Clock Synchronization
Chapter 9
Acoustic Detection (Shooter Detection)

- Sound travels much slower than radio signal (331 m/s)
- This allows for quite accurate distance estimation (cm)
- Main challenge is to deal with reflections and multiple events
Rating

- **Area maturity**
  - First steps: No apps
  - Text book: Mission critical

- **Practical importance**
  - No apps: Mission critical
  - Not really: Must have

- **Theoretical importance**
  - Not really: Must have
Overview

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

- Synchronizing time 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

*terms are explained on following slides
Properties of Clock 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

- One-shot versus continuous synchronization
  - Periodic synchronization required to compensate clock drift

- A-priori versus a-posteriori
  - A-posteriori clock synchronization triggered by an event

- Global versus local synchronization (explained later)

- Accuracy versus convergence time, Byzantine nodes, …
Clock Sources

• Radio Clock Signal:
  – Clock signal from a reference source (atomic clock) is transmitted over a long wave 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

<table>
<thead>
<tr>
<th>Platform</th>
<th>System clock</th>
<th>Crystal oscillator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mica2</td>
<td>7.37 MHz</td>
<td>32 kHz, 7.37 MHz</td>
</tr>
<tr>
<td>TinyNode 584</td>
<td>8 MHz</td>
<td>32 kHz</td>
</tr>
<tr>
<td>Tmote Sky</td>
<td>8 MHz</td>
<td>32 kHz</td>
</tr>
</tbody>
</table>
Sender/Receiver Synchronization

- Round-Trip Time (RTT) based synchronization

- Receiver synchronizes to the sender's clock
- Propagation delay $\delta$ and clock offset $\theta$ 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
Some Details

- Different radio chips use different paradigms:
  - Left is a CC1000 radio chip which generates an interrupt with each byte.
  - Right is a CC2420 radio chip that generates a single interrupt for the packet after the start frame delimiter is received.

- In sensor networks propagation can be ignored (<1 μs for 300m).

- Still there is quite some variance in transmission delay because of latencies in interrupt handling (picture right).
General Framework

- The clock synchronization framework must provide the abstraction of a correct logical time to the application. This logical time is based on the (inaccurate) hardware clock, and calibrated by exchanging messages with other nodes in the network.
Reference-Broadcast Synchronization (RBS)

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

\[
t_2 = t_1 + S_s + A_s + P_{s,A} + R_A \\
t_3 = t_1 + S_s + A_s + P_{s,B} + R_B \\
\theta = t_2 - t_3 = (P_{s,A} - P_{s,B}) + (R_A - R_B)
\]

- 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
- E.g., least-square linear regression to tackle clock drifts
- Multi-hop?
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

Why this random timer?
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. grid)?
  - Two neighbors may have bad synchronization?
Flooding Time Synchronization Protocol (FTSP)

- Each node maintains both a local and a global time
- Global time is synchronized to the local time of a reference node
  - Node with the smallest id is elected as the reference node
- Reference time is flooded through the network periodically

- Timestamping at the MAC Layer is used to compensate for deterministic message delays
- Compensation for clock drift between synchronization messages using a linear regression table
From single-hop to multi-hop

- Many protocols don’t even handle single-hop clock synchronization well. On the left figures we see the absolute synchronization errors of TPSN and RBS, respectively. The figure on the right presents a single-hop synchronization protocol minimizing systematic errors.

- Even perfectly symmetric errors will sum up over multiple hops.
  - In a chain of $n$ nodes with a standard deviation $\sigma$ on each hop, the expected error between head and tail of the chain is in the order of $\sigma\sqrt{n}$. 
Best tree for tree-based clock synchronization?

- Finding a good tree for clock synchronization is a tough problem
  - Spanning tree with small (maximum or average) stretch.

- Example: Grid network, with \( n = m^2 \) nodes.

- No matter what tree you use, the maximum stretch of the spanning tree will always be at least \( m \) (just try on the grid figure right...)

- In general, finding the minimum max stretch spanning tree is a hard problem, however approximation algorithms exist [Emek, Peleg, 2004].
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:
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 move forward.
   • Sometimes faster, sometimes slower is OK.
   • But there should be a minimum and a maximum speed.
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 $\mu$
  – Jitter $\varepsilon$ 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 $\varepsilon$ off.

• Hence, two nodes at distance $D$ may have clocks which are $\varepsilon D$ 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.
Local/Gradient Clock Synchronization

• Model
  
  – Each node has a hardware clock $H_i(\cdot)$ with a clock rate $h_i(t)$ such that $(1-\epsilon)t \leq h_i(t) \leq (1+\epsilon)t$
  – The hardware clock 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 logical clock using the hardware clock and neighboring messages
  – The message transmission delay is in $(0,1]$
Synchronization Algorithms: $A^{\text{max}}$

- **Question:** How to update the logical 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 local property:** Fast propagation of the largest clock value could lead to a large skew between two neighboring nodes
  - First all messages take 1 time unit, then we have a fast message!

[Diagram showing clock updates and skew]

Clock value: $D+x$  
Old clock value: $D+x-1$  
New time is $D+x$

Old clock value: $x+1$  
New time is $D+x$

Old clock value: $x$  
Skew $D$!
Synchronization Algorithms: $A_{\text{max'}}$

- The problem of $A_{\text{max}}$ is that the clock is always increased to the maximum value.
- Idea: Allow a constant slack $\gamma$ between the maximum neighbor clock value and the own clock value.
- The algorithm $A_{\text{max'}}$ 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 $\gamma$!

