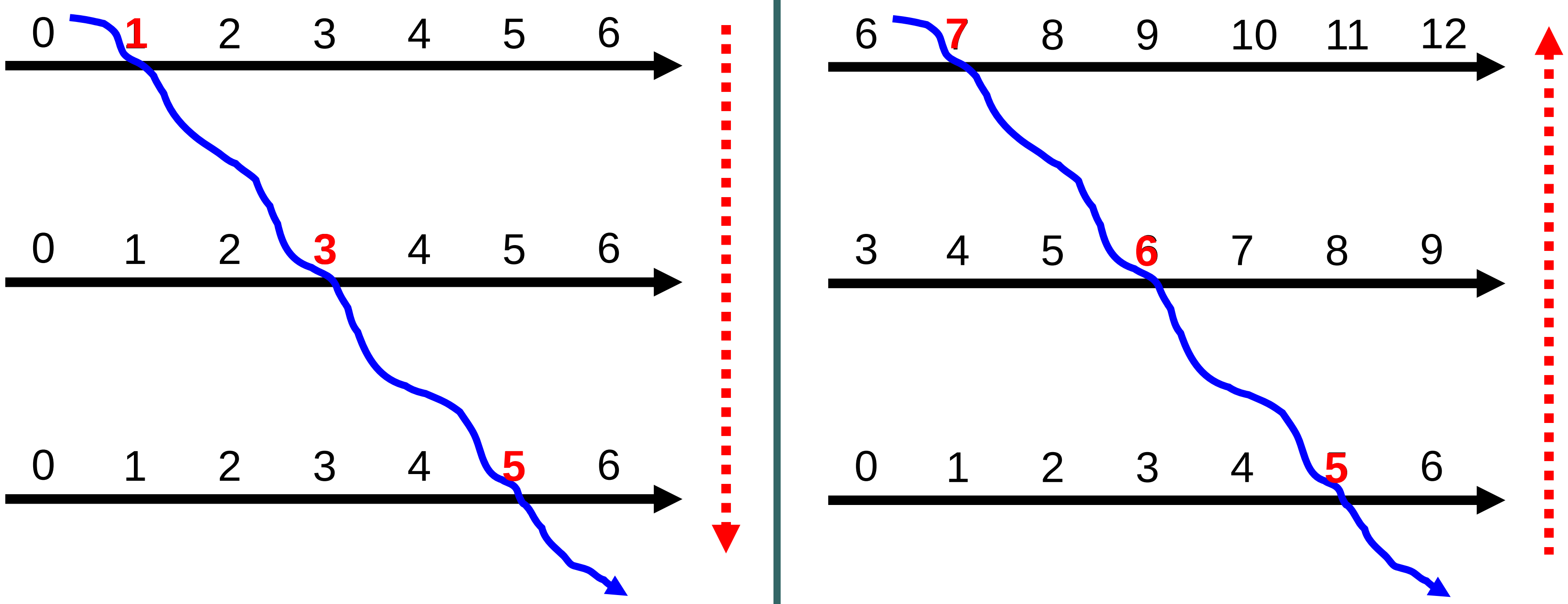
Secure and Self-Stabilizing Clock Synchronization in Sensor Networks

Jaap-Henk Hoepman, **Andreas Larsson**
Elad M. Schiller, Philippas Tsigas

13 November 2007


2.4 GHZ
5250
Без товаров, подлежащих обложению пошлиной
Intet å fortolle
Nothing to declare
С товарами, подлежащими обложению пошлиной
Varer til fortolling
Goods to declare

Attacks against clocks



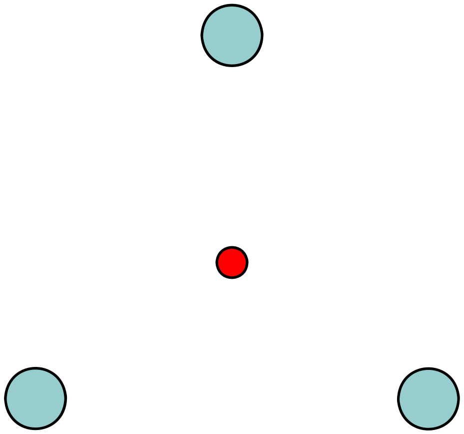
Outline

- **Motivation**
- Implementation
- Attacks
- Correctness
- Earlier work
- Conclusion

