

Clock Synchronisation

Physical Clocks

Marco Aiello, Ilche Georgievski

University of Groningen

DS, 2014/2015

- 1 Introduction
- 2 Computer clocks
 - Clock drift and clock skew
- 3 Synchronisation
 - Cristian's algorithm
 - Berkeley algorithm
 - Network Time Protocol
 - Precision Time Protocol

Why do we need clocks?

- temporal ordering of events produced by concurrent processes
- synchronisation between senders and receivers of messages
- serialisation of concurrent access to shared objects
- *to know the time an event occurred at a computer*
- *no global clock in a distributed system*

Logical vs. physical clocks

- Logical clock ➞ keeps track of ordering of events
- Physical clocks ➞ keep and coordinate time of day

Physical clocks

1927 Quartz clocks

- crystal oscillator
- accuracy:
 - ◆ standard oscillator - 6 parts per million at 31°C (< 0.29 sec/day)
 - ◆ good oscillator - about one second in 10 years

1955 Atomic clocks

- the most accurate time and frequency standards
- a *second* = duration of 9 192 631 770 cycles of radiation = transition between two energy levels of the caesium-133 atom
- accuracy:
 - ◆ 1 part in 10^{14} , i.e., < 1 second in 6 million years
- use:
 - ◆ primary standards for international time distribution services
 - ◆ control the wave frequency of television broadcasts
 - ◆ global navigation satellite systems
- International Atomic Time (TAI)

Time standard

1961 Coordinated Universal Time (UTC)

- derived from TAI, but adjusted to UT1 by adding a *leap second*
 - ◆ including 2012, a total of 25 leap seconds added (30 June 2012)

☺ Check *The One-second War (What Time Will You Die?)*

- broadcast
 - ◆ land-based signals are accurate to about 0.1-10 milliseconds
 - ◆ satellite-based signals are accurate to about one microsecond
- **UT0** - obtained from astronomical observations
- **UT1** - UT0 corrected for polar motion, i.e., the actual Earth's rotation with respect to the solar time
- **UT2** - UT1 corrected for seasonal variations in Earth's rotation

Article development led by [acmqueue](http://acmqueue.queue.acm.org)
queue.acm.org

Finding a lasting solution to the leap seconds problem has become increasingly urgent.

BY POUL-HENNING KAMP

The One-Second War

THANKS TO A secretive conspiracy working mostly below the public radar, your time of death may be a minute later than presently expected. But don't expect to live any longer, unless you happen to be responsible for time synchronization in a large network of computers, in which case this coup will lower your stress level a bit every other year or so.

We're talking about the abolishment of leap seconds, a crude hack added 40 years ago to paper over the fact that planets make lousy clocks compared with quantum mechanical phenomena.

Timekeeping used to be astronomers' work, and the trouble it caused was very academic. To the rural

population, sunrise, midday, and sunset were plenty precise for all relevant purposes.

Timekeeping became a problem for non-astronomers only when ships started to navigate where they could not see land. Finding your latitude is easy: measure the height of the midday sun over the horizon, look at the table in your almanac, done. Finding your longitude is possible only if you know the time of day precisely, and the sun will not tell you that unless you know your longitude.

If you know your longitude, however, the sun will tell you the time very precisely. Using that time, you can make tables of other nonsolar astronomical events—for example, the transits of the



Physical & software clocks

- Physical clock
 - ◆ CMOS clock circuit that counts oscillations of a quartz
 - ◆ after a specified number of oscillations, the clock increments a register, thereby adding one *clock-tick* to a counter that represents the passing of time: $H_i(t)$
 - ◆ battery backup to continue measuring when computer power is off
- Software clock - to timestamp events

$$C_i(t) = \alpha H_i(t) + \beta$$

- $C_i(t)$ approximates the real physical time t at process p_i
- e.g., a 64-bit number giving nanoseconds since some base time

Clock resolution

- Period between the updates of the clock value.
- Successive events can be distinguished if the clock resolution is smaller than the time interval between the two events.

Physical clocks in distributed systems

- local processes obtain the value of the current time
- processes on different computers can timestamp their events
- but clocks on different computers may give different times
- difficult to synchronise: even if clocks on all computers in a distributed system are set to the same time, their clocks will eventually vary quite significantly unless corrections are applied

Problem

Computer clocks hardly ever agree on time!

