Adaptive Internal Clock Synchronization

SRDS 2008

Zbigniew Jerzak, Robert Fach, and Christof Fetzer

October 6, 2008

{Zbigniew.Jerzak, Robert.Fach, Christof.Fetzer}@inf.tu-dresden.de

Introduction

Build safer distributed systems using weaker assumptions [FC03]

In this paper/presentation:

- Relaxed specification for internal clock synchronization
- New clock synchronization algorithm
- New way to deal with crash failures

Current Approach

1. Assume a bounded deviation Δ_{max}
2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks' speeds fluctuate with time
   - processes crash
3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system

Current Approach

1. Assume a bounded deviation Δ_{max}
2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks' speeds fluctuate with time
   - processes crash
3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system

Current Approach

1. Assume a bounded deviation Δ_{max}
2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks' speeds fluctuate with time
   - processes crash
3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system

Current Approach

1. Assume a bounded deviation Δ_{max}
2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks' speeds fluctuate with time
   - processes crash
3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system

Our Approach

- No upper bound for the deviation (Δ_{max})
  - compute deviation between processes in real-time
  - use computed deviation to adapt system behavior
- No bound on the maximum number of crashed processes
  - mask crashes using clock readings extrapolation

Result:

An adaptive, safer system

Our Approach

- No upper bound for the deviation (Δ_{max})
  - compute deviation between processes in real-time
  - use computed deviation to adapt system behavior
- No bound on the maximum number of crashed processes
  - mask crashes using clock readings extrapolation

Result:

An adaptive, safer system

Our Approach

- No upper bound for the deviation (Δ_{max})
  - compute deviation between processes in real-time
  - use computed deviation to adapt system behavior
- No bound on the maximum number of crashed processes
  - mask crashes using clock readings extrapolation

Result:

An adaptive, safer system

Use Case

- p is assigned a time slot $[S, T]$ for access to **R**
- $[S, T]$ defined w.r.t. some abstract clock
- p is not synchronized with a priori known Δ_{max}, instead...
- Use E_p – maximum local clock synchronization error
  - an upper bound on the deviation from some abstract clock
  - calculated by every process
  - propagated to application layer for error handling
- p uses **R** only within $[S + E_p, T - E_p]$
  - p knows how well it is synchronized
  - p knows when it is allowed to access **R**

Use Case

- p is assigned a time slot $[S, T]$ for access to **R**
- $[S, T]$ defined w.r.t. some abstract clock
- p is not synchronized with a priori known Δ_{max}, instead...
- Use E_p – maximum local clock synchronization error
  - an upper bound on the deviation from some abstract clock
  - calculated by every process
  - propagated to application layer for error handling
- p uses **R** only within $[S + E_p, T - E_p]$
  - p knows how well it is synchronized
  - p knows when it is allowed to access **R**

Use Case

- p is assigned a time slot $[S, T]$ for access to **R**
- $[S, T]$ defined w.r.t. some abstract clock
- p is not synchronized with a priori known Δ_{max}, instead...
- Use E_p – maximum local clock synchronization error
  - an upper bound on the deviation from some abstract clock
  - calculated by every process
  - propagated to application layer for error handling
- p uses **R** only within $[S + E_p, T - E_p]$
  - p knows how well it is synchronized
  - p knows when it is allowed to access **R**

Processes and Hardware Clocks

based on TADSM [CF99]

- Set of $\mathbb{N} = \{p_0, p_1, \ldots, p_n\}$ processes
- Connected via communication bus
- Hardware clocks – $H_p(t)$:
  - bounded drift:
    $\forall_{p \in \mathbb{N}} \forall t : \quad |\rho_p(t)| \leq \rho_{\max}$
  - within linear envelope of real time:
    $(t - s)(1 - \rho_{\max}) \leq H_p(t) - H_p(s) \leq (t - s)(1 + \rho_{\max})$

Generic Internal Clock Synchronization Algorithm

```
1  ClockVal A_p; // current adjustment
2  ClockVal T; // end of current round

4  void init() {
5      (A_p,T) = initialAdjustment();
6      // every P starting at T
7      schedule(synchronizer, P, T);
8  }

10 void synchronizer() {
11     // N - number of processes
12     ClockVal clk[N],
13     ClockVal err[N];
14     // remote clock reading
15     readClocks(clk, err);
16     // adjustment for the next round
17     A_p = adjust(A_p, T, clk, err);
18     // set T to next round
19     T = T + P;
20 }
```

Software Clocks and Remote Clock Reading

- We do not apply A_p directly to the $H_p(t)$
- Instead we use software clocks:
  $S_p(t) = H_p(t) + a_p(t)$

- A_p is calculated based on the remote clock readings
  - probabilistic remote clock reading [FC99b]
  - provides values of the remote clocks `clk[]`
  - and accompanying remote clock reading errors `err[]`

Failure Model

- Messages are supposed to be delivered within δ
  - might be delayed
  - delivered out of order
  - get dropped
- Arbitrary hardware clock failures: $|\rho_p(t)| > \rho_{\max}$
  - converted to stop failures [FC99a]
- Processes' value (byzantine) failures
  - handled by Software Encoded Processing [WF07]

Specifically:

No upper bound
on the frequency of communication and process failures

Failure Model

- Messages are supposed to be delivered within δ
  - might be delayed
  - delivered out of order
  - get dropped
- Arbitrary hardware clock failures: $|\rho_p(t)| > \rho_{\max}$
  - converted to stop failures [FC99a]
- Processes' value (byzantine) failures
  - handled by Software Encoded Processing [WF07]

Specifically:

No upper bound
on the frequency of communication and process failures

Problem Statement

$$\forall_{p,q \in \mathbb{N}} : \quad S_p(t) = S_q(t) \tag{1}$$

- However, achieving 1 is not possible
- Minimize & bound difference between software clocks:
  - $\forall_{p,q \in \mathbb{N}: p,q \ \neg \mathrm{crashed}(t)} \forall_t : \ |S_p(t) - S_q(t)| \leq E_p(t) + E_q(t)$

Problem Statement

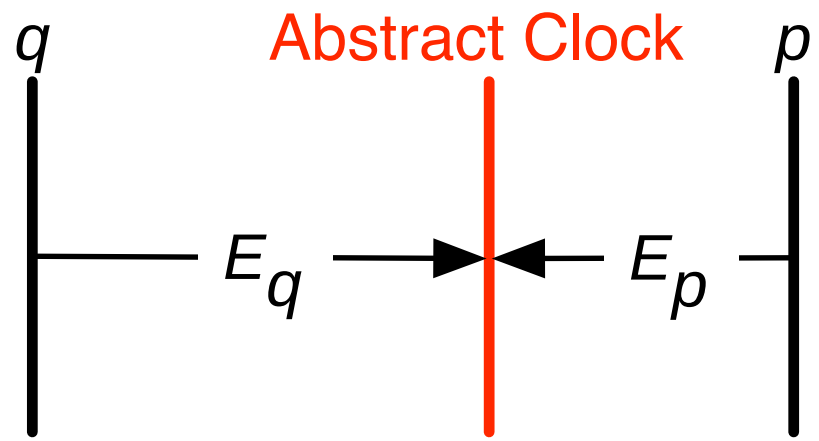
$$\forall_{p,q \in \mathbb{N}} : \quad S_p(t) = S_q(t) \tag{1}$$

