Time, Synchronization, and Wireless Sensor Networks
Part I

Ted Herman
University of Iowa

Presentation: Part I

synchronization and clocks

[detour: we review NTP]

clock hardware in sensor networks

technical approaches to clock synchronization between sensors

Synchronization

- used throughout distributed system software, middleware, and network protocols
- sensor networks: are they different from our usual model of (mobile) *ad hoc* networks?

→ yes : more limited resources, sensor and actuator events, energy constraints; many sensor networks do not have mobile nodes

Synchronization Techniques

- messages, tokens, permissions, locks, semaphores, synchronized object methods --- often too heavyweight for sensor networks (also, sensor networks are more faulty)
- time-division, wakeups, alarms, time-triggered events --- more practical in sensor networks because protocol stack is "thin", closer to hardware, where clocks are available.
- BUT, typical sensor network operating systems are not "hard real time" systems!

→ may need to add fault tolerance to applications that depend on time synchronization.

Using Clocks in Sensor Networks

- typical purpose of sensor networks: collect sensor data, log to database *and correlate with time, location, etc.* Notice: this is a "non-synchronization" use of time.
- future purpose of sensor networks: coordinated actuation, reacting to sensed events & command/control in real time.

Needed Clock Properties

- No agreement on this point! So many different kinds of sensor applications with different needs. → impossible to specify what is "perfect" clock generally
- Taxonomy of Clock Properties
  - logical time or real time ?
  - bounded or unbounded ?
  - synchronized to UTC (GPS) or internal time only ?
  - monotonic or backward correction allowed ?
  - δ-synchronized wrt neighbors, hop-distance ?

Special Requirements

- Efficiency
  - will clock algorithm fit into memory/processor constraints?
  - will clock algorithm burn up the batteries too fast?
- Scalability – will clock protocol fail or perform badly for large networks?
- Robustness – will clock protocol work when some sensor nodes are faulty, dynamically moved or replaced?
- Modes of synchrony: "on demand", "post facto", "regional time"
- Application-specific: are clocks only needed for "basestation data collect", or for arbitrary patterns of sensor data collection, sensor actuation, and such?

Controversy?

- Claim: GPS is solution to all problems of keeping time, synchronizing clocks
  - we will see this claim is doubtful for many wireless sensor networks, for several reasons
- Claim: Synchronizing clocks of nodes in sensor networks is not needed for applications that only collect data
  - this claim is actually true for some specific cases

GPS (and other Radio Beacons)

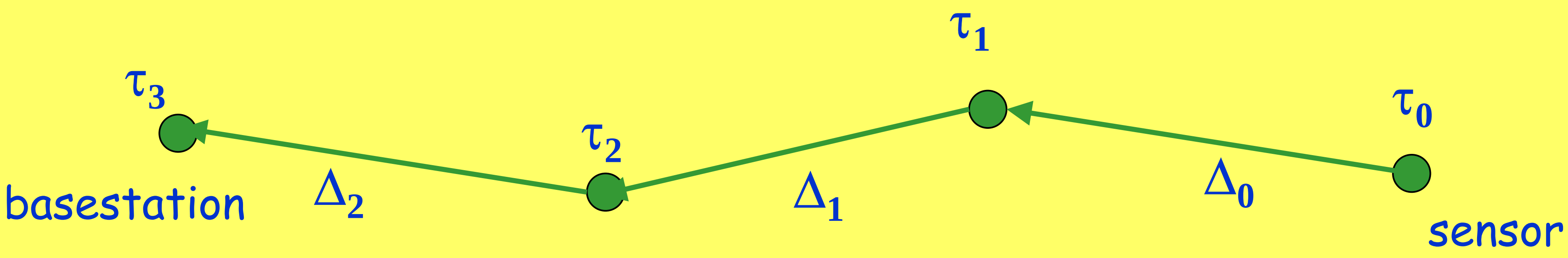
- Relatively high-power (GPS)
- Need special GPS / antenna hardware
- Need “clear view” to transmissions
  - ironically, mobility is an advantage!
- *Precision* of transmitted message is in seconds (not millisecond, microsecond, etc)
- “Pulse-per-Second” (PPS) can be highly precise (1/4 microsecond), but not easy to use
- other radio techniques: WWVB, GOES, ACTS

GPS hardware can be optimized for time synchronization

- PPS accurate to within one microsecond
- PPS requires extra hardware & interrupt service
- for timing, only one satellite needed in view
- agreement with UTC to nearest second *without PPS* based on ASCII NMEA message containing UTC time/date (using filter algorithms, could probably synchronize to within 25 milliseconds, depending on hardware GPS implementation)
- pulse later (after ASCII message) signals actual UTC second boundary

Timestamping without Sync

Suppose all delays can be accurately measured (and all clocks run at same rate)



message arriving to collection point (base station) contains data field with

$\tau_0 + \Delta_0 + \tau_1 + \Delta_1 + \tau_2$

→ highly dependent on implementation details

Presentation: Part I

synchronization and clocks

[detour: we review NTP]

clock hardware in sensor networks

technical approaches to clock synchronization between sensors

Interlude: Review NTP

- Can we learn how to synchronize time in sensor networks by studying NTP ?
- How does NTP use GPS to synchronize ?
- We can contrast NTP's approach with other time synchronization methods

How NTP uses GPS

- NTP: the Internet timekeeper
- Like GPS, there is a unique "leader" clock
- NTP/GPS is a two-network solution to synchronized clocks in a distributed system
  - NTP uses pull : clients request current time from servers (servers arranged in hierarchy of *strata*)
  - GPS uses push: atomic clock is broadcast to satellites, which relay time/pulses to Earth
  - Some NTP servers have attached GPS units for PPS signals, which regulate clock rates

NTP Technical Difficulties

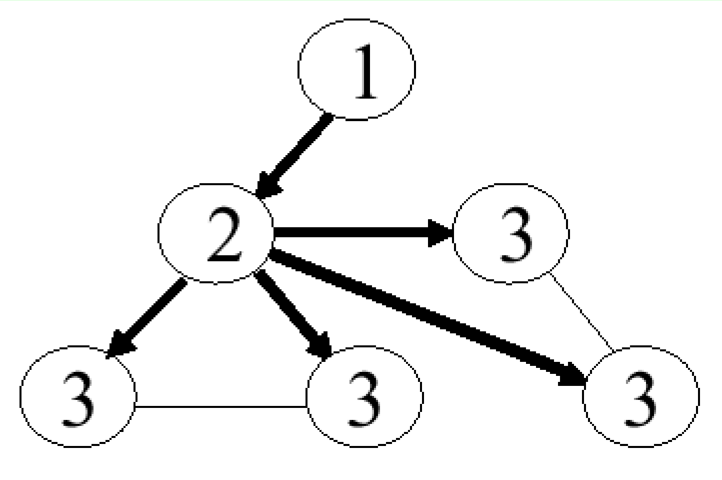
- The two goals of Clock Synchronization
  - Correct the displacement from leader clock

