Time Synchronization

Chapter 9

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Rating

- Area maturity

- Practical importance

- Theoretical importance

Overview

- Motivation
- Clock Sources
- Reference-Broadcast Synchronization (RBS)
- Time-sync Protocol for Sensor Networks (TPSN)
- Gradient Clock Synchronization

Motivation

- Time synchronization is essential for many applications
  - Coordination of wake-up and sleeping times (energy efficiency)
  - TDMA schedules
  - Ordering of collected sensor data/events
  - Co-operation of multiple sensor nodes
  - Estimation of position information (e.g. shooter detection)

- Goals of clock synchronization
  - Compensate *offset* between clocks
  - Compensate *drift* between clocks

Properties of Synchronization Algorithms

- External versus internal synchronization
  - External sync: Nodes synchronize with an external clock source (UTC)
  - Internal sync: Nodes synchronize to a common time
    - to a leader, to an averaged time, or to anything else
- Instant versus periodic synchronization
  - Periodic synchronization required to compensate clock drift
- A-priori versus a-posteriori
  - A-posteriori clock synchronization triggered by an event
- Local versus global synchronization

Clock Sources

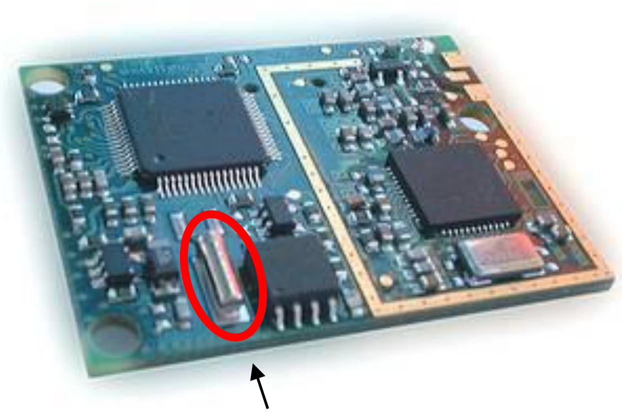
- Radio Clock Signal:
  - Clock signal from a reference source (atomic clock) is transmitted over a longwave radio signal
  - DCF77 station near Frankfurt, Germany transmits at 77.5 kHz with a transmission range of up to 2000 km
  - Accuracy limited by the distance to the sender, Frankfurt-Zurich is about 1ms.
  - Special antenna/receiver hardware required

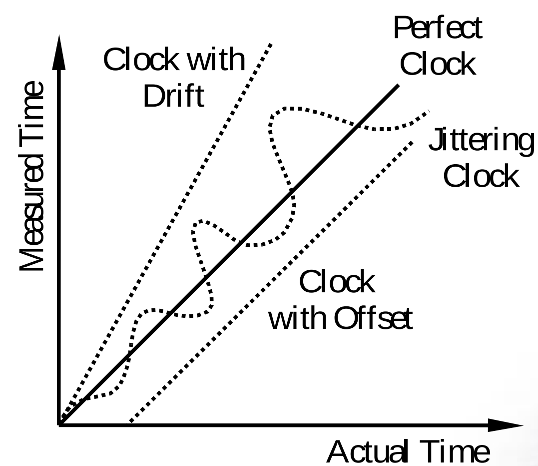
- Global Positioning System (GPS):
  - Satellites continuously transmit own position and time code
  - Line of sight between satellite and receiver required
  - Special antenna/receiver hardware required

Clock Devices in Sensor Nodes

- Structure
  - External oscillator with a nominal frequency (e.g. 32 kHz)
  - Counter register which is incremented with oscillator pulses
  - Works also when CPU is in sleep state
- Accuracy
  - Clock drift: random deviation from the nominal rate dependent on power supply, temperature, etc.
  - E.g. TinyNodes have a maximum drift of 30-50 ppm at room temperature

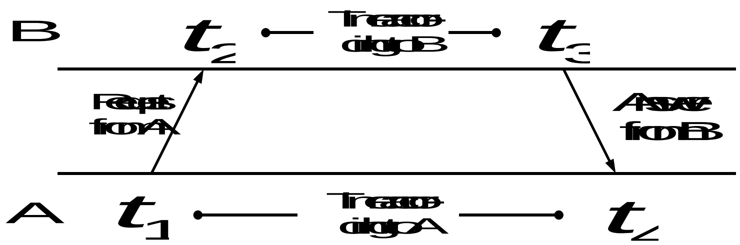
This is a drift of up to 50 μs per second or 0.18s per hour





