

Clock Synchronization

Marco Platania

www.dis.uniroma1.it/~platania
platania@dis.uniroma1.it

Sapienza University of Rome

Time notion

- Each computer is equipped with a **physical (hardware) clock**
- It can be viewed as a counter incremented by ticks of an oscillator
- At time t , the Operating System (OS) of a process i reads the hardware clock $H(t)$ of the processor
- Then, it generates the **software clock** $C = a H(t) + b$
- C approximately measures the time t of process i

Time in Distributed Systems (DS)

- Time is a key factor in a DS to analyze how distributed executions evolve
- Problems:
 - Lacking of a global reference time: it's hard to know the state of a process during a distributed computation
 - However, it's important for processes to share a common time notion
- The technique used to coordinate a common time notion among processes is known as **Clock Synchronization**

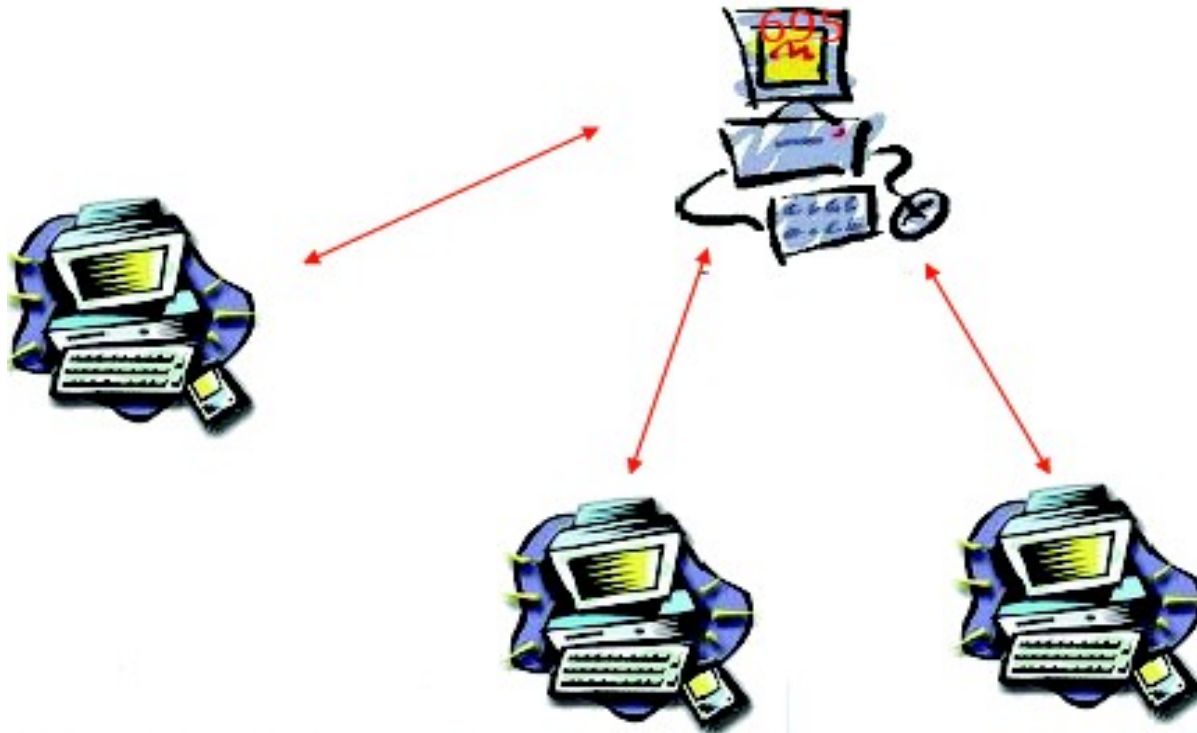
Clock Synchronization [10]

- The hardware clock of a set of computers (system nodes) may differ because they count time with different frequencies
- Clock Synchronization faces this problem by means of synchronization algorithms
 - Standard communication infrastructure
 - No additional hardware
- Clock synchronization algorithms
 - External
 - Internal
 - Heartbeat (pulse)

External Clock Sync

- Minimizes the distance between clock C_i of process $i = 1, 2, \dots, n$ and the time provided by an authoritative time source S , within a synchronization bound Δ :

$$|C(t) - S(t)| \leq \Delta$$



Internal Clock Sync

- No external reference time server
- Minimizes the distance of any two clock C_i and C_j of processes $i, j = 1, 2, \dots, n$ within a synchronization bound Δ :

$$| C_i(t) - C_j(t) | \leq \Delta$$

- Notes:
 - A set of processes P externally synchronized within a bound Δ is also internally synchronized within a bound 2Δ
 - It follows directly from definition of external and internal clock synchronization
 - A set of process P internally synchronized are not necessarily externally synchronized because they may drift collectively from the external time source

Heartbeat (pulse) sync

- Aims to synchronize, for each node, the time in which a synchronization round starts, rather than the clock value
- System nodes periodically generate a local heartbeat approximately in the same instant
- They start and finish a synchronization round approximately in the same instant, with an error that is at least one order of magnitude smaller than the round length

Clock sync algorithms

- Clock sync algorithms are divided in three groups
 - Deterministics
 - Probabilistics
 - Statistics
- Deterministics algorithms assume the presence of an upper bound on transmission delay and guarantee an upper bound on the difference between two any clocks (**precision**)

Probabilistic algorithms

- No assumption on transmission delay, but they guarantee a constant maximum deviation between synchronized clocks
- At any time, a process knows if its clock is synchronized with others
- But there is a probability greater than 0 that a process clock goes out of synchronization if a lot of communication errors occurs (messages loss)

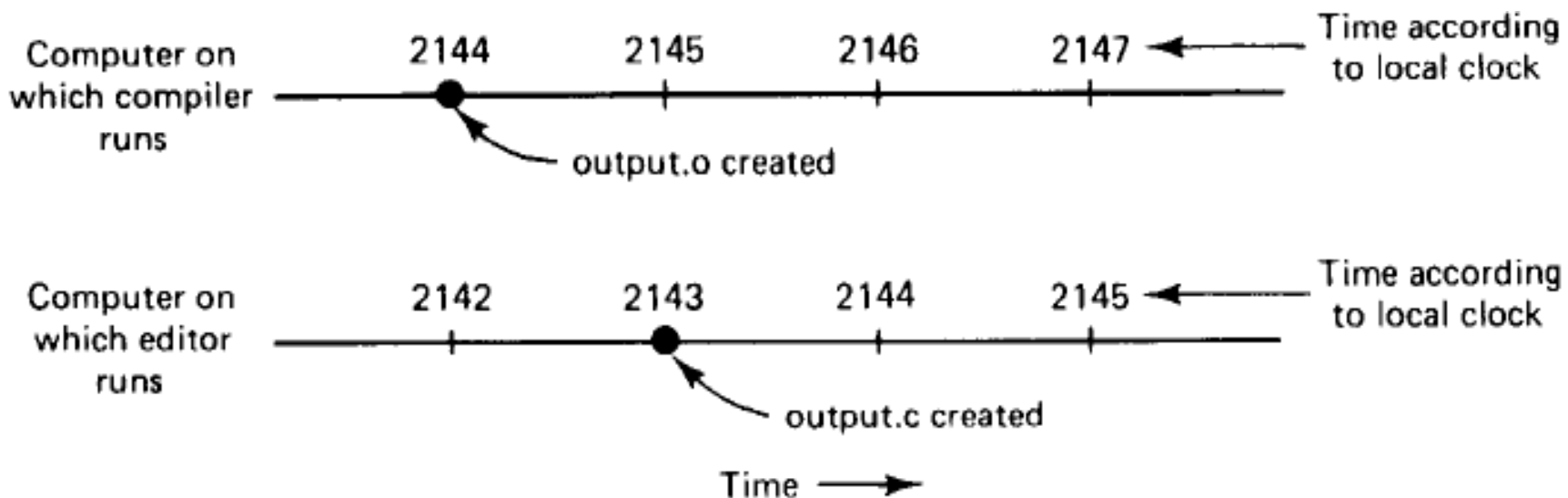
Statistics algorithms

- Assumes that the standard deviation and the expected value of the transmission delay are known
- Processes don't know how far their clocks are from others
- Anyway, they guarantee that at each time two any clocks are within a given constant maximum deviation with a certain probability
- Then, in probabilistics and statistics algorithms precision can be guaranteed only with a non zero probability of failure

Why clock sync is important? An example

- UNIX *make* command is used to compile source code
- Typically, a large UNIX program is splitted in several files
- A change to one source file only requires that file to be recompiled
- *make* goes through all the source files to find out which ones need to be recompiled
- It examines the time in which source and object files were last modified
- If the source file was last modified after the object file, *make* calls the compiler to recompile it

- Let's suppose to have editor and compiler running on two different machines, with no global agreement on time
- Suppose that output.o has time 2144 and shortly after output.c is modified but is assigned time 2143 because the clock on that machine is slower
- *make* will not call the compiler
- The executable program will contain a mixture of code that probably will lead to an unpredictable behavior
- Lack of synchronization is dramatic!



Some definition

■ Scale of the system

■ LAN (Local Area Network)

- typically composed by few hundreds of nodes
- It allows a deep data traffic analysis in order to derive an upper bound on the transmission delay
- In the following we'll consider clique-based LANs (each node is connected to every other)
- Clique is not scalable: adding new nodes augments the load of the system

■ WAN (Wide Area Network)

- Up to millions of nodes
- No data traffic analysis is feasible
- Transmission delay finished but unknown
- Random graph topology: each node is connected to a subset of nodes in the network
- High scalability: a node interacts only with its neighbors

■ Failures

■ None

- reliable system

■ Crash-stop

- a crashed node leaves the system forever

■ Byzantine

- a byzantine node is one that arbitrarily either follows correctly the algorithm, or executes different steps. In clock sync, byzantines aim to destroy the system convergence

■ Dynamism

■ Static network

- Nodes leave the network only due to crashes

■ Dynamic network

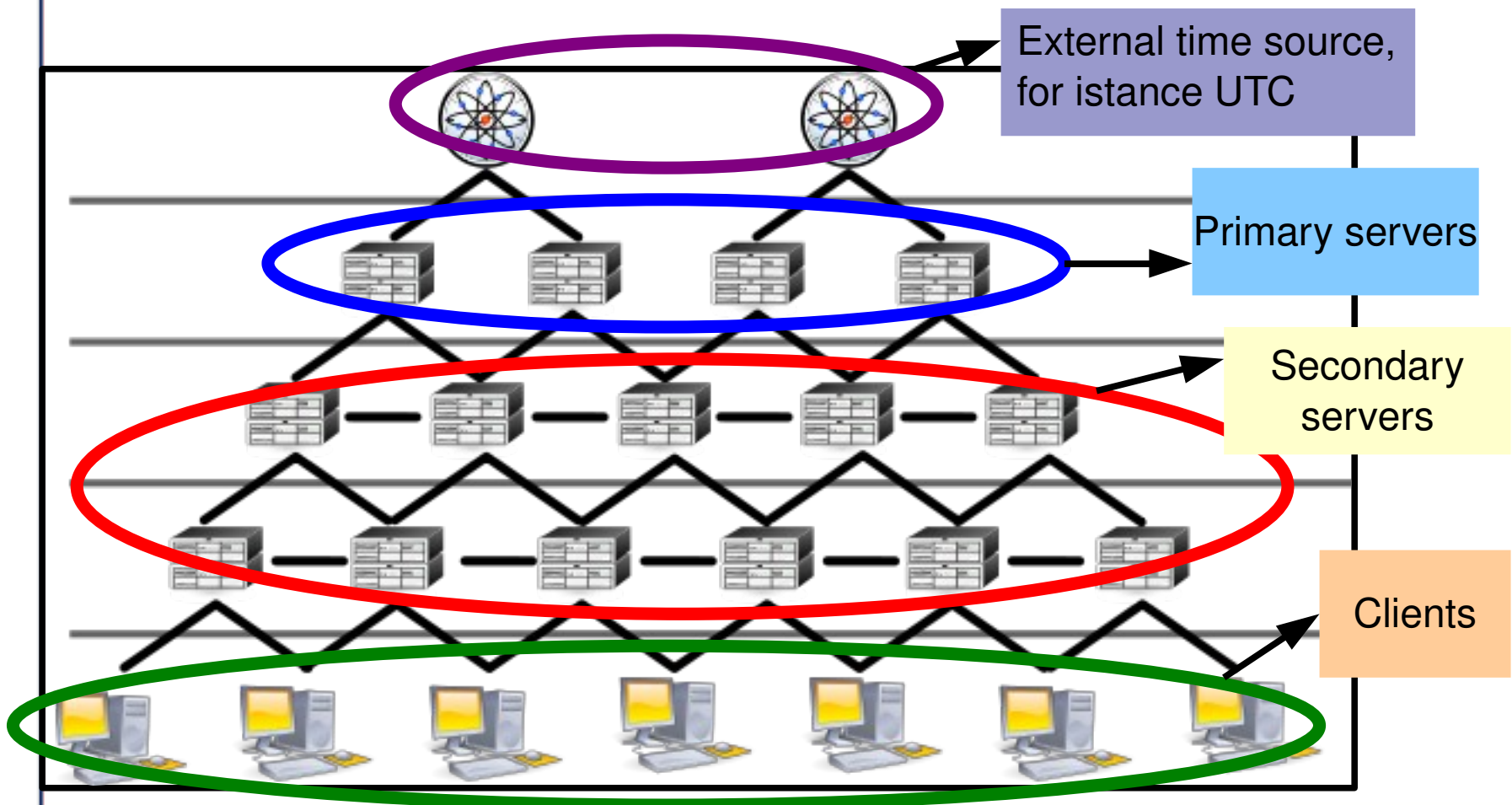
- Nodes can join and leave the network at any time, voluntarily or due to crashes