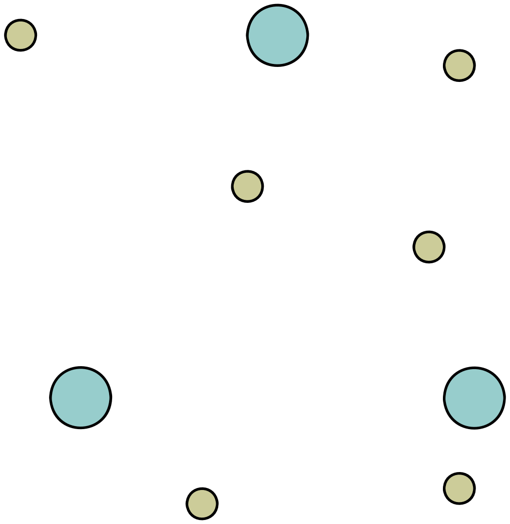
The need for clock synchronization

- Pinpointing events geographically
- Time division message scheduling
- Radio shutoff periods
- Certain mathematical functions
- …

Need for precision



Required result



Result of traditional protocols

Adversary

- Much more powerful than the nodes
  - Intercepting
  - Replaying
  - Delaying
- Capturing nodes and impersonating

Self-stabilization, Security & Fault tolerance

- Dealing with transient faults
- Security needs self-stabilization
  - Security under certain assumptions
  - Attacks eventually violate assumptions

Arbitrary starting configuration

- Fault tolerance – message loss
  - Noise
  - Collisions

Outline

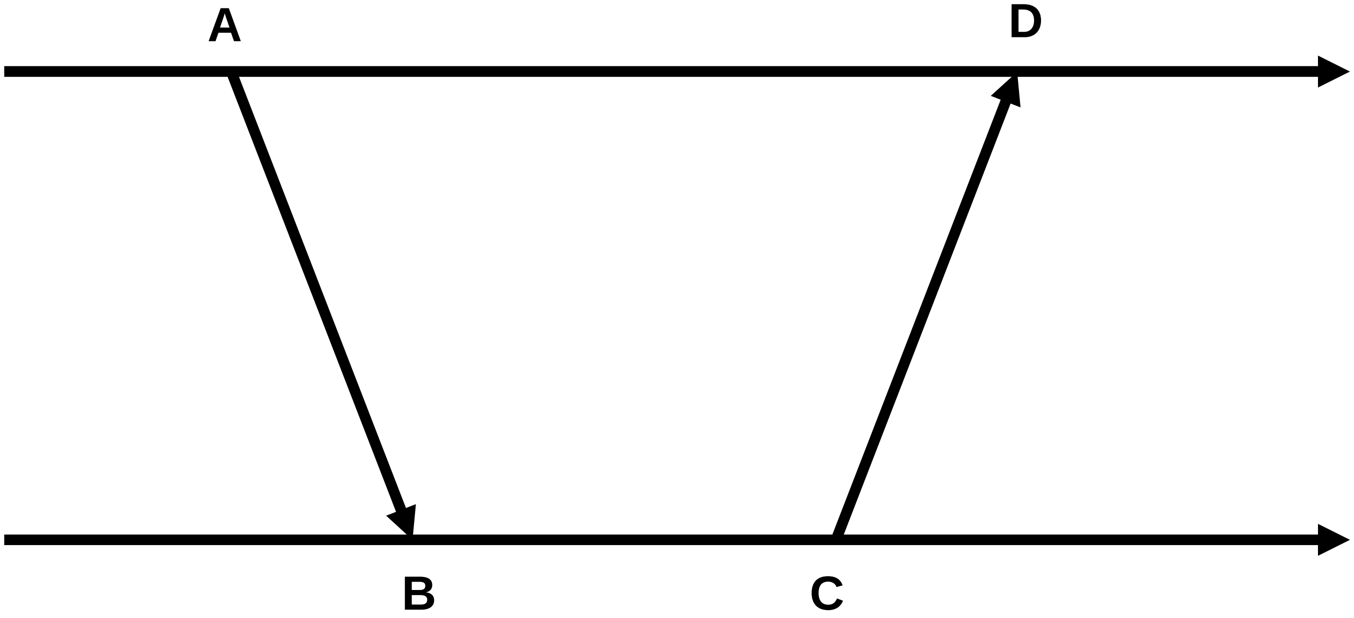
- Motivation
- **Implementation**
- Attacks
- Correctness
- Earlier work
- Conclusion