- However, achieving 1 is not possible
- Minimize & bound difference between software clocks:
  - $\forall_{p,q \in \mathbb{N}: p,q \ \neg \mathrm{crashed}(t)} \forall_t : \ |S_p(t) - S_q(t)| \leq E_p(t) + E_q(t)$

Problem Statement

$$\forall_{p,q \in \mathbb{N}} : \quad S_p(t) = S_q(t) \tag{1}$$

- However, achieving 1 is not possible
- Minimize & bound difference between software clocks:
  - $\forall_{p,q \in \mathbb{N}: p,q \ \neg \mathrm{crashed}(t)} \forall_t : \ |S_p(t) - S_q(t)| \leq E_p(t) + E_q(t)$



E_p – Maximum Clock Synchronization Error

$$E_p(T) = |C_p(T, p) - \mathrm{cfn}()| + \mathcal{E}_p(T)$$

- $\mathrm{cfn}() = \mathrm{mid}\left(\left[L_p^i, U_p^i\right]\right)$
  - determines the A_p
- $\mathcal{E}_p(T) = \max\left\{\mathrm{cfn}() - \overleftarrow{M}_p(T), \overrightarrow{M}_p(T) - \mathrm{cfn}()\right\}$
  - determines the midpoint estimation error

E_p – Maximum Clock Synchronization Error

$$E_p(T) = |C_p(T, p) - \mathrm{cfn}()| + \mathcal{E}_p(T)$$

- $\mathrm{cfn}() = \mathrm{mid}\left(\left[L_p^i, U_p^i\right]\right)$
  - determines the A_p
- $\mathcal{E}_p(T) = \max\left\{\mathrm{cfn}() - \overleftarrow{M}_p(T), \overrightarrow{M}_p(T) - \mathrm{cfn}()\right\}$
  - determines the midpoint estimation error

Convergence Functions: $\mathrm{cfn}() = \mathrm{mid}\left(\left[L_p^i, U_p^i\right]\right)$

$E_p(T) = |C_p(T, p) - \mathrm{cfn}()| + \mathcal{E}_p(T)$

- A convergence function:
  - determines an abstract clock value which...
  - ...should be reached at the end of the next round
- Specifically:
  - $\mathrm{cfn}_{V0}()$ — based on [WL88]
  - $\mathrm{cfn}_{V1}()$ — based on [CF94]

Convergence Functions: $\mathrm{cfn}() = \mathrm{mid}\left(\left[L_p^i, U_p^i\right]\right)$

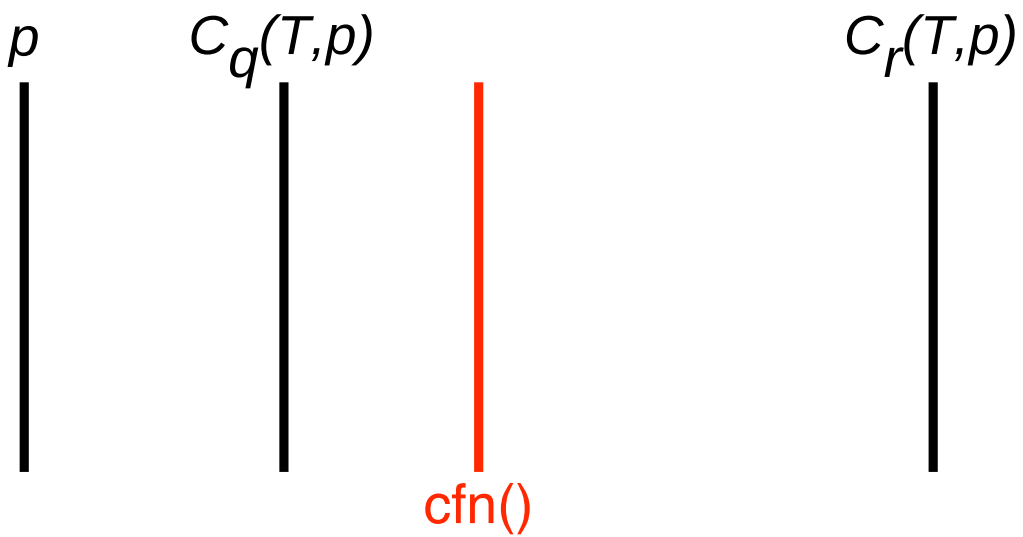
$E_p(T) = |C_p(T, p) - \mathrm{cfn}()| + \mathcal{E}_p(T)$

- A convergence function:
  - determines an abstract clock value which...
  - ...should be reached at the end of the next round
- Specifically:
  - $\mathrm{cfn}_{V0}()$ — based on [WL88]
  - $\mathrm{cfn}_{V1}()$ — based on [CF94]

Convergence Functions: $\mathrm{cfn}() = \mathrm{mid}\left(\left[L_p^i, U_p^i\right]\right)$

$E_p(T) = |C_p(T, p) - \mathrm{cfn}()| + \mathcal{E}_p(T)$

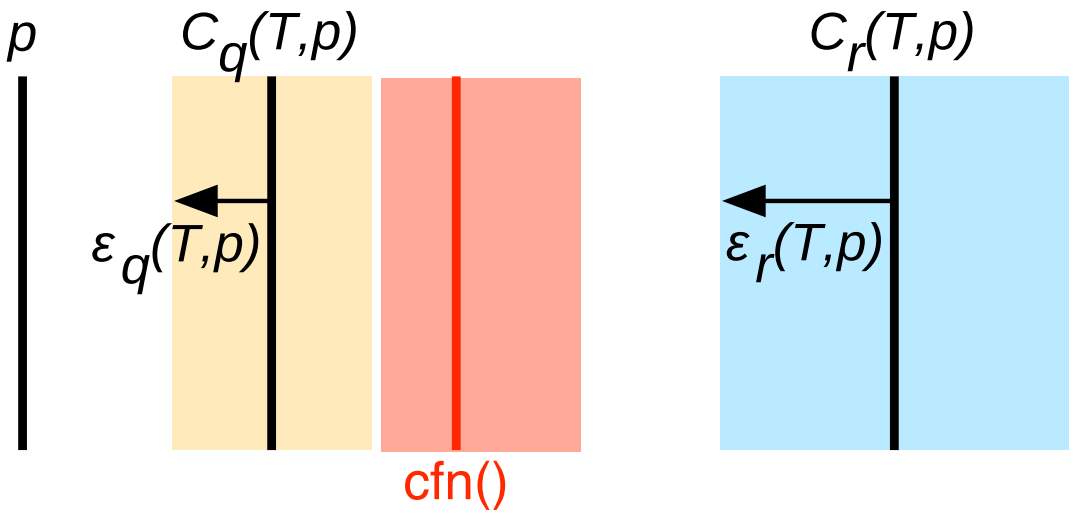
- Perfect World (no transmission delays, no drift):



Convergence Functions: $\mathrm{cfn}() = \mathrm{mid}\left(\left[L_p^i, U_p^i\right]\right)$

$E_p(T) = |C_p(T, p) - \mathrm{cfn}()| + \mathcal{E}_p(T)$

- Real Life (transmission delays, drift):