Clock sync in literature: a state of the art

| Authors | Algorithm Paradigm | Synchronization Paradigm | Fault Tolerance | Communication Topology | Assumption on message delay |
|--------------------|--------------------|--------------------------|-----------------|-----------------------------|-----------------------------|
| Lamport | Deterministic | Internal | None | Clique | Known bound |
| Cristian | Probabilistic | Internal | Byz. (3F+1) | Star (Master-Slave) | Known distribution |
| Mills (NTP) | Probabilistic | External | Byz. (1) | Hierarchical (Master-Slave) | Known distribution |
| Rodrigues et al. | Probabilistic | External | Crash | Hierarchical (Master-Slave) | Known distribution |
| O. Babaoglu et al. | Probabilistic | HeartBeat | Crash | Random Graph | None |
| Van Steen et al. | Probabilistic | External | Crash | Random Graph | None |
| Baldoni, et al. | Probabilistic | Internal | Byz. Prob. | Random Graph | None |
| D. Dolev et al. | Deterministic | HeartBeat | Byz. (3F+1) | Clique | Known bound |

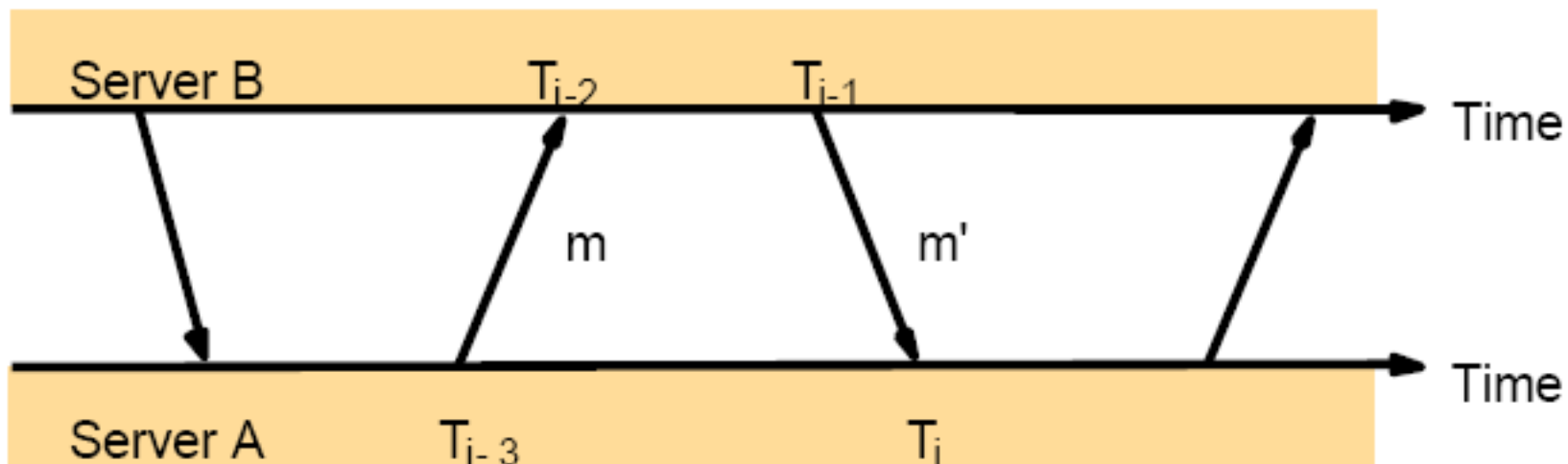
NTP – Network Time Protocol [7]

- Provides external synchronization
- The most used algorithm both in LAN and in WAN
- It is based on a hierarchical structure



- Primary servers are directly connected to external time source
- Secondary servers are connected to one or more primary servers
- Secondary servers read time from primary servers and distribute the information to clients
- Time estimation
 - For each pair of messages exchanged by two servers, NTP estimates the total transmission time as:

$$T_{i-2} - T_{i-3} + T_i - T_{i-1}$$



■ Fault tolerance

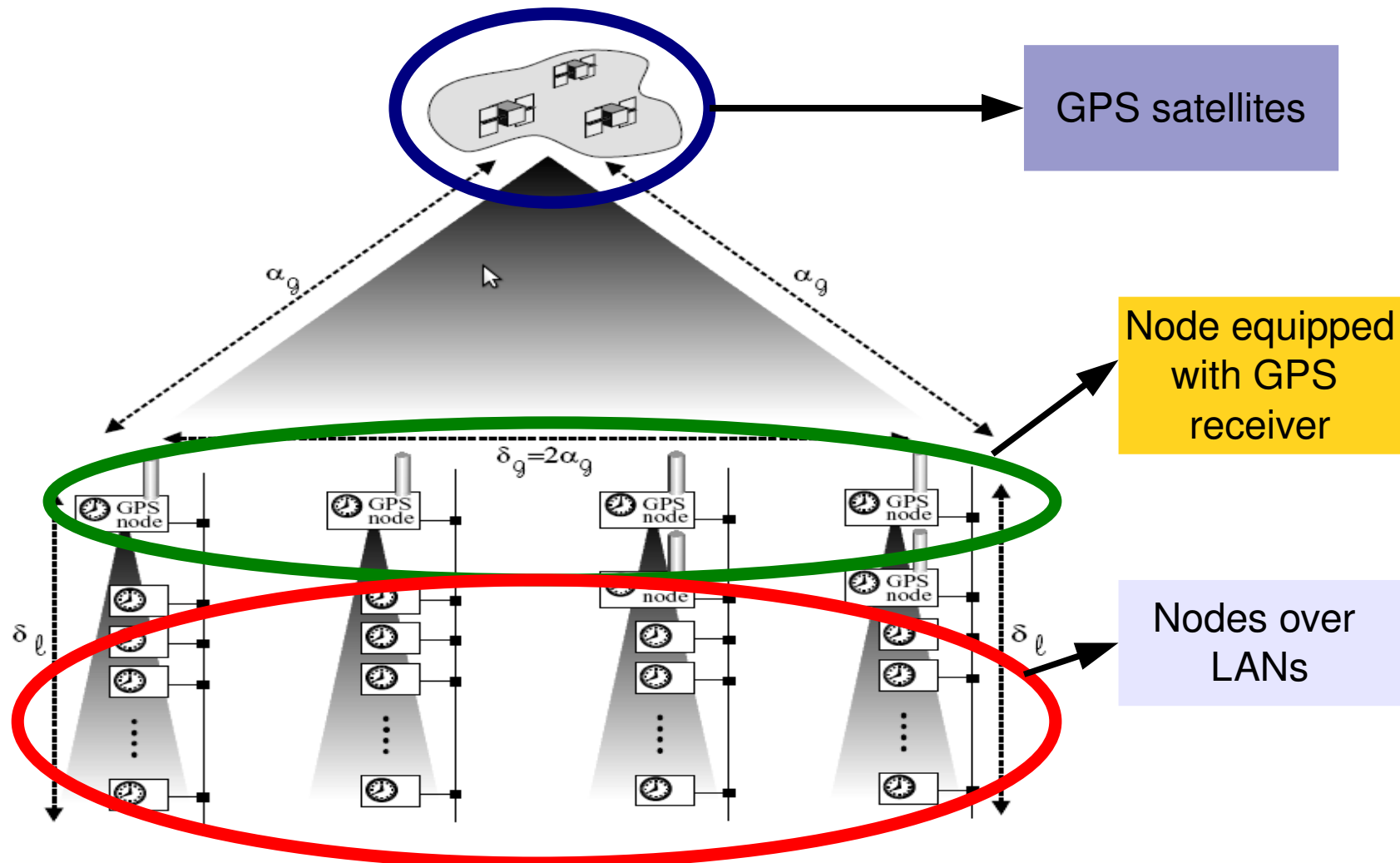
- If a primary server disconnects from the time source, then it becomes a secondary server
- If a secondary server loses connection from a primary server, then it connects to another primary server
- NTP tolerates byzantine failure by means of servers replication and cryptography

■ Accuracy of NTP

- 10 milliseconds on Internet paths
- 1 millisecond in LAN

CESIUM SPRAY – Rodrigues et al. [8]

- Hybrid external/internal synchronization
- Uses a hierarchical structure based on WANs of LANs



