Synchronization in Sensor Networks

Blerta Bishaj
Helsinki University of Technology

Overview

- Introduction
- Characterizing Time Synchronization
- Reasons for clock desynchronization
- Algorithms
- Time synchronization attacks in sensor networks
- Conclusions

Introduction

- Sensor networks are **implemented** in climate studies, military implementations, agriculture, maintenance of machinery,urban didaster prevention, etc.
- **Time synchronization** particularly **important**, because most often used for **gathering data**.
- Sensor networks have major resource constraints: **smaller nodes**, **harsh environment**.

Characterizing Time Synchronization

These metrics are important:

- **Precision.** The maximum error in relation to a standard.
- **Lifetime**. Sync can last for a moment (for an event), or the whole network lifetime.
- **Scope and availability**. The extent of the sync.
- **Efficiency.** Time and energy needed for the sync.
- **Cost and form**. Some netw. might need very small, low-cost nodes.

These metrics cannot be all optimized:

GPS – precision (200ns), but poor scope and availability

small nodes sync with a signal – poor precision, but fast and eergy-efficient

Note: high freqs consume more => to be used only when needed

Causes of clock desynchronization

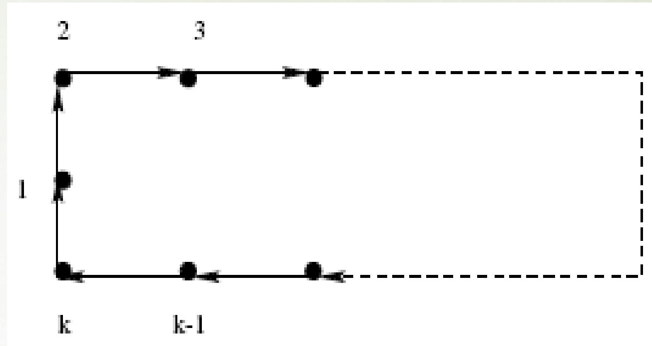
Clocks – quartz crystal vibrates when electricity is applied

Two nodes‘ clocks are **desynced**:

- Offset. Started at different times
- Skew. They run at different freqs, over time they diverge
- Drift. The freq changes over time
  - **Short-term drift**. Enironment (shock, temperatures, voltage)
  - Long-term drift. Crystal aging

All-node-based synchronization

- Assumptions:
  - **Clock cycles** are the **same**
  - **Clock tick** time **longer** than **transmission time** (otherwise it consumes energy)
  - Message transmission and **processing the same in all nodes** (can hold at first)
- The algorithm has two phases:
  - Phase **one**:
    - A **sync packet** is **sent** in a loop
    - **Initiator** marks **start and end time**
    - **Intermediaries** mark **current no of hops**
  - Phase **two**:
    - **Initiator** sends **a correction packet**, with **start** and **end time**, as well as total **no of hops**
    - **Each node** computes how to **adjust its clock**



The relative clock error proved to be at most three clock cycles

Cluster-based synchronization

- In all-nodes based algorithm, **all nodes** in **one session ?!**
- **Nodes** can be **organized** in **clusters**
- **Two rounds** of **sync**
  - round **one:** cluster **heads sync**
  - round **two:** cluster **heads** initiate **cluster sync**
  - **each round uses all-nodes based algorithm**
- **Algorithm** becomes more **flexible** and **scalable**
- **maximum error** increases to **6 clock cycles**
- **not scalable** for large entworks
- **not fault-tolerant**, the init. node may fail

The Rate-based Synchronous Diffusion Algorithm

- We can describe the **network** as a graph **G(V,E)**
- The time readings at each are: $C = (c_1^t, c_2^t, \cdots, c_n^t)^T$
- If $c_i > c_j$, we want to decrease c_i and increase c_j
  - diffusion value proportional to (c_i – c_j)
  - diffusion rate rij > 0 (rij = 0 if n_i and n_j are not neighbors)
  - $\Sigma_{j<>i}\, r_{ij} <= 1$
- Algorithm:

  Do the following with some given frequency
  for each sensor *ni* in the network **do**
      Exchange clock times with *ni*’s neighbors
      for each neighbor *nj* **do**
          Let the time difference between *ni* and *nj* be *ti −tj*
          Change *ni*’s time to *ti* − *rij*(*ti* − *tj*)
- It can be proved that it converges

The Asynchronous Diffusion Algorithm

- The previous alg requires rounds not to be interrupted
- The algorithm:
  - Each node asks neighbors about their time
  - Computes average
  - Sends back the result
  - A node takes part in one such operation at one time
- It can be proved that it converges

Reference Broadcast Synchronization (RBS)

- It is based on an analysis of mesage latency:
  - **Send time**. Time spent at the sender to construct and transfer it to the NIC
  - **Access time.** The **delay** incurred by the **MAC** protocol, for **access** to the **transmit channel**
  - **Propagation delay.** Time needed for the message to **reach the destination**, once it has left the sender
  - **Receive time**. **Receiver processing**. The overhead of packet processing is eliminated if the arrival time of the packet is **timestamped** in a **low layer** of the receiver's stack,

