# Efficient Time Synchronization in WSNs by Adaptive Value Tracking

**Kasım Sinan YILDIRIM**

Department of Information Engineering, University of Padova, İtaly
Department of Computer Engineering, Ege University, Turkey

Summer School of Information Engineering, July, 2013


# Outline

# Outline

# Need for Synchronization

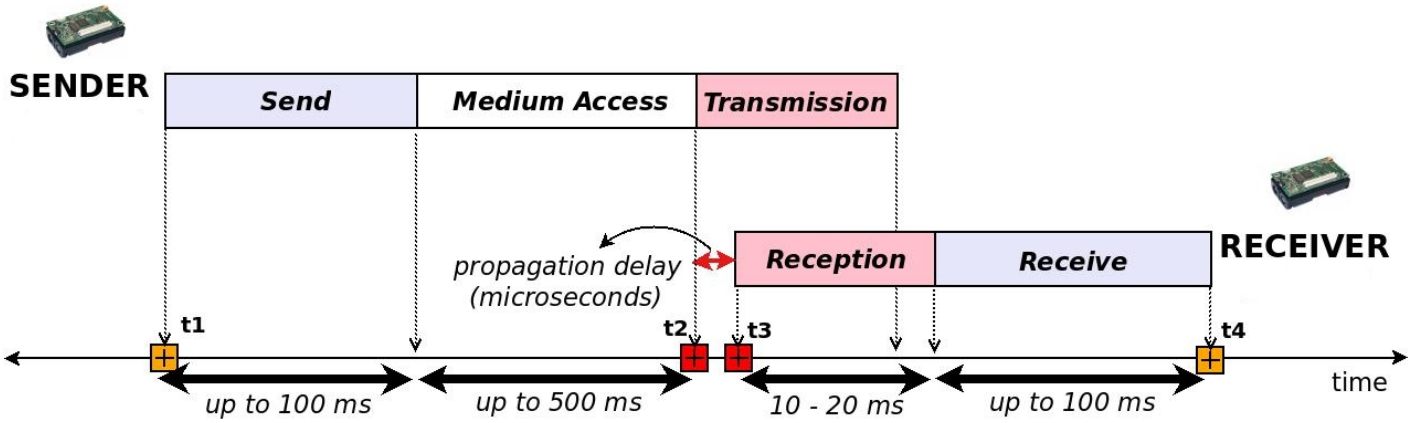
- Read-only ***Hardware clock*** $H(t)$ frequently **drifts** apart ($\pm 100$ ppm)
  - $\frac{dH(t)}{dt} = h(t)$:***rate*** (speed) of hardware clock at time $t$

# Need for Synchronization

- Read-only ***Hardware clock*** $H(t)$ frequently **drifts** apart ($\pm 100$ ppm)
  - $\frac{dH(t)}{dt} = h(t)$:***rate*** (speed) of hardware clock at time $t$
    - ★ **Local time notion** - **TDMA, Localization, Duty Cycling !!!**

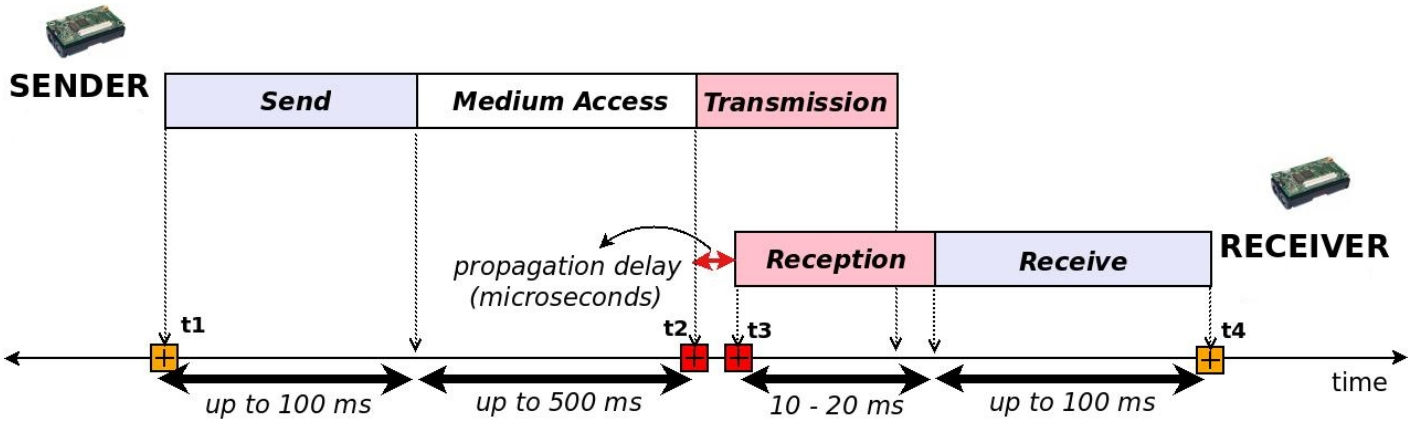
# Need for Synchronization

- Read-only ***Hardware clock*** $H(t)$ frequently **drifts** apart ($\pm 100$ ppm)
  - $\frac{dH(t)}{dt} = h(t)$:***rate*** (speed) of hardware clock at time $t$
    - ★ **Local time notion** - **TDMA, Localization, Duty Cycling !!!**
- ***Message delay*** (**non-deterministic** components) $\sim \mathcal{N}(\mu, \sigma^2)$



# Need for Synchronization

- Read-only ***Hardware clock*** $H(t)$ frequently **drifts** apart ($\pm$100 ppm)
  - $\frac{dH(t)}{dt} = h(t)$:***rate*** (speed) of hardware clock at time $t$
    - ★ **Local time notion** - **TDMA, Localization, Duty Cycling !!!**
- ***Message delay*** (**non-deterministic** components) $\sim \mathcal{N}(\mu, \sigma^2)$