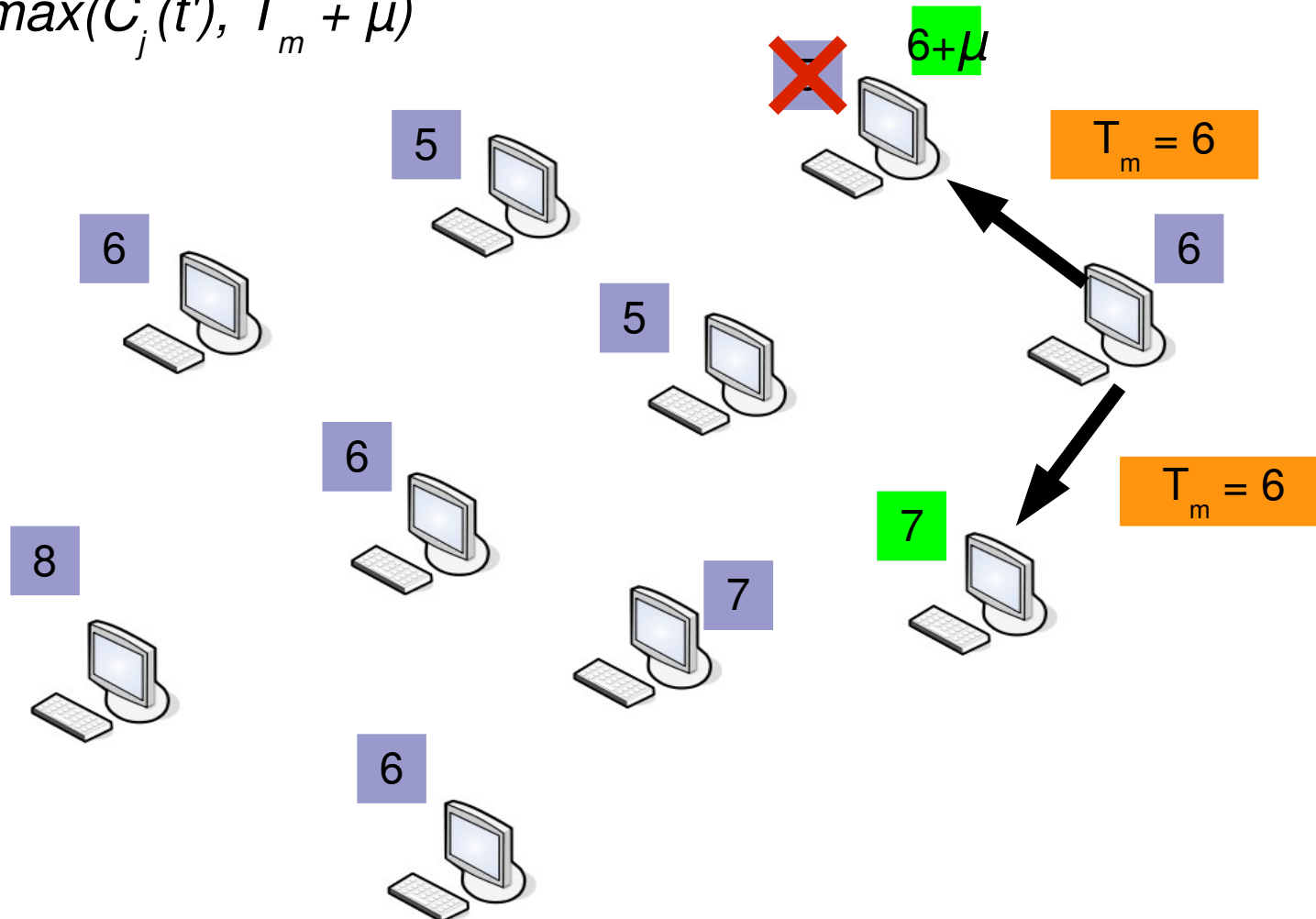
- Assumption: each LAN is equipped at least with a node with GPS receiver (GPS node)
- GPS satellites “spray” time information over GPS nodes
- GPS nodes further “spray” time information over LANs through broadcasts (an error-free broadcast is guaranteed to occur)
- Nodes within LANs apply a function (average, median, ...) to the set of value obtained to update the clock value
- WANs of LANs approach:
 - Accuracy of synchronization in LANs even if a disconnection from time source occurs
 - Low error rate
 - Transmission delay bounded
 - Transmission delay negligible without errors
- Fault tolerance
 - Replication of GPS nodes for external synchronization
 - Enough node in LANs for internal synchronization

Lamport [1]

- Internal clock synchronization
- Reliable LAN (no failures)
- Clique-based topology
- Synchronization required only for interacting nodes
 - Lacking of synchronization is not observable for non interacting nodes
- Idea: agreement not on the clock value, but on the order of events
- Assumption:
 - The recipient of a message knows a minimum value for the transmission delay $\mu \geq 0$

Algorithm:

- Process i sends a message m at time t with timestamp $T_m = C_i(t)$
- Process j receives m at time t' and sets its clock $C_j(t')$ as $\max(C_j(t'), T_m + \mu)$



Cristian e Fetzer [2]

- Provides accurate internal clock synchronization
- The following two conditions hold:
 - At any time, the deviation between two correct clocks is bounded by a constant δ ([maximum deviation](#))
 - Clocks drift rate bounded by a constant $r \gg 1$
- Assumption
 - Clique-based topology
 - At most f byzantine nodes
 - At least $3f + 1$ nodes in the system
 - Remote procedure method to read clock values
 - Bounded error in clock reading
 - Round-based algorithm
 - Eventually, each process goes through the next round
 - Initially any two correct clocks are within a bound d

■ Algorithm:

- Each process maintains a clock value that is the sum between the hardware clock $H(t)$ and an adjustment A

$$C(t) = H(t) + A$$

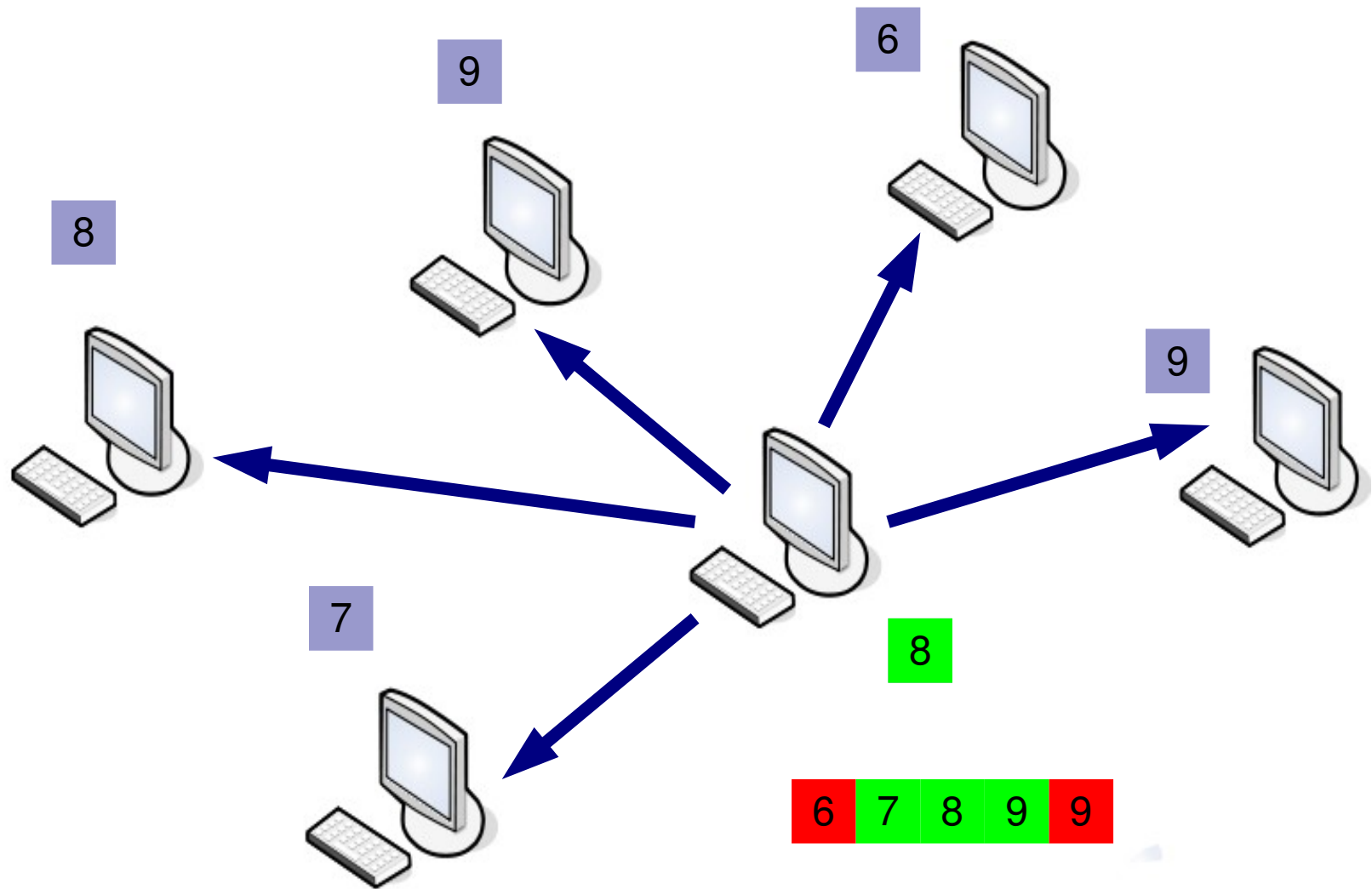
- At each round, any node:

- Reads the clock values of all other nodes through the remote procedure method
- Inserts all read values in a sorted list
- Discards the f highest and lowest values because there are at most f byzantine failures
- Applies a midpoint function on remaining $f + 1$ values

$$mid(x,y) = (x + y) / 2$$

where x and y are the extreme values of the interval

- Modifies A according to the value obtained by the midpoint function
- Algorithm ensures an optimal maximum deviation
- However, the clique-based topology makes the algorithm not scalable



Why at least $3f + 1$ nodes in the system?

- With cryptographic primitives just the majority of correct processes is needed to let the algorithm converge ($2f + 1$)
- Without cryptographic primitives $3f + 1$ nodes in order to discard the f highest and lowest and to maintain only correct processes
- Among the $2f$ discarded nodes, f are correct
- Trade-off: best removing a portion of correct nodes in order to discard all faulty nodes
- Formal proof [3]

“Self stabilizing” pulse sync – Dolev et al. [4]

- Aims to synchronize the beginning and the end of a synchronization round of correct nodes
- Self stabilizing: despite clock values initially sparse, eventually nodes will reach a stable state, from which they proceed at unison
- Clique-based topology
- At least $3f + 1$ nodes in the system to tolerate up to f byzantine faults
- Definitions
 - Pulse: re-synchronization event
 - Round: time interval I between two following pulses
 - Two correct nodes i and j are pulse synchronized if:
$$| I_i(t) - I_j(t) | \leq \Delta$$
 - It is similar to the internal clock sync condition

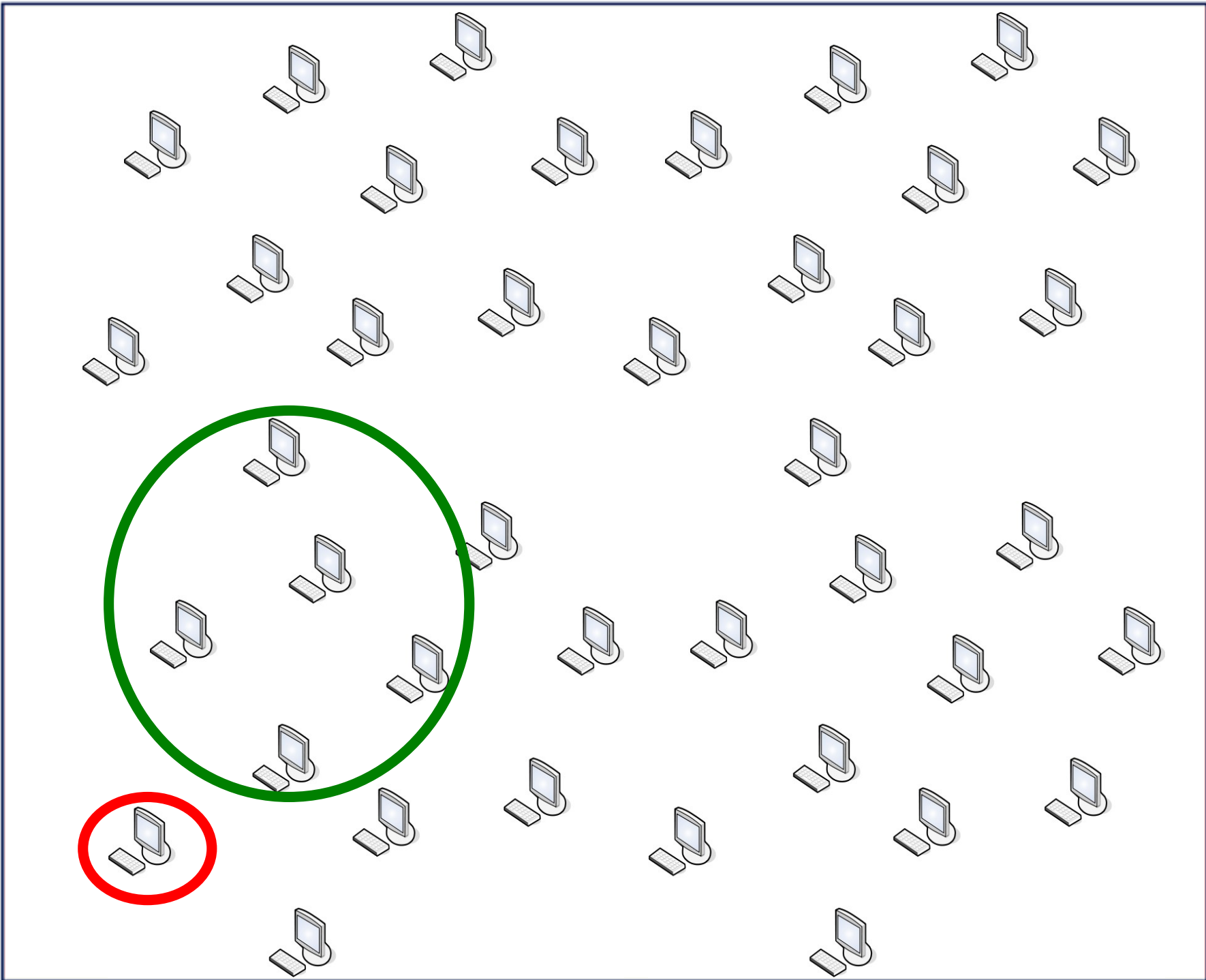
- The algorithm is composed by two main blocks
 - Pulse-sync: two way to send pulse proposals
 - Timeout expiring or at the reception of $f + 1$ different proposals
 - Each proposal is broadcasted
 - At the reception of $n - f$ different proposals, a pulse event is invoked and the timeout is resetted
 - After few rounds, nodes pulse at unison
 - Consensus
 - After a pulse event, a consensus request is invoked
 - At the reception of $n - f$ messages, a node decides the value of its clock
- Drawback: consensus increases the number of messages sent during a computation than other clock synchronization algorithms