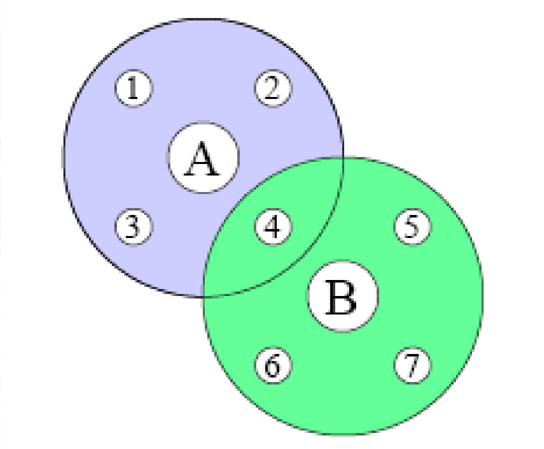
Reference Broadcast Synchronization (RBS)

- The algorithm:
  - **a node** broadcasts a **message** to **two reveivers**
  - they **record the time** they **received** it
  - they **exchange** the **time** of **receiveing** ysdfg

- Analysis:
  - eliminates non-determinism on the sender side (**send time** and **access time** not in the computations)
  - assumes **propagation time** is 0 (broadcast, no intermediary nodes)
  - **receive time** minimized if early timestamping

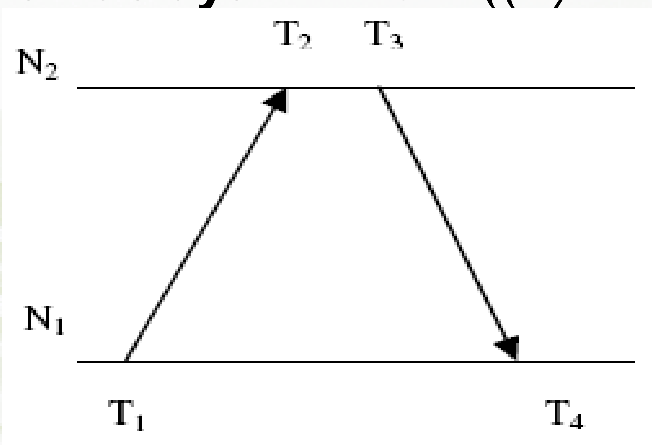
Reference Broadcast Synchronization (RBS)

- Precision of computation
  - **precision** can be **increased** if **m messages** are sent
  - each receiver computes the **average of offsets** for m messages
- Another way: **least-squares linear regression method**
  - this method would help find the **skew** also, not only the **offset**
- Multi-hop synchronization
  - node 4 can relate the two clocks



Time-sync Protocol for Sensor Networks (TPSN)

- The algorithm
  - a tree is built
  - each node establishes the level based on parent info
  - sync between child and parent (initiated by child)
    - **child** sends a mesage with the time **$\underline{T_1}$**
    - **parent** responds with reception **$\underline{T_2}$** and transmission time **$\underline{T_3}$**
    - **child** records receive time **$\underline{T_4}$**
  - **Offset** $\Delta = ((T_2 - T_1) - (T_4 - T_3)) / 2$
  - **Propagation delays** $d = ((T_2 - T_1) + (T_4 - T_3)) / 2$



- Topology changes affect the protocol

Flooding Time Synchronization Protocol (FTSP)

- Algorithm:
  - a **root** node **broadcasts**: *rootID*, *seqNum*, *sendingTime*
  - **nodes** hearing it **calculate** their **offset**
  - if there are several such messages, nodes can also **calculate** their **skew**

- FTPS multi-hop time synchronization:
  - a node receives a messages, then **sends another** for other nodes
  - messages that count for a node:
    - *seqNum* field is **bigger** than the biggest one received so far
    - the *rootID* of the message **not bigger** than the last received *rootID*

- More robust than TPSN, no topology, more resilient

Attacks on RBS

- Algorithm:
  - a reference message sent
  - two overhearing nodes exchange receiving time
  - a malicious node feeds wrong information
- The **multi-hop version** of **RBS** can also be attacked
  - the nodes nodes at the boundary of two overlapping regions
- Robust Estimation
  - **breakdown point** – smallest fraction of contamination in the data that can diverge the result
  - the **average** and the **linear regression** methods have a **low breakdown point**
  - another method might be used

Attacks on TPSN

- The algorithm is **tree-based**
- A node can **decept** all the **branch** that originates at it
- A node acn also pretend to be at a lower level, to contaminate more nodes

Attacks on FTSP

- A malicious node an become a root if it sends messages with ID 0 and a higher sequence number
- Then it tampers the sending time in the messages

Conclusions

- Time-synchronization protocols are crucial to sensor networks
- Higher accuracy and synchronization costs:
  - more clock ticks
  - more messages
  - more computations

  A compromise has to be made
- Security should be considered when designing these protocols