Clock drift

- Clocks tick at different rates
- **Clock drift rate** is the relative amount that a computer clock differs from a perfect clock
- Recall:
 - ✌ Typical quartz clocks *drift rate* is about 10^{-6} secs/sec
 - ✌ High-precision quartz clocks *drift rate* is about 10^{-8} or 10^{-9} secs/sec

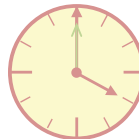
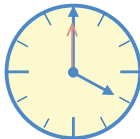
What happens to clocks when their batteries become very low?

Clock skew

- **Clock skew** defines the difference between the times on two clocks: $|C_i(t) - C_j(t)|$
- Reasons:
 - ◆ clock drift
 - ◆ the clocks may have been set differently on different machines

Example

1 Sep 04:00:00



3 Oct 04:00:00

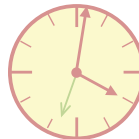


04:01:16

skew = +76 secs

+76 secs/32 days

drift = 2,38 sec/day



04:01:34

skew = +94 secs

+94 secs/32 days

drift = 2,94 sec/day

Clock correctness

- H_i is said to be **correct** if its drift rate is within a bound $\rho > 0$ (e.g., 10^{-6} secs/sec)
- The error in measuring the interval between real times t and t' is bounded:
 - $(1 - \rho)(t' - t) \leq H_i(t') - H_i(t) \leq (1 + \rho)(t' - t)$ (where $t' > t$)
 - forbids jumps in time readings of physical clocks
- Weaker condition of *monotonicity*
 - $t' > t \Rightarrow C(t') > C(t)$
 - e.g., required by UNIX *make* operation
 - can achieve monotonicity with a physical clock that runs fast by adjusting the values of α and β in $C_i(t) = \alpha H_i(t) + \beta$
- A **faulty** clock is one that does not obey its correctness condition
 - * *crash* failure - a clock stops ticking
 - * *arbitrary* failure - any other failure e.g., jumps in time

Consider the ‘Y2K bug’ - what sort of clock failure would that be?


Get accurate time

- Put a GPS receiver to each computer
 - ▢ ≈ 1 microsecond of UTC



Cost, power, convenience, environment.

Get accurate time

- ~~Put a GPS receiver to each computer~~
 ~~≈ 1 microsecond of UTC~~



Cost, power, convenience, environment.

Get accurate time

- ~~Put a GPS receiver to each computer~~
~~▢▢▢▢▢ ≈ 1 microsecond of UTC~~
- Put a radio receiver
▢▢▢▢ ≈ 0.1 -10 milliseconds of UTC

Cost, power, convenience, environment.

Get accurate time

- Put a GPS receiver to each computer
▢ ≈ 1 microsecond of UTC
- Put a radio receiver
▢ ≈ 0.1 -10 milliseconds of UTC

Cost, power, convenience, environment.

Get accurate time

- Put a GPS receiver to each computer
▢ ≈ 1 microsecond of UTC
- Put a radio receiver
▢ ≈ 0.1 -10 milliseconds of UTC

Cost, power, convenience, environment.

Get accurate time

Synchronise with another machine, one with more accurate clock!

Synchronising computer clocks

External synchronisation

Process's clock is synchronised with an external authoritative time source S

- $|S(t) - C_i(t)| < \delta, \delta > 0,$
 $i = 1, 2, \dots, N, t \in I, I$ is an interval of real times
- The clock C_i is **accurate** to within the bound δ

Internal synchronisation

Processes' clocks are synchronised with one another

- $|C_i(t) - C_j(t)| < \delta, \delta > 0,$
 $i = 1, 2, \dots, N, t \in I, I$ is an interval of real times, and C_i and C_j are clocks at processes p_i and p_j
- The clocks C_i and C_j **agree** within the bound δ

- Are internally synchronised clocks also externally synchronised?
- If a set of processes P is synchronised externally within a bound δ , is then P also internally synchronised? If yes, what is the bound?

