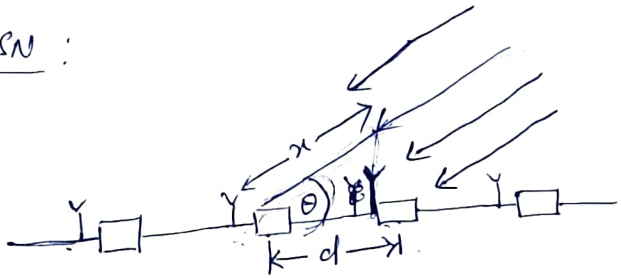
Time Synchronization:

Time is an important aspect for WSNs, nodes can measure time through local oscillators (clocks). Due to random phase shifts and drifts of oscillators, the local timing of nodes starts to differ and there is loss of synchronization. Time synchronization is a standard problem in distributed systems.

Need for time synchronization in WSN:

As shown in figure the acoustic wavefront generated by sound source a large distance away impinges onto array of acoustic sensors and the angle of arrival is to be estimated.



(gives x)

→ Each sensor knows its exact position and records time of arrival of sound event. When the length of d & x are known, the angle θ can be determined as $x = d\sin\theta$ or $\theta = \arcsin\frac{x}{d}$. The sensor distance d can be derived from known position of sensors and distance x can be derived from time difference Δ_t, bet^n sensor readings and speed c = 330 m/s. Using $x = c \cdot \Delta_t$: Let d = 1m, Δ_t = 0.001 sec. then $\theta \approx 0.336$ (in radians). If time difference is in range bet^n 500μs & 1500μs, the estimate for θ range bet^n $\theta \approx 0.166$ and $\theta \approx 0.518$. Therefore, a seemingly small error in time synchronization lead to significantly biased estimators.

→ There are two ways to get reliable timing information estimate. First one to keep sensors clocks as tightly synchronized as possible through dedicated time synchronization algorithms. The second one is to combine the readings of multiple sensors and "average out" the estimation errors.

Node clocks & the Problem of accuracy

→ Node possess an oscillator of specified frequency and a counter register, which is incremented in hardware after a certain number of oscillator pulses.

→ The node's software has access to the value of this register & time between two increments determine time resolution.

Let $H_i(t)$ be the value of hardware clock of node i at real time t. Then the local software clock $L_i(t)$ from this value is obtained through affine transformation to hardware clock as

$$L_i(t) = \theta_i H_i(t) + \phi_i$$

θ_i is drift rate and ϕ_i is phase shift. Coefficients θ_i and ϕ_i are changed to do clock adjustment.

External Synchronization : The nodes $1, 2, \ldots, n$ are said to be accurate at time t within a bound δ if

$$|L_i(t) - t| < \delta \quad \text{holds for all nodes } i \in \{1, 2, \ldots n\}$$

Internal synchronization : The nodes $1, 2, \ldots, n$ are said to agree on the time with a bound of δ if

$$|L_i(t) - L_j(t)| < \delta \text{ holds for all } i, j \in \{1, 2, \ldots n\}$$

To achieve external synchronization, a reliable source of real time or coordinated universal time must be available, for example GPS receiver. If nodes $1, 2, \ldots n$ are externally synchronized with bound δ, they are also internally synchronized with bound 2δ.

There are 3 problems:

(i) The phase ϕ_i is also random

(ii) oscillators have random deviation from nominal frequency, called drift.

(iii) oscillator freq. is also time variable implies time synchronization algorithms need to resynchronize once every few minutes to keep track of changing frequencies.

Time synchronization in WSNs

The requirements for time synchronizatn algorithms

→ An algorithm must scale to large multihop n/wks of severely energy constrained nodes.
→ Precision need to be diverse, ranging from microseconds to seconds.
→ Use of extra hardware removed due to cost.
→ Mobility is low
→ No fixed upper bound for packet delivery delay
→ Propagation delay betn neighboring nodes is negligible.
→

Clocks in WSN nodes:

Mostly a hardware clock is present.

→ The oscillator generates pulses at a fixed nominal frequency.

→ The counter register is incremented after a fixed no of pulses
  → only content of register is available to software
  → Register change rate gives achievable time resolution.