Sender/Receiver Synchronization

- Round-Trip Time (RTT) based synchronization



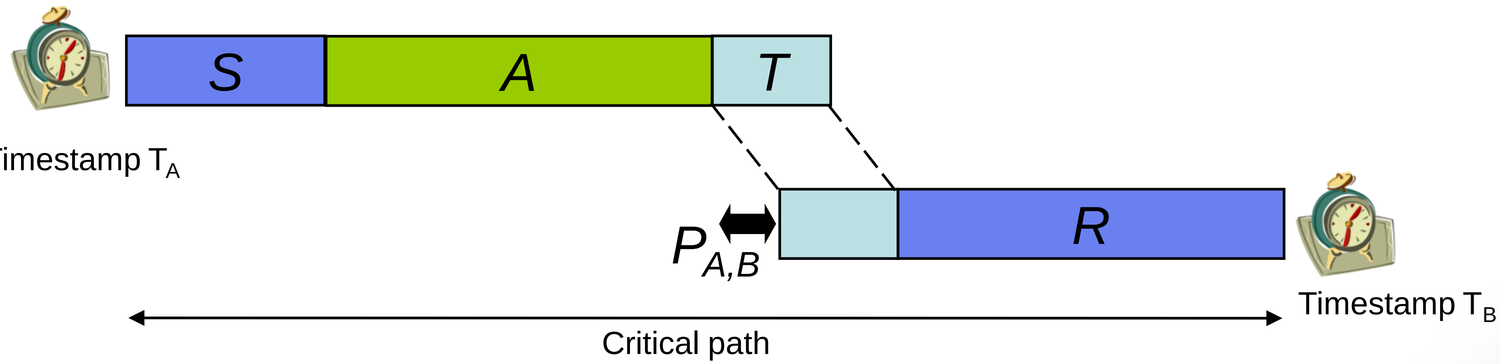
- Receiver synchronizes to the sender‘s clock
- Propagation delay δ and clock offset θ can be calculated

$$\delta = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$$

$$\theta = \frac{(t_2 - (t_1 + \delta)) - (t_4 - (t_3 + \delta))}{2} = \frac{(t_2 - t_1) + (t_3 - t_4)}{2}$$

Disturbing Influences on Packet Latency

- Influences
  - Sending Time *S* (up to 100ms)
  - Medium Access Time *A* (up to 500ms)
  - Transmission Time *T* (tens of milliseconds, depending on size)
  - Propagation Time $P_{A,B}$ (microseconds, depending on distance)
  - Reception Time *R* (up to 100ms)



- Asymmetric packet delays due to *non-determinism*
- Solution: timestamp packets at MAC Layer

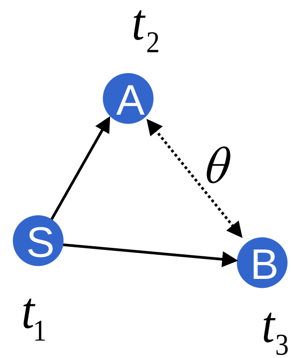
Reference-Broadcast Synchronization (RBS)

- A sender synchronizes a set of receivers with one another
- Point of reference: beacon's arrival time

$$t_2 = t_1 + S_S + A_S + P_{SA} + R_A$$

$$t_3 = t_1 + S_S + A_S + P_{SB} + R_B$$

$$\theta = t_2 - t_3 = (P_{SA} - P_{SB}) + (R_A - R_B)$$

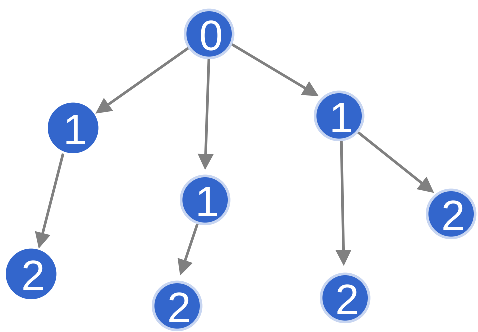


- Only sensitive to the difference in propagation and reception time
- Time stamping at the interrupt time when a beacon is received
- After a beacon is sent, all receivers exchange their reception times to calculate their clock offset

- Post-synchronization possible
- Least-square linear regression to tackle clock drifts

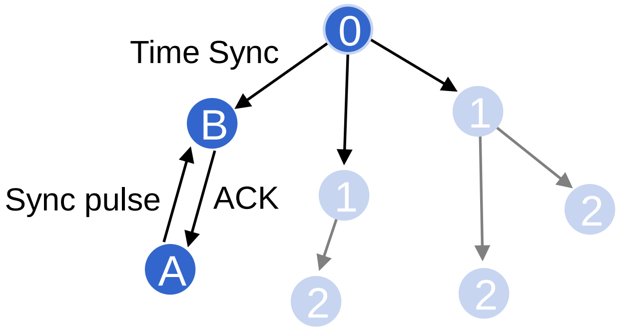
Time-sync Protocol for Sensor Networks (TPSN)

- Traditional sender-receiver synchronization (RTT-based)
- *Initialization phase: Breadth-first-search flooding*
  - Root node at level 0 sends out a *level discovery* packet
  - Receiving nodes which have not yet an assigned level set their level to +1 and start a random timer
  - After the timer is expired, a new level discovery packet will be sent
  - When a new node is deployed, it sends out a *level request* packet after a random timeout



Time-sync Protocol for Sensor Networks (TPSN)