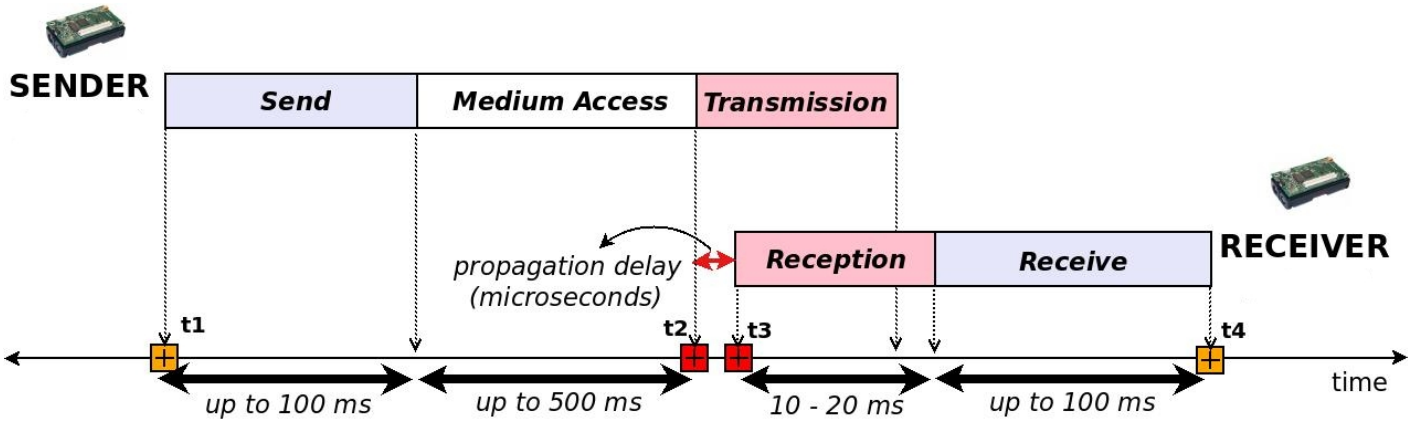



## Participate in *time synchronization*

- exchange current time information periodically (e.g. hw clock value)

# Need for Synchronization

- Read-only ***Hardware clock*** $H(t)$ frequently **drifts** apart ($\pm 100$ ppm)
  - $\frac{dH(t)}{dt} = h(t)$:***rate*** (speed) of hardware clock at time $t$
    - ★ **Local time notion** - **TDMA, Localization, Duty Cycling !!!**
- ***Message delay*** (**non-deterministic** components) $\sim \mathcal{N}(\mu, \sigma^2)$




## Participate in *time synchronization*

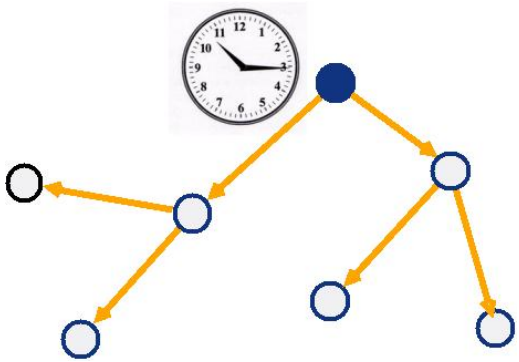
- exchange current time information periodically (e.g. hw clock value)
- calculate a ***logical clock*** $L(t)$ (represents the **common time**).
  - $\frac{dL(t)}{dt} = h(t)l(t)$ :***rate multiplier*** of $L$.

# Flooding Based Time Synchronization

- **Flooding Time Information**
  - A common approach - **external synchronization**

# Flooding Based Time Synchronization

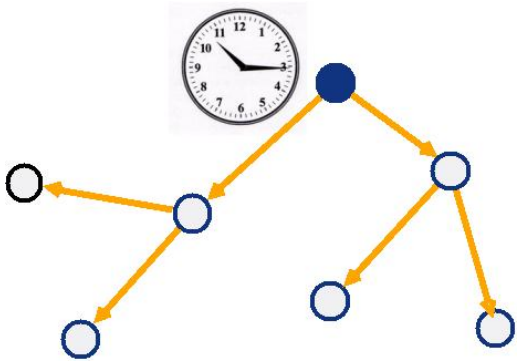
- **Flooding Time Information**
  - A common approach - **external synchronization**



- A **reference** (master) node
  - dynamically **elected** or predefined
  - **periodically floods** its current time

# Flooding Based Time Synchronization

- **Flooding Time Information**
  - A common approach - **external synchronization**

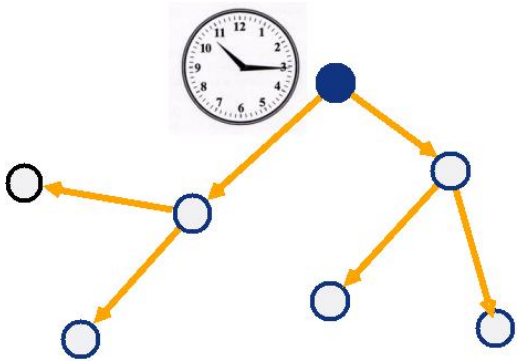


- A **reference** (master) node
  - dynamically **elected** or predefined
  - **periodically floods** its current time

- Slaves **establish a relationship** - their clock $\Longleftrightarrow$ received time
  - the relationship - **logical clock function**
  - **predict** future time - without communicating frequently
  - broadcast clock values - **network-wide synchronization**

# Flooding Based Time Synchronization

- **Flooding Time Information**
  - A common approach - **external synchronization**



- A **reference** (master) node
  - dynamically **elected** or predefined
  - **periodically floods** its current time

- Slaves **establish a relationship** - their clock $\Longleftrightarrow$ received time
  - the relationship - **logical clock function**
  - **predict** future time - without communicating frequently
  - broadcast clock values - **network-wide synchronization**
- Minimize ***clock skew*** (*synchronization error*) at any time instant
  - **Global Skew** - skew between **arbitrary** nodes - **MAIN GOAL!**

# Focus of This Talk

- We reveal the **major drawbacks** of flooding based synchronization schemes

# Focus of This Talk

- We reveal the **major drawbacks** of flooding based synchronization schemes
  - **poor synchronization performance - FTSP** [Maróti et al., 2004]
  - demanding **rapid-flooding - PulseSync** [Lenzen et al., 2009]
  - **keeping track** of the neighboring nodes **- FCSA** [Yildirim and Kantarci, 2013]
  - considerable overhead in terms of **computation** and **memory allocation - ALL Protocols!!!**

# Focus of This Talk

- We reveal the **major drawbacks** of flooding based synchronization schemes
  - **poor synchronization performance - FTSP** [Maróti et al., 2004]
  - demanding **rapid-flooding - PulseSync** [Lenzen et al., 2009]
  - **keeping track** of the neighboring nodes - **FCSA** [Yildirim and Kantarci, 2013]
  - considerable overhead in terms of **computation** and **memory allocation - ALL Protocols!!!**

**Our Question**

Is it possible to achieve high quality time synchronization **without having these drawbacks**?