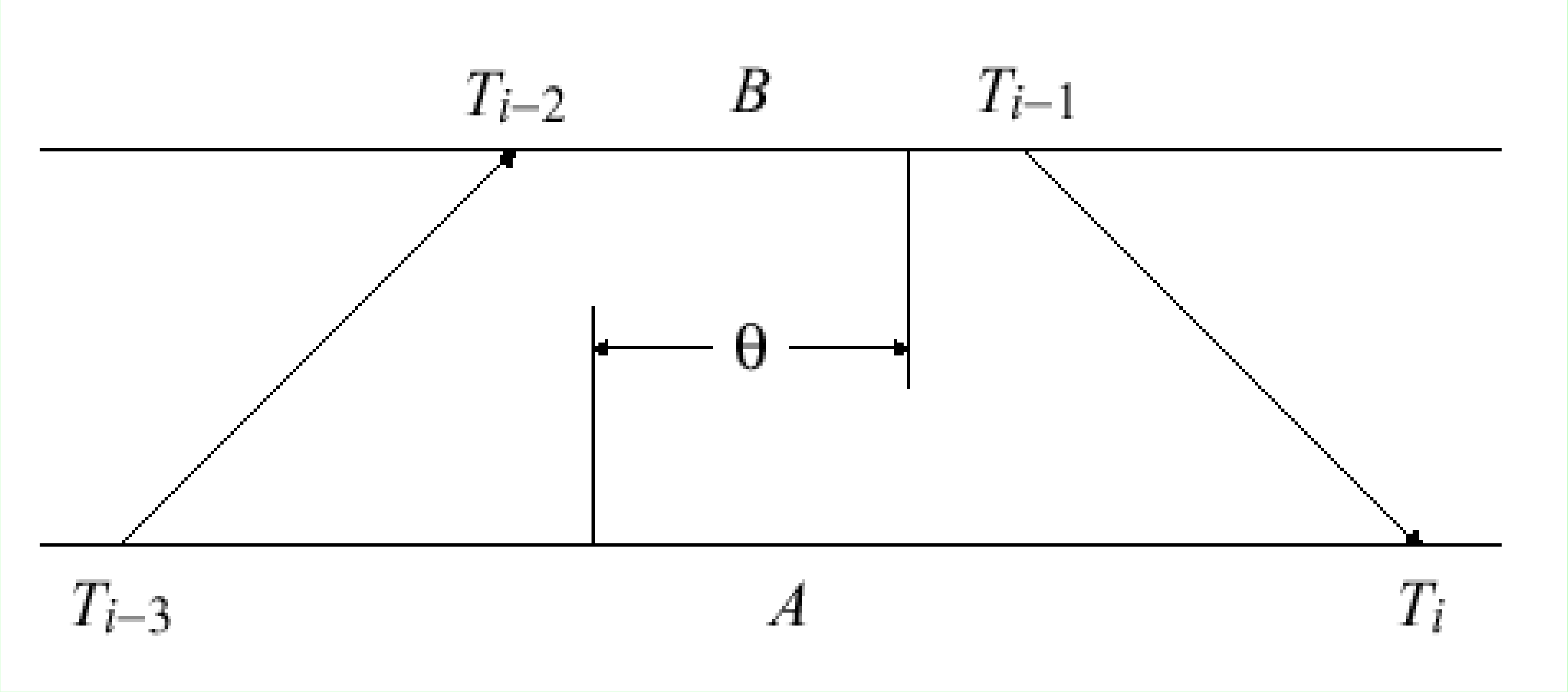
    offset
  - Compensate for incorrect local clock rate

    skew, drift
- To correct offset, use Internet protocol (pull)
- To correct for skew, use GPS/PPS (push)
- For efficiency, use hierarchy of Time Servers
- Extensive statistical techniques to overcome Internet nondeterministic delays