The clock model

- Offset is arbitrary
- Rate, ρ, is varying
  - Manufacturing variations
  - Environmental variations
- Clock rate stays within a certain interval

$$\rho_{\min} < \rho < \rho_{\max}$$

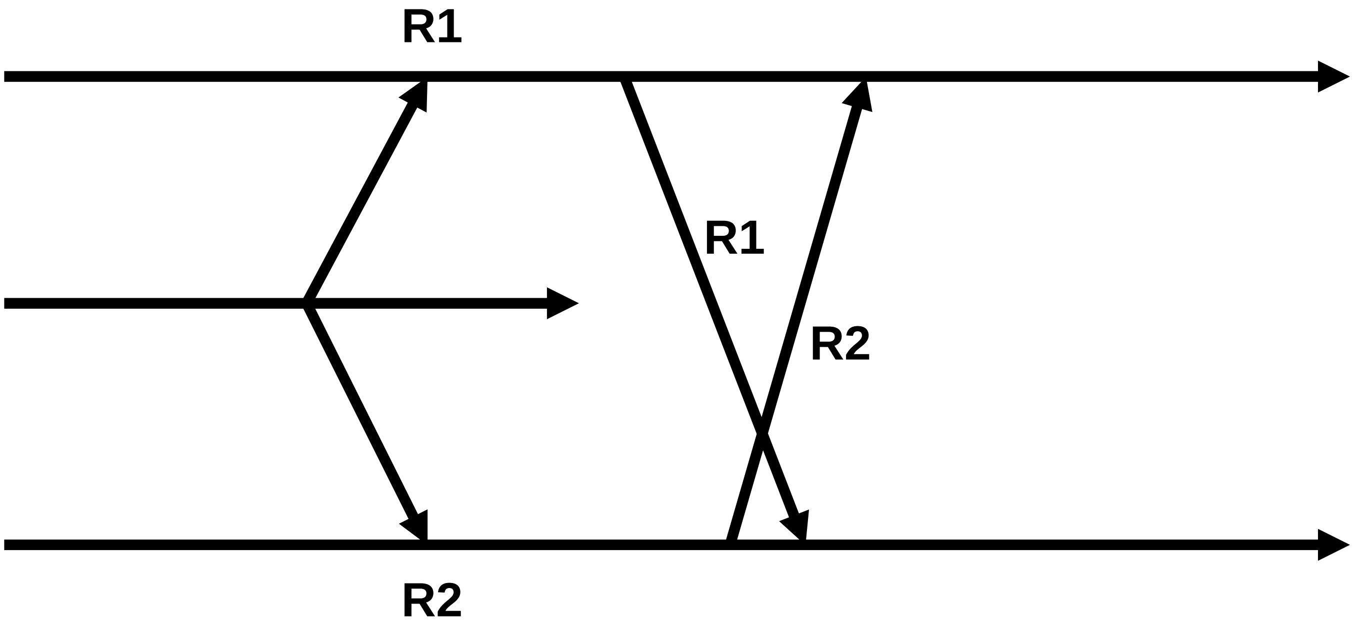
Roundtrip synchronization



Offset

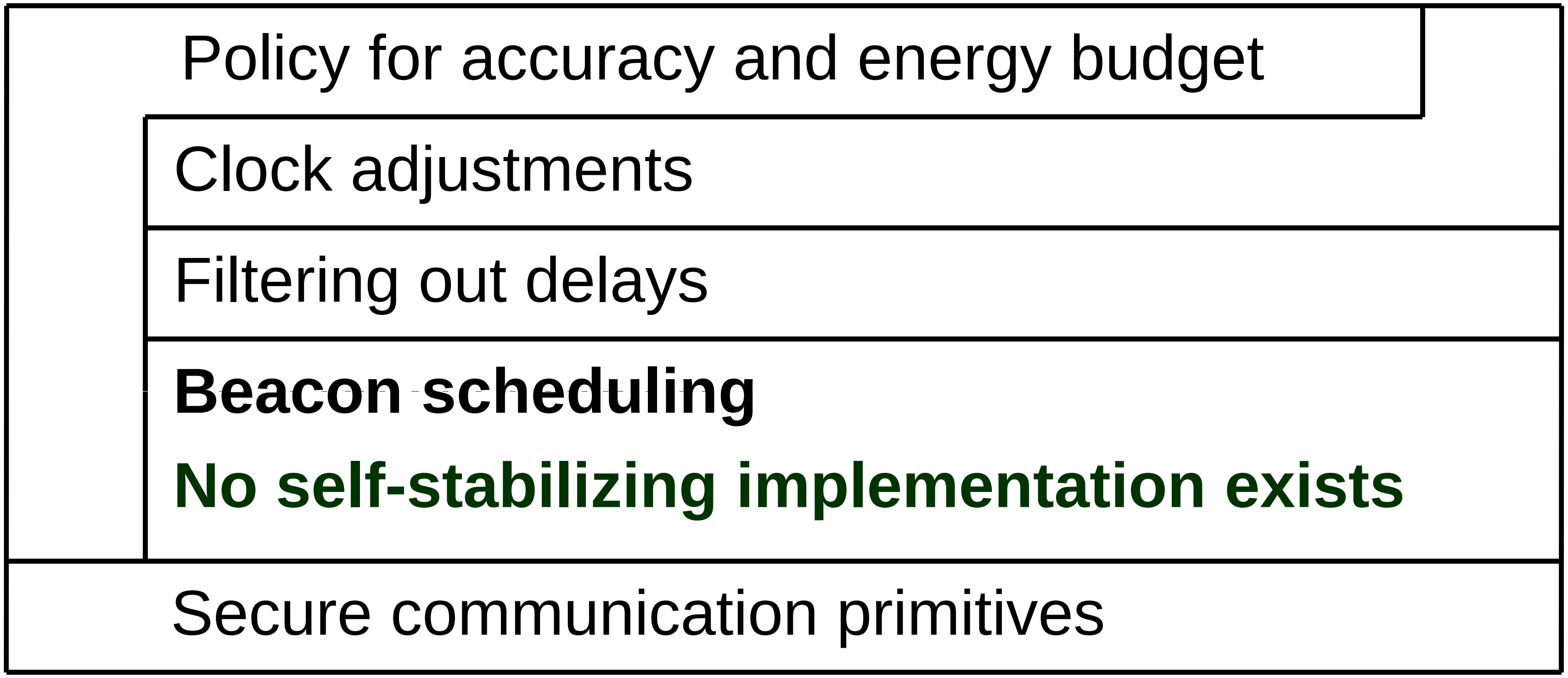
Delay

Reference Broadcast



Offset with higher precision

The protocol layers



Policy for accuracy and energy budget

Clock adjustments