# Focus of This Talk

- We reveal the **major drawbacks** of flooding based synchronization schemes
  - **poor synchronization performance - FTSP** [Maróti et al., 2004]
  - demanding **rapid-flooding - PulseSync** [Lenzen et al., 2009]
  - **keeping track** of the neighboring nodes - **FCSA** [Yildirim and Kantarci, 2013]
  - considerable overhead in terms of **computation** and **memory allocation - ALL Protocols!!!**

**Our Question**

Is it possible to achieve high quality time synchronization **without having these drawbacks**?

- We introduce a new time synchronization protocol whose main component is **adaptive-value tracking** - **AVTS protocol**

# Focus of This Talk

- We reveal the **major drawbacks** of flooding based synchronization schemes
  - **poor synchronization performance - FTSP** [Maróti et al., 2004]
  - demanding **rapid-flooding - PulseSync** [Lenzen et al., 2009]
  - **keeping track** of the neighboring nodes - **FCSA** [Yildirim and Kantarci, 2013]
  - considerable overhead in terms of **computation** and **memory allocation - ALL Protocols!!!**

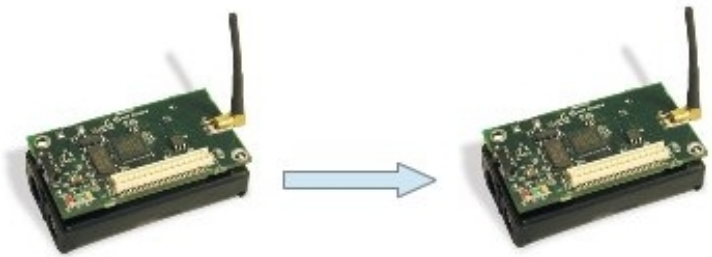
**Our Question**

Is it possible to achieve high quality time synchronization **without having these drawbacks**?

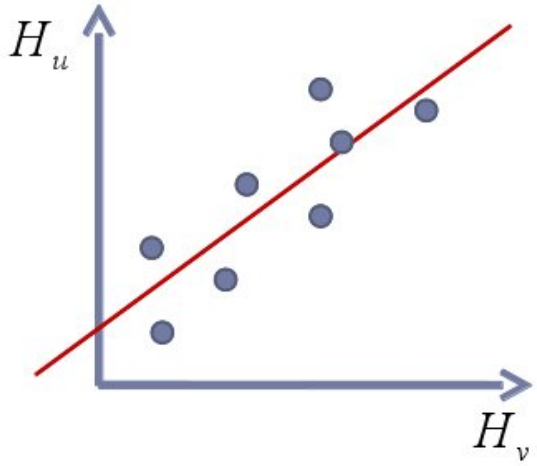
- We introduce a new time synchronization protocol whose main component is **adaptive-value tracking - AVTS protocol**
  - identical synchronization with an approximately **97% less CPU overhead** and **80% less memory allocation**

# Outline

# Master - Slave Synchronization

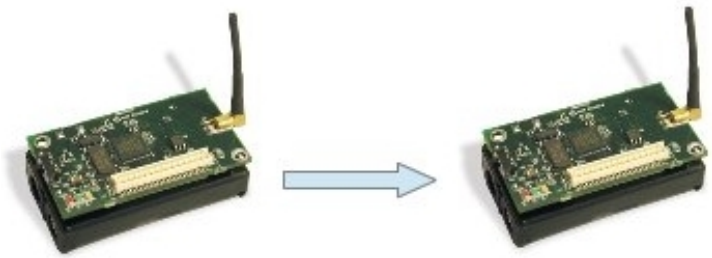


- ***Send*** time $t_1$: $\boxed{H_u(t_1)}$
- ***Receive*** time $t_2$:
  $\boxed{H_v(t_2), \hat{H}_u(t_2)}$

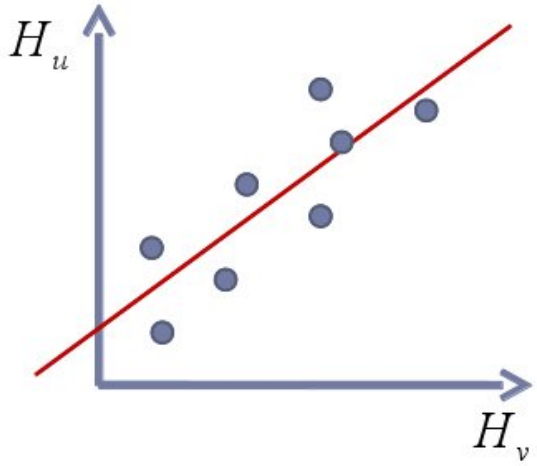



- Collected $(x_i, Y_i)$ pairs

# Master - Slave Synchronization



- ***Send*** time $t_1$: $\boxed{H_u(t_1)}$
- ***Receive*** time $t_2$: $\boxed{H_v(t_2), \hat{H}_u(t_2)}$