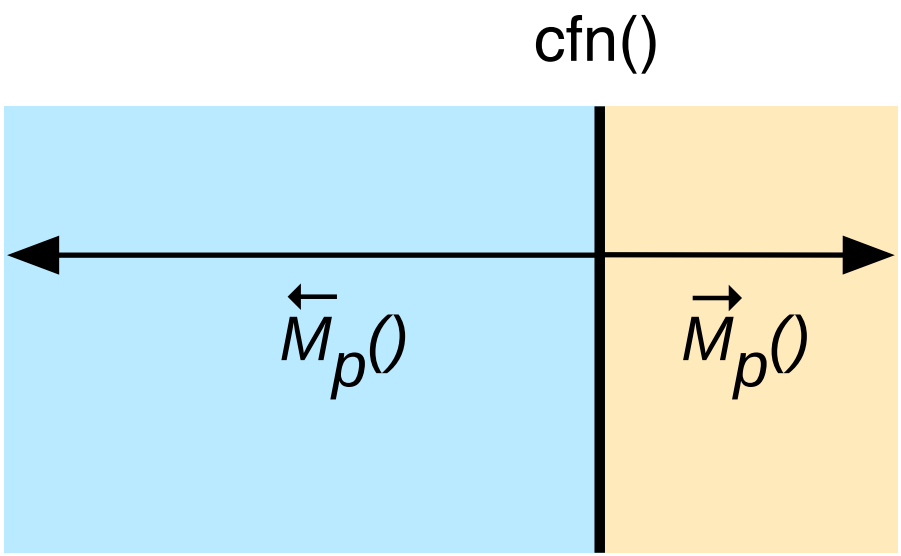
Midpoint Estimation Error: $\overrightarrow{M}_p(T), \overleftarrow{M}_p(T)$

$E_p(T) = |C_p(T, p) - \mathrm{cfn}()| + \mathcal{E}_p(T)$

- Possible values a midpoint can take:

$$\overleftarrow{M}_p(T) = \mathrm{mid}\left(\left[\overleftarrow{L}_p(T), \overleftarrow{U}_p(T)\right]\right)$$

$$\overrightarrow{M}_p(T) = \mathrm{mid}\left(\left[\overrightarrow{L}_p(T), \overrightarrow{U}_p(T)\right]\right)$$



Extrapolation & Intersection

- Re-use clock readings from previous round
- Mask transient clock reading failures (extrapolation)
- Improve readings with large error (intersection)
- Extrapolation:
  - in round i use the reading from round $i-1$
  - shift clock value by P, extend error by $(k+2)P\rho_{\max}$
- Intersection:
  - intersect readings from round i with extrapolation from $i-1$
  - both correct $\longrightarrow$ result also correct

Extrapolation & Intersection

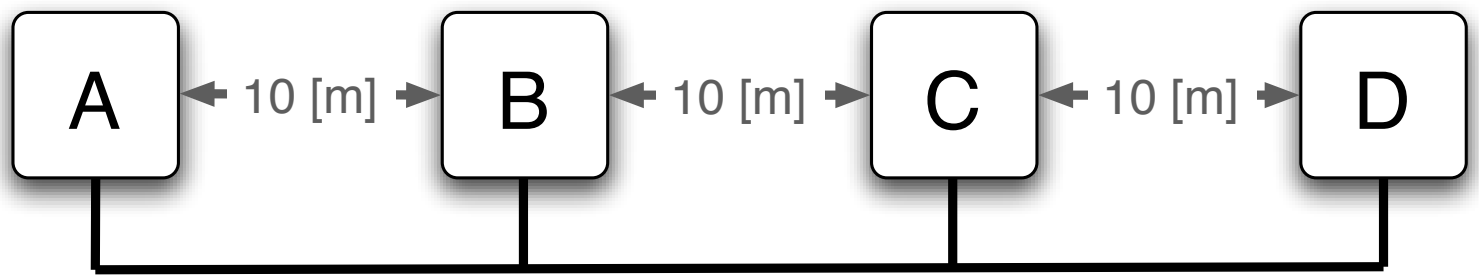
- Re-use clock readings from previous round
- Mask transient clock reading failures (extrapolation)
- Improve readings with large error (intersection)
- Extrapolation:
  - in round i use the reading from round $i-1$
  - shift clock value by P, extend error by $(k+2)P\rho_{\max}$
- Intersection:
  - intersect readings from round i with extrapolation from $i-1$
  - both correct $\longrightarrow$ result also correct

Extrapolation & Intersection

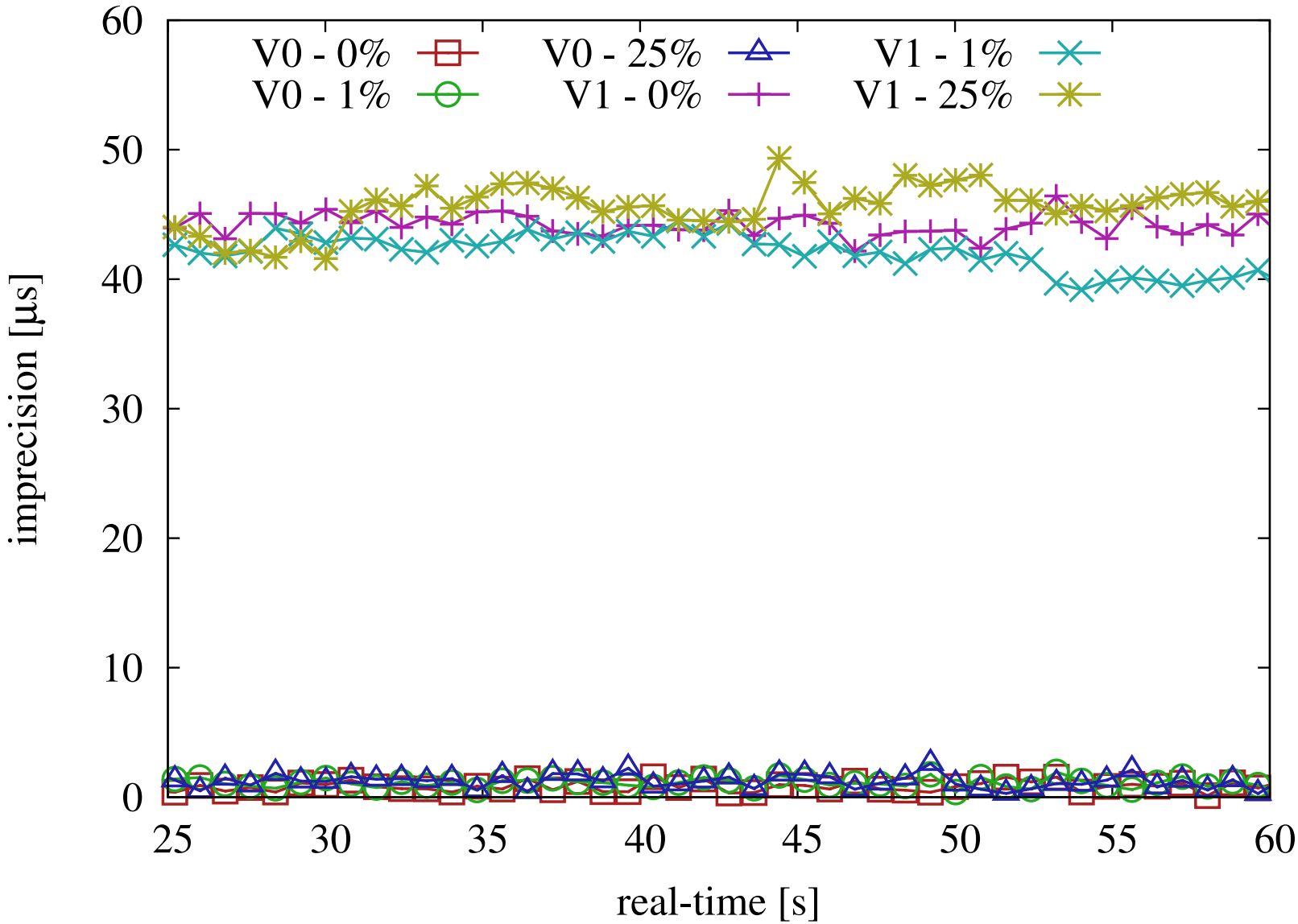
- Re-use clock readings from previous round
- Mask transient clock reading failures (extrapolation)
- Improve readings with large error (intersection)
- Extrapolation:
  - in round i use the reading from round $i-1$
  - shift clock value by P, extend error by $(k+2)P\rho_{\max}$
- Intersection:
  - intersect readings from round i with extrapolation from $i-1$
  - both correct $\longrightarrow$ result also correct