Filtering out delays

Beacon scheduling

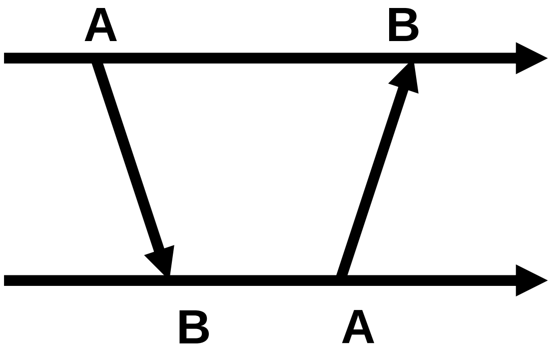
No self-stabilizing implementation exists

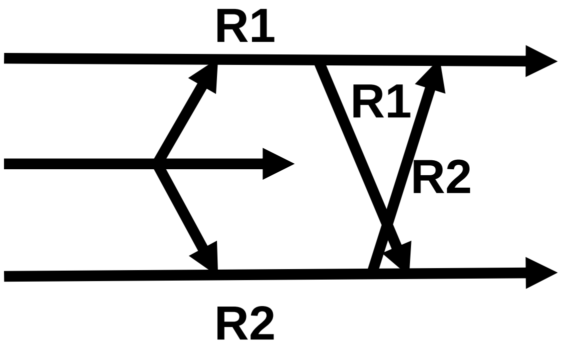
Secure communication primitives

Combining the two approaches

Beacon sent by node i:

R_0	…	R_{i-1}	A_i	R_{i+1}	…	R_n





Dealing with message loss

0	R_0	…	R_{i-1}	A_i	R_{i+1}	…	R_n
1	R_0	…	R_{i-1}	A_i	R_{i+1}	…	R_n
⋮				⋮			
Q-1	R_0	…	R_{i-1}	A_i	R_{i+1}	…	R_n
Q	R_0	…	R_{i-1}	A_i	R_{i+1}	…	R_n