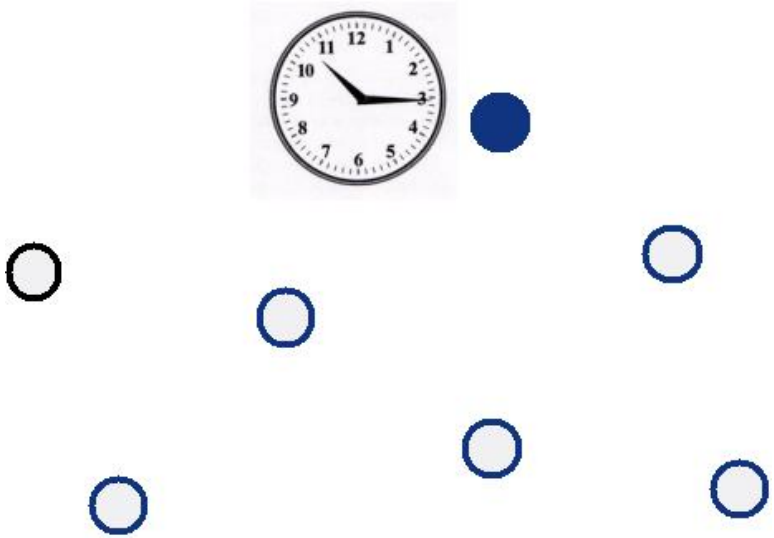

- Collected $(x_i, Y_i)$ pairs

## Least-squares line

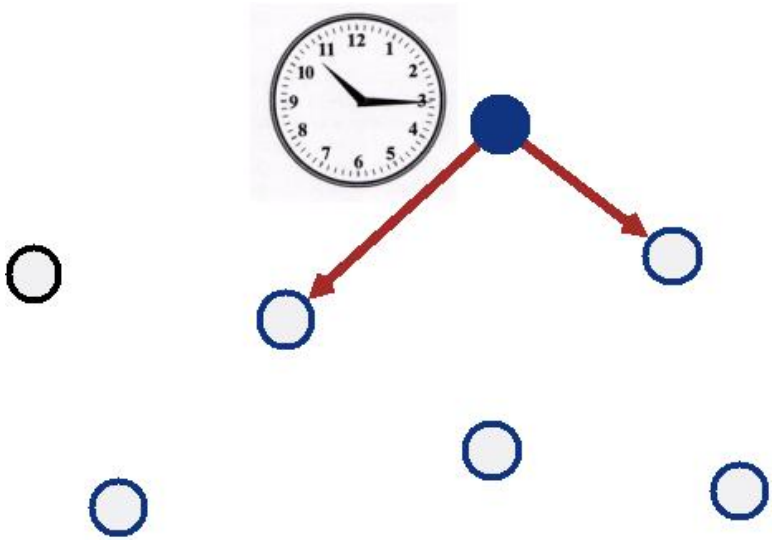
- ***Linear relationship*** is modeled as $Y_i = \alpha + \beta x_i + \varepsilon_i \sim \mathcal{N}(\mu, \sigma^2)$

- **Logical Clock** $L_v = \hat{\alpha} + \hat{\beta} H_v$
  - $\hat{\beta} = \frac{h_u}{h_v} = \frac{\sum(x_i - \bar{x})(Y_i - \bar{Y})}{\sum(x_i - \bar{x})^2}$ - **rate multiplier**
  - $\hat{\alpha} = \overline{Y} - \hat{\beta}\overline{x}$ - **offset**

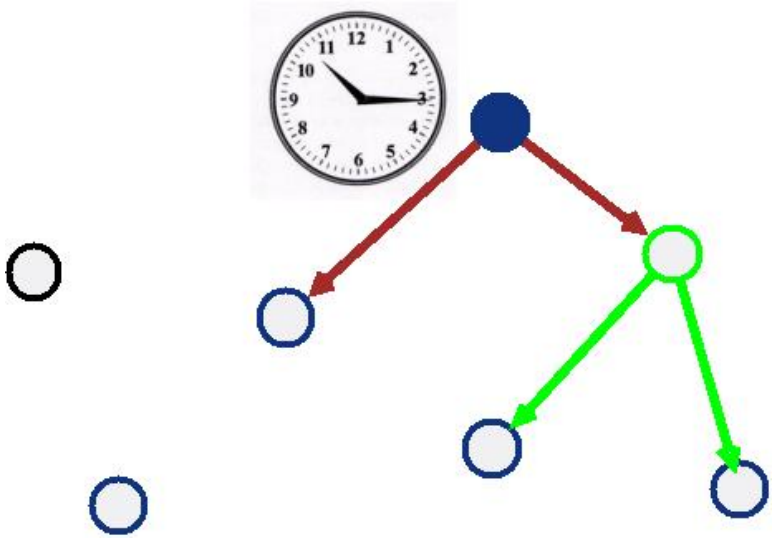
# Flooding Time Synchronization Protocol (FTSP) [Maróti et al., 2004]



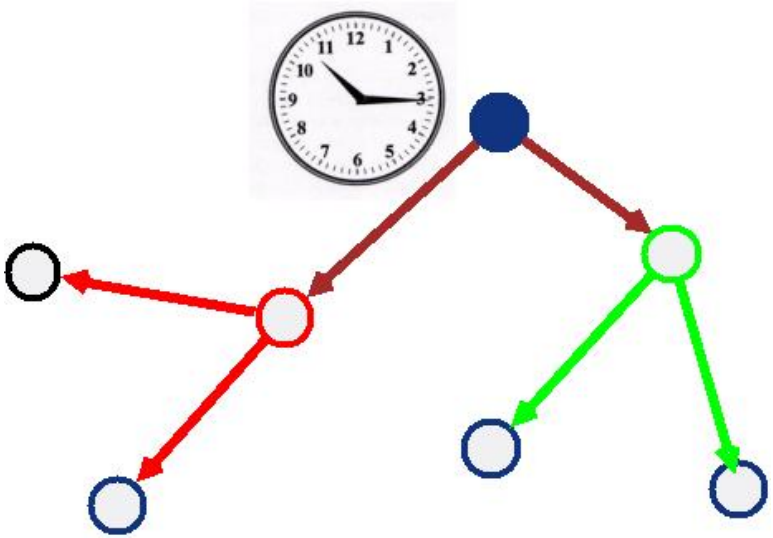
# Flooding Time Synchronization Protocol (FTSP) [Maróti et al., 2004]