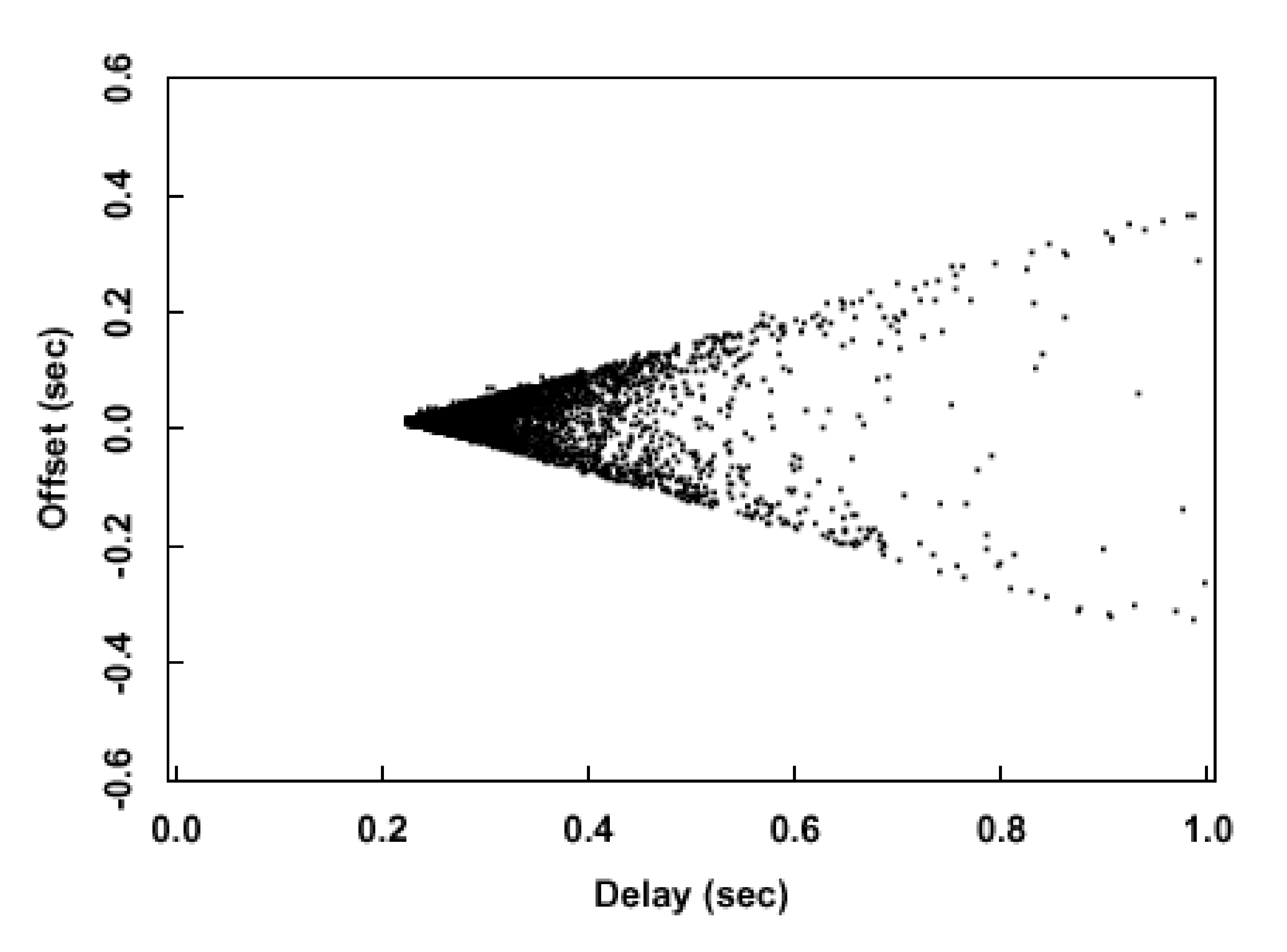
NTP Servers

request/reply accounts for round-trip delay

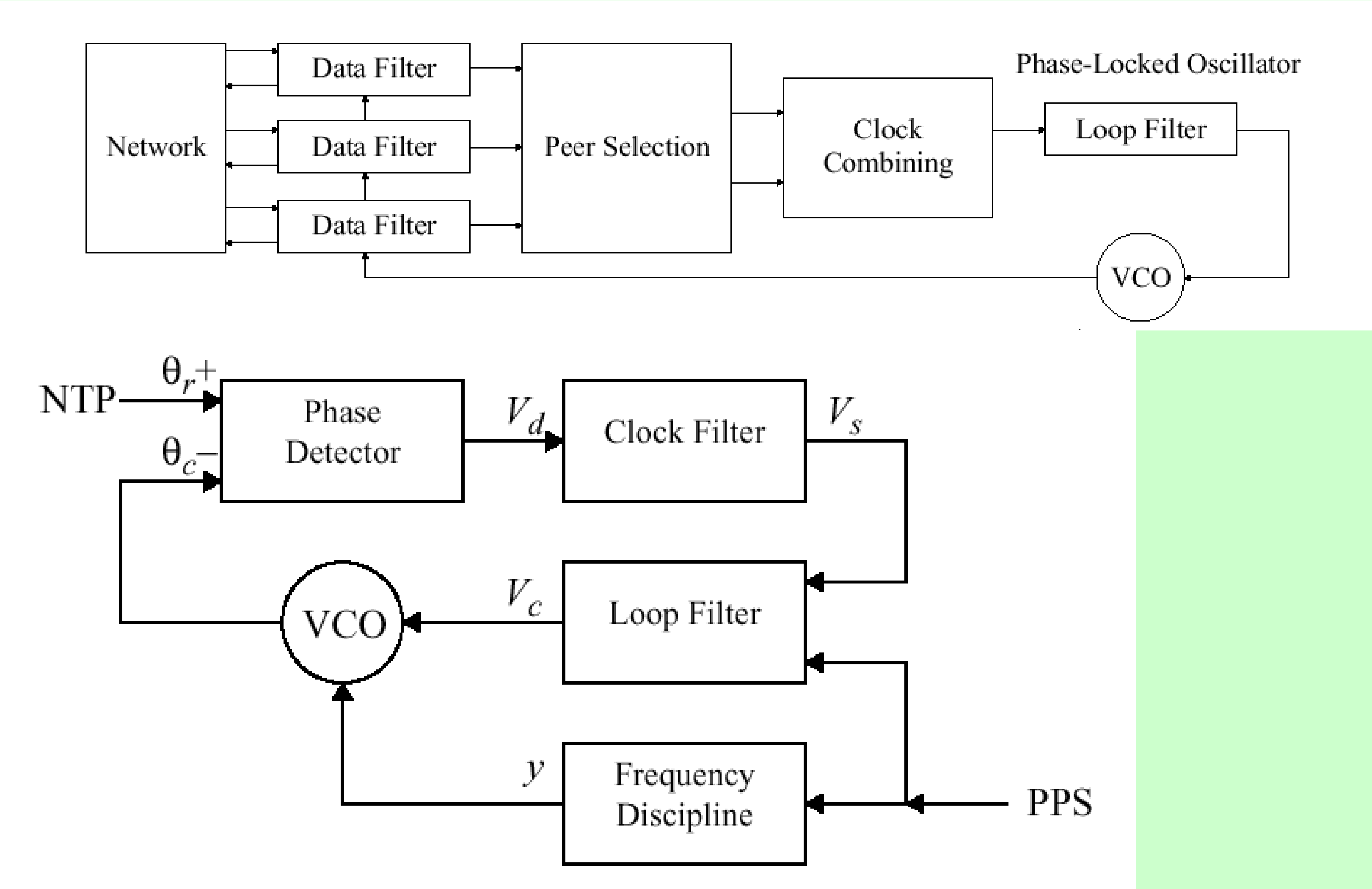




NTP statistics



NTP server logic



NTP characteristics

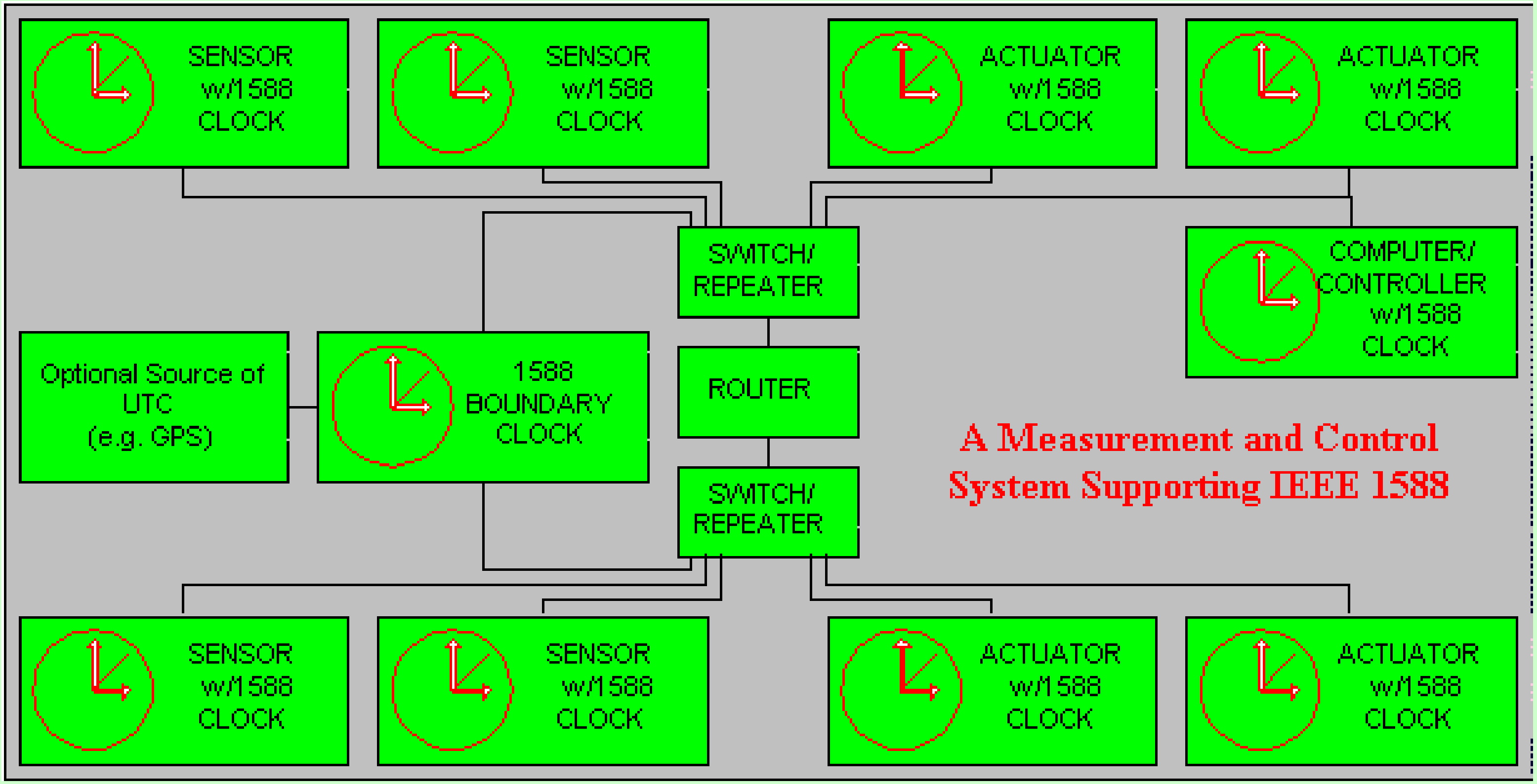
- Can take a long time to synchronize a clock
- No guarantee on accuracy --- however, 2-100 milliseconds is typical

  (see http://www.ntp.org/ntpfaq/NTP-s-algo.htm)
- Exploits availability of many servers
- Statistical techniques require significant computation and memory

→ characteristics not well suited to wireless sensor networks

Another standard: IEEE 1588

from http://ieee1588.nist.gov/ (Kang Lee)



not designed for wireless sensor networks

End of Detour: Conclusion

- NTP uses clever statistical techniques
  (probably too heavyweight for most sensor networks)
- NTP shows how PPS corrects for skew
- At stratum 1, specialized "time-GPS" hardware can synchronize to GPS/UTC within microseconds
  - only requires one satellite in view
- Idea of hierarchy, with "leader clock" at top will be useful for sensor networks

Presentation: Part I

synchronization and clocks

[detour: we review NTP]