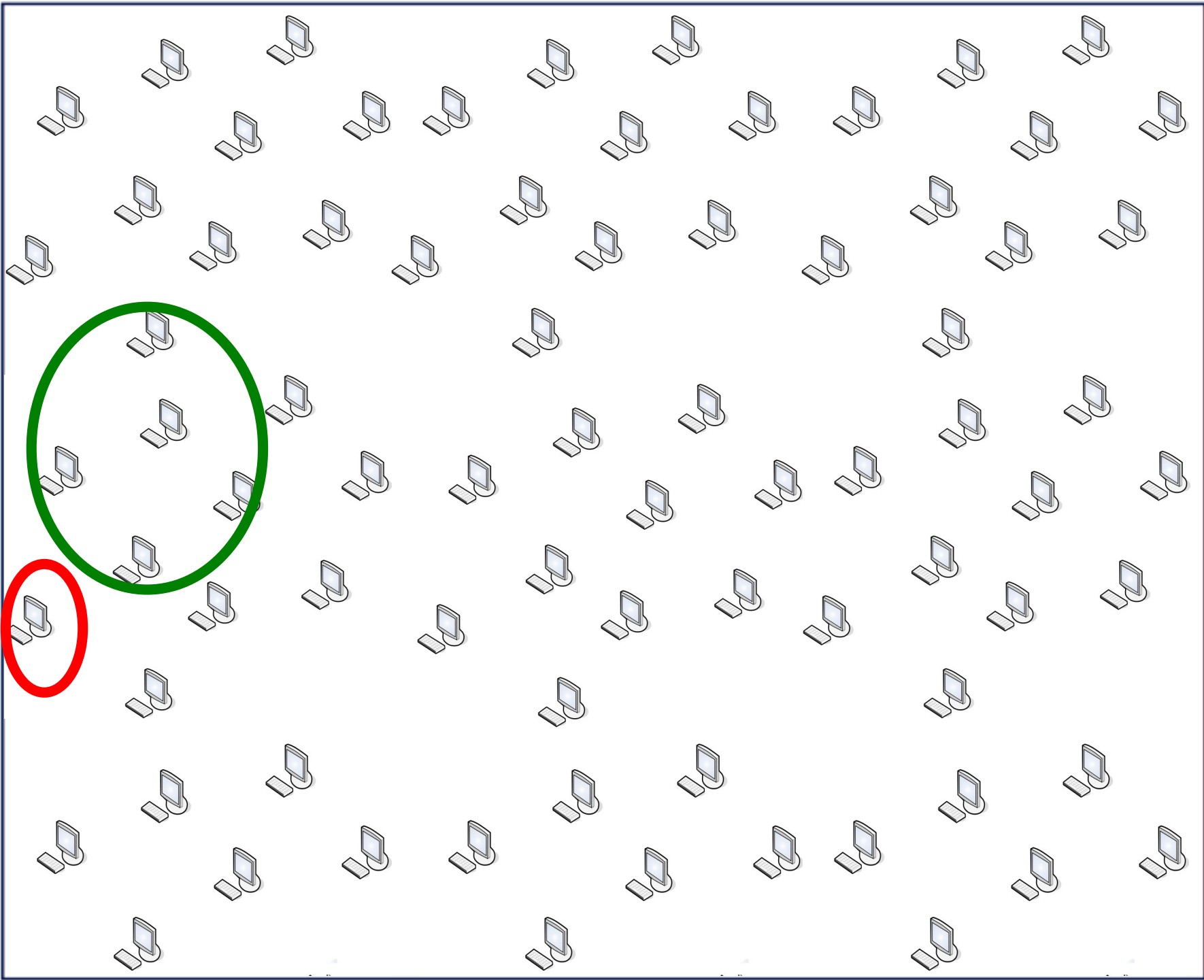
Clock sync in Large Scale Dynamic Systems

- Up to now, we studied external clock synchronization or clock synchronization in LAN environment
- Now we concentrate on clock synchronization in Large Scale Dynamic Systems
- We focus on
 - Environment
 - Problems
 - Dynamic
 - Security
 - Methodologies
 - Gossip protocol
 - Peer Sampling Service
 - Their use in clock sync algorithms

Environment

- Clique-based topology is not scalable
- Scalability is a fundamental issue in large systems
- Solution: random graph topology
- Random graph
 - Each node is connected to a portion of the system
 - It has a partial knowledge
 - It interacts only with its neighborhood
 - If the size of the system increases, the load is not increased
 - The best to face churn
- Transmission delay finite but unknown





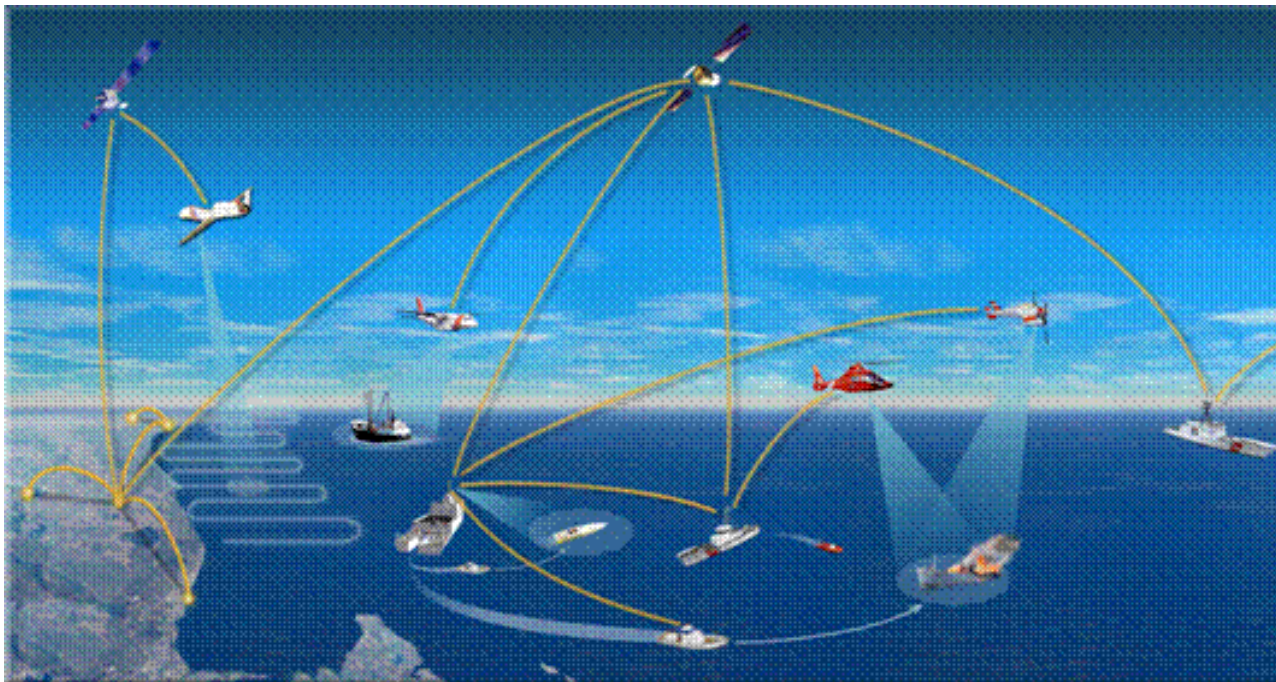
Dynamic

- In large scale systems applications have to
 - Be scalable
 - Be churn resilient
 - Make no assumption on the network infrastructure
- This suggests the Peer-to-Peer (P2P) approach
 - No centralized point, scalability improvement
 - No costly infrastructure, direct communication among peers
- In this kind of system, the clock synchronization problem is not yet completely solved

Clock sync in large scale systems

- A fundamental requirement to ensure Quality of Service (QoS)
 - Timeliness: time constraint on message delivery
 - Ordering: ordered communication
- Dynamic is one of the main challenges because makes harder the synchronization process

- A particular scenario: smart environment
 - Composed by heterogeneous devices
 - Fixed: PCs, workstations, ...
 - Mobile: laptops, PDAs, sensors, ...
 - It can be classified according to two parameters
 - Dynamic in time: leaves/joins modifies network structure
 - Dynamic in space: mobile devices move from a subnetwork to another one
 - Dynamic in space seems to be the more attractive challenge

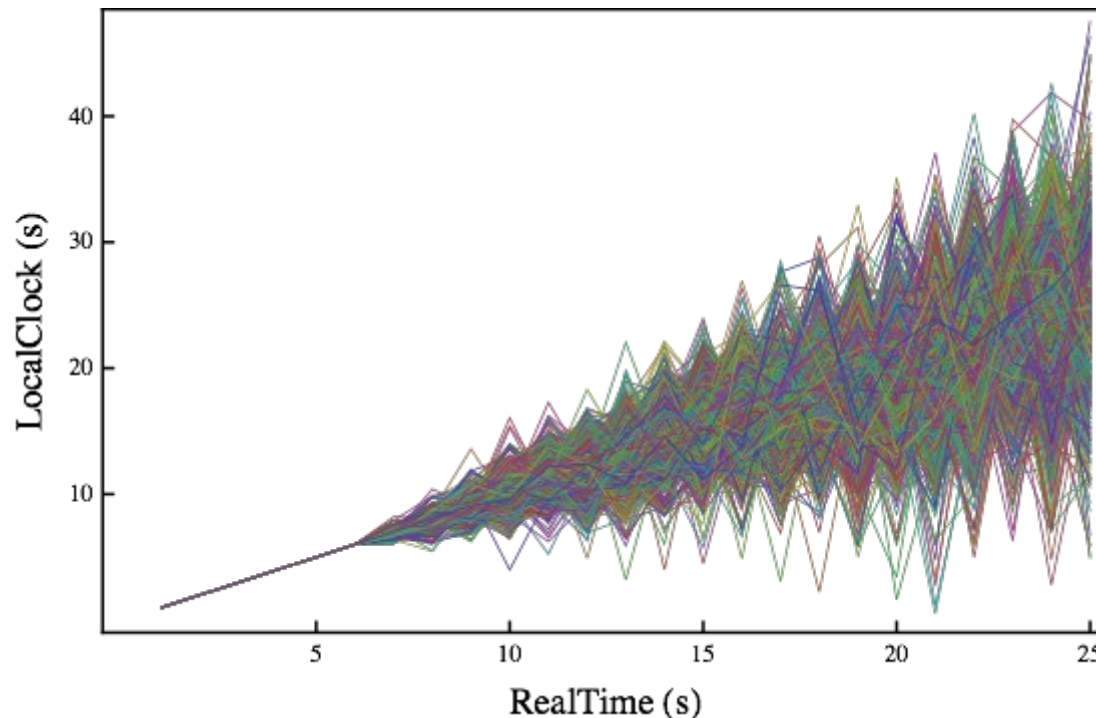


Security

- Application have to provide dependable results even if under attack
- Attack: range of failures ranging from crashes to byzantine faults and external intrusion, in which the intruder aims to manipulate the system as he wants
- Gossip protocols to face crashes
- Byzantine ?
 - In clique-based network a static filter and the assumption of $3f + 1$ nodes at least are used
 - Unfeasible in large systems due to scalability