Delivering to upper layer

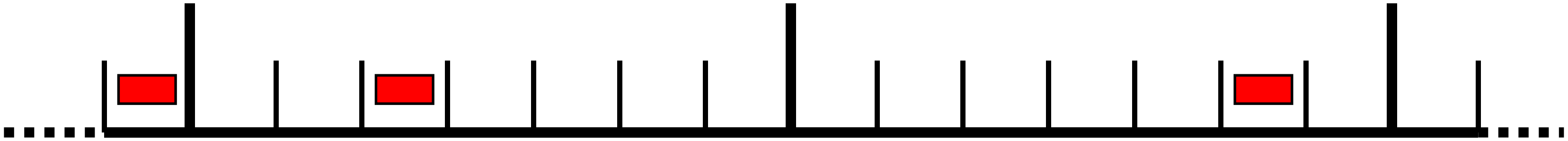
- Data held by a node
  - Its beacon send times
  - Its receive times of beacons
  - The corresponding data received from others
- Delivery to upper layer is delayed
  - Collect as much as possible before reporting

Randomized beacon scheduling

Partition time

Divide partitions into slots ($n \log^2 n$)

Randomly send one beacon per partition



Time complexity

n nodes send a message of size *O*(*n*) each

Optimal	**Our randomized strategy**
n^2	$n^2 \log^2 n$

n = bound on degree of nodes

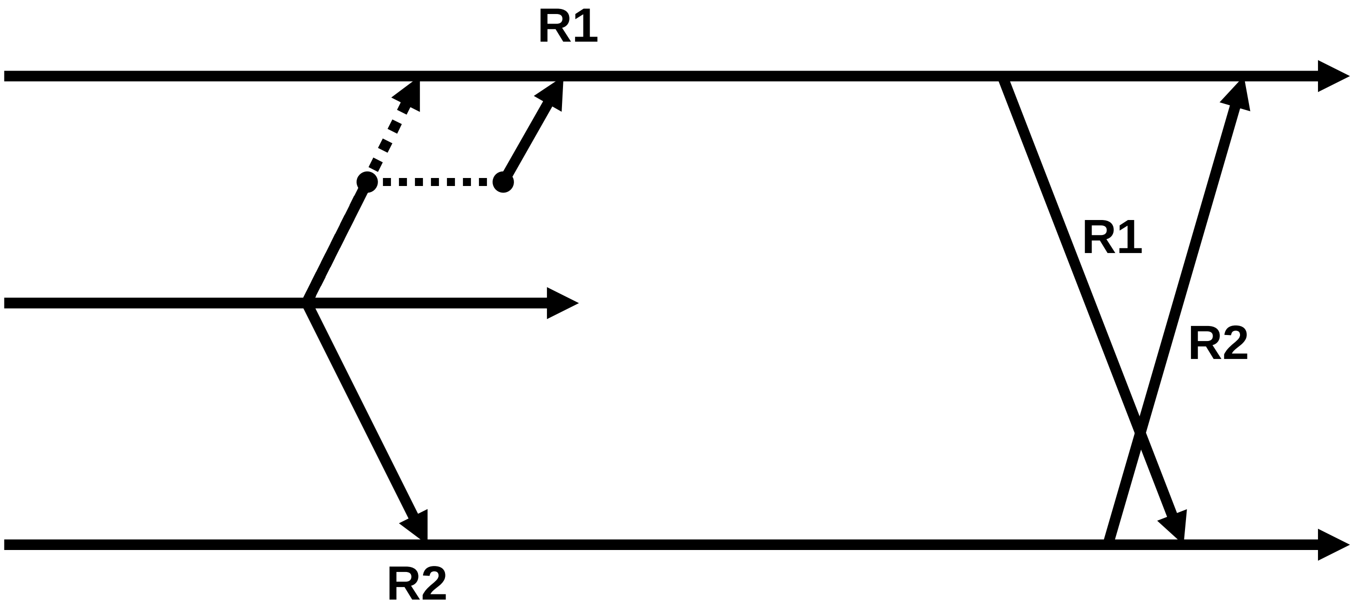
Outline

- Motivation
- Implementation
- **Attacks**
- Correctness
- Earlier work
- Conclusion