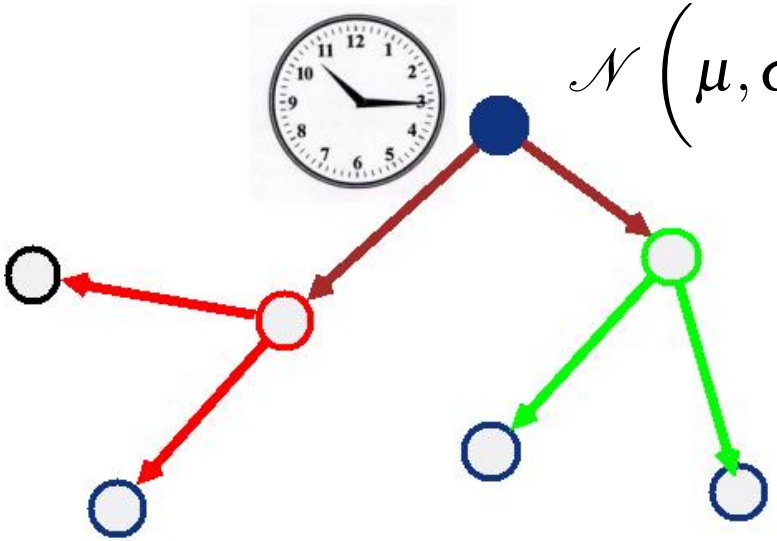
# Flooding Time Synchronization Protocol (FTSP) [Maróti et al., 2004]



# Flooding Time Synchronization Protocol (FTSP) [Maróti et al., 2004]

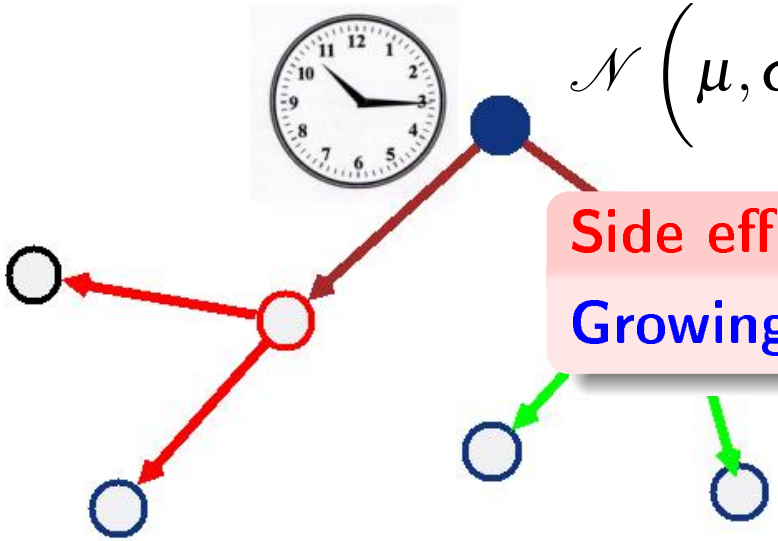


# Flooding Time Synchronization Protocol (FTSP) [Maróti et al., 2004]



$$\mathcal{N}\left(\mu, \sigma^2\left(1+\frac{1}{N}+\frac{(x^{'}-\bar{x})^2}{\sum(x_i-\bar{x})^2}\right)\right)$$

# Flooding Time Synchronization Protocol (FTSP) [Maróti et al., 2004]

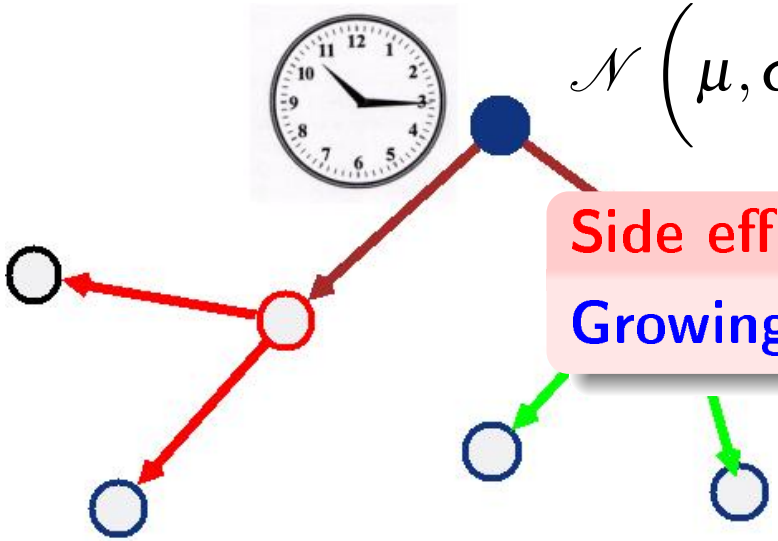


$$\mathcal{N}\left(\mu, \sigma^2\left(1+\frac{1}{N}+\frac{(x^{'}-\bar{x})^2}{\sum(x_i-\bar{x})^2}\right)\right)$$

**Side effect of the waiting times**

**Growing Prediction Error!!!**

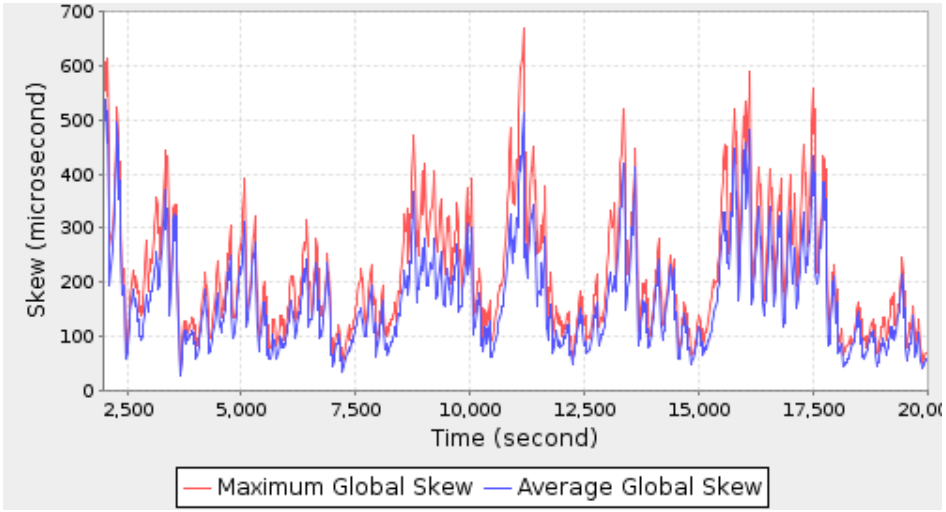
# Flooding Time Synchronization Protocol (FTSP) [Maróti et al., 2004]