* Node i's register value at real time t is $H_i(t)$

→ The convention for real physical time (t) and capital T denote time stamps for anything else visible to nodes.

→ A (node-local) software clock is derived from hardware clock as

$$L_i(t) = \theta_i H_i(t) + \phi_i$$

θ_i and ϕ_i are the drift rate and phase shift respectively.

The time synchronization algorithms modify θ_i & ϕ_i but not the counter register.

Synchronization:

External Synchronization:

→ Synchronization with external real time scale

→ Nodes $i = 1, \ldots n$ are accurate at time t within bound δ when.

$|L_i(t) - t| < \delta$ for all i.

Hence, at least one node must have access to external time scale.

Internal synchronization:

→ No external timescale, nodes must agree on common time.

Nodes $i = 1, \ldots n$ agree on time within bound δ when

$|L_i(t) - L_j(t)| < \delta$ for all i, j.

Sources of inaccuracies:

→ Nodes are switched on at random times, Phase θ_i is also random.

→ Actual oscillators have random deviations from nominal frequency drift.

→ Oscillator frequency depends on time (Oscillator ageing) and environment (temp., pressure, supply voltage).

General Properties of time synchronization algorithms:

→ Physical time vs logical time

→ External vs internal synchronization

→ Global vs local algorithms
Keep all nodes of a WSN synchronized or only a local neighborhood.

→ Absolute vs relative time

→ Hardware vs software based mechanism.
A GPS receiver would be a hardware sol^n, but often too heavyweight/ costly / energy consuming in WSN nodes, and in addition a line-of-sight to at least four satellites is required.

→ A-priori vs a-posteriori synchronizat^n
Is time synchronization achieved before or after an interesting event?
Post-facto synchronization takes place after the important event.

→ Local clock update
Should backward jumps of local clocks be avoided.
Avoid sudden jumps?

Performance metrics for time synchronization:

Metrics: 1) Precision: Of is the max^m synchronized error for deterministic algorithms, / error mean / Std dev / quntiles for stochastic ones
2) Energy costs, eg. of exchanged packets, computational costs.

3) Memory requirements
4) Fault tolerance – what happens when node die?

Fundamental building blocks of time synchronization algorithms:

1) Resynchronization event detection block: When to trigger a time synchronization round? Whether periodically, ?, or after external event?
2) Remote clock estimation block: Figuring out the other nodes clocks with the help of exchanging packets.
3) Clock correction block: Compute adjustments for own local clock based on estimated clocks of other nodes.
4) Synchronization mesh setup block: Figure out which node synchronizes with which other nodes.

Constraints for Time synchronization (TS) in WSNs:

→ An TS algorithm should scale to large networks of unreliable nodes.
→ Algorithm must have diverse precision requirements, from ms to tens of sec
→ Use of extra hardware (like GPS receivers) is not the option.
→ low mobility
→ The packet delivery time don't have fixed upper bound (due to delays in MAC, buffering)
→ Negligible Propagation delay bet^n neighboring nodes.
→ Manual node configuration is not option (Adaptive, hence tuning is critical)

Post-facto synchronization:

→ Basic idea is as all nodes synchronized all the times results in significant costs due to need for frequent resynchronization. This if true when network is active rarely due to event occurance (Forest fire)
→ When the node observes the external event at time t, it stores it local timestamp $L_i(t)$, achieves synchronization with neighbor node/ Sink node & converts $L_i(t)$ accordingly.

Protocols based on Sender/receiver Synchronization

→ In these protocols, a receiver synchronizes to the clock of a sender. Two steps are considered

(a) Pairwise Synchronization: How a single receiver synchronizes to a single sender.

(b) Networkwide synchronizatn: How to figure out who synchronizes with whom to keep whole n/wk or parts of it synchronized.

LTS - Light weight Time synchronizatn.