- *Synchronization phase*
  - Root node issues a *time sync* packet which triggers a random timer at all level 1 nodes
  - After the timer is expired, the node asks its parent for synchronization using a *synchronization pulse*
  - The parent node answers with an *acknowledgement*
  - Thus, the requesting node knows the round trip time and can calculate its clock offset
  - Child nodes receiving a synchronization pulse also start a random timer themselves to trigger their own synchronization

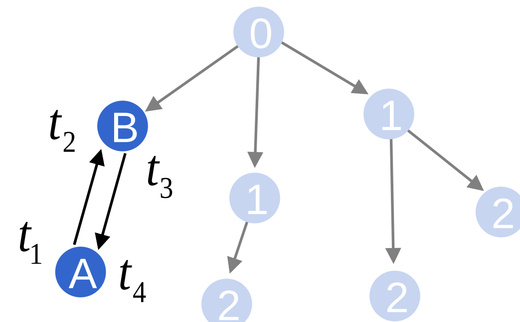


Time-sync Protocol for Sensor Networks (TPSN)

$$t_2 = t_1 + S_A + A_A + P_{A,B} + R_B$$

$$t_4 = t_3 + S_B + A_B + P_{B,A} + R_A$$

$$\Theta = \frac{(S_A - S_B) + (A_A - A_B) + (P_{A,B} - P_{B,A}) + (R_B - R_A)}{2}$$



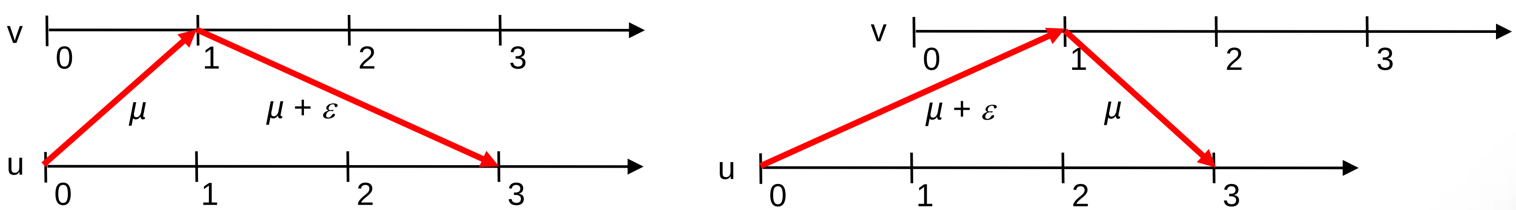
- Time stamping packets at the MAC layer
- In contrast to RBS, the signal propagation time might be negligible
- Authors claim that it is “about two times” better than RBS
- Again, clock drifts are taken into account using periodical synchronization messages

- Problem: What happens in a non-tree topology (e.g. ring)?!?
  - Two neighbors may have exceptionally bad synchronization

Theoretical Bounds for Clock Synchronization

- Network Model:
  - Each node *i* has a local clock $L_i(t)$
  - Network with *n* nodes, diameter *D*.
  - Reliable point-to-point communication with minimal delay μ
  - Jitter ε is the uncertainty in message delay

- Two neighboring nodes u, v cannot distinguish whether message is faster from u to v and slower from v to u, or vice versa. Hence clocks of neighboring nodes can be up to ε off.

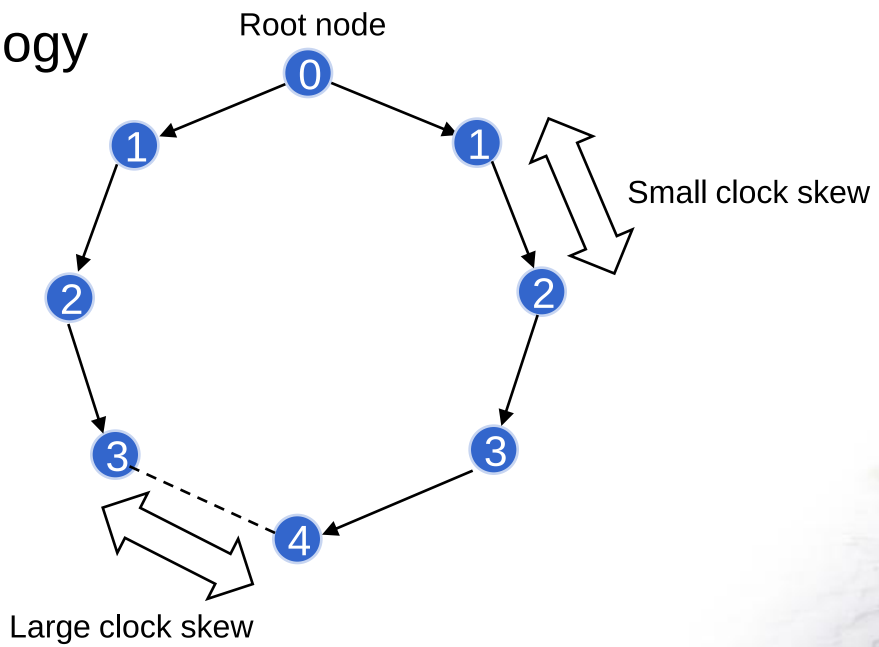


- Hence, two nodes at distance *D* may have clocks which are εD off.
- This can be achieved by a simple flooding algorithm: Whenever a node receives a new minimum value, it sets its clock to the new value and forwards its new clock value to all its neighbors.