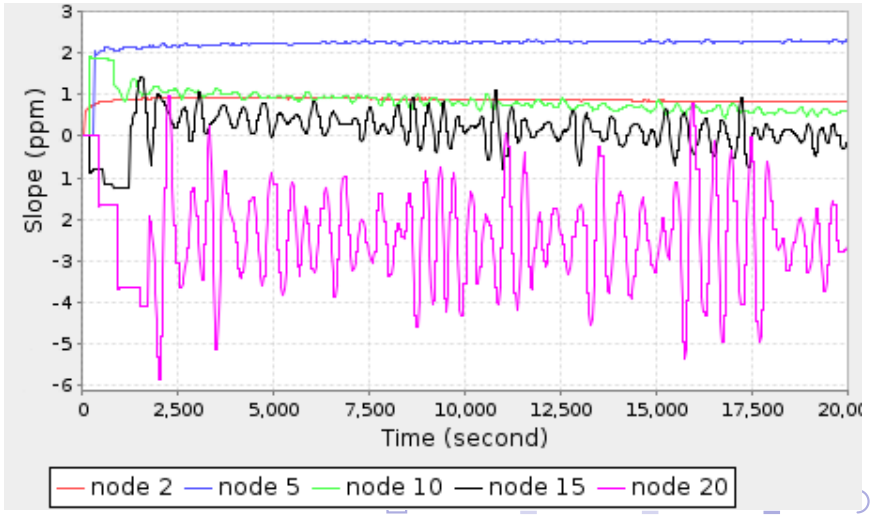


$$\mathcal{N}\left(\mu, \sigma^2\left(1+\frac{1}{N}+\frac{(x^{'}-\bar{x})^2}{\sum(x_i-\bar{x})^2}\right)\right)$$

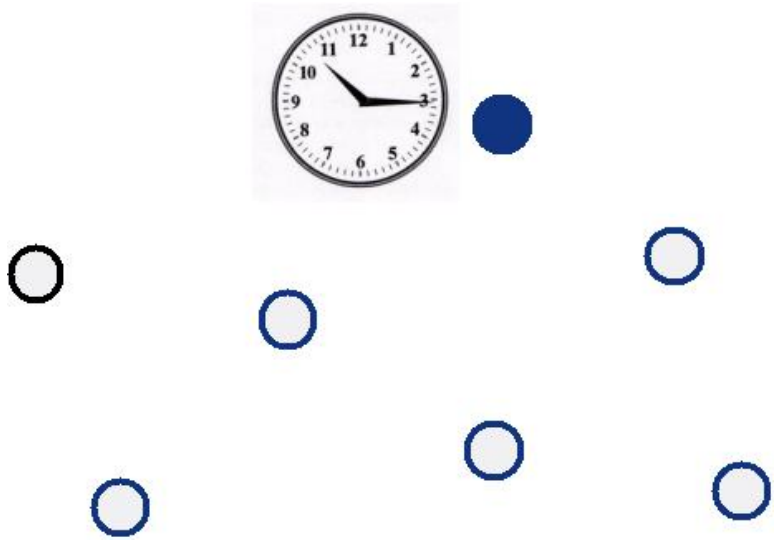
**Side effect of the waiting times**

**Growing Prediction Error!!!**

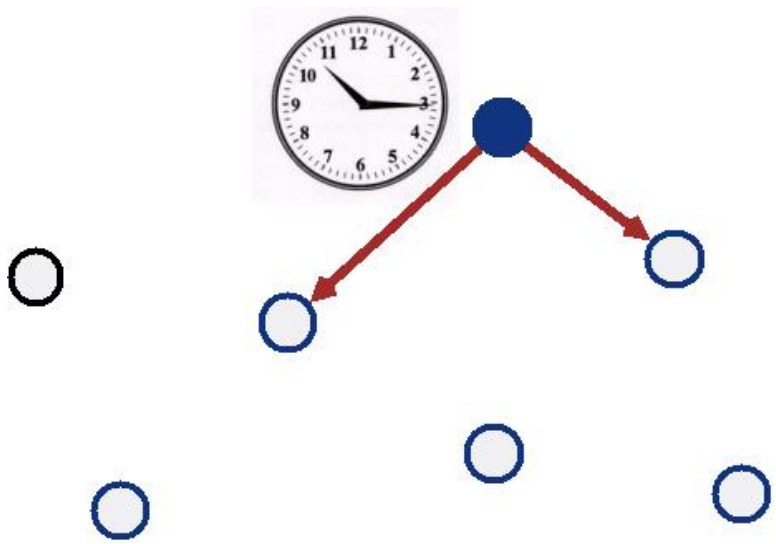


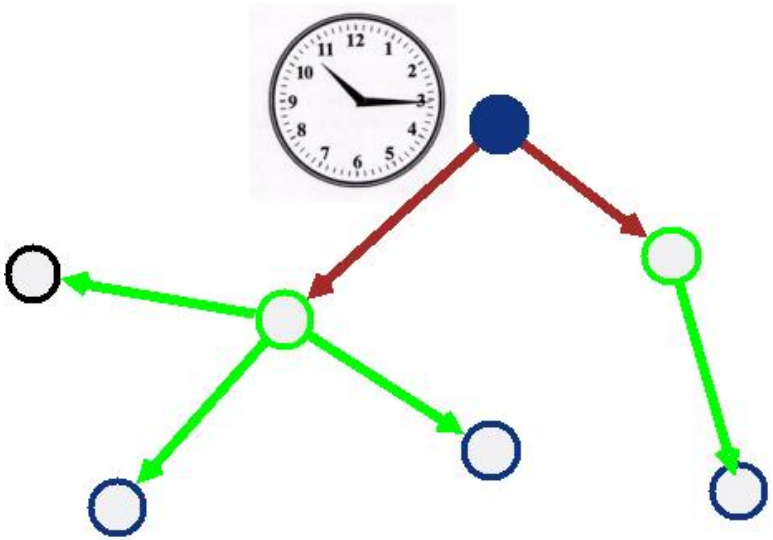
# PulseSync [Lenzen et al., 2009]