Evaluation Environment

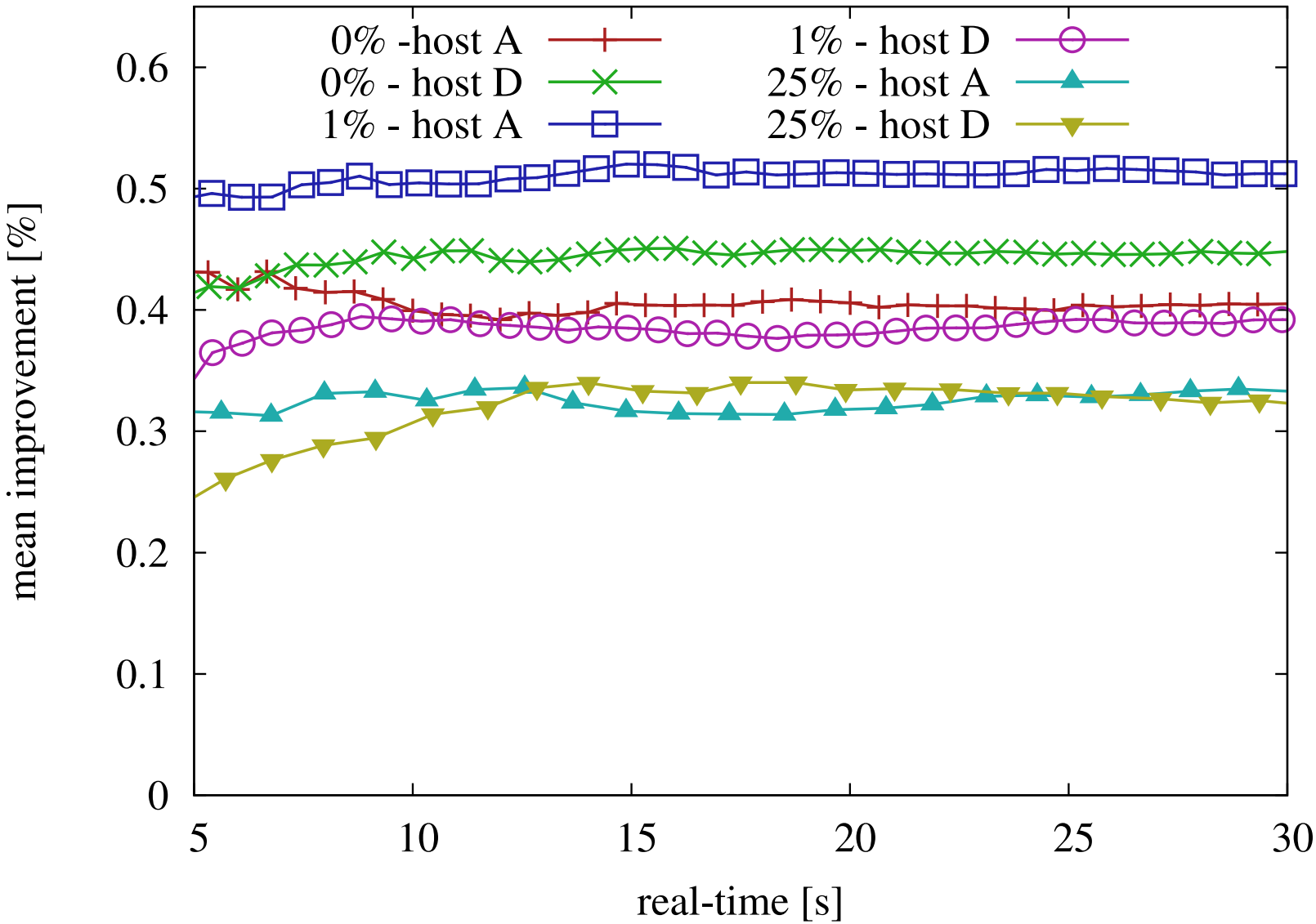
- OMNeT++ discrete event based simulator
  - global observer view only possible in simulation
- TDMA
- 10 Mbit, half-duplex, shared Ethernet
- $2 \cdot 10^5$ km/s signal propagation speed
- 4 hosts:



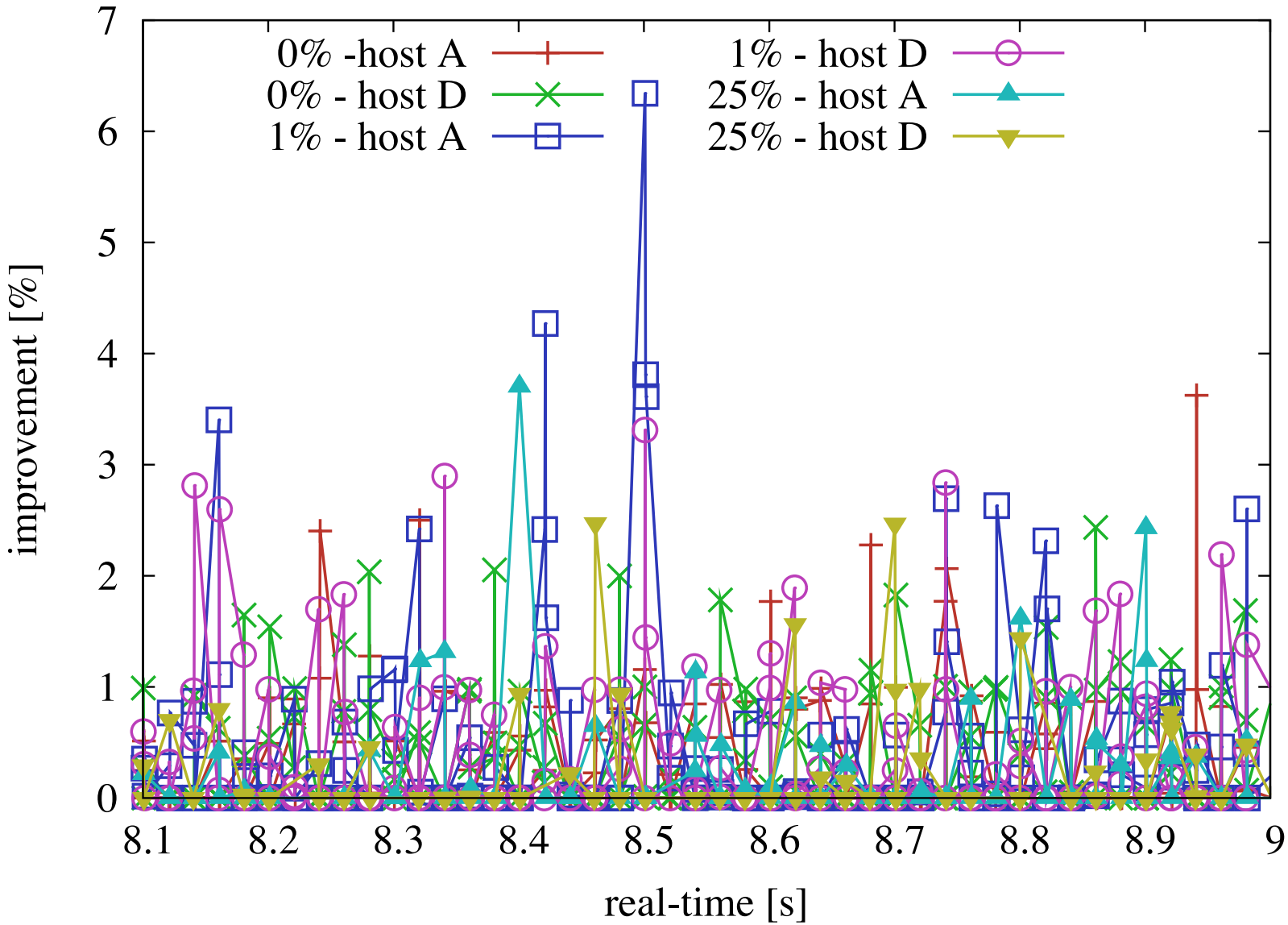
Imprecision – $\mathrm{cfn}_{V0}()$ & $\mathrm{cfn}_{V1}()$