Byzantine failures

- Related to byzantines, is the problem of false information dissemination
- For the clock synchronization point of view, false information dissemination means network partitioning
 - Correct nodes
 - Correct nodes influenced by byzantines



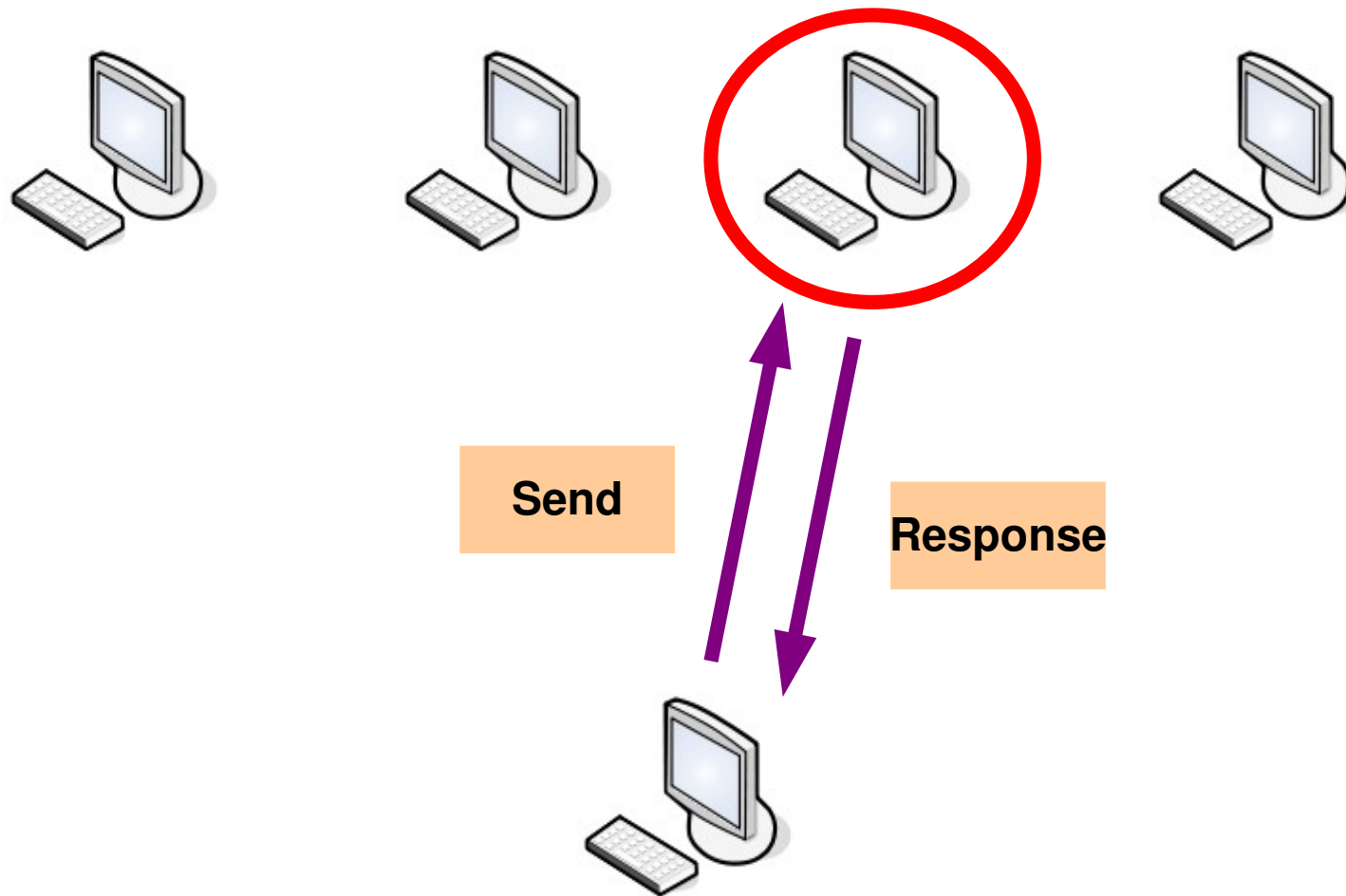
Methodologies

- Increasing interest to P2P systems in the last few years
- Main instruments in P2P environment
 - Gossip protocols
 - Peer Sampling Service
- How these instruments can be applied to clock synchronization algorithms?

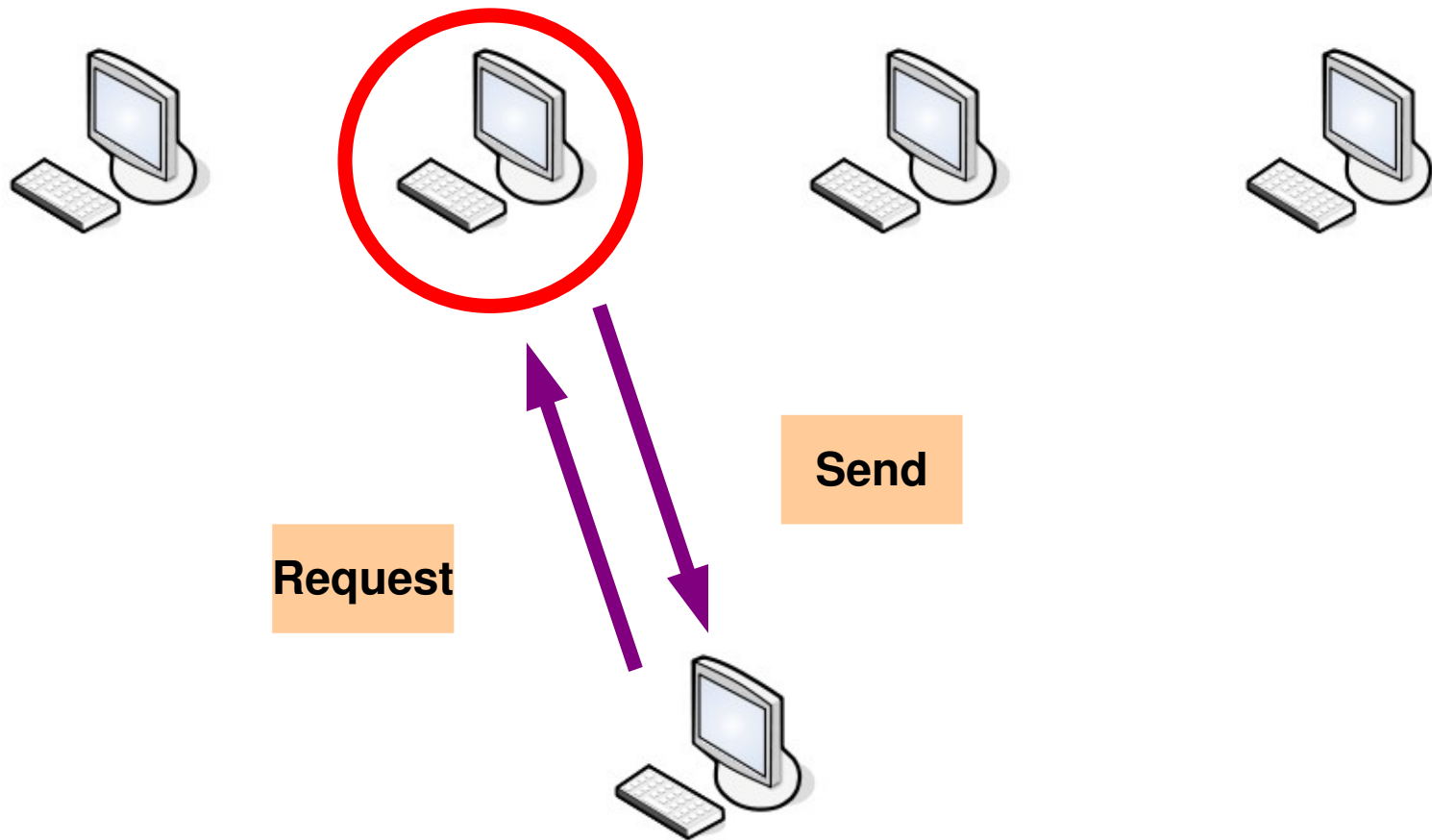
Gossip protocols

- A gossip protocol is based on the information exchange between a pair of nodes
 - It is a useful approach for large and/or unstructured networks (i.e. Random graph)
- Algorithm
 - A node select a neighbor
 - Randomly
 - Age mechanism
 - An information exchange occurs
 - Push-based: the information is sent to the contacted node, that will send a response
 - Pull-based: the information is required to the contacted node, that will send it

Push-based gossip



Pull-based gossip



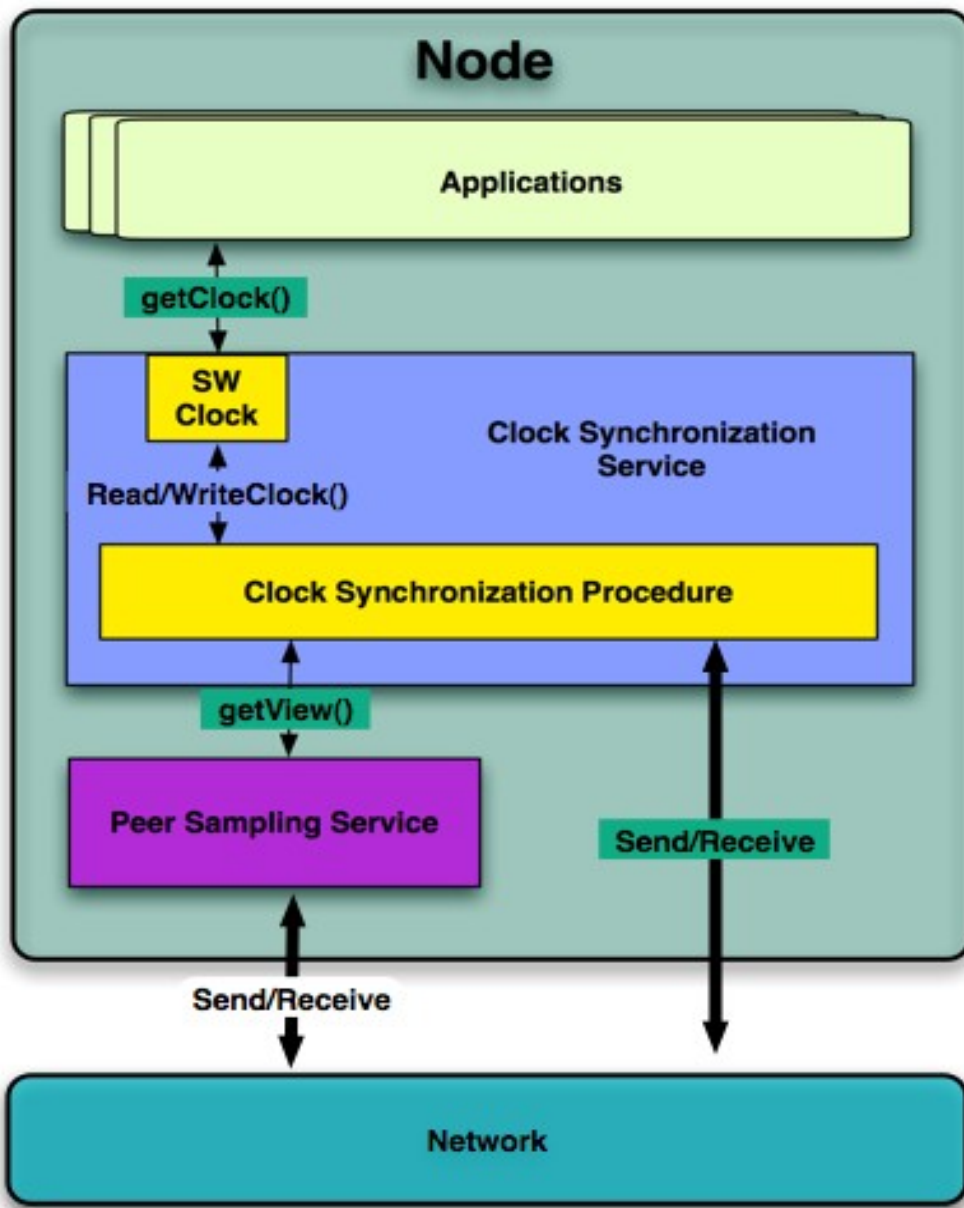
- After an information exchange, a node updates its state