→ Overall goal: Synchronize the clocks of all sensor nodes/ a subset of nodes to one reference clock (ref. clock equipped with GPS receivers)

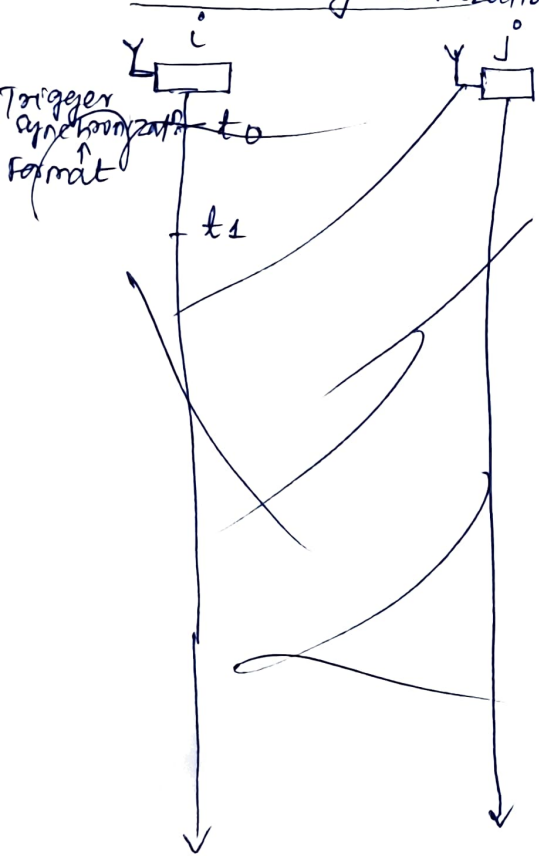
→ Allows to synchronize the whole n/wk, Parts of it and also supports post-facto synchronization.

→ It considers only Phase shifts and doesnot try to correct different drift rates.

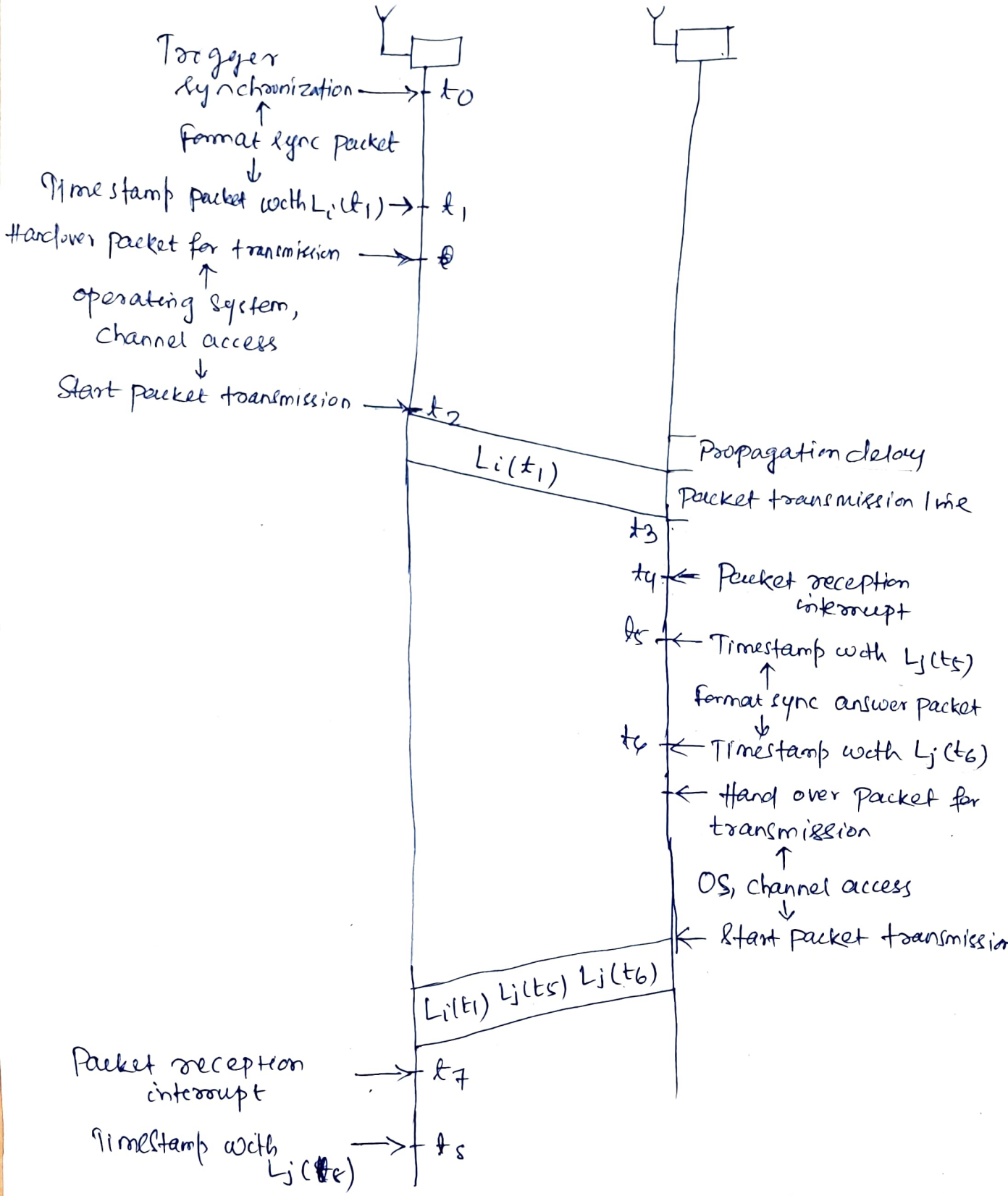
It has two components:

(i) Pairwise synchronization: Based on Sender/receive techniques.

(ii) Networkwide synchronization: Maint. spanning tree construction with reference node as root.



LTS Pairwise Synchronization



→ Here node i wants to synchronize its clock to node j's clock in steps:

(i) Node i triggers resynchronization at t_0, formats packet, timestamps it at t_1 with $L_i(t_1)$ and hands it over to transmission (with $L_i(t_1)$ as payload).

→ At t_2 the first bit appears on channel, at t_3 the receiver receives last bit, Packet reception signaled at t_4, and at t_5 node j timestamps it with $L_j(t_5)$.

→ Node j formats answer packet timestamps it at time t_6 with $L_j(t_6)$ and handover for transmission as payload $L_i(t_1)$, $L_j(t_5)$ & $L_j(t_6)$.

→ Arrival of answer packet is signaled at time t_7 to node i, and i timestamps it afterwards with $L_i(t_8)$.

→ After time t_8, node i possesses four values: $L_i(t_1)$, $L_j(t_5)$, $L_j(t_6)$ and $L_i(t_8)$ and estimates it's clock offset to node j.

Assumptions: If node i's aim is to estimate true offset $O = \Delta(t_1) = L_i(t_1) - L_j(t_1)$, assuming negligible drift the algorithm estimates $\Delta(t_5)$ and assume $\Delta(t_5) = \Delta(t_1)$. Hence node i estimates $\Delta(t_5) = L_i(t_5) - L_j(t_5)$, and estimates $L_j(t_5)$ i.e in between t_1 to t_8.

$$L_j(t_5) = \frac{L_i(t_1) + \tau + t_p + L_i(t_8) - \tau - t_p - (L_j(t_6) - L_j(t_5))}{2}$$

$$\therefore O = \Delta(t_5) = L_i(t_5) - L_j(t_5) = \frac{L_i(t_8) + L_i(t_1) - L_j(t_6) - L_j(t_5)}{2}$$

Discussion:

1) Node i figure out offset to node j based on $L_i(t_1)$, $L_j(t_5)$, $L_j(t_6)$, $L_i(t_8)$
2) This exchange takes two packets.
3) The uncertainty is in the interval $I = [L_i(t_1) + \tau + t_p, L_i(t_8) - \tau - t_p - (L_j(t_6) - L_j(t_5))]$

LTS - Network wide Synchronization:

Here one reference node R is given to which all other nodes/subset of nodes wants to synchronize.

→ R's direct neighbors (level-1 neighbors) synchronize with R
→ Two-hop (level-2) neighbors synchronize with level-1 neighbors.

→ This way a spanning tree is created.