clock hardware in sensor networks

technical approaches to clock synchronization between sensors

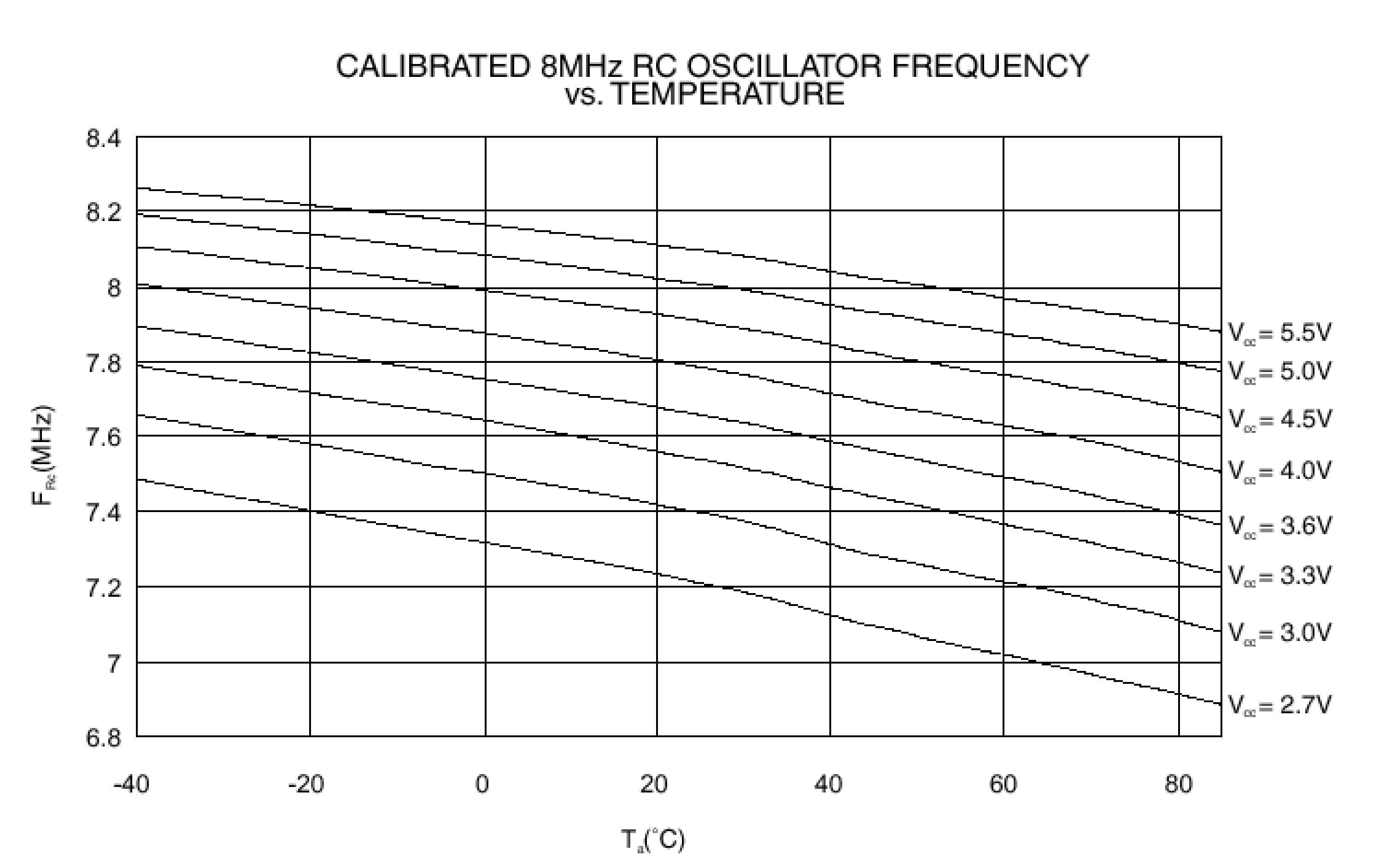
Clock Hardware in Sensors

- Sensors do not have clocks ! (construction is simpler, less expensive without)
- Typical sensor CPU has counters that increment by each cycle, generating interrupt upon overflow → we can keep track of time, but managing interrupts is error-prone
- External oscillator (with hardware counter) can increment, generate interrupt → another way to keep track of time --- even when CPU is "off" to save power

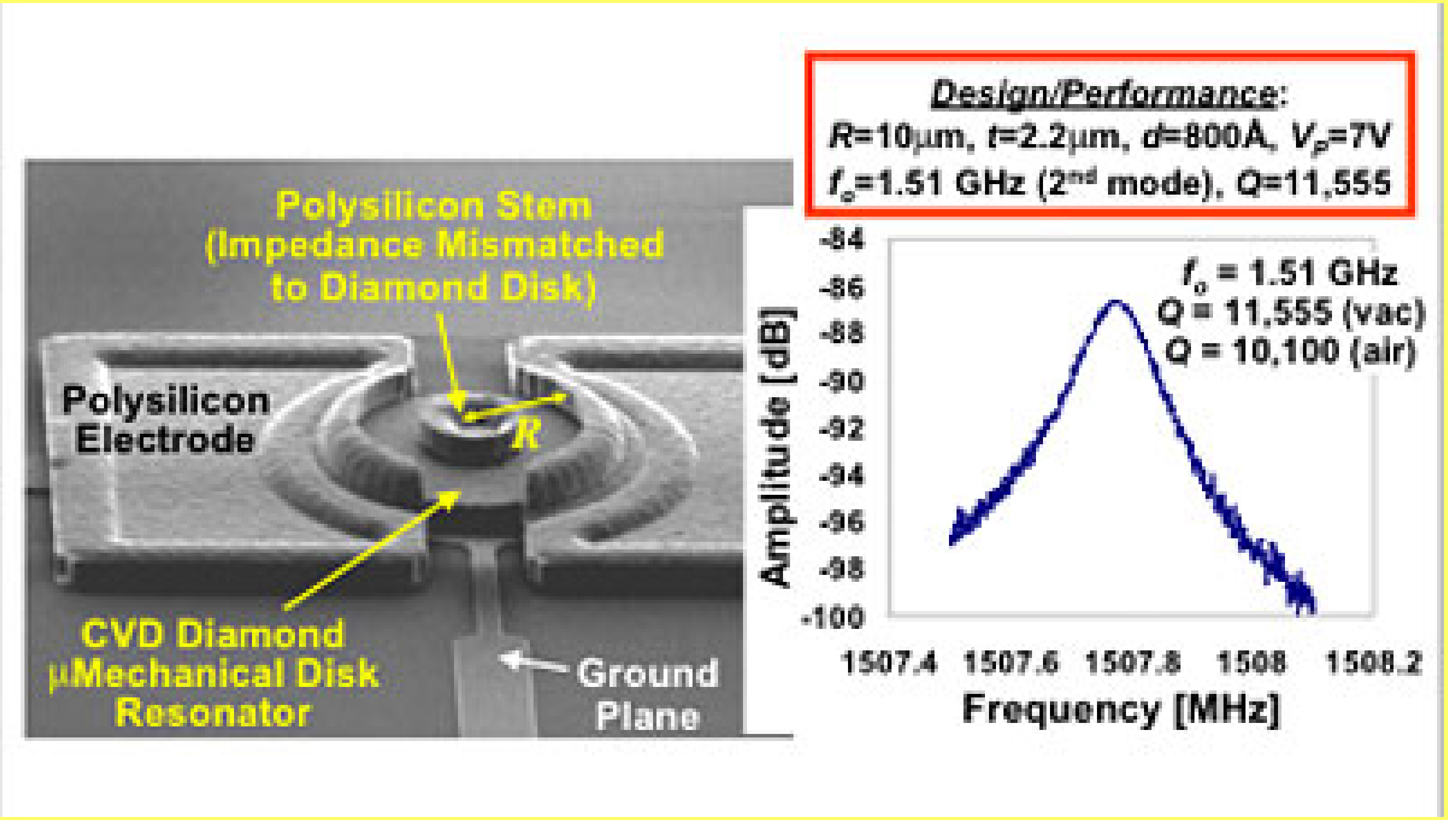
Oscillator Characteristics

- cheap, off-the-shelf components --- can deviate from ideal oscillator rate by one unit per 10^{-5} (for a microsecond counter, accuracy could diverge by 10 microseconds each second)
- oscillator rates vary depending on power supply, temperature

Typical Oscillator Data



Science Fiction or Future?