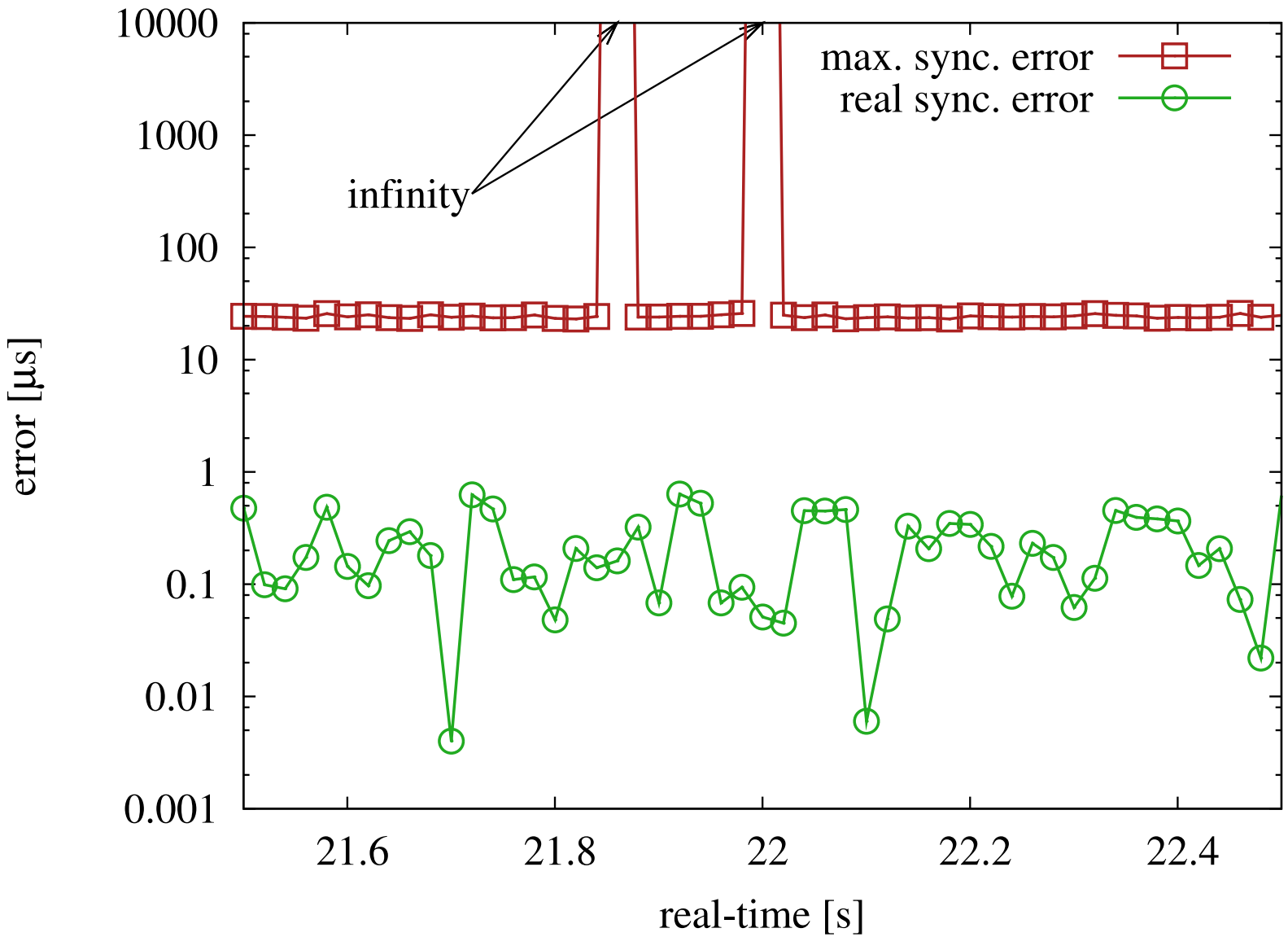
Intersection Gain



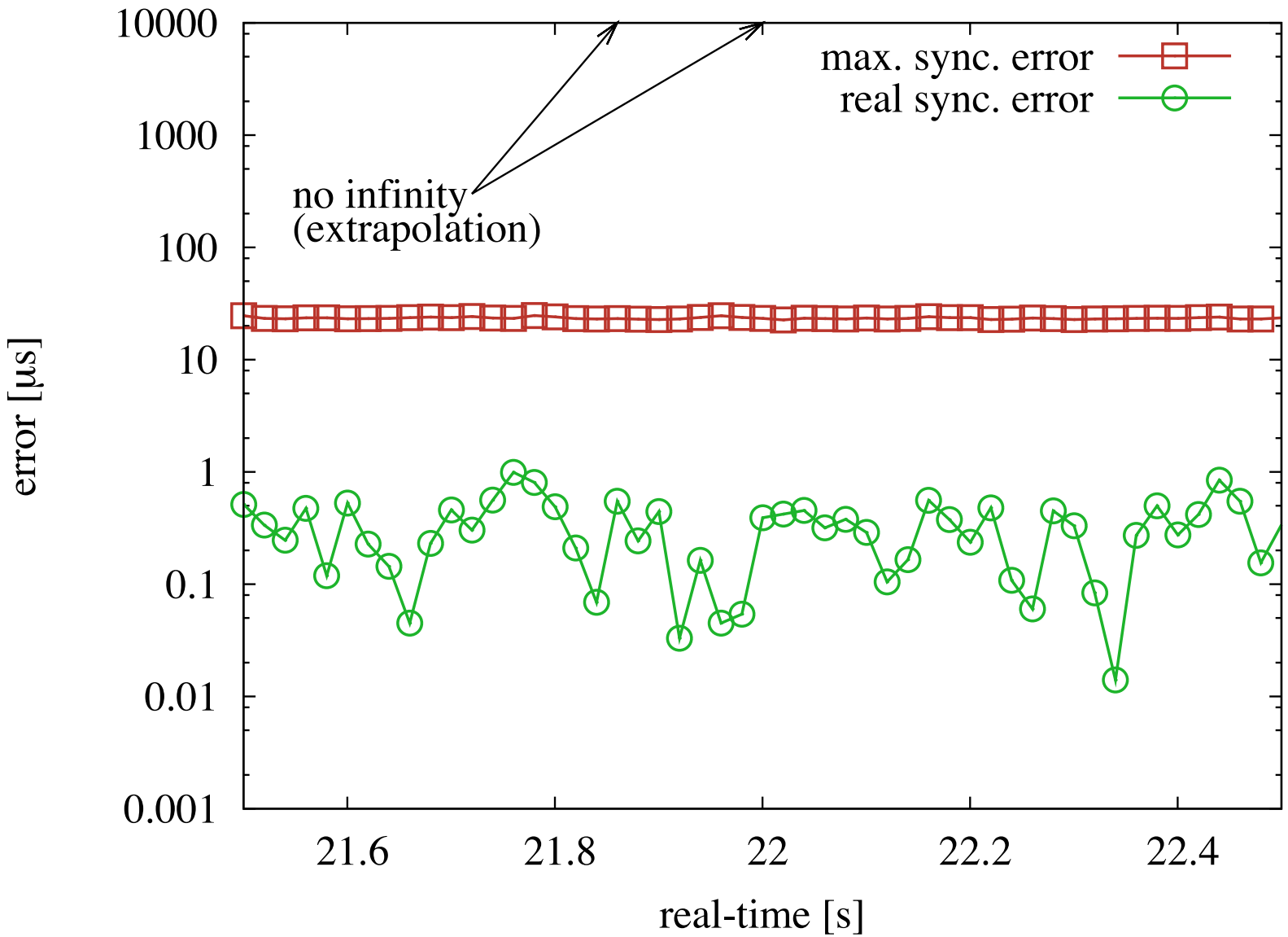
Intersection Gain



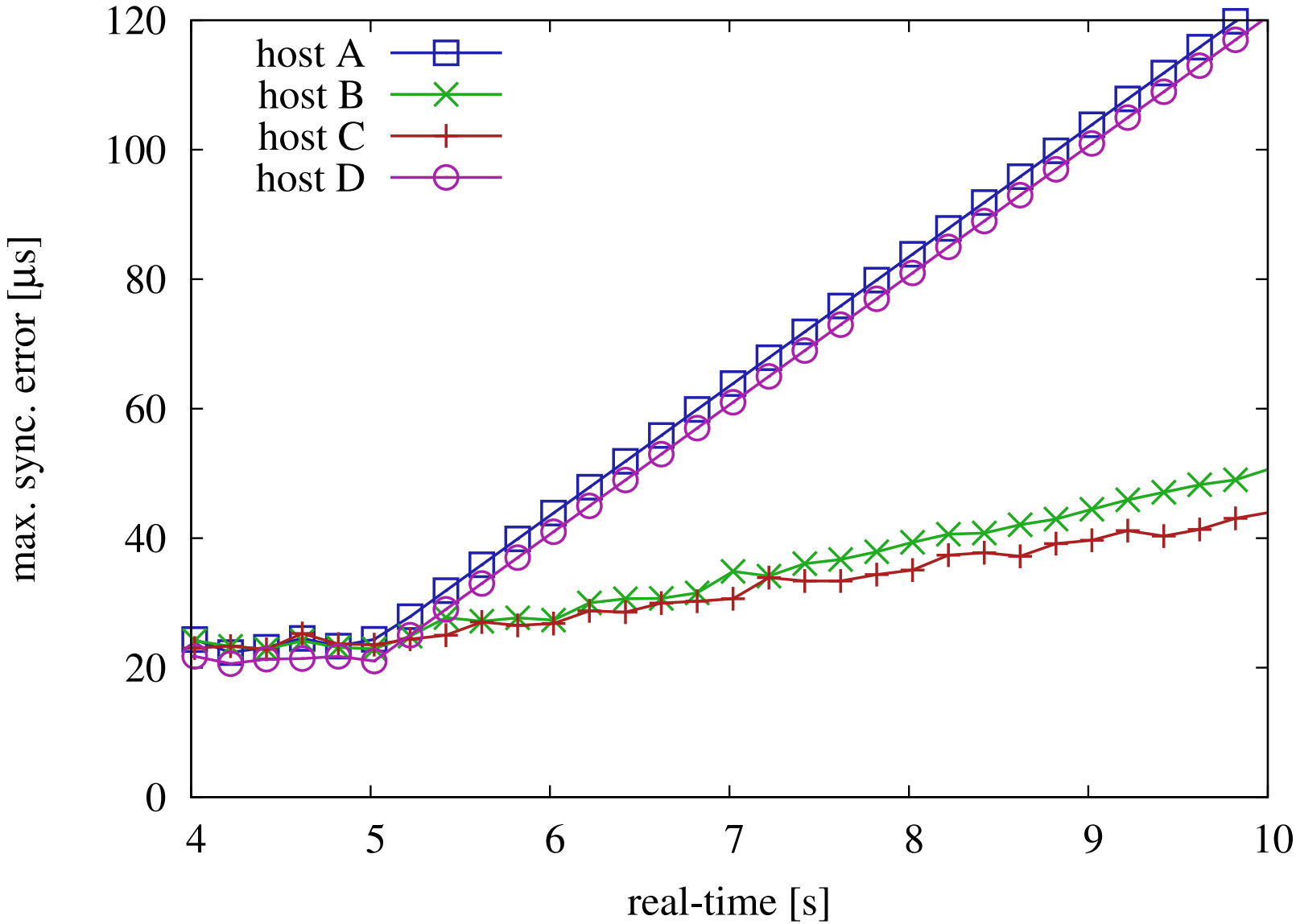
Extrapolation and Omissions



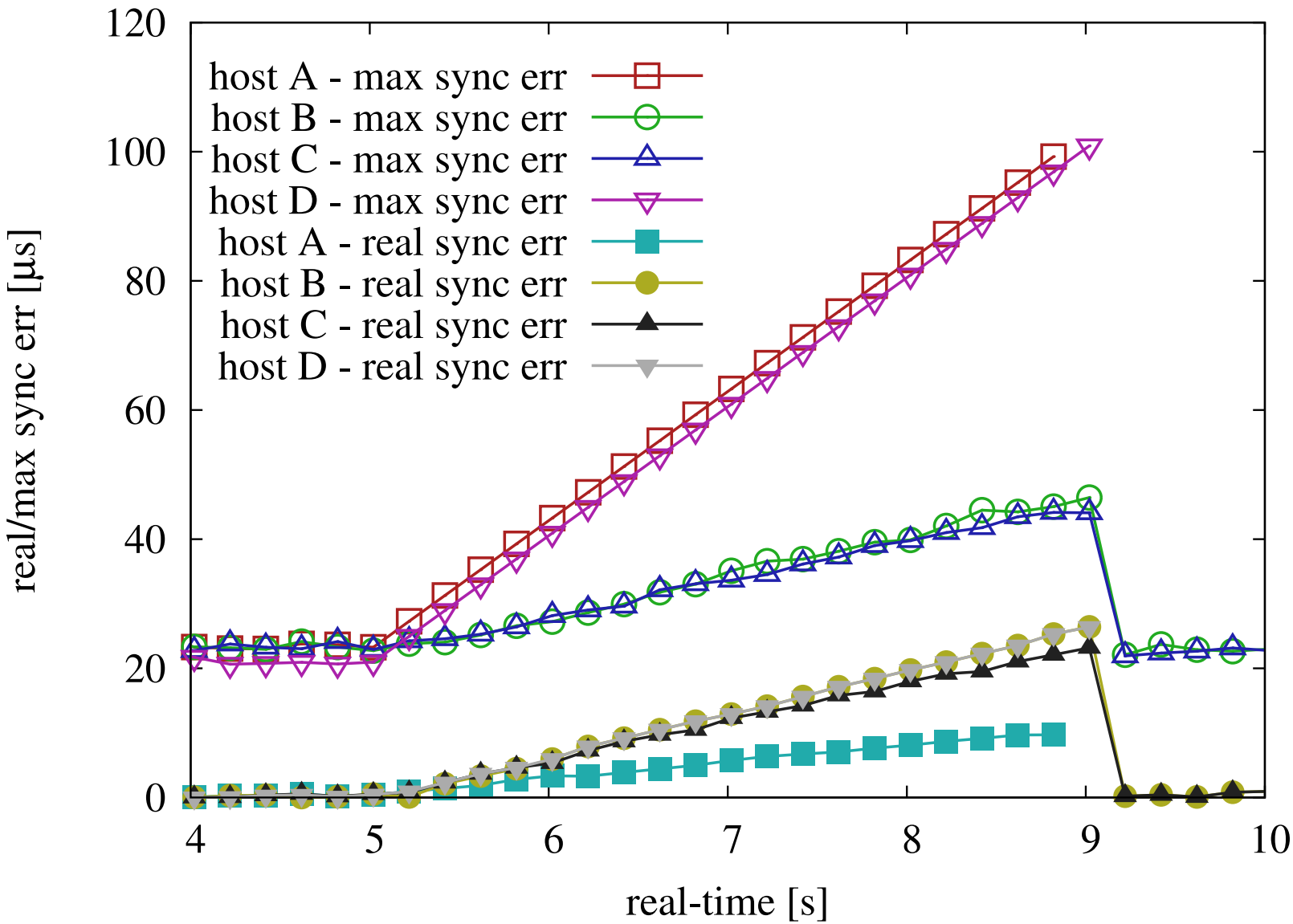
Extrapolation and Omissions



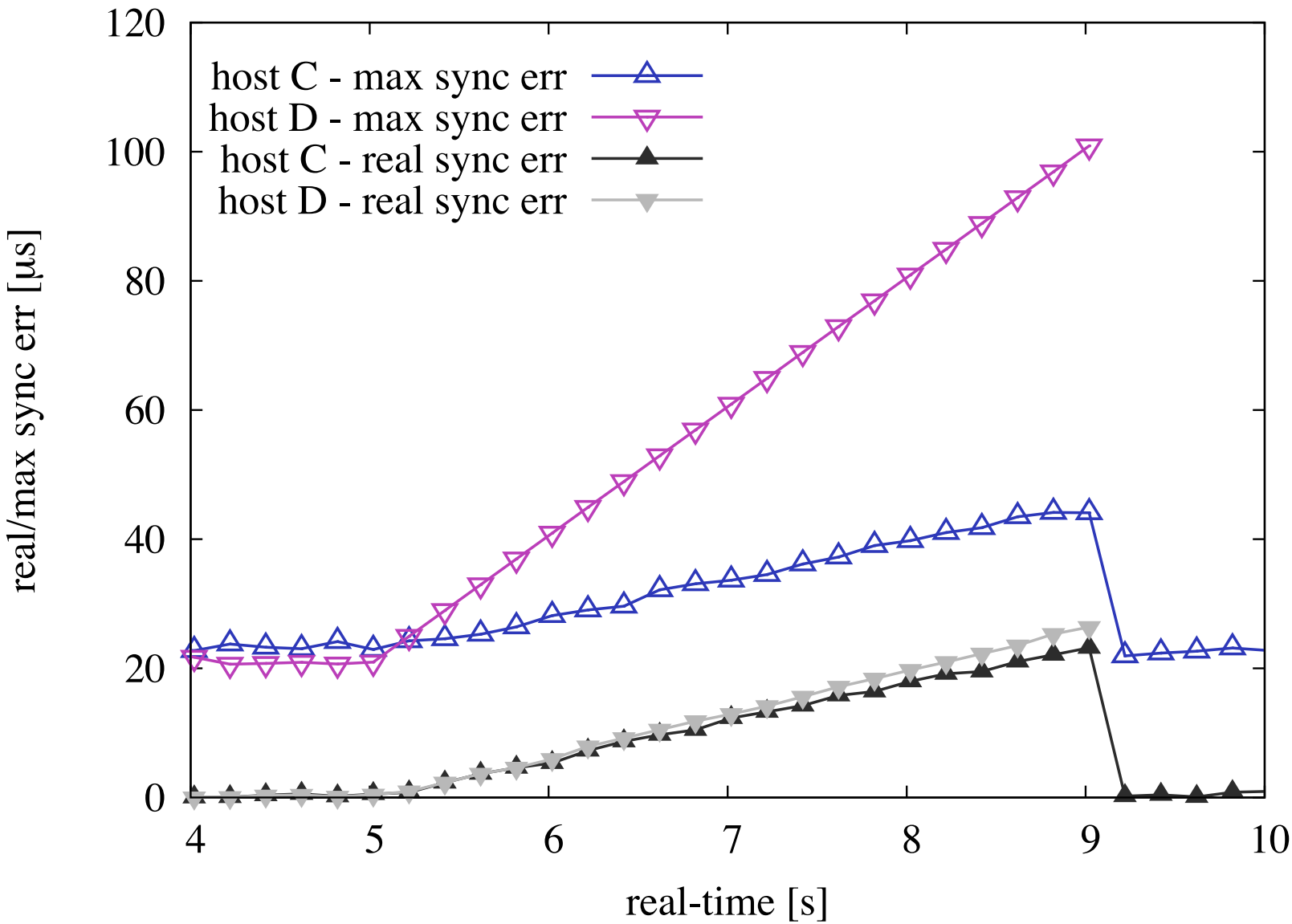
Asymmetric Permanent Omissions



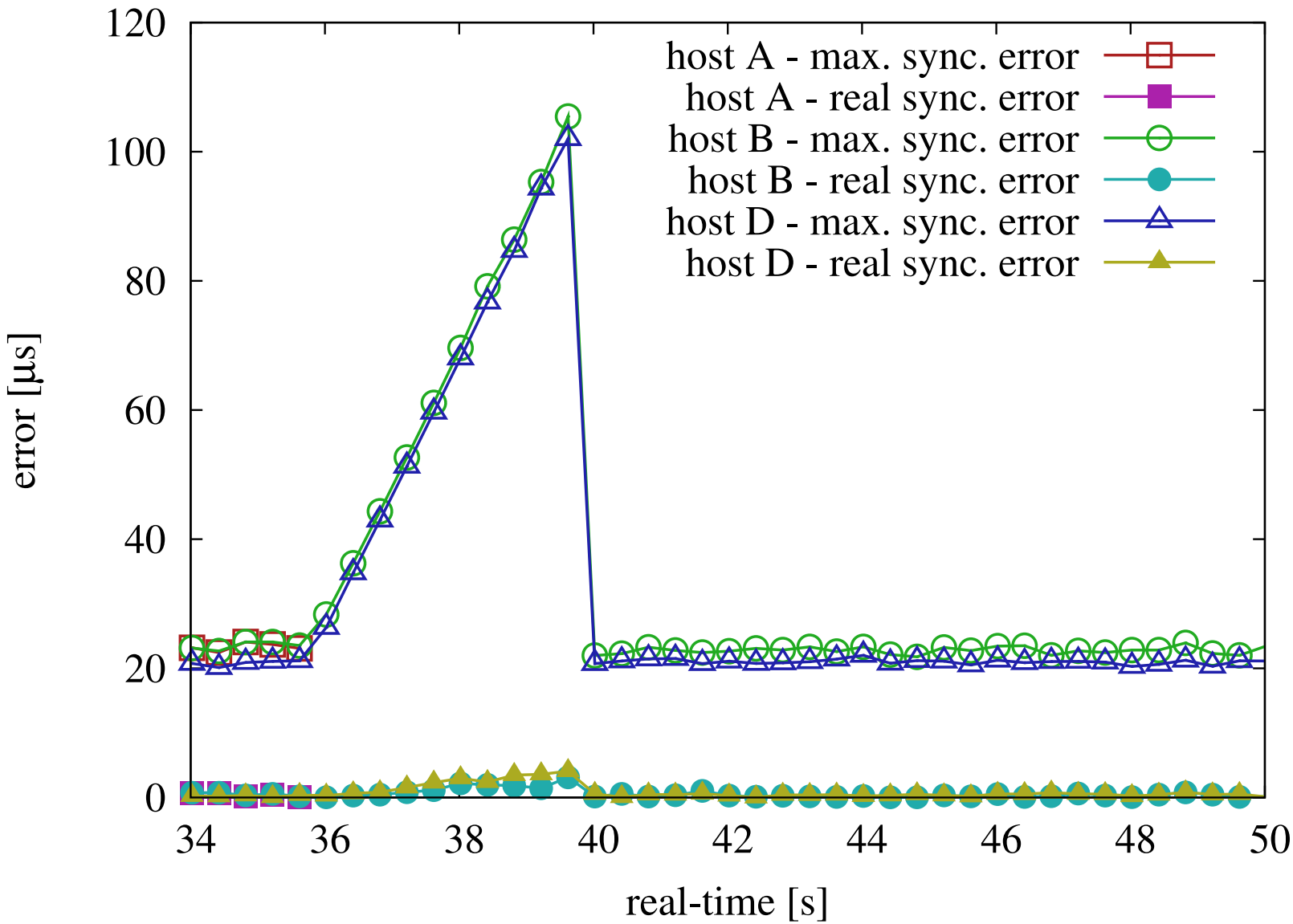
Asymmetric Permanent Omissions



Asymmetric Permanent Omissions



Crash Failures



Summary

- Need to build critical systems, however:
  - underlying components are not synchronous
  - need to use COTS to cut cost
  - need to use wireless to cut weight
- We propose Adaptive Internal Clock Synchronization
  - copes with message delays and process failures
  - bounds deviation $E_p(T)$ from correct clocks in the system
  - propagates the deviation $E_p(T)$ to upper level applications
- Applications use the $E_p(T)$ to adapt their behavior
  - no more synchronization layer crashes
  - the system is safer

Summary

- Need to build critical systems, however:
  - underlying components are not synchronous
  - need to use COTS to cut cost