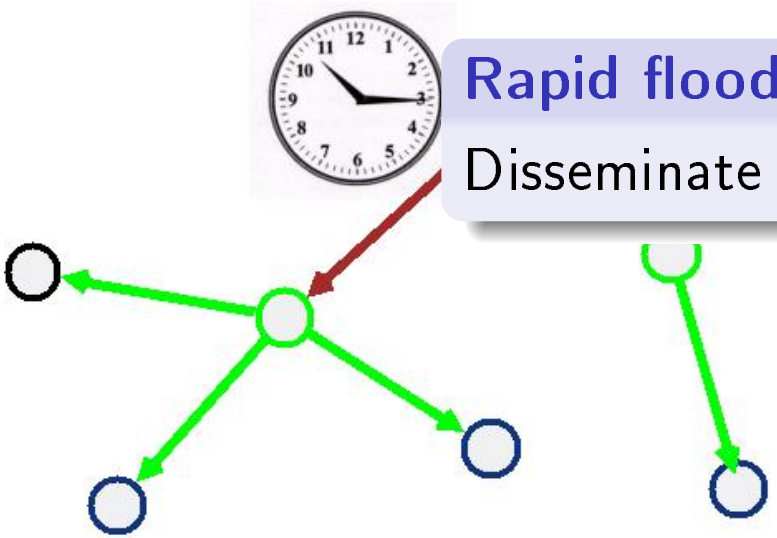
# PulseSync [Lenzen et al., 2009]



# PulseSync [Lenzen et al., 2009]



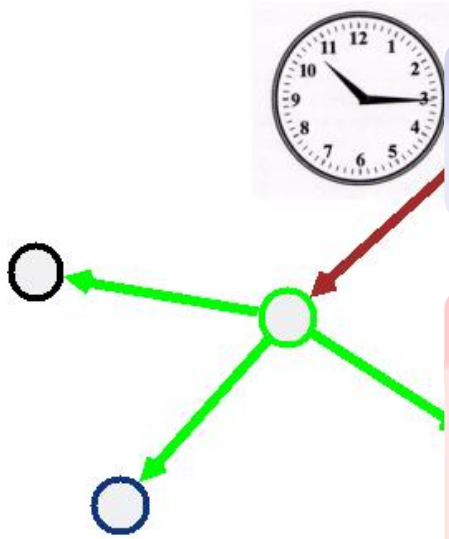
# PulseSync [Lenzen et al., 2009]



**Rapid flooding**

Disseminate as fast and reliable as possible

# PulseSync [Lenzen et al., 2009]



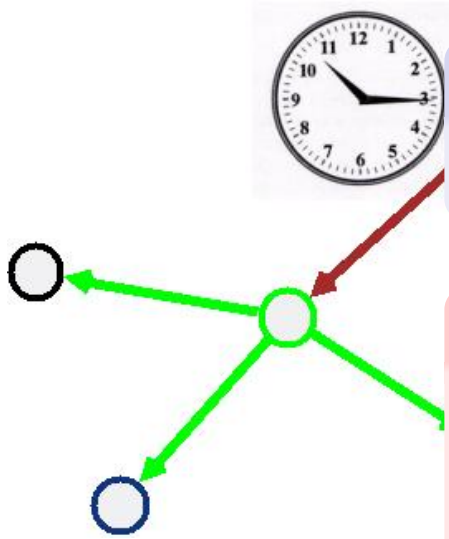
## Rapid flooding

Disseminate as fast and reliable as possible

## Drawbacks

- *Contention* - trasmission scheduling - density
- *Reliable rapid flooding* - packet losses

# PulseSync [Lenzen et al., 2009]



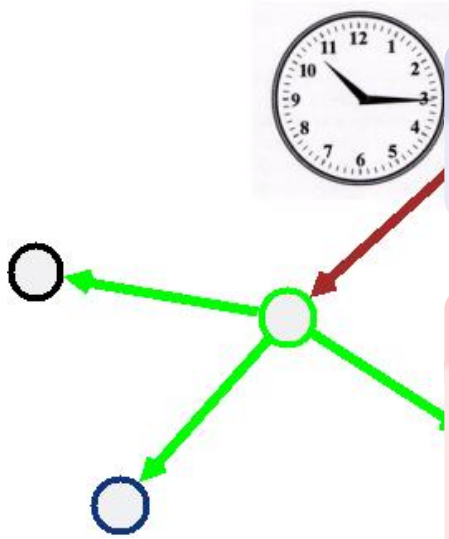
## Rapid flooding

Disseminate as fast and reliable as possible

## Drawbacks

- ***Contention*** - **trasmission scheduling** - **density**
- ***Reliable rapid flooding*** - **packet losses**
- ***constructive interference (CI)*** [Ferrari et al., 2011]
  - ***Scalability problem!!!***[Wang et al., 2012]