[Clark Nguyen, University of Michigan]

MEMS-scale atomic clocks solve oscillator variance problems

Presentation: Part I

synchronization and clocks

[detour: we review NTP]

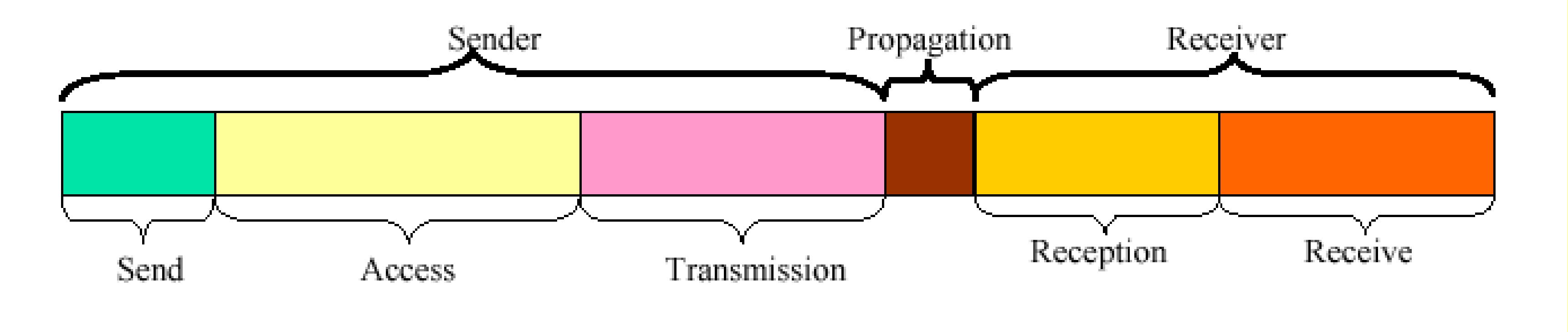
clock hardware in sensor networks

technical approaches to clock synchronization between sensors

Effects of Network/MAC layer

- Some sensor networks allow operating system to participate in radio transmission at bit-granularity → can get very accurate timing
- Some sensor networks use radio chipsets that handle packets and framing → lower timing resolution available to operating system
- Most wireless sensor networks now use randomized delay to manage fair access and collision management → variable delays make it more difficult to synchronize clocks

Delays between Sensor Nodes



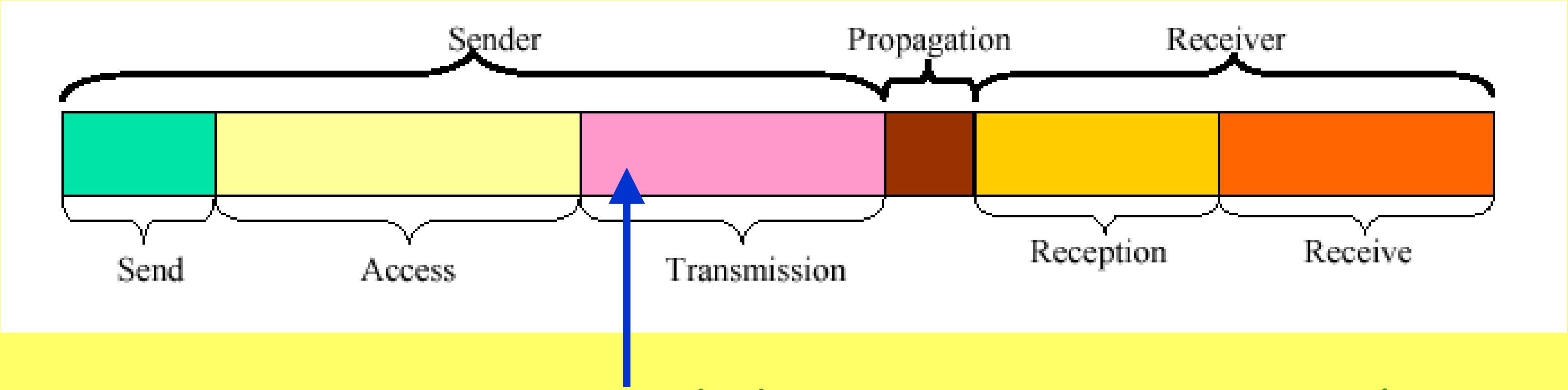
- Except for Access delay and multiprocess scheduling delays (not shown above), we can calculate the delays.
- Notice that propagation delay insignificant (reverse of Internet, satellite communication models)

Accounting for Delays

- Each sensor node can send "timesync" message to other node(s) --- message contains timestamp generated near the instant of sending message
- Receiver of timesync message can record local timestamp at instant of receiving message (and compensate for known delays) → enables sender/receiver synchronization
- Timestamping techniques depend on MAC protocol implementation

Technique #1 – low level timestamp

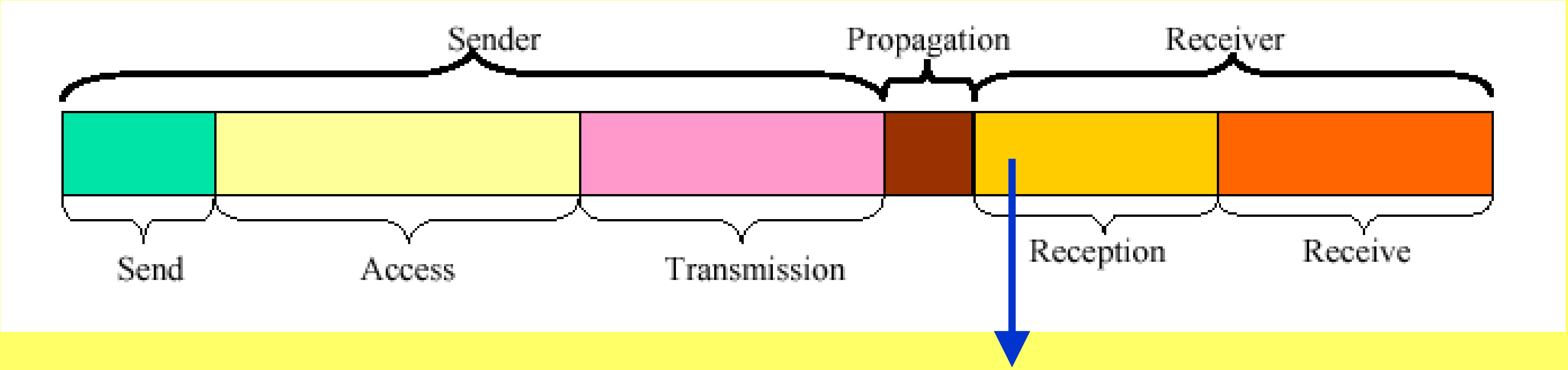
- If radio protocol stack allows system to interact with message/bit transmission, sender could generate timestamp very nearly the instant of transmission.