The attacker model

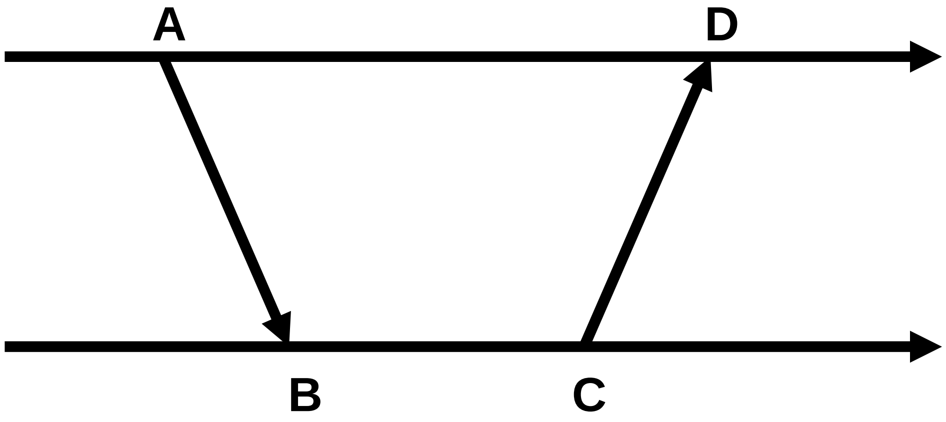
- Interception of messages
  - Stop receival
  - Replay later
- Capturing nodes
  - Get data including keys
  - Stop nodes
  - Impersonate nodes

Delay attacks



- Cryptography does not help
- Nonce does not help

Dealing with delay attacks



- Locally calculate delay
- Filter out over-delayed beacons
  - Byzantine agreement [Ganeriwal et al. 05]
  - Outlier filtering [Song et al. 06]

Dealing with captured nodes

- Impersonated nodes send misleading data
  - Send at one time, claim another
- Filter out misleading beacons
  - Byzantine agreement [Ganeriwal et al. 05]
  - Outlier filtering [Song et al. 06]

Outline

- Motivation
- Implementation
- Attacks
- **Correctness**
- Earlier work
- Conclusion

Correctness proof

- Beacon scheduler
  - Partially synchronous system
  - Message collision and omission
- Probabilistic delivery guarantees
  - Every node sends a beacon that every node receives
  - Every node receives a response to its beacon from every node
- Beacon aggregation (appears in TR)

Outline

- Motivation
- Implementation
- Attacks
- Correctness
- **Earlier work**
- Conclusion

Self-stabilizing but not Secure

- [Herman and Zhang 06]
  - a model for clock synchronization in sensor network
  - show that the converge-to-max approach is stabilizing
- A single captured node attack
  - At any time introduce the maximal clock value
- Adversary sends the clock “far into the future”
  - Preventing a continuous time approximation function

Secure but not Self-stabilizing

- No existing secure and self-stabilizing implementations
  - Many implementations require initial clock synchronization prior to the first pulse-delay attack
- The adversary can risk detection and intercept all beacons for a long period
  - As a result: arbitrary clock offsets
  - The system has to use global restart
  - No global restart after deployment!

Secure but not Self-stabilizing

- [Sun et al. 05] cluster-wise synchronization
    - Based on synchronous rounds
    - Byzantine agreement
    - Synchronized clock at the starting configuration
- We make no assumptions on synchronous rounds or start

Secure but not Self-stabilizing

- [Manzo et al. 05]
  - Consider attacks on unsecured clock synchronization
  - Suggest counter measures
  - Use a randomly selected “core” of nodes to minimize the effect of captured nodes
  - Do not consider the cases in which the adversary captures nodes after the core selection
- We make no assumption regarding the distribution of the captured nodes

Secure but not Self-stabilizing

- [Farrugia and Simon 06]
  - A cross-network spanning tree in which the clock values propagate for global clock synchronization
  - No pulse-delay attacks are considered
- [Sun et al. 06]
  - Use external source nodes to increase the resilience against an attack that compromises source nodes
- We use no source nodes

Outline

- Motivation
- Implementation
- Attacks
- Correctness
- Earlier work
- **Conclusion**

Conclusion

- System settings of traditional networks
  - cannot be assumed
- Designer assumptions
  - cannot hold forever
- Self-stabilization can provide self-defense capabilities

Thank you for your attention

Thank you for your attention

Andreas Larsson
larandr@cs.chalmers.se
Chalmers University of Technology