Gradient Clock Synchronization

1. Global property: Minimize clock skew between any two nodes
2. Local (gradient) property: Small clock skew between two nodes if the distance between the nodes is small.
3. Clock should not be allowed to jump backwards
   - You don't want new events to be registered earlier than older events.

- Example: Ring topology



Trivial Solution: Let t = 0 at all nodes and times

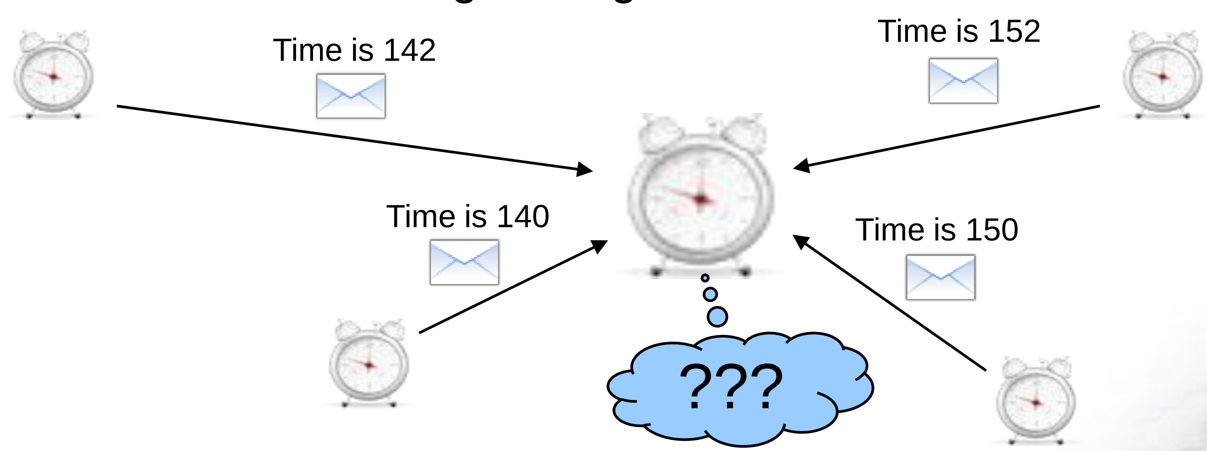
1. Global property: Minimize clock skew between any two nodes
2. Local (gradient) property: Small clock skew between two nodes if the distance between the nodes is small.
3. Clock should not be allowed to jump backwards

- To prevent trivial solution, we need a fourth constraint:

4. Clock should always to move forward.
    - Sometimes faster, sometimes slower is OK.
    - But there should be a minimum and a maximum speed.

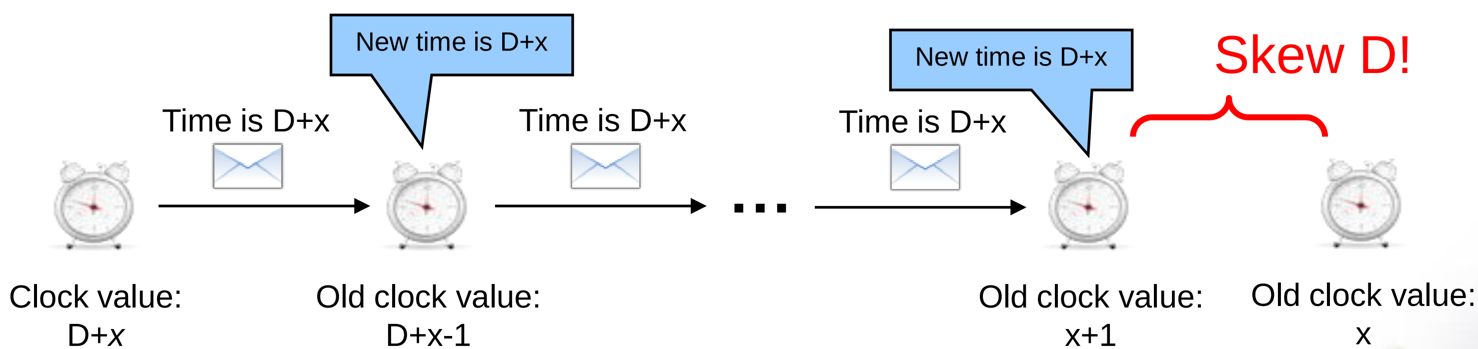
Gradient Clock Synchronization

- Model
  - Each node has a hardware clock $H_i(\cdot)$ with a clock rate $h_i(t) \in [L, U]$ where $0 < L < U$ and $U \geq 1$
  - The time of node *i* at time *t* is $H_i(t) = \int_0^t h_i(t)\,dt$
  - Each node has a logical clock $L_i(\cdot)$ which increases at the rate of $H_i(\cdot)$
  - Employ a synchronization algorithm A to update the local clock with fresh clock values from neighboring nodes (clock cannot run backwards)
  - Nodes inform their neighboring nodes when local clock is updated