Timestamp generated during transmission (rate of transmission determines delay calculation)

Technique #1 – low level timestamp

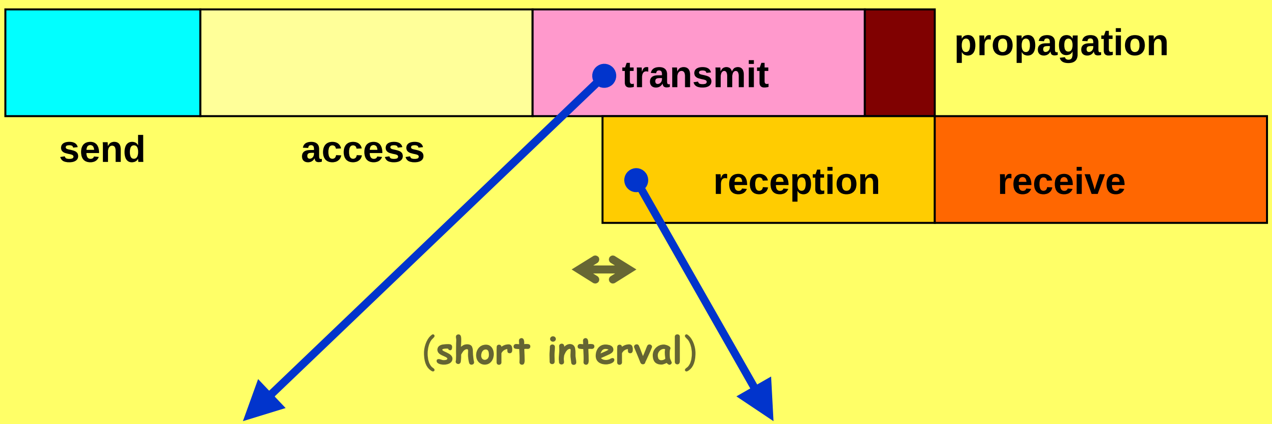
- Operating system also enabled to record current clock/counter during message reception



receiver's timestamp and sender's timestamp are very close in time, tight synchronization is possible

Technique #1 – “concurrent view”

- transmission and reception actually overlap

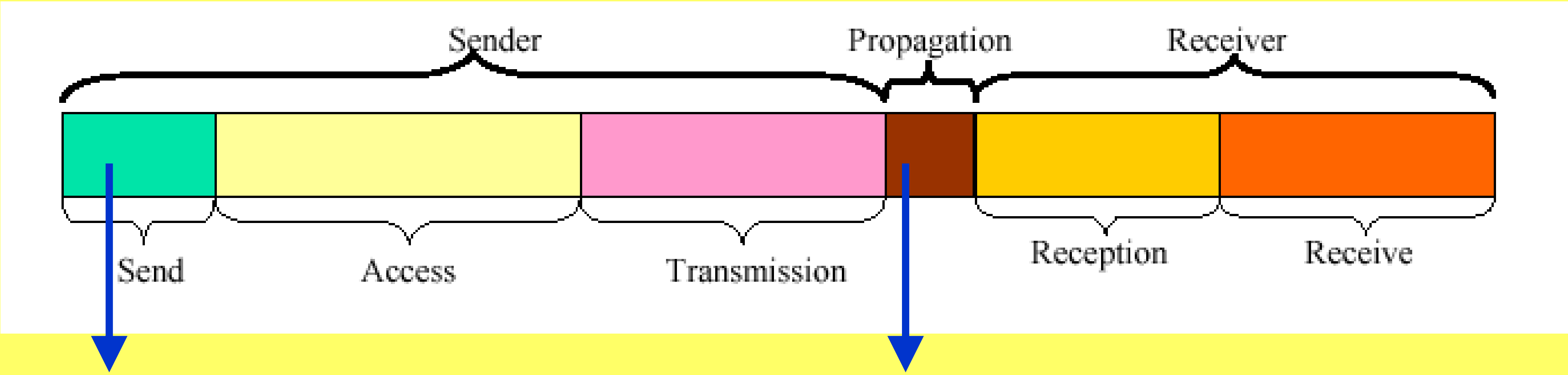


sender timestamp generated during transmit

receiver timestamp generated during reception

Technique #2 - delayed timestamp

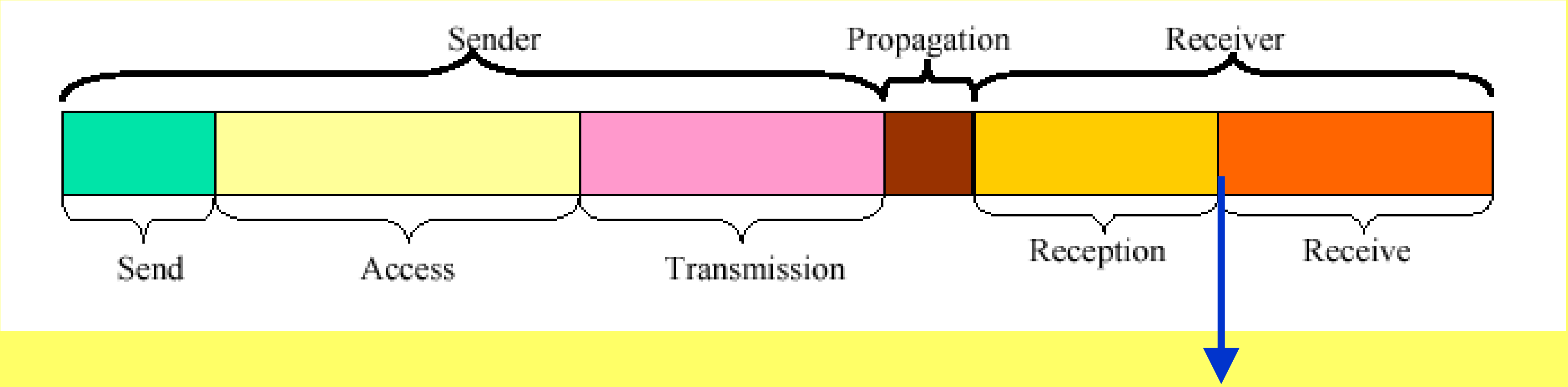
- Operating system also enabled to record current clock/counter just *after* message transmission



sender puts timestamp in message at time of send, then, too late, learns true timestamp at instant when transmission completes ☹