  - need to use wireless to cut weight
- We propose Adaptive Internal Clock Synchronization
  - copes with message delays and process failures
  - bounds deviation $E_p(T)$ from correct clocks in the system
  - propagates the deviation $E_p(T)$ to upper level applications
- Applications use the $E_p(T)$ to adapt their behavior
  - no more synchronization layer crashes
  - the system is safer

Summary

- Need to build critical systems, however:
  - underlying components are not synchronous
  - need to use COTS to cut cost
  - need to use wireless to cut weight
- We propose Adaptive Internal Clock Synchronization
  - copes with message delays and process failures
  - bounds deviation $E_p(T)$ from correct clocks in the system
  - propagates the deviation $E_p(T)$ to upper level applications
- Applications use the $E_p(T)$ to adapt their behavior
  - no more synchronization layer crashes
  - the system is safer

Thank You!

http://wwwse.inf.tu-dresden.de/

References

Flaviu Cristian and Christof Fetzer.

Probabilistic internal clock synchronization.
In *Proceedings of the Thirteenth Symposium on Reliable Distributed Systems (SRDS1994)*, pages 22–31, October 1994.

Flaviu Cristian and Christof Fetzer.

The timed asynchronous distributed system model.
IEEE Trans. on Parallel and Distr. Systems, 10(6):642–657, June 1999.

Christof Fetzer and Flaviu Cristian.

Building fault-tolerant hardware clocks.
In *Proceedings of the Seventh IFIP Int. Working Conf. on Dependable Computing for Critical Applications*, pages 59–78, San Jose, USA, Jan 1999.

Christof Fetzer and Flaviu Cristian.

A fail-aware datagram service.
In Iain Bate and Alan Burns, editors, *IEE Proceedings - Softw. Eng.*, volume 146, pages 58–74. IEE, April 1999.

Christof Fetzer and Flaviu Cristian.

Fail-awareness: An approach to construct fail-safe systems.
Journal of Real-Time Systems, 24(2):203–238, March 2003.

Ute Wappler and Christof Fetzer.

Hardware failure virtualization via software encoded processing.
In *5th IEEE Int. Conf. on Industrial Inf. (INDIN 2007)*, volume 2, pages 977–982, June 2007.

Jennifer Lundelius Welch and Nancy Lynch.

A new fault-tolerant algorithm for clock synchronization.
Information and Computing, 77(1):1–36, 1988.