- Gossip protocols advantages
 - The information exchanged has small size
 - After an interaction, the state of one of the interacting nodes is changed (no a ping mechanism)
 - No reliable communication required
 - Interactions frequency is less if compared to typical messages latency

- Why gossip protocols are important?
 - Information dissemination
 - Building block for applications

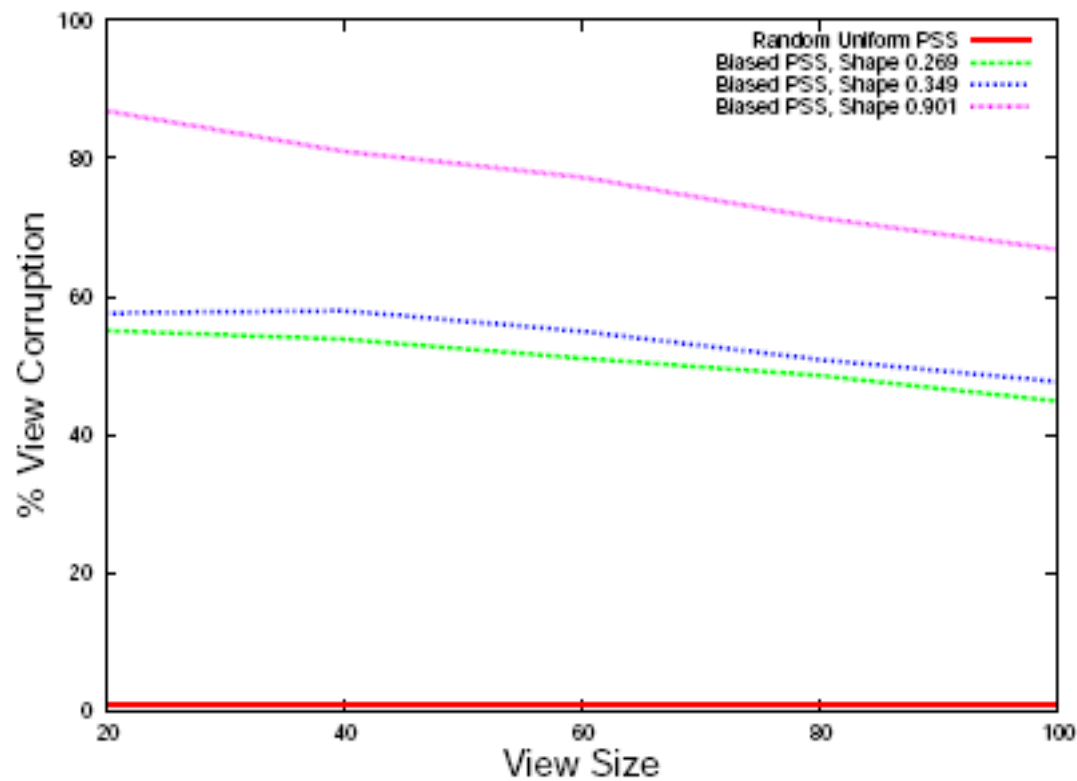
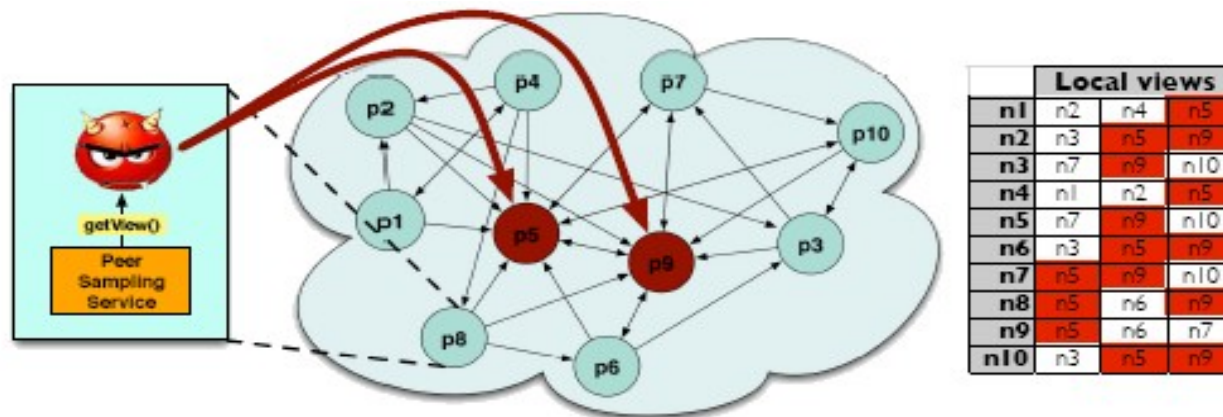
Peer Sampling Service (PSS)

- Random graph
 - Scalable
 - Partial view of the system for each node, but...
- How to provide a neighborhood for each node?
 - PSS
- PSS aims to guarantee network connectivity and to maintain system global informations
- Typically is based on a gossip protocol
 - At each round a node selects a neighbor
 - They exchange a subset of their view, chosen randomly or according to a certain politic



- Why gossip?
 - To maintain large networks
 - Churn resilience
 - Low management cost
- Challenge: uniform PSS [11]
 - Typically node degree in real networks follows a power-law distribution
 - Very few nodes have a huge number of links, while many nodes have few links
 - The probability for a node to be selected is not the same for all nodes
 - Churn
 - May generate inconsistencies (PSS might return offline nodes)
- Uniform PSS allows:
 - Search, replication and routing efficiently
 - Load balancing
 - Agreement in presence of byzantines

■ Non uniform PSS: a problem



Methodologies in clock synchronization algorithms

- How gossip protocols and PSS are applied in clock sync?

- Two examples
 - Van Steen's algorithm for external clock synchron [9]
 - Baldoni's algorithm for internal clock synchron [6]

- Gossip protocol
 - Algorithm [9] uses a push-based gossip protocol
 - Each node selects a neighbors at time
 - Variable gossip frequency
 - If nodes are synchronized, then the frequency is lower
 - If nodes are not synchronized or a huge number of nodes joins the network, then the frequency is higher
 - Nodes who notice first the need of synchronization maintain the frequency high to let other nodes synchronize

■ Gossip protocol

■ Algorithm [6] uses a pull-based approach

- Different from [9], each node exchanges information with all neighbors
- This is motivated by the different way in which nodes in the two algorithms update their clock values
- Fixed frequency

■ PSS

■ Both algorithms uses CYCLON [12]

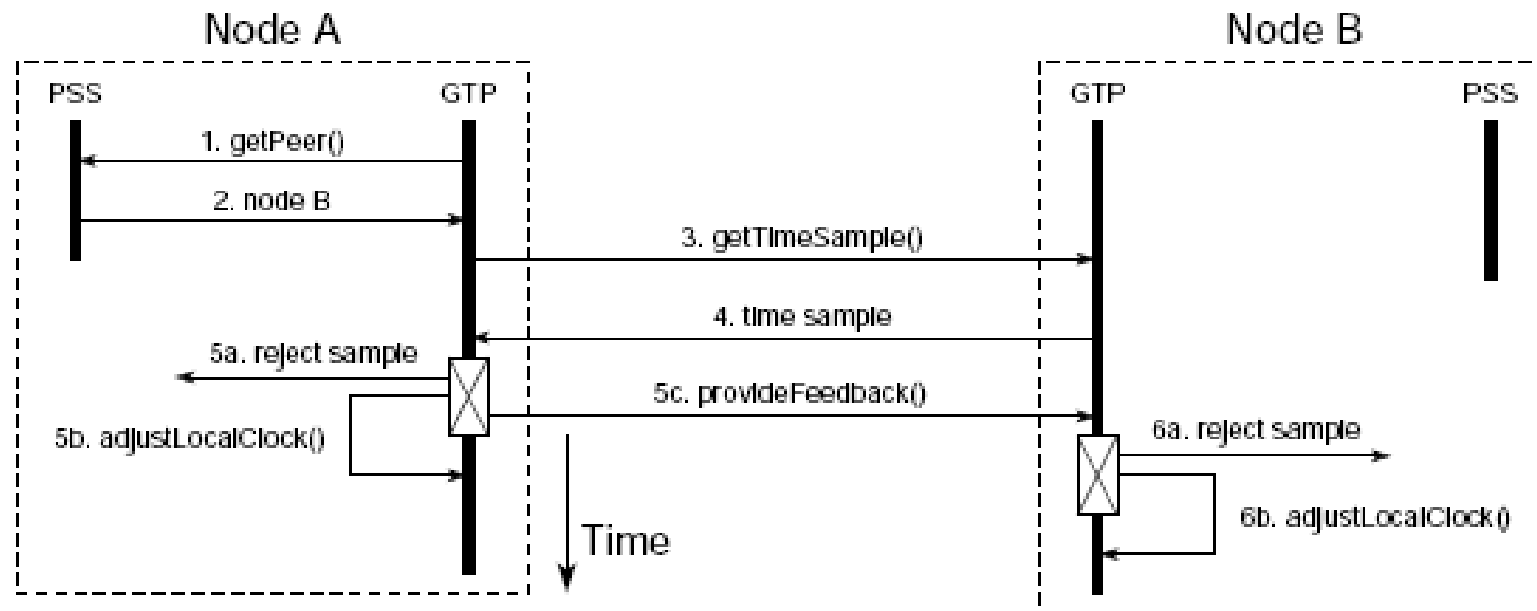
- Each node exchange a random subset of its view elements with the oldest neighbor
- Age-based mechanism to refresh view: old nodes replaced by young ones
- Useful to maintain up-to-date view with respect to dynamic

Clock synchronization algorithms in WAN

- In the following we present three clock synchronization algorithms for large scale dynamic systems
 - Van Steen's algorithm [9]
 - External clock synchronization
 - Babaoglu's algorithm [5]
 - Heartbeat synchronization
 - Baldoni's algorithm [6]
 - Internal clock synchronization

Van Steen et al. [9]

- Provides external synchronization
- Assumption:
 - At least one robust and accurate time source
 - Uniform Peer Sampling Service (PSS)
- Time information disseminated first to nodes directly connected to the source, then to the whole network
- Round-based algorithm



■ Sample evaluation

■ Hop count metric H

- Due to network delays, accuracy degrades along path
- Hence, if H_A is greater than H_B , then A accepts a sample from B and sets $H_A = H_B + 1$
- Eventually, each node will read the clock value directly from the source ($H = 1$) due to the uniform PSS
- Reading again from the source may take time; then, re-synchronization is needed

■ Dispersion metric

- Evaluates clock errors inherited from previous reading through an empirical function

■ Fault tolerance

- Crash, due to enough node in the network
- Time source as bottleneck and single point of failure
- Resilient to churn due to peer-to-peer (P2P) interaction

■ Gossip frequency

- Dissemination by means of a gossip algorithm (see later)
- When nodes closer to source note a change in the time value, they start a synchronization process
- The gossip frequency is maintained high to let synchronize farer nodes
- When most of the node in the network are synchronized, the gossip frequency is decreased (enough to let synchronize the rest of the network)
- Small number of messages exchanged

Babaoglu et al. - Heartbeat synchronization [5]