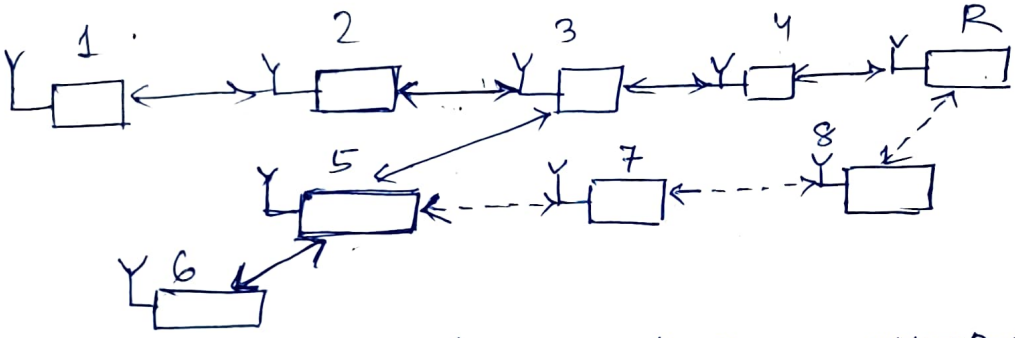
Considering a node i with hop distance h_i to the root node.

Assumption are : all synchronization errors are independent, ii) all synchronization errors are identically normally distributed with zero mean and variance $\gamma\sigma^2$.

Two possibilities

Centralized multihop LTS : Here reference node R triggers construction of spanning tree, it first synchronizes its neighbors, then the first level neighbors synchronize second-level neighbors and so on.

Distributed Multihop LTS :- No explicit construction of spanning tree needed, but each node knows identity of reference nodes and routes to them.



→ When node 1 wants to synchronize with R, an appropriate request travels to R - following that 4 synchronizes to R, 3 synchronizes to 4, 2 synchronizes to 3, 1 synchronizes to 2. By-product nodes 2,3 and 4 synchronized with R.

→ when node 5 wants to synchronize with R it issue its own synchronization request using route over 3,4 and put additional synchronization burden on them. Therefore, it asks in neighbourhood whether someone is synchronized or has ongoing synchronization request. Hence, it uses path over 7 & 8 to also synchronize those nodes.

Protocols based on receiver/receiver synchronization

→ In these protocols receivers of packets synchronize among each other not with transmitter of the packet

RBS is called: RBS (Reference Broadcast Synchronization)

