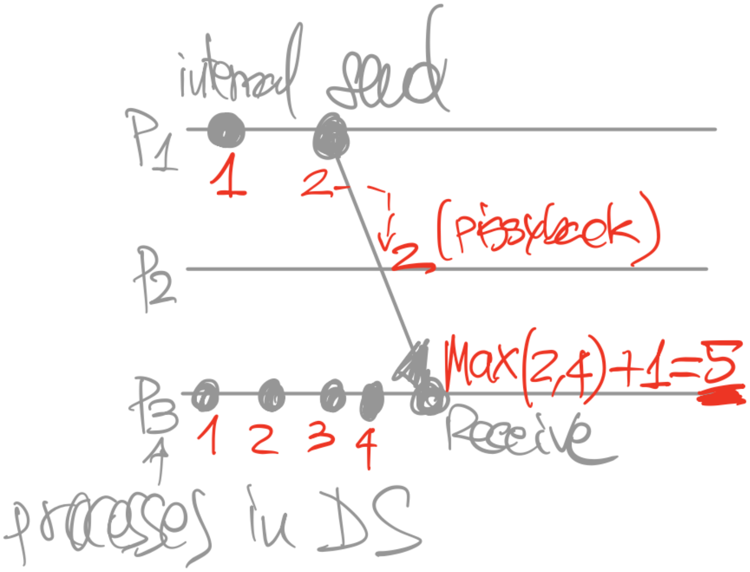
LC Predicates

① Before executing event e:

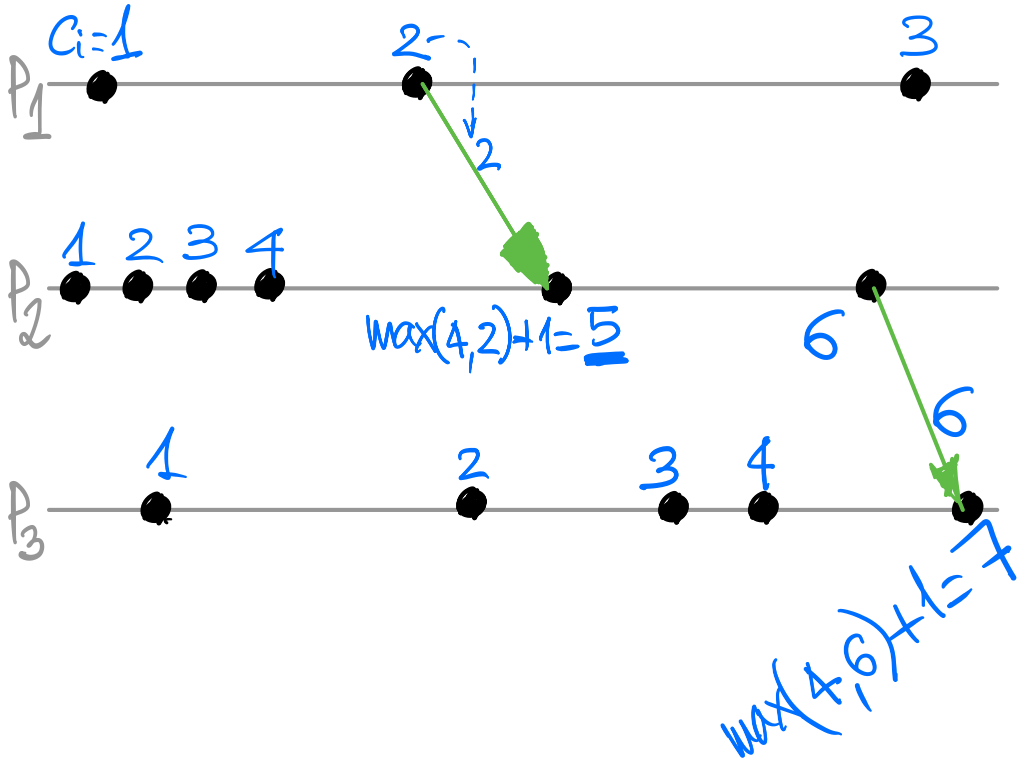
$$\underbrace{C_i}_{\text{timestamp of process } P_i} = C_i + d \quad (d > 0, \text{ typically } 1)$$

② . $C_i = \max(\underset{\text{Receiver}}{C_i}, \underset{C_i \text{ of sender}}{C_{msg}})$

- Do ①
- Deliver message



Example of DS comprised of three processes each of which executes a number of events:



NOTES ON VECTOR CLOCKS ON NEXT PAGE

(**V.1.5**, 16/Nov/2022)

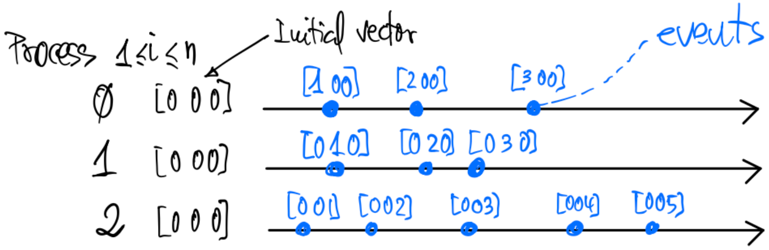
DS 10·NOV·22 VECTOR CLOCKS José María Foces Morán

- The two LC values of two events cannot reveal whether they are causally ordered: $LC(A) < LC(B) \not\Rightarrow A \prec B$
- Vector clocks do detect causality

not necessarily a vector space

called vectors

- DEFINITION:
  - n processes · events V · A n-tuple of $\mathbb{N}$ $(\mathbb{Z}_+ \cup \{0\})$
  - Vector clock $VC: V \to A$, a mapping from V on A.
  - For example, assume n processes n=3, each vector has three components, one for each n: 1..3. Example VC: [10, 5, 7]