Technique #2 – delayed timestamp

- receiver records timestamp at instant after message received

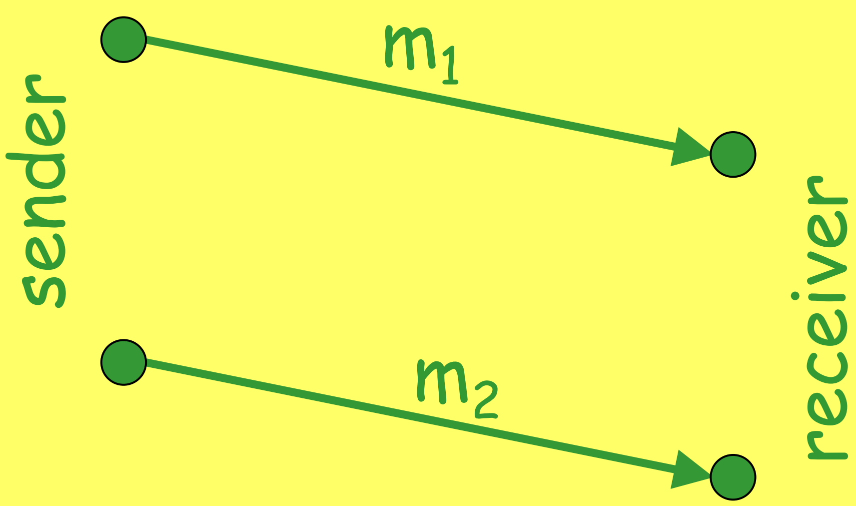


but receiver cannot trust timestamp contained in message from sender, because it was generated before access/transmission delays

what to do?

Technique #2 - delayed timestamp

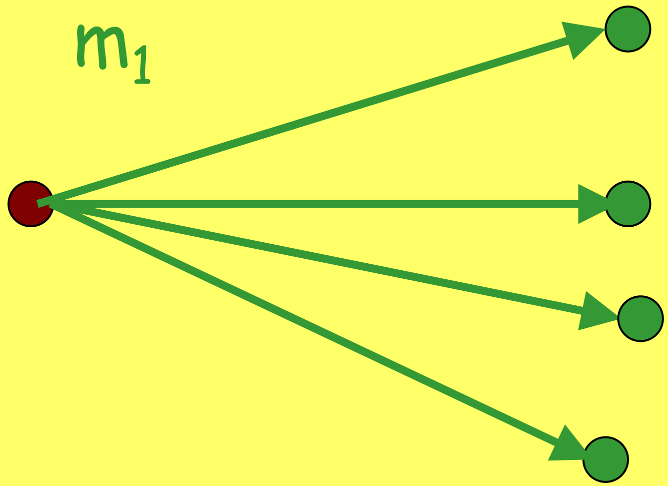
- correction part: use consecutive messages to account for delays



let m_2 contain timestamp correction when m_1 was finally transmitted, so receiver can determine corrected value for m_1's timestamp

Technique #3 – multiple reception

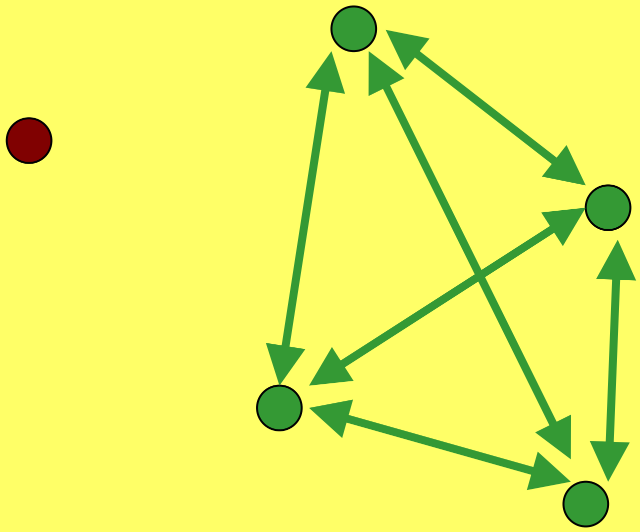
- when operating system cannot record instant of message transmission (access delay unknown), but can record instant of reception



in wireless sensor network, m_1 could be received simultaneously by multiple receivers: each records a timestamp value contained in m_1

Technique #3 - multiple reception

- after getting m_1, all receivers share their local timestamps at instant of reception



now, receivers come to consensus on a value for synchronized time: for example, each adjusts local clock/counter to agree with average of local timestamps

Technique #4 – filtering

- what if operating system cannot record timestamp at instant of message reception?
  - record timestamp as close as possible to reception
  - experimentally determine delay distribution
  - using model of distribution (Gaussian or other), calculate sampling size for desired confidence
  - iterate Technique #2/#3 to gather samples
  - use statistical techniques to reduce error, get accurate estimate of unknown delays

Comparison of Techniques

- #1 – timestamps during bit transmission → most accurate, but high "software overhead" and mixing of system/radio design
- #2 – timestamp at end of transmission → requires two consecutive messages, can be as accurate as #1, but is slower in adjustment
- #3 – multiple receivers (called RBS in literature) → considerable overhead for extra communication
- #4 – filtering (delay approximation) → more processing resource, but fewer system hacks

Presentation: Part I

conclusions

- sensor networks have variable synchronization requirements, so there can be multiple solutions to time synchronization
- traditional timekeeping protocols may not be the answer to how time synchronization should work on sensor networks
- Some low-level issues of communication and MAC protocols influence the design of neighborhood clock synchronization

remaining topics for Part II

what about multi-hop synchronization, scalability, robustness?