Synchronization Algorithms: A^{max}

- Question: How to update the local clock based on the messages from the neighbors?
- Idea: Minimizing the skew to the fastest neighbor
  - Set the clock to the maximum clock value received from any neighbor (if greater than local clock value)
- Poor gradient algorithm: Fast propagation of the largest clock value could lead to a large skew between two neighboring nodes



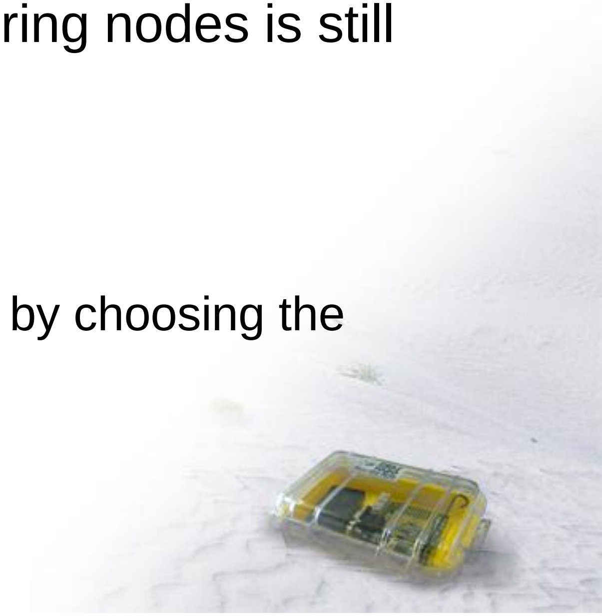
Synchronization Algorithms: $A^{max'}$

- The problem of A^{max} is that the clock is always increased to the maximum value
- Idea: Allow a constant slack γ between the maximum neighbor clock value and the own clock value
- The algorithm $A^{max'}$ sets the local clock value $L_i(t)$ to

$$L_i(t) := \max(L_i(t), \max_{j \in N_i} L_j(t) - \gamma)$$

→ Worst-case clock skew between two neighboring nodes is still $\Theta(D)$ independent of the choice of γ!

- How can we do better?
  - Idea: Take the clock of all neighbors into account by choosing the average value

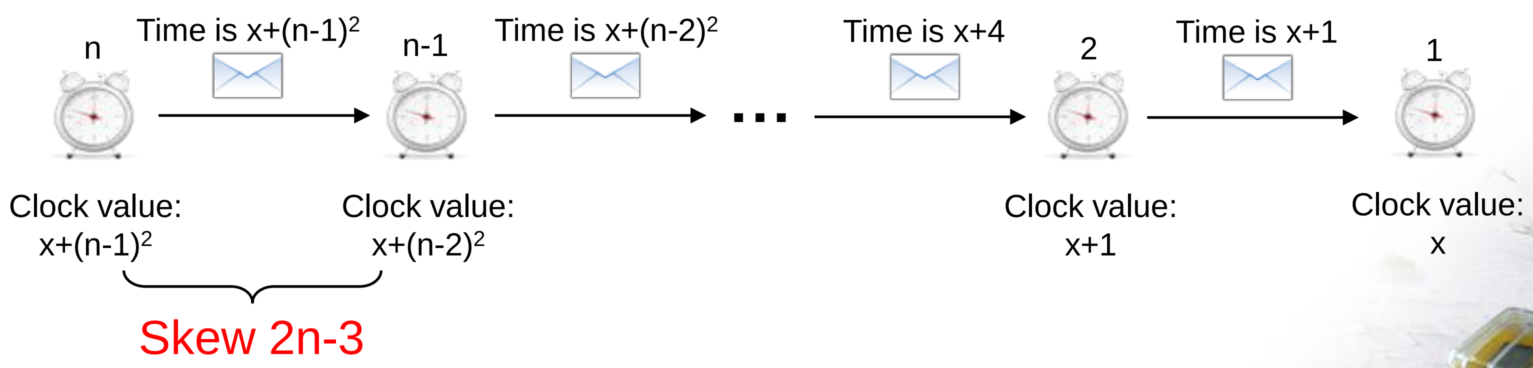


Synchronization Algorithms: A^{avg}

- A^{avg} sets the local clock to the average value of all neighbors:

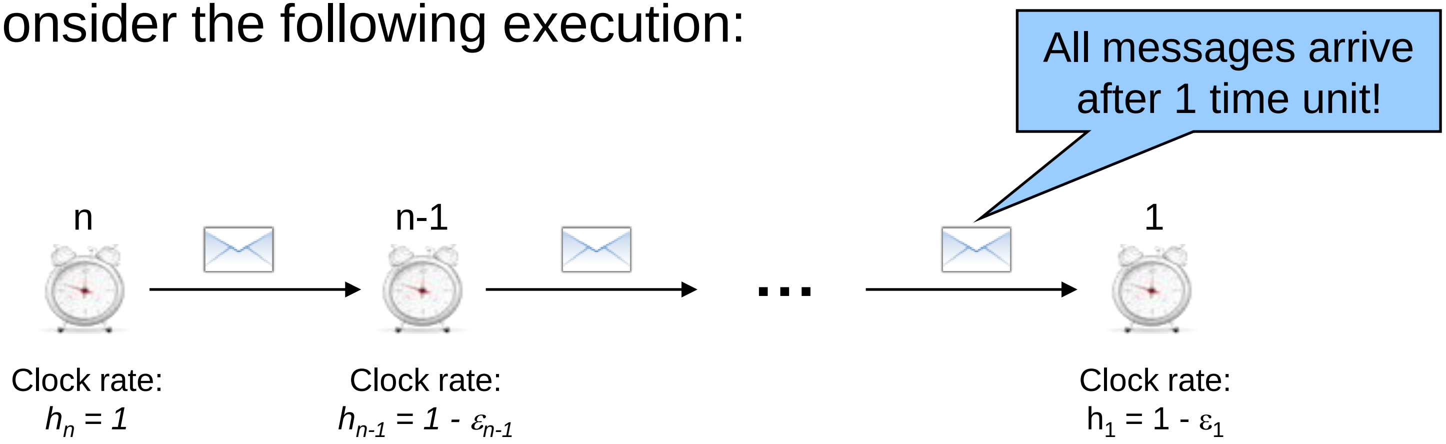
$$L_i(t) := \max\left(L_i(t), \frac{1}{|N_i|} \sum_{j \in N_i} L_j(t)\right)$$

- Surprisingly, this algorithm is even worse!
- We will now proof that in a very natural execution of this algorithm, the clock skew becomes large!



Synchronization Algorithms: A^{avg}

Consider the following execution:



All ε_i for $i \in \{1,\dots,n-1\}$ are arbitrary values in the range (0,1)

→ The clock rates can be viewed as *relative* rates compared to the fastest node n!

Theorem: In the given execution, the largest skew between neighbors is $2n-3 \in O(D)$.

Synchronization Algorithms: A^{avg}

We first prove two lemmas:

Lemma 1: In this execution it holds that $\forall$ t $\forall$ i $\in$ {2,…,n}:
$L_i(t) - L_{i-1}(t) \leq 2i - 3$, independent of the choices of $\varepsilon_i > 0$.

Proof:
Define $\Delta L_i(t) := L_i(t) - L_i(t-1)$. It holds that $\forall$ t $\forall$ i: $\Delta L_i(t) \leq 1$.
$L_1(t) = L_2(t-1)$ as node 1 has only one neighbor (node 2).
Since $\Delta L_2(t) \leq 1$ for all t, we know that $L_2(t) - L_1(t) \leq 1$ for all t.

Assume now that it holds for $\forall$ t $\forall$ j ≤ i: $L_j(t) - L_{j-1}(t) \leq 2j - 3$.
We prove a bound on the skew between node i and i+1:
For t = 0 it is trivially true that $L_{i+1}(t) - L_i(t) \leq 2(i+1) - 3$.

Synchronization Algorithms: A^{avg}

Assume that it holds for all t’ ≤ t. For t+1 we have that

$$
\begin{aligned}
L_i(t+1) &\geq \frac{L_{i+1}(t) + L_{i-1}(t)}{2} \\
&\geq \frac{L_{i+1}(t) + L_i(t) - (2i-3)}{2} \\
&\geq \frac{L_{i+1}(t) + L_i(t+1) - 1 - (2i-3)}{2} \\
&\geq L_{i+1}(t+1) - (2(i+1) - 3).
\end{aligned}
$$

- The first inequality holds because the logical clock value is always at least the average value of its neighbors.
- The second inequality follows by induction.
- The third and fourth inequalities hold because $\Delta L_i(t) \leq 1$.

Synchronization Algorithms: A^{avg}

Lemma 2: $\forall$ i $\in$ {1,…,n}: $\lim_{t \to \infty} \Delta L_i(t) = 1$.

Proof:

Assume $\Delta L_{n-1}(t)$ does not converge to 1.

Case (1):

$\exists\, \varepsilon > 0$ such that $\forall$ t: $\Delta L_{n-1}(t) \leq 1 - \varepsilon$.

As $\Delta L_n(t)$ is always 1, if there is such an ε, then

$\lim_{t \to \infty} L_n(t) - L_{n-1}(t) = \infty$, a contradiction to Lemma 1.

Case (2):

$\Delta L_{n-1}(t) = 1$ only for some t, then there is an unbounded number of times t' where $\Delta L_{n-1}(t) < 1$, which also implies that

$\lim_{t \to \infty} L_n(t) - L_{n-1}(t) = \infty$, again contradicting Lemma 1.