Synchronous distributed system

A distributed system is *synchronous*, if:

- 1 the time to execute each step of a process has known lower and upper bounds
- 2 each process has a local clock whose drift rate from the real time has a known bound (ρ)
- 3 each message transmitted over a channel is received within a known bounded time (min, max)

Synchronisation

- Server S sends its local time T_{server} to a client C in a message m
- C could set its clock to $T_{server} + T_{trans}$, where T_{trans} is the time to transmit m
- T_{trans} is unknown, but $min \leq T_{trans} \leq max$
- Let u be the uncertainty in the message transmission time, so that $u = max - min$. Then, client sets the clock to $T_{server} + (max + min)/2$, so that $skew \leq u/2$

A synchronous system

Internet

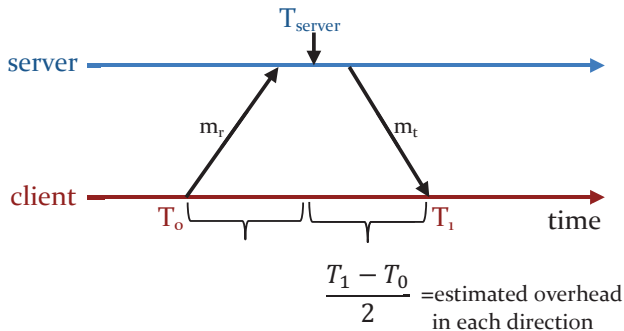
We can only say $T_{trans} = min + x$, where $x \geq 0$.

- Cristian's algorithm
- Berkeley algorithm
- Network Time Protocol
- Precision Time Protocol

Cristian's algorithm (1989)

Steps

- ① A time server receives signals from a UTC source
- ② Client requests time in m_r and receives T_{server} in m_t from the server
- ③ The client sets its clock to $T_{new} = T_{server} + \frac{T_{round}}{2}$

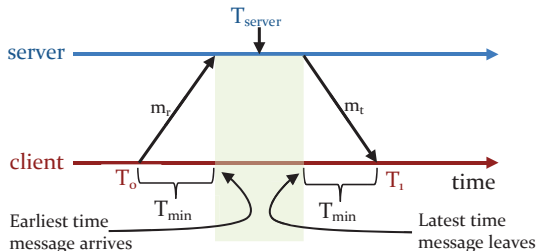


T_{round} is the round-trip time recorded by the client, i.e., $T_{round} = T_1 - T_0$

Accuracy

Let T_{min} be the minimum message delay (transition time). Then,

- T_{min} is the earliest time the server puts T_{server} in message m_t after the client sent m_r , and
- T_{min} is the latest time before m_t arrived at the client, and
- $[T_{server} + T_{min}, T_{server} + T_{round} - T_{min}]$ is the range of the time by the server's clock when m_t arrives.

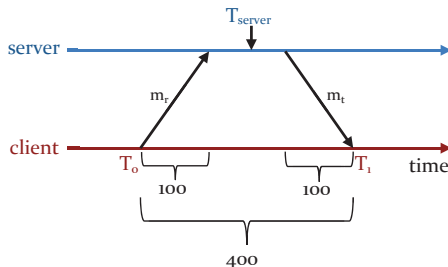


$$\text{Accuracy} = \pm(T_{round}/2 - T_{min})$$

What is the potential problem in using a single time server?

Example

- 1 request sent at (T_0): 04:01:16.200
- 2 response received at (T_1): 04:01:16.600
 - ▢▢▢▢▢ response contains time (T_{server}): 04:02:30.200
- round time: $T_1 - T_0$
 - ▢▢▢▢▢ $04:01:16.600 - 04:01:16.200 = 400$ msecs
- best guess: the timestamp was generated 200 msecs ago
- set time (T_{new}): $04:02:30.200 + 200 = 04:02:30.400$



$$accuracy = \pm \frac{600 - 200}{2} - 100 = \pm 200 - 100 = \pm 100$$

Berkeley algorithm (1989)

About

- Internal synchronisation of a group of computers
- Assumes no machine has an accurate time source
- Each machine runs **time daemon**
- One machine represents a server (**master**), others are **slaves**