- How can we do better?
  - Adjust logical clock speeds to catch up with fastest node (i.e. no jump)?
  - 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:

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

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

```
Clock value: x+(n-1)^2
Time is x+(n-1)^2

Clock value: x+(n-2)^2
Time is x+(n-2)^2

Clock value: x+1
Time is x+1

Clock value: x
Time is x
```

skew 2n-3
Synchronization Algorithms: $A^{avg}$

- Consider the following execution:

  ![Clock rates diagram](image)

  Clock rate: $h_n = 1$
  Clock rate: $h_{n-1} = 1 - \varepsilon_{n-1}$
  Clock rate: $h_1 = 1 - \varepsilon_1$

- All $\varepsilon_i$ for $i \in \{1, \ldots, n-1\}$ are arbitrary values with $\varepsilon_i > 0$.
- The clock rates can be viewed as relative rates compared to the fastest node $n$. We will show:

  Theorem: In the given execution, the largest skew between neighbors is $2n-3 \in \Theta(D)$. Hence, the global skew is $\Theta(D^2)$.
Synchronization Algorithms: $A^{avg}$

We first prove two lemmas:

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

**Proof:**
Define $\Delta L_i(t) := L_i(t) - L_{i-1}(t)$. It holds that $\forall t \forall i$: $\Delta L_i(t) \leq 1$.
$L_1(t) = L_2(t-1)$, because 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 $\forall t, \forall j \leq i$: $L_j(t) - L_{j-1}(t) \leq 2j - 3$.
We prove a bound on the skew between node $i$ and $i+1$:
For $t = 0$ it is trivially true that $L_{i+1}(t) - L_i(t) \leq 2(i+1) - 3$,
since all clocks start with the same time.
Synchronization Algorithms: $A^{avg}$

- Assume that it holds for all $t' \leq t$. For $t+1$ we have that

$$L_i(t + 1) \geq \frac{L_{i+1}(t) + L_{i-1}(t)}{2}$$

$$\geq \frac{L_{i+1}(t) + L_i(t) - (2i - 3)}{2}$$

$$\geq \frac{L_{i+1}(t) + L_i(t + 1) - 1 - (2i - 3)}{2}$$

$$\geq L_{i+1}(t + 1) - (2(i + 1) - 3).$$

- 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:** $\forall \ i \in \{1,\ldots, n\}: \lim_{t \to \infty} \Delta L_i(t) = 1.$

**Proof:**

- Assume $\Delta L_{n-1}(t)$ does not converge to 1.
- Argument for simple case:
  $\exists \ \varepsilon > 0$ 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 $\varepsilon$, then
  $\lim_{t \to \infty} L_n(t) - L_{n-1}(t) = \infty$, a contradiction to Lemma 1.

- A bit more elaborate argument:
  $\Delta L_{n-1}(t) = 1$ 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.

  Again, $\lim_{t \to \infty} \Delta L_{n-1}(t) = 1$.

- Applying the same argument to the other nodes, it follows
  inductively that $\forall \ i \in \{1,\ldots, n\}: \lim_{t \to \infty} \Delta L_i(t) = 1.$
Synchronization Algorithms: $A^{avg}$

**Theorem:** The skew between neighbors $i$ and $i-1$ converges to $2i-3$.

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

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

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 $\Theta(n) = \Theta(D)$ results in $\Theta(D)$ waiting time
  - $\Theta(D)$ skew!

![Diagram showing synchronization process]
Synchronization Algorithms: \( A^{\text{root}} \)

- 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

$$
\begin{align*}
\text{Waiting time } &O(\sqrt{D}) \\
\text{Skew } &O(\sqrt{D}) \\
\text{Node with fast clock} &\quad \text{Node with slow clock} \\
\{ \quad O(\sqrt{D}) \quad \}
\end{align*}
$$

Chain of length \( O(\sqrt{D}) \)
Synchronization Algorithms: $A^{\text{root}}$

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

  \[
  \begin{align*}
  &\text{max} := \text{Maximum clock value of all neighboring nodes} \\
  &\text{min} := \text{Minimum clock value of all neighboring nodes} \\
  \text{if } (\text{max} > \text{own clock} \text{ and } \text{min} + U\sqrt{D+1} > \text{own clock}) \\
  &\quad \text{own clock} := \min(\text{max}, \text{min} + U\sqrt{D+1}) \\
  &\quad \text{inform all neighboring nodes about new clock value} \\
  \text{end if}
  \end{align*}
  \]

- This algorithm guarantees that the worst-case clock skew between neighbors is bounded by $A^{\text{root}}(\sqrt{D+1})$. 
Some Results

- All natural/proposed clock synchronization algorithms seem to fail horribly, having at least square-root skew between neighbor nodes.

- Indeed [Fan, Lynch, PODC 2004] show that when logical clocks need to obey minimum/maximum speed rules, the skew of two neighboring clocks can be up to \( \Omega(\log D / \log \log D) \), where \( D \) is the diameter of the network.

- Nice open problem…? Unfortunately not! In 2008 a \( O(\log D) \) clock skew algorithm was presented at [Lenzen et al., FOCS 2008]. Also, the lower bound seems to be \( \Omega(\log D) \)…
Theory vs. Practice

• Can these theoretical findings be applied to practice?
  – Do the theoretical models represent reality?
• Example: Experimental evaluation on a ring topology

![Ring Topology Diagram](image)

- Node 8 and Node 15 are leaves of two different subtrees

• Results: Synchronization error between Node 8 and Node 15
  – Tree-based synchronization (FTSP, left) leads to a larger error than a simple gradient clock synchronization algorithm (right)
Open Problem

- As listed on slide 9/6, clock synchronization has lots of parameters. Some of them (like local/gradient) clock synchronization have only started to be understood.

- **Local clock synchronization** in combination with other parameters are not understood well, e.g.
  - accuracy vs. convergence
  - fault-tolerance in case some clocks are misbehaving [Byzantine]
  - clock synchronization in dynamic networks