Hence, $\lim_{t \to \infty} \Delta L_{n-1}(t) = 1$. Applying the same argument to the other nodes, it follows inductively that $\forall$ i $\in$ {1,…,n}: $\lim_{t \to \infty} \Delta L_i(t) = 1$.

Synchronization Algorithms: A^{avg}

Theorem: In the given execution, the largest skew between neighbors is 2n-3.

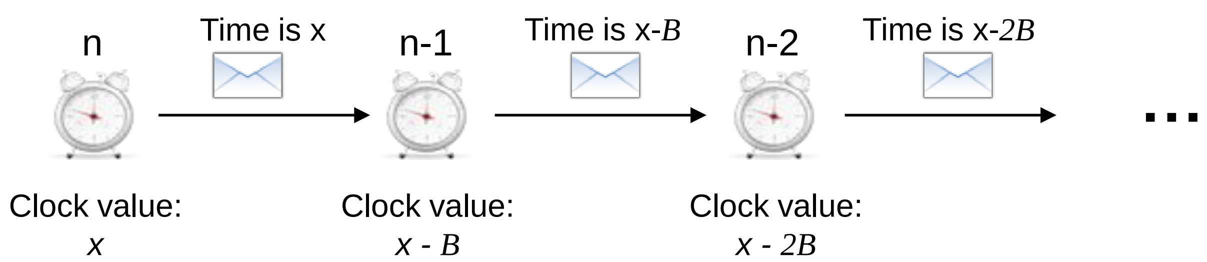
Proof:
We show that $\forall$ i $\in$ {2,…,n}: $\lim_{t \to \infty} L_i(t) - L_{i-1}(t) = 2i - 3$.
Since $L_1(t) = L_2(t-1)$, it holds that $\lim_{t \to \infty} L_2(t) - L_1(t) = \Delta L_1(t) = 1$, according to Lemma 2.
Assume that $\forall$ j ≤ i: $\lim_{t \to \infty} L_j(t) - L_{j-1}(t) = 2j - 3$.
According to Lemma 1 & 2, $\lim_{t \to \infty} L_{i+1}(t) - L_i(t) = Q$ for a value $Q \le 2(i+1) - 3$. If (for the sake of contradiction) $Q < 2(i+1) - 3$, then

$$\begin{aligned}\lim_{t\to\infty} L_i(t) &= \lim_{t\to\infty} \frac{L_{i-1}(t-1) + L_{i+1}(t-1)}{2} \\ &= \lim_{t\to\infty} \frac{2L_i(t-1) - (2i-3) + Q}{2}\end{aligned}$$

and thus $\lim_{t \to \infty} \Delta L_i(t) < 1$, a contradiction to Lemma 2.