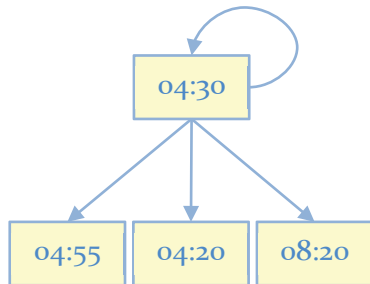
Steps

- ① A master polls to collect clock values from slaves
- ② Machines response with their clock values
- ③ Once responses are gathered, the master computes **an average**
 - the average includes master's own time
 - the average is fault-tolerant – it ignores faulty clocks
 - **an average eliminates the tendencies of machine's clocks to run slow or fast**
- ④ Master sends the required adjustment (offset) to the slaves
 - ▮ better than sending the time. Why?

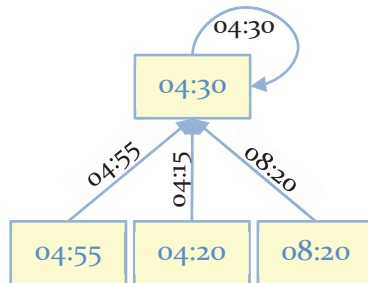
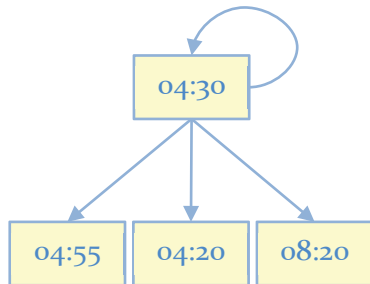
Observation:

- 15 computers, clock synchronisation 20-25 milliseconds, drift rate $< 2 \times 10^{-5}$

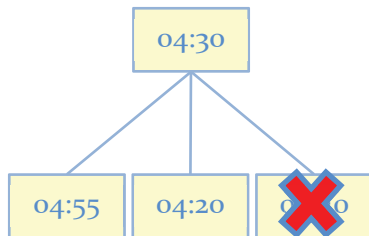
Example



Example

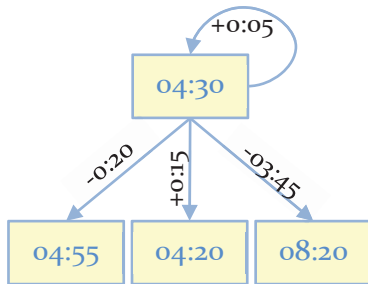
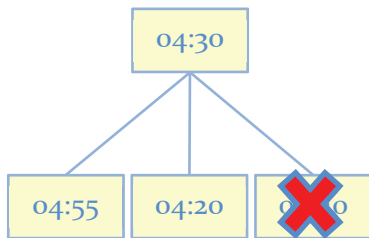


Example



$$\text{fault-tolerant average} = \frac{04:30 + 04:55 + 04:20}{3} = 04 : 35$$

Example



$$\text{fault-tolerant average} = \frac{04:30 + 04:55 + 04:20}{3} = 04 : 35$$

Network Time Protocol

About

A time service that played a large role in time synchronisation by keeping networked computer clocks synchronised to within *milliseconds* of each other.

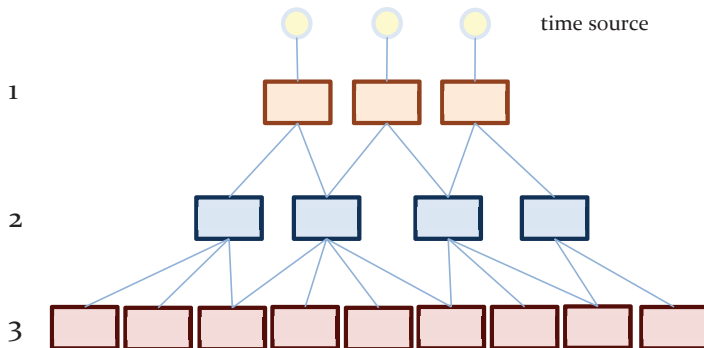
- 1988 (v1): RFC 1059, clock filter, selection and discipline algorithms, client/server and symmetric modes
- 1989 (v2): RFC 1119, formal model, pseudo-code, Control Message Protocol, cryptographic authentication,
- 1992 (v3): RFC 1305, formal error analysis, timekeeping quality, broadcast mode, reference clock drivers
- 2010 (v4): RFC 5905-5908, IPv4, IPv6 and OSI support, improved accuracy to tens of *microseconds*, dynamic server discovery, multicast mode

Design goals

- Enable clients across the Internet to be synchronised accurately to UTC
- Provide a reliable service that can handle extensive losses of connectivity
- Enable clients to resynchronise frequently enough
- Protect against interference with the time service

Synchronisation subnet

- Stratum 1: primary servers connected directly to UTC
- Stratum 2: secondary servers synchronised to primary servers
- ...



