Chapter 11
TIME SYNC

Distributed Computing Group

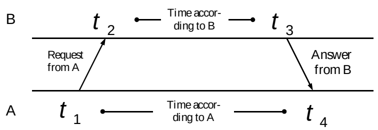
Mobile Computing
Winter 2005 / 2006

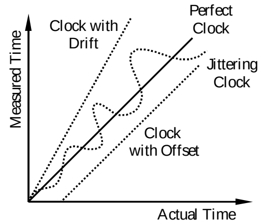
Overview

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

Motivation

- Time synchronization is essential for many applications
  - Coordination of wake-up and sleeping times
  - TDMA schedules
  - Ordering of sensed events in habitat environments
  - Estimation of position information
  - …
- Scope of a Clock Synchronization Algorithm
  - *Packet delay / latency*
  - *Offset* between clocks
  - *Drift* between clocks





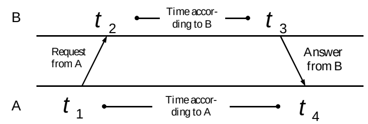
Disturbing Influences on Packet Latency

- Influences
  - Sending Time *S*
  - Medium Access Time *A*
  - Propagation Time $P_{A,B}$
  - Reception Time *R*
- Asymmetric packet delays due to *non-determinism*
- Example: RTT-based synchronization

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

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

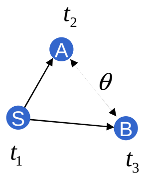
$$= \frac{(t_2 - t_1) + (t_3 - t_4)}{2}$$



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

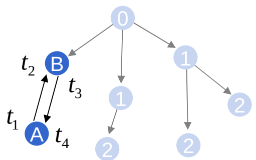
- Post-synchronization possible
- Least-square linear regression to tackle clock drifts

Time-sync Protocol for Sensor Networks (TSPN)

- 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
- *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 (TSPN)

$$t_2 = t_1 + S_A + A_A + P_{A,B} + R_B$$
$$t_4 = t_3 + S_B + A_B + P_{B,A} + R_A$$
$$\theta = \frac{(S_A - S_B) + (A_A - A_B) + (P_{A,B} - P_{B,A}) + (R_B - R_A)}{2}$$



- Time stamping packets at the MAC layer
- In contrast to RBS, the signal propagation time might be negligible
- About "two times" better than RBS
- Again, clock drifts are taken into account using periodical synchronization messages

- Problem: What happens in a ring?!?
  - Two neighbors will have exceptionally badly synchronization

Theoretical Bounds for Clock Synchronization

- Network Model:
  - Each node has a private clock
  - *n* node network, with diameter $\Delta \le n$.
  - 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 Δ might have clocks which are $\varepsilon\Delta$ 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

- It could happen that a clock has to jump back to a much lower value
  - Think again about a ring example, assume that in one leg of the ring messages are forwarded fast all of a sudden.

- Problem: At a node, you don't want a clock to jump back all of a sudden.
  - You don't want new events to be registered earlier than older events.
  - Instead, you want your clock always to move forward. Sometimes faster, sometimes slower is OK. But there should be a minimum and a maximum speed.
  - This is called "gradient" clock synchronization in [Fan and Lynch, PODC 2004] .

- 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

$$\Omega\left(\frac{\log \Delta}{\log \log \Delta}\right)$$