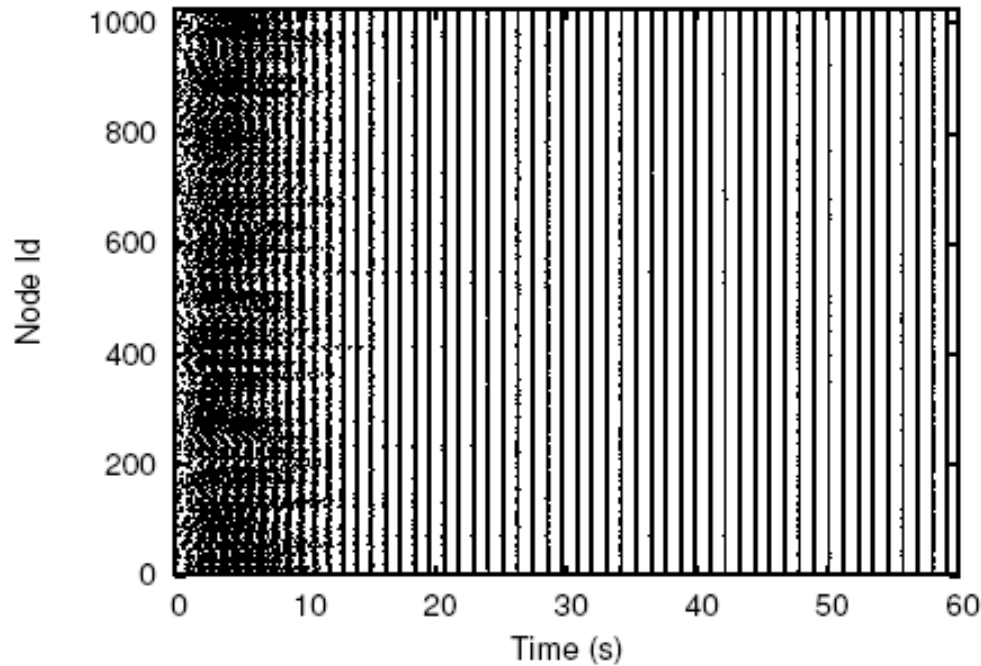
- Heartbeat sync: agreement on the cycle ID rather than on the clock value
- Nodes start and end a cycle approximately at the same time
- The error at least one order of magnitude lesser than the cycle length: no overlapping cycles
- Algorithm inspired by the biological fireflies phenomenon
 - Fireflies synchronize their flashes only by observing a small portion of the swarm

■ Assumption

- Nodes connected through an existing network (i.e. Internet)
- Interaction only with neighbors
- Uniform PSS to guarantee nodes to have a uniform random sample of the network as neighborhood
- Churn
- Crashes (no byzantines)

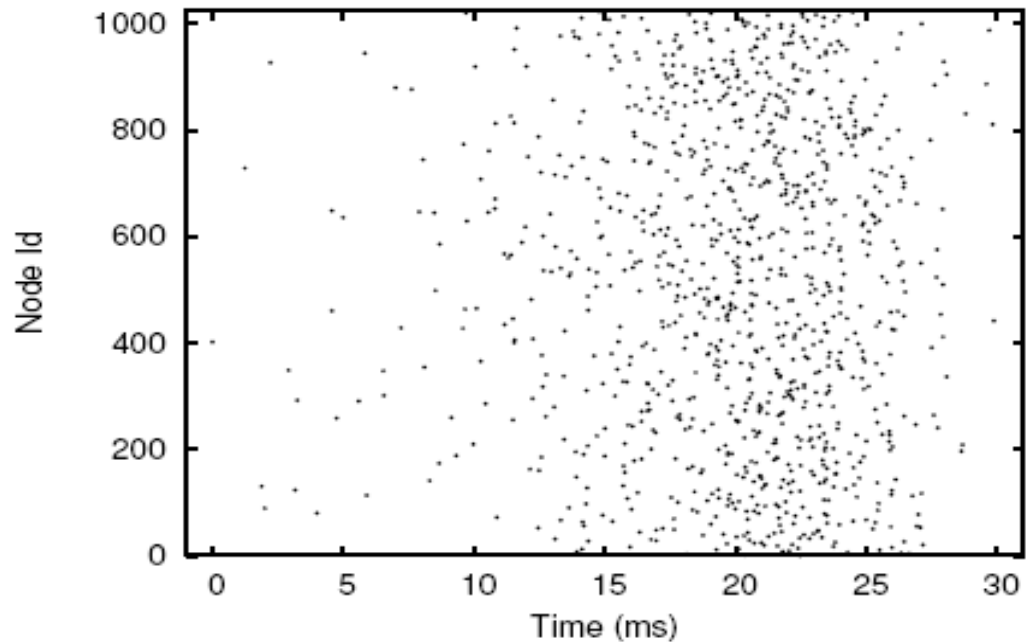
■ Algorithm

- Each nodes is characterized by two parametres
 - The phase Φ : a sawtooth function within the interval $[0,1]$
 - The length of a cycle Δ
 - Initially, each node has a different Δ within an a priori established interval I
- When $\Phi = 1$, the node sends sync messages to neighbors (flash)
- When the node receives a flash, it moves Δ within I , in order to be aligned for the next flash
- Eventually the system converges, emitting flashes at unison



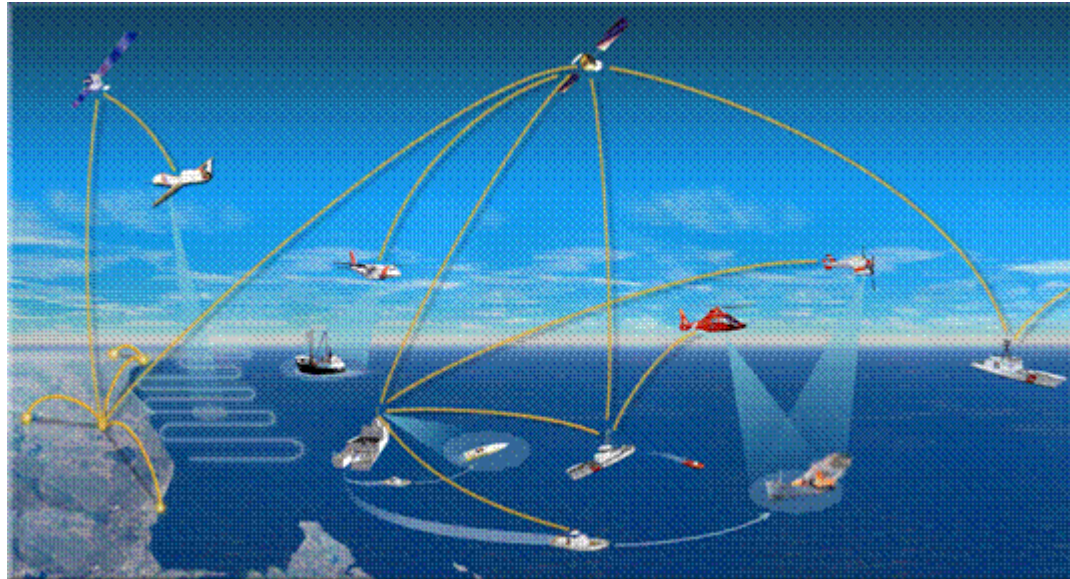
- System convergence
- After 10 sec, the system flashes at unison

- Synchronization error
- Error interval about 15 milliseconds



Baldoni et al. [6]

- Internal clock synchronization
- We refer to large scale dynamic systems where we have
 - Heterogeneous actors
 - Workstation
 - Data centers
 - Laptop
 - Palmtop
 - Sensors
 - ...
 - Heterogeneous QoS requirements
 - Timeliness
 - Ordering
 - Reliable delivery
 - ...

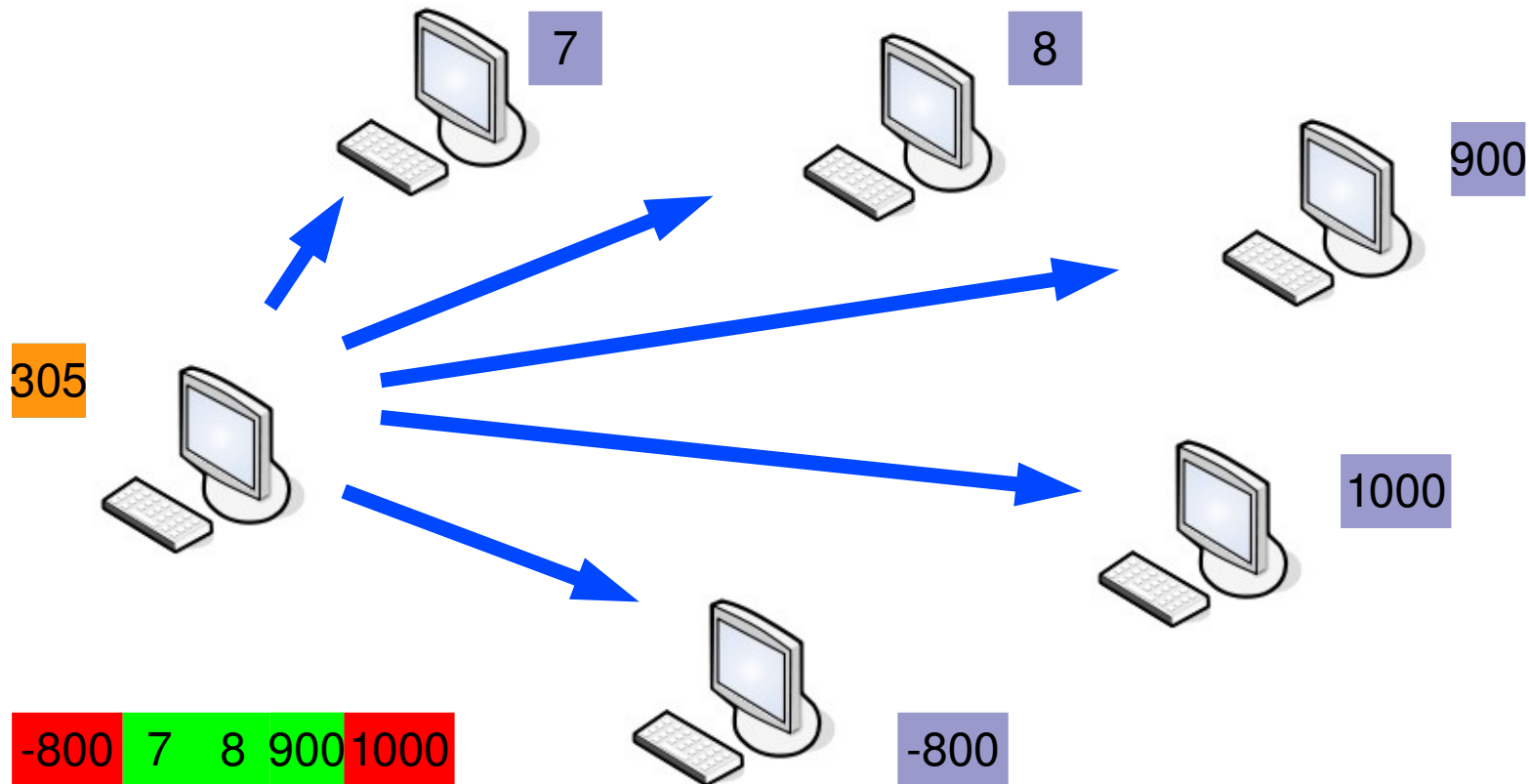


- Our goal: design an internal clock sync algorithm for large scale dynamic systems, where applications are required
 - To operate without assumption on the deployed functionalities
 - Due to security issues or limited assumption on the infrastructure
 - Being able to tolerate network dynamic
 - The continuous arrival/departure of nodes
 - Scaling on ten of thousands of nodes

- We use a fully decentralized paradigm, in which nodes implement all functionalities by means of a gossip algorithm
 - Each nodes exchanges time informations only with its neighbors
 - It is inspired by biological phenomena: coupling oscillators, fireflies, pacemaker network cell in the heart, ...