Synchronisation modes

- **Multicast** mode

- ▢ a server within a high-speed LAN multicasts time
- ▢ not very accurate, but efficient

- **Procedure-call** mode

- ▢ similar to Cristian's algorithm
- ▢ higher accuracy, useful if no hardware multicast

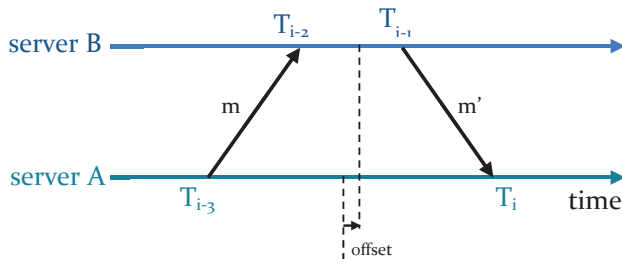
- **Symmetric** mode

- ▢ pairs of servers exchange messages containing time information
- ▢ very high accuracy

In all modes, *UDP* is used!

Synchronisation strategy (message exchange)

- Each message bears timestamps of recent events:
 - T_{i-3} : local time when the previous message was sent
 - T_{i-2} : local time when the previous message was received
 - T_{i-1} : local time when the current message was sent
- Recipient notes the time of receiving the message, T_i
- In the symmetric mode, there can be a non-negligible delay between messages



Accuracy

- For each pair of messages between two servers, NTP calculates an *estimated offset* Θ_i and a *delay* δ_i .

Let two servers exchange a pair of messages, and let Θ be the *actual offset* between the servers' clocks and t, t' the transmission times of the messages. Then,

$$T_{i-2} = T_{i-3} + t + \Theta \text{ and } T_i = T_{i-1} + t' - \Theta.$$

The *delay* equals the total transmission time

$$\delta_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}.$$

The *estimated offset* is

$$\Theta_i = \frac{(T_{i-2} - T_{i-3}) + (T_i - T_{i-1})}{2}.$$

The *actual offset* is $\Theta = \Theta_i + \frac{(t' - t)}{2}$.

Since $t > 0$ and $t' > 0$, the value of $t' - t$ must be between δ_i and $-\delta_i$. Therefore,

$$\Theta_i - \frac{\delta_i}{2} \leq \Theta \leq \Theta_i + \frac{\delta_i}{2},$$

So, Θ_i is an estimate of the offset and δ_i is a measure of the accuracy.

Accuracy

- NTP servers filter pairs $\langle \Theta_i, \delta_i \rangle$ by estimating reliability from variation, which allows them to select peers. The eight most recent pairs are saved. It selects the Θ_i with the smallest δ_i (the smaller delay, the better accuracy).
- Accuracy of tens of milliseconds over Internet paths and 1 millisecond on LANs (v3)

Precision Time Protocol

About

A packet-based synchronisation mechanism able to synchronise LAN-networked computer clocks within tens of *nanoseconds* of each other.