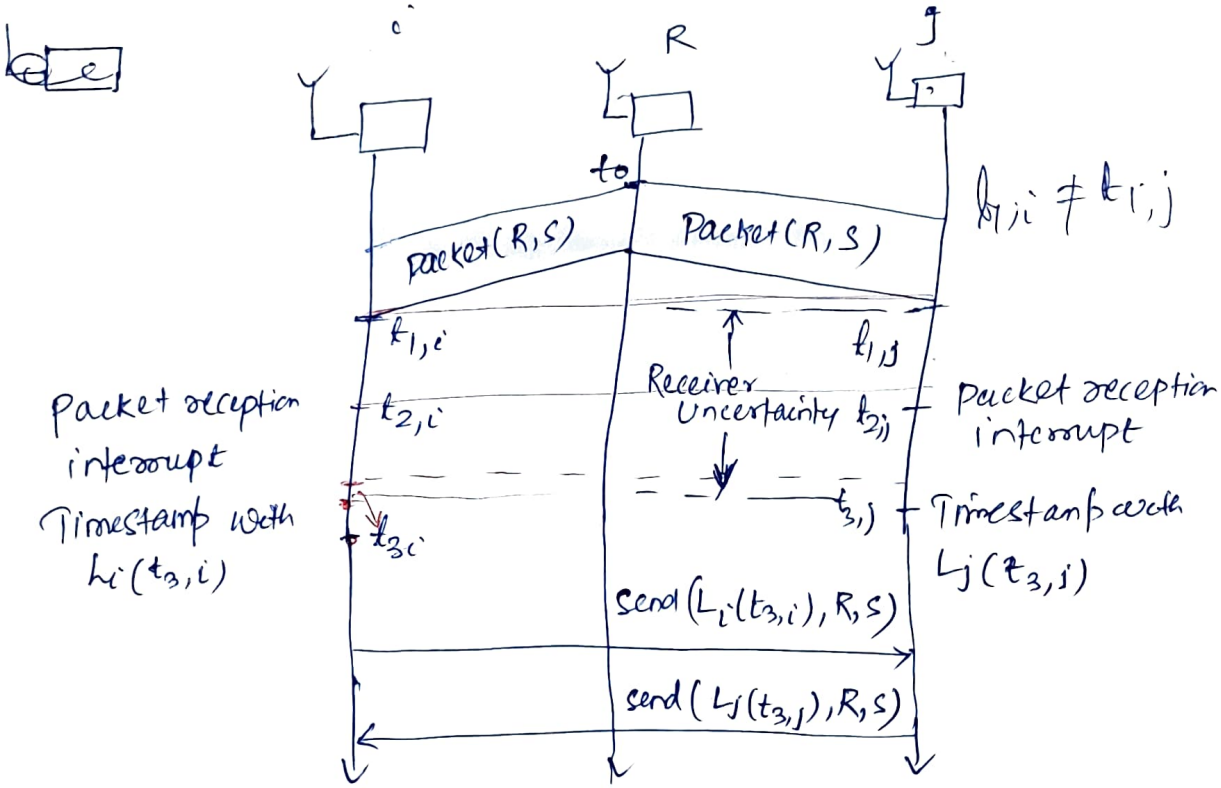
→ RBS has 2 components

(i) synchronize receivers within a single broadcast domain

(ii) A scheme for relating timestamps betⁿ nodes in different domains.

→ RBS doesnot modify the local clocks of nodes, but computes a table of conversion Parameters for each peer in broadcast domain

→ RBS allocos Post facto synchronization.



The goal is to synchronize i's and j's clock to each other

Timeline:

→ Reference node R broadcasts at time t_0 some synchronization packet carrying it's identification R and a sequence numbers.

→ Receiver i receives the last bit at time $t_{1,i}$ gets the packet interrupt at time t_{2i} and time stamps it at $t_{3,i}$.

→ Receiver j is doing the same.
→ After some time node i transmits its observation $(L_i(t_{3,i}), R, S)$ to node j
→ After some time node j transmits its observation $(L_j(t_{3,j}), R, S)$ to node i.
→ This procedure is repeated periodically, the reference node transmits its synchronization packets with increasing sequence numbers

Under the assumption $t_{3,i} = t_{3,j}$ node j can find the offset $O_{i,j} = L_j(t_{3,j}) - L_i(t_{3,i})$ after receiving node i's final packet

RBS - synchronizatn. in broadcast domain:
~~the synchronizatn error in this~~ scheme has 2 causes.

→ There is a difference betn $t_{3,i}$ and $t_{3,j}$
→ Drift betn. $t_{3,i}$ and time coherc. node i transmits its observations to j.