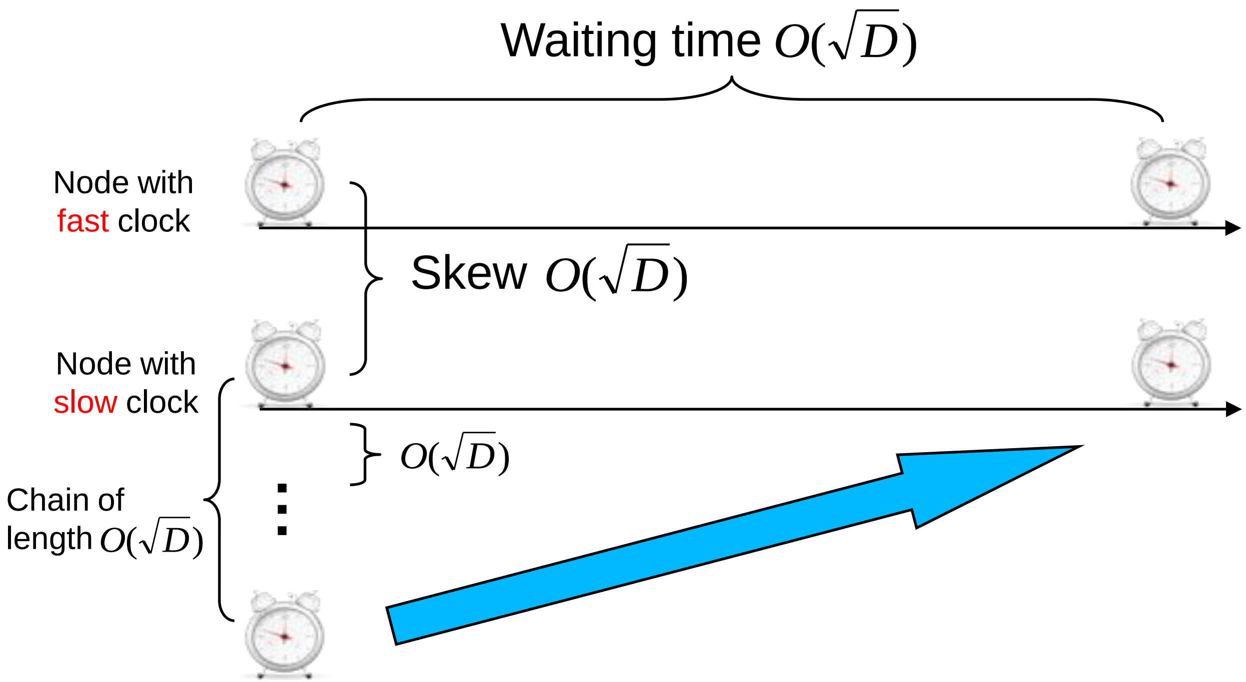
Synchronization Algorithms: A^{bound}

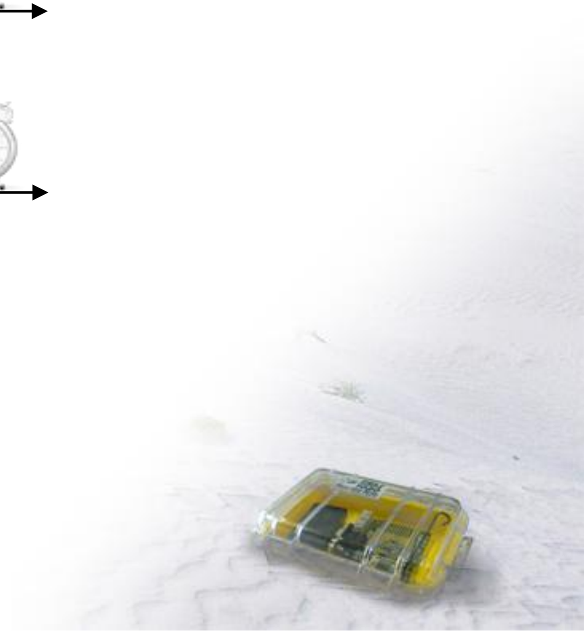
- Idea: Minimize the skew to the slowest neighbor
  - Update the local clock to the maximum value of all neighbors as long as no neighboring node's clock is more than B behind.
- Gives the slowest node time to catch up
- Problem: Chain of dependency
  - Node *n-1* waits for node *n-2*, node *n-2* waits for node *n-3*, …
    → Chain of length Θ(n) = Θ(D) results in Θ(D) waiting time
    → Θ(D) skew!



Synchronization Algorithms: A^{root}

- How long should we wait for a slower node to catch up?
  - Do it smarter: Set $B = O(\sqrt{D})$ → skew is allowed to be $O(\sqrt{D})$ → waiting time is at most $O(D/B) = O(\sqrt{D})$ as well





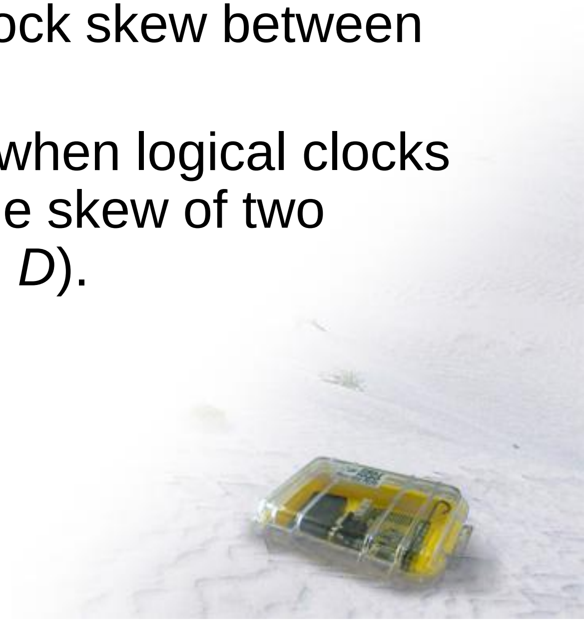
Synchronization Algorithms: A^{root}

- When a message is received, execute the following steps:

max := Maximum clock value of all neighboring nodes
min := Minimum clock value of all neighboring nodes

if (*max* > own clock and *min* + $U\sqrt{D+1}$ > own clock
    own clock := min(*max*, *min* + $U\sqrt{D+1}$)
    inform all neighboring nodes about new clock value
end if

- This algorithm guarantees that the worst-case clock skew between neighbors is bounded by $O(\sqrt{D})$.
- In [Fan and Lynch, PODC 2004] it is shown that when logical clocks need to obey minimum/maximum speed rules, the skew of two neighboring clocks can be up to $\Omega(\log D / \log \log D)$.



Open Problem

- The obvious open problem is about gradient clock synchronization.

- Nodes in an arbitrary graph are equipped with an unmodifiable hardware clock and a modifiable logical clock. The logical clock must make progress roughly at the rate of the hardware clock, i.e., the clock rates may differ by a small constant. Messages sent over the edges of the graph have delivery times in the range [0, 1]. Given a bounded, variable drift on the hardware clocks, design a message-passing algorithm that ensures that the logical clock skew of adjacent nodes is as small as possible at all times.

- Indeed, there is a huge gap between upper bound of $\sqrt{D}$ and lower bound of $\log D / \log\log D$.