- PARTIAL ORDER DEFINITION: $VC(a) < VC(b)$ iff:
  - 1. $\forall i,\ 0 \le i \le n-1,\ VC_i(a) \le VC_i(b)$

    event a

    ith component of VC(a)
  - 2. $\exists j,\ 0 \le j \le n-1,\ VC_j(a) < VC_j(b)$

Examples: $[7,8,2] \overset{?}{<} [7,8,1]$ Ⓐ

- 1. $7 \le 7$ is true · $8 \le 8$ true; $2 \le 1$ false $\Rightarrow$ Ⓐ is false
  no need to check rule 2. above

$[10\ 8\ 11] \overset{?}{<} [10\ 9\ 11]$ Ⓑ

- 1. $10 \le 10$ ✓; $8 \le 9$ ✓; $11 \le 11$ ✓ $\Big\}$ $\Rightarrow$ Ⓑ is true
- 2. $8 < 9$

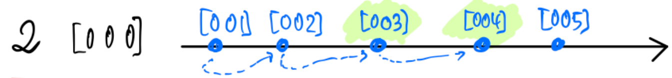
IMPLEMENTATION OF VECTOR CLOCKS

- We seek $\boxed{a \prec b \iff VC(a) < VC(b)}$!

$$\boxed{\begin{array}{l} \text{if } a \prec b \Rightarrow VC(a) < VC(b) \\ \qquad\qquad \text{and} \\ \text{if } VC(a) < VC(b) \Rightarrow a \prec b \end{array}}$$

- if a and b are causally ordered then the $VC(a) < VC(b)$
  and
- if $VC(a) < VC(b)$ we can claim that they are causally ordered

- n processes, $0 \leq i \leq n-1$ $VC[0] = 0$
- Local, internal events: within a specific process. Example:

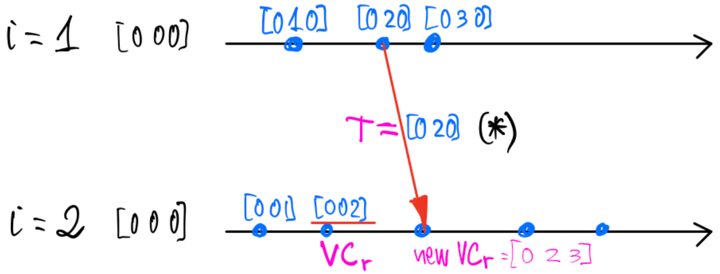


$$\boxed{VC_i[i] = VC_i[i] + 1}$$

$[0\ 0\ 3]$; $VC_2[2] + 1 = 3 + 1 = 4$: $[0\ 0\ 4]$

$i = 2$

- Send events: Piggyback the current vector (*)



- Receive events: Create a new event and compute its VC by using T and the current VC of the receiving process VC_r

$VC_r = [0\ 0\ 2]$ Since $i = 2$ we fetch component 2 which value is $\underline{\underline{2}}$

component 0 1 2

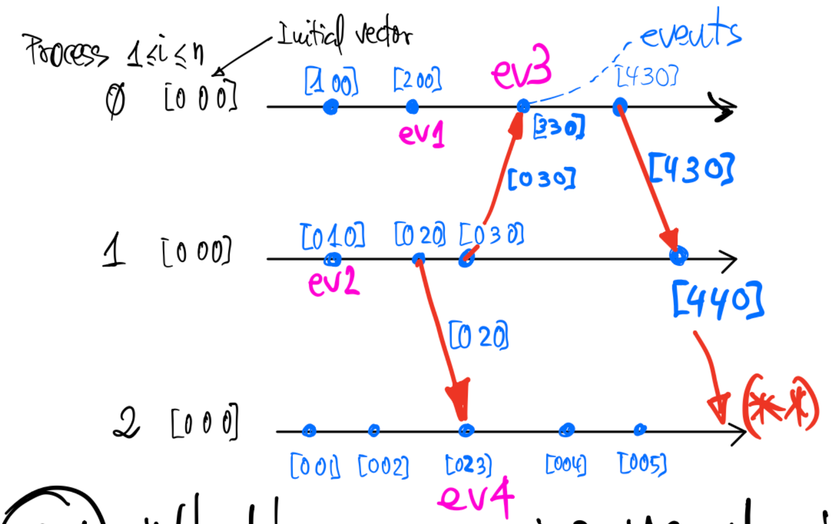
① $VC_r[r] = VC_r[r] + 1$ in this case $VC_2[2] = VC[2]+1 = \underline{\underline{2}} + 1 = \underline{\underline{3}}$ new $VC_2[2]$

② $\forall r$: $0 \leq r \leq n-1$ :: $VC_r[i] := \max\left(T_r, VC_r[i]\right)$ in this case:

$[0\ 2\ 0]$, $[0\ 0\ 3]$

$$\left.\begin{array}{l}\max(0,0)=0\\ \max(2,0)=2\\ \max(0,3)=3\end{array}\right\} \to \text{new } VC_r = [0\ 2\ 3]$$

EXAMPLE AND QUESTIONS



Q1 What's process i=2 VC at point (**)?

Q2 Are events ev1 and ev2 causally ordered? Otherwise, are ev1 and ev2 concurrent?

Q3 Are events ev1 and ev3 causally ordered?

Q4 Are events ev1 and ev4 concurrent?