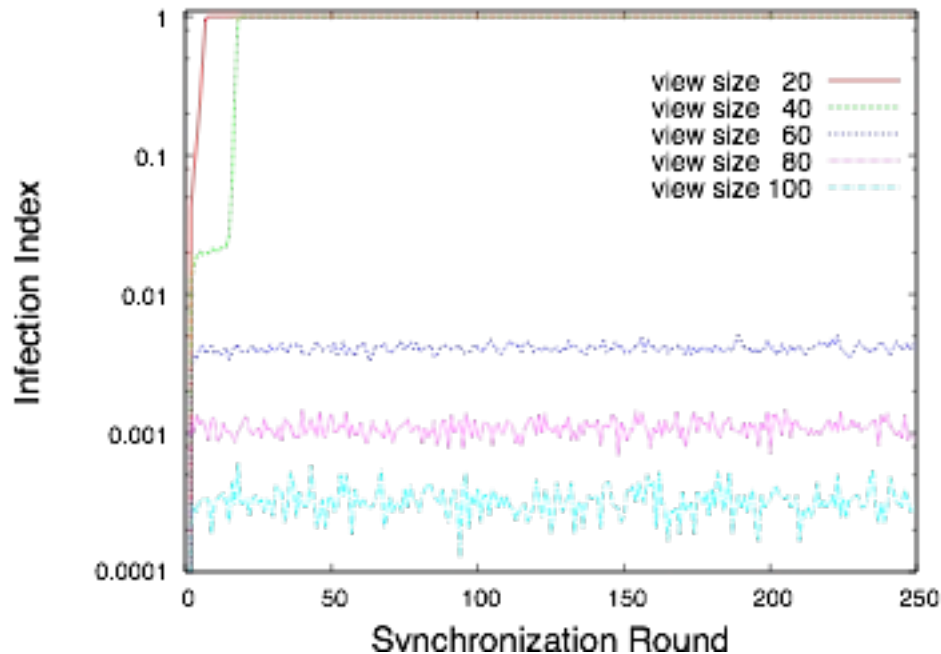
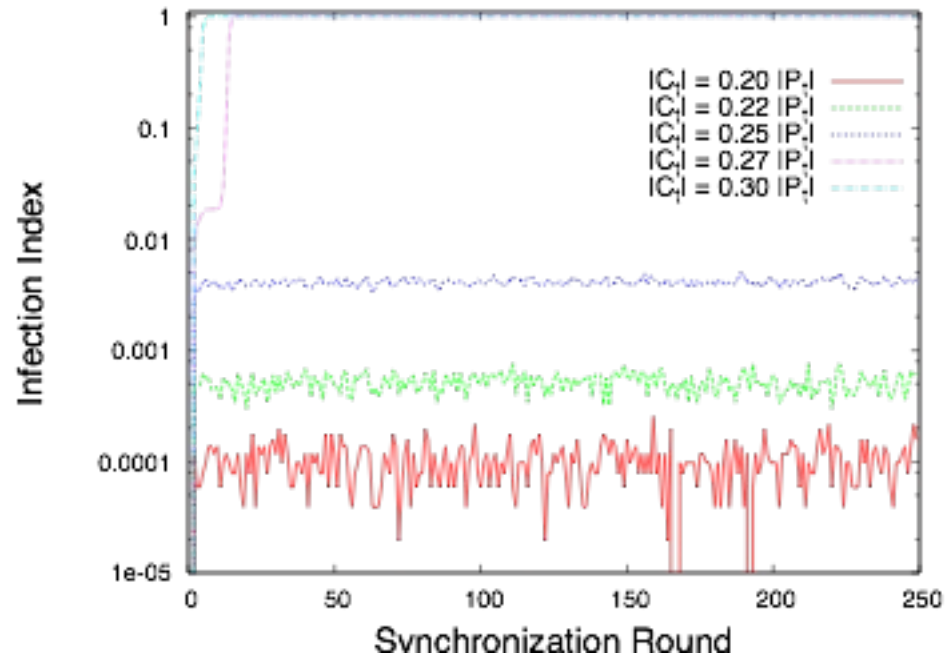
■ A demo!

- The novelty is that **our algorithm is able to tolerate byzantine faults**
- It is the first in internal clock synchronization algorithm in large scale dynamic systems
- We apply the same static filter used in the Cristian and Fetzer's algorithm [2] to local views



■ Problems

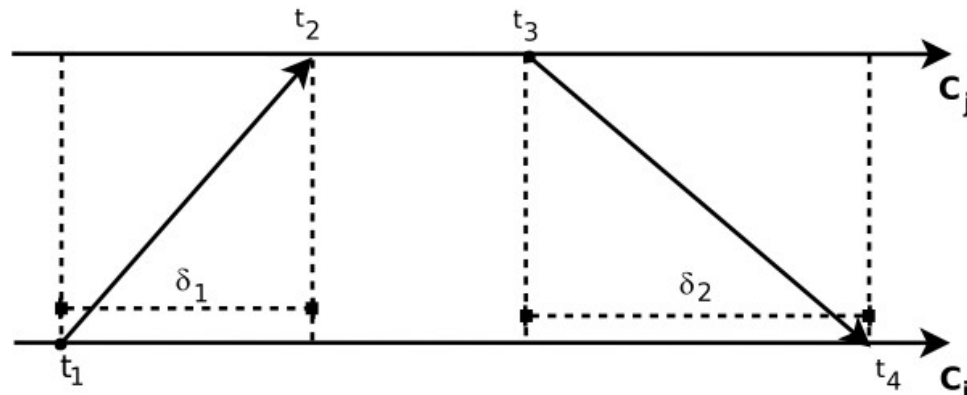
- It is possible that not all byzantines are filtered
 - A percentage of error is still present in the system



■ Local views

- Each node has only a partial knowledge of the network
- Filtering is not with respect to the whole system

- Algorithm analysis: it is made without byzantine nodes (all correct nodes)
- Clock updating



$$C_i(n+1) = C_i(n) + f_i \Delta T +$$

$$+ \frac{K_i}{N_i} \sum_{j=1}^{N_i} [(C_j(n) - C_i(n)) * edge(i, j)] +$$

$$+ \frac{K_i}{N_i} \sum_{j=1}^{N_i} \left[\left(\frac{\delta_{i,j} - \delta_{j,i}}{2} \right) * edge(i, j) \right], \quad i = 1..N$$

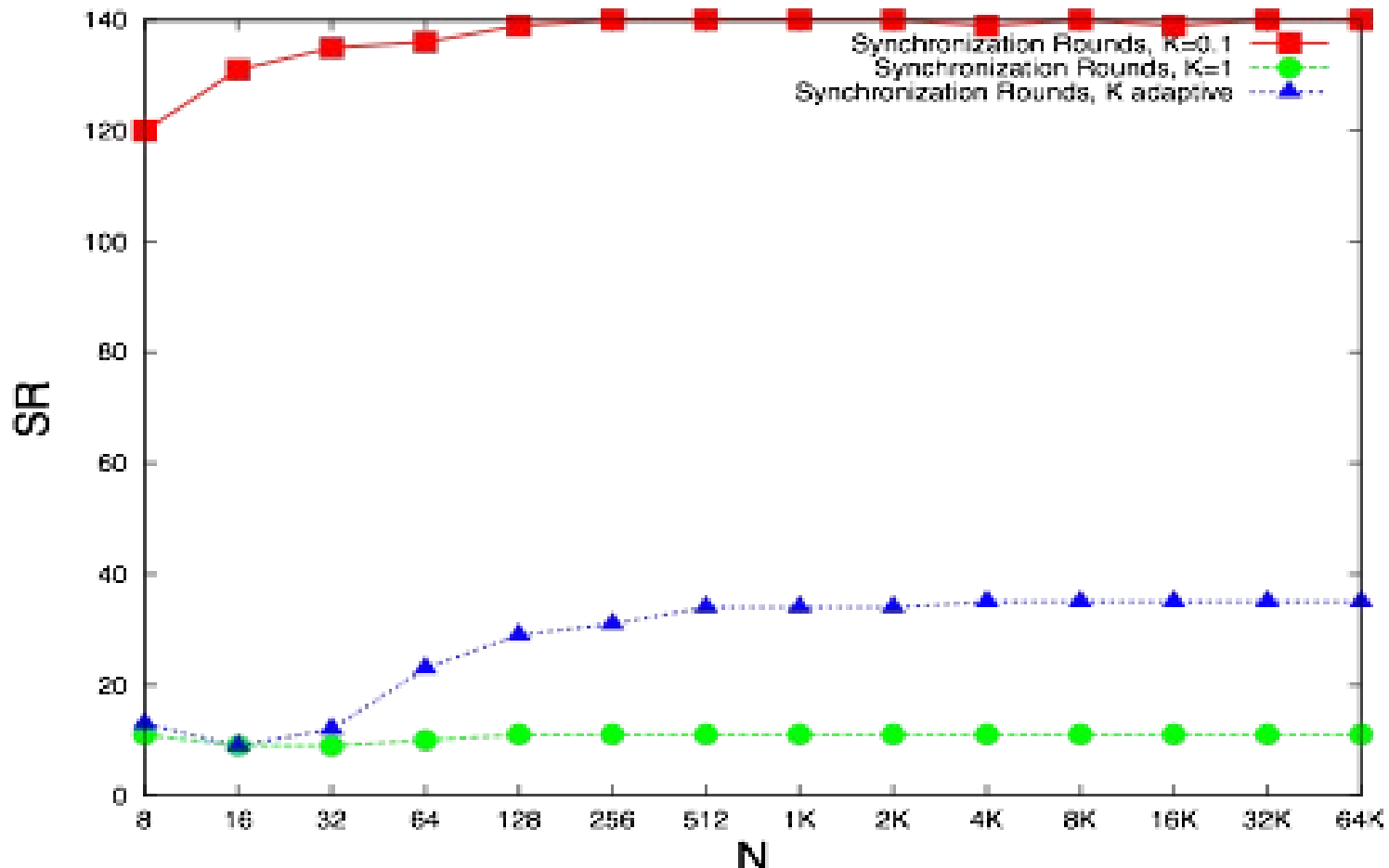
Coupling factor!

- Coupling factor measures the strength of interaction between nodes
 - How a node is influenced by neighbors
- It plays a key role
- It assumes values in $(0,1]$
 - A lower coupling factor leads to a better synchronization despite system perturbations (network transmission delay and churn). The differences in clock values are negligible
 - An higher coupling factor leads to a faster convergence but clock values are more sparse. The sensitivity of interaction augments, but augments the influence of system perturbations

- We use an adaptive coupling factor
- We want to reduce the impact of system perturbations and to maintain a slow convergence time
 - High coupling factor for nodes joining the network in order to let them synchronize soon
 - System perturbations have a small effect if the clock values differences are high
 - Low coupling factor for node already in the network, in order to guarantee a better synchronization
 - The impact of system perturbations is limited

■ Scalability

- The convergence time remains the same with few or several thousands of nodes
- Higher coupling factor provides faster convergence time



■ Bibliography

- 1) *Time, clocks and Ordering of Events in a Distributed System*, Lamport, 1978
- 2) *An Optimal Internal Clock Synchronization Algorithm*, Fetzer, Cristian, 1995
- 3) *On the Possibility and Impossibility of Achieving Clock Synchronization*, Dolev, Halpern, Strong, 1984
- 4) *Self-stabilizing Pulse Synchronization Inspired by Biological Pacemaker Networks*, Daliot, Dolev, Parnas, 2003
- 5) *Fireflies-inspired Heartbeat Synchronization in Overlay Networks*, Babaoglu, Binci, Montresor, Jelasity, 2006
- 6) *A Peer-to-Peer Filter-based Algorithm for Internal Clock Synchronization in Presence of Corrupted Processes*, Baldoni, Platania, Querzoni, Scipioni, 2008
- 7) *Network Time Protocol Version 4: Reference and Implementation Guide*, Mills, 2006
- 8) *CESIUM SPRAY: a Precise and Accurate Global Time Service for Large-scale Systems*, Verissimo, Rodrigues, Casimiro, 1997
- 9) *Gossip-based Clock Synchronization for Large Decentralized Systems*, Iwanicki, van Steen, Voulgaris, 2006
- 10) *Performance Evaluation of Clock Synchronization Algorithms*, Anceaume, Puant, 1998
- 11) *On Unbiased Sampling for Unstructured Peer-to-Peer Networks*, Stutzbach, Rejaie, Duffield, Sen, Willinger, 2006
- 12) *CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays*, Voulgaris, Gavidia, van Steen

Thank You