- 2002 (v1): local networks
- 2008 (v2): improved accuracy, precision and performance

Design goals

- Provide sub-microsecond synchronisation of real-time clocks in distributed (measurement and control) systems
- Applicable to LANs supporting multicast communication
- Provide a simple, administration-free installation
- Support heterogeneous systems of clocks
- Impose minimal resource requirements on networks and hosts

Approach

- Master-slave hierarchy
- Master
 - time reference for one or more slaves
 - selected by Best Master Clock (BMC) algorithm
- Devices: ordinary clock, boundary clock, transparent clocks, *etc.*

Devices

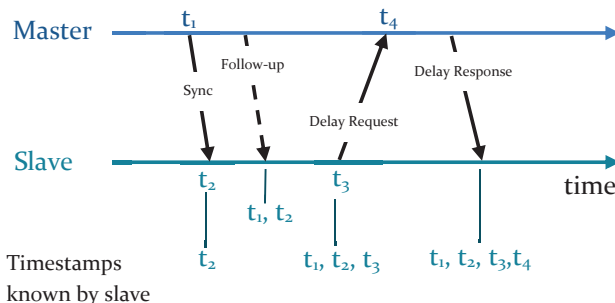
- Ordinary clock
 - a device with a single network connection
 - either a source or destination for a synchronisation reference
- Boundary clock
 - a device with multiple network connections
 - accurately bridges synchronisation from one network segment to another – distributes a master clock to different parts of the network
 - contains a timekeeper and multiple ports
- Transparent clock
 - a device that connects a group of devices without segmenting the PTP network
 - exposes its slave devices to the PTP master
 - improves distribution accuracy

Configurations and synchronisation modes

- *Software-only* configuration – ordinary clocks
- *Hardware timestamping* configuration – ordinary clocks, boundary clocks, transparent clocks
- **End-to-end** mode – a slave issues a *Delay Request* message and a master responds with a *Delay Response*
- **Peer-to-peer** mode – a device issues a *Peer Delay Request* message to an immediate neighbour (not necessarily master) which responds with *Peer Delay Response*
 - ▢▢▢▢➡ better performance when network traffic
 - ▢▢▢▢➡ better accuracy

Software-only configuration in end-to-end mode

- Master: periodically transmits a *Sync* message by UDP multicast
 - t_1 : time when the *Sync* message is sent by the master
 - t_2 : time when the *Sync* message is received by the slave
 - t_1 : actual time when the *Sync* left the master contained in the *Follow-up* message
 - t_3 : time when the *Delay Request* message is sent by the slave
 - t_4 : time when the *Delay Request* message is received by the master
- Master: sends a *Delay Response* to the slave containing t_4



NTP vs. PTP

<i>Property</i>	NTP	PTP
<i>Network</i>	WAN (LAN)	LAN (WAN)
<i>Network topology</i>	Well-defined	Not necessarily well-defined
<i>Security</i>	Good	Low
<i>Special hardware</i>	No	Boundary clocks, transparent switches
<i>Server discovery</i>	Multicast (LAN)	Unicast (WAN)
<i>Network reorganisation</i>	Semi-autonomous	Autonomous
<i>Synchronisation methodology</i>	clock offset, message delays, servo algorithm to adjust a device's timekeeper	clock offset, message delays, no servo algorithm
<i>Timescale</i>	UTC	TAI and UTC offset
<i>Performance (accuracy)</i>	Milliseconds (WAN)	Nanoseconds (LAN, hardware), sub-milliseconds (LAN, software-only), milliseconds (WAN)

Summary

- Clocks on different systems will always tick differently
- Accurate timekeeping is important for distributed systems
- Algorithms synchronise clocks in spite of their drift and the variability of message delays
- Timestamped messages, estimated delay of message transmission, estimated offset between different clocks, synchronised to UTC or to a local source
- For ordering of an arbitrary pair of events at different computers, clock synchronisation is not always practical

References

- Coulouris, J. Dollimore, and T. Kindberg, Distributed Systems: Concepts and Design (5th Edition), Chapter 14, Addison-Wesley Longman Publishing Co., Inc., 2012
- F. Cristian, Probabilistic clock synchronization, *Distributed Computing*, 3:146-158, Springer-Verlag, 1989
- R. Gusella and S. Zatti, The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD, *IEEE Transactions on Software Engineering*, vol. 15(7), 1989
- D. L. Mills, A Brief History of NTP Time: Memoirs of an Internet Timekeeper, *SIGCOMM Comput. Commun. Rev.*, vol. 33(2), 2003
- R. Ratzel and R. Greenstreet, Toward Higher Precision, *Commun. ACM*, vol. 55(10), 2012
- ☺ P. Kamp, The One-second War (What Time Will You Die?), *Queue*, vol. 9(4), 2011