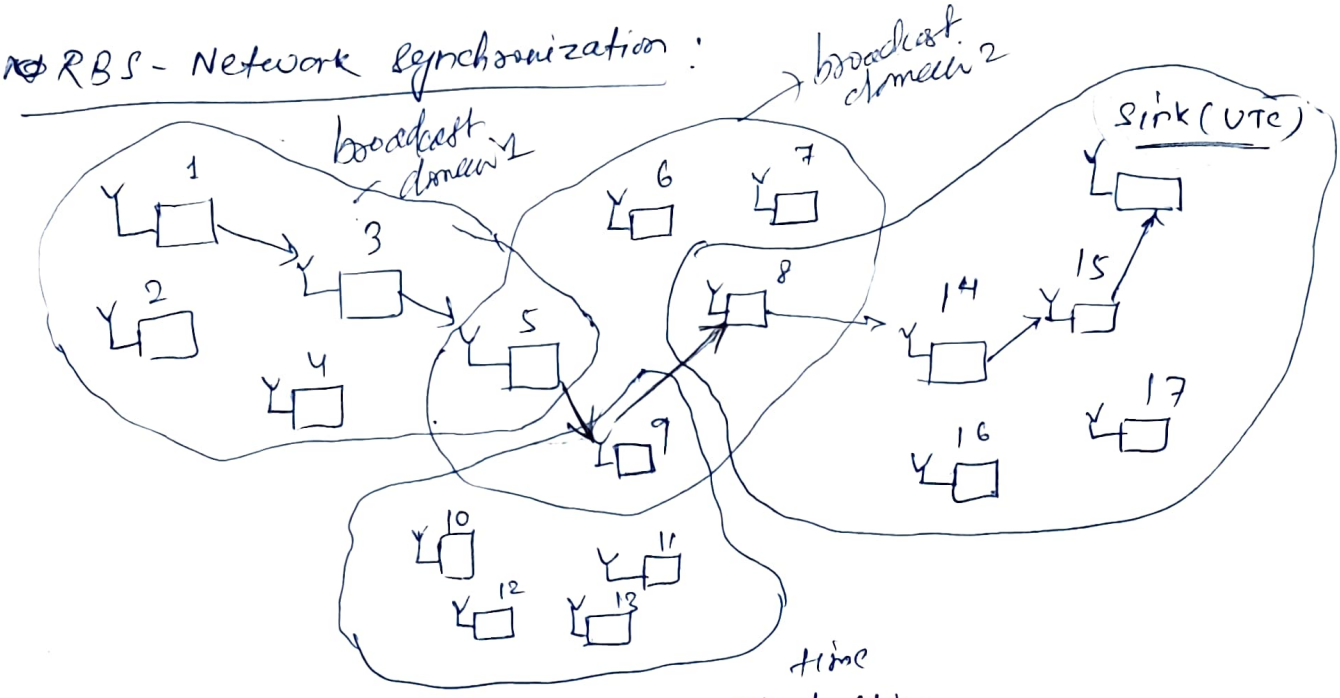
But
→ in small broadcast domains & when received packets are timestamped as early as possible the difference betn $t_{3,i}$ and $t_{3,j}$ is very small
• As compared to sender/receiver based schemes where MAC delay and operating system delays experienced by the reference node play no role
→ Drift can be neglected when observations are exchanged quickly after reference packets.
→ Drift can be estimated jointly with offset O when a number of periodic observations of $O_{i,j}$ have been collected.

Different comm^n. costs are.
Let m the no. of nodes in broadcast domain
First scheme: Reference node collects the observations of the nodes, computes offsets & sends them back → 2m packets.
second scheme: Reference node collects the observation of the nodes, computes the offset and keep them, but it has responsibility for timestamp conversions and forwarder selection → m packets.
Third scheme: Each node transmits its observation individually to other members of broadcast domain → m(m-1) packets

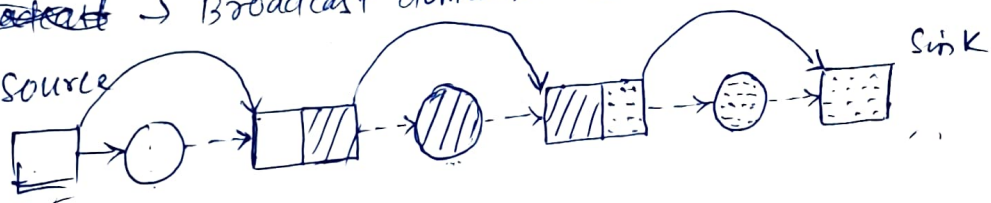
Fourth Scheme : Each node broadcasts its observation → n packets, but unreliable delivery.

* Collisions are a problem

RBS - Network Synchronization :



→ Let node 1 has detected an event at $L_1(t)$. (time)
→ Sink is connected to a GPS receiver and has UTC timescale.
→ Node 1 wants to inform the sink about event such that sink receives a timestamp in UTC timescale.
→ Broadcast domains are indicated by circles.



Timestamp conversion approach:

→ idea : Donot synchronize all nodes to UTC time, but convert timestamps as packet is forwarded from node 1 to the sink → avoids global synch.
→ Node 1 picks node 3 as forwarder - as both are in same broadcast domain, node 1 can convert the time stamp $L_1(t)$ into $L_3(t)$.
→ Node 3 picks node 5 in same way.
→ Node 5 is member in two broadcast domains and knows also the conversion parameters for next forwarder 9. and so on.
→ Result: The sink receives a timestamp in UTC timescale.
→ Nodes 5, 8, and 9 are gateway nodes.