# PulseSync [Lenzen et al., 2009]

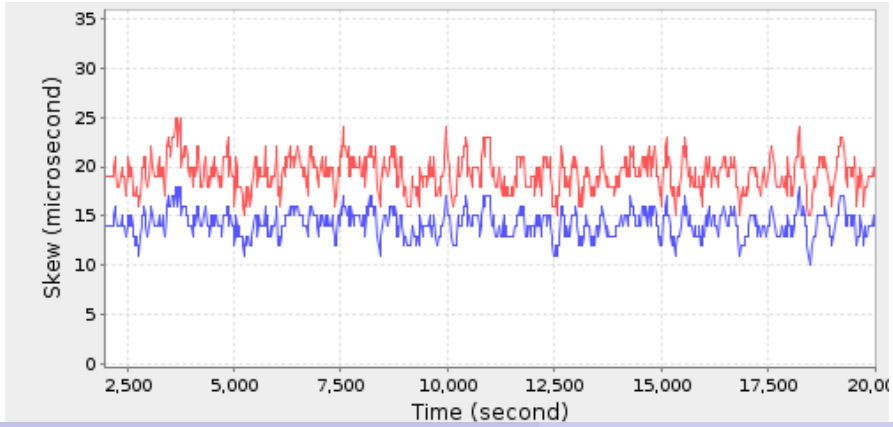


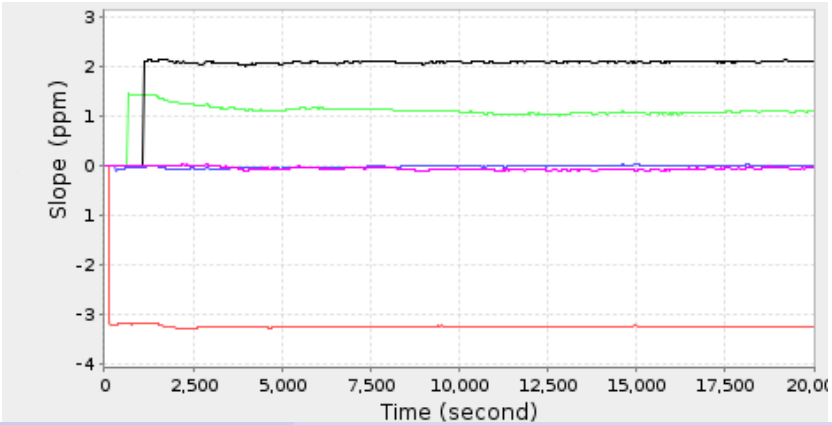
## Rapid flooding

Disseminate as fast and reliable as possible

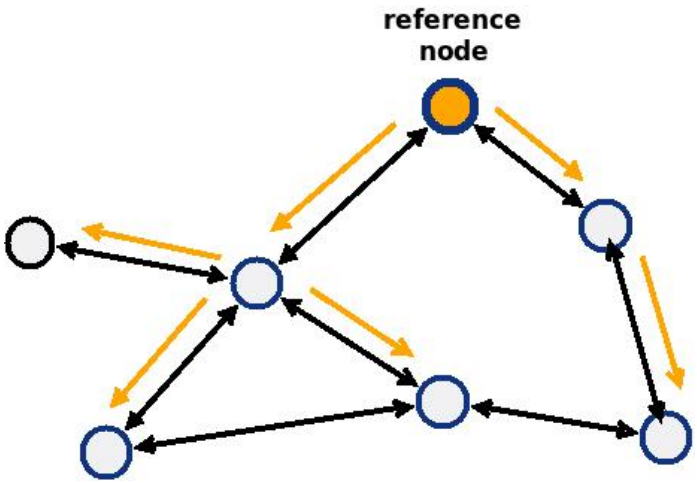
## Drawbacks

- ***Contention*** - **trasmission scheduling** - **density**
- ***Reliable rapid flooding*** - **packet losses**
- ***constructive interference (CI)*** [Ferrari et al., 2011]
  - ***Scalability problem!!!***[Wang et al., 2012]

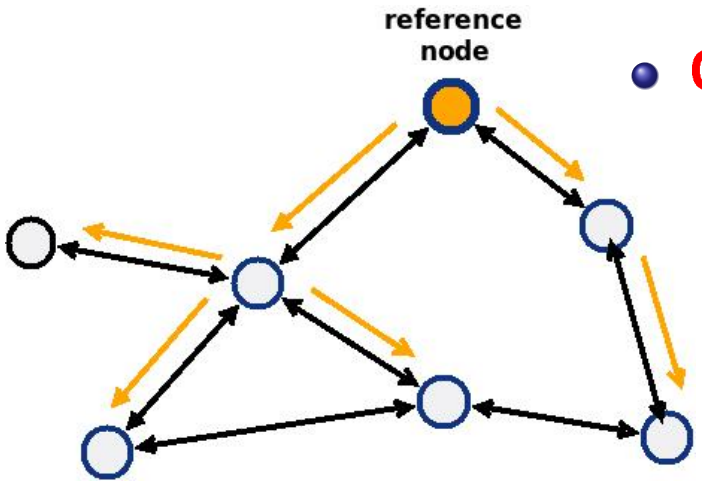


# FCSA [Yildirim and Kantarci, 2013]

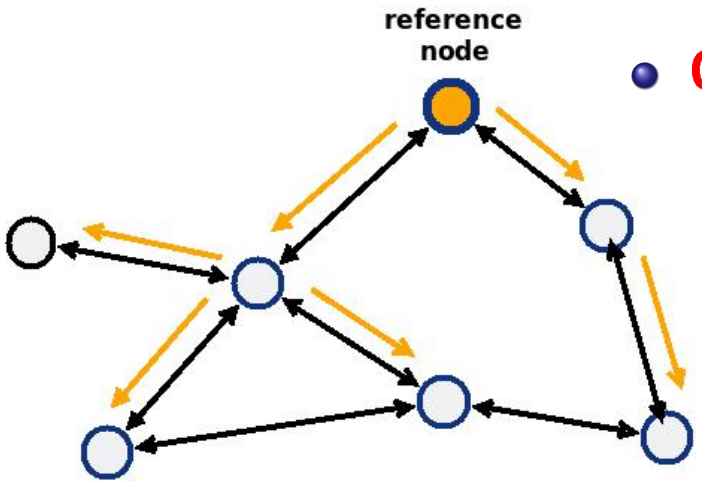


# FCSA [Yildirim and Kantarci, 2013]



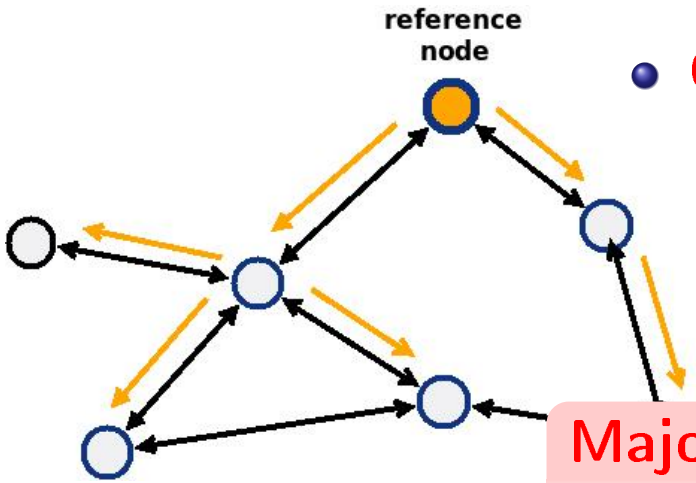

- **Clock Speed Agreement**
  - $l_v(t^+) = \frac{l_v(t) + \sum_{u \in \mathcal{N}_v} \frac{h_u}{h_v}(t).l_u(t)}{|\mathcal{N}_v|+1}$

# FCSA [Yildirim and Kantarci, 2013]




- **Clock Speed Agreement**
  - $l_v(t^+) = \frac{l_v(t)+\sum_{u\in\mathcal{N}_v} \frac{h_u}{h_v}(t).l_u(t)}{|\mathcal{N}_v|+1}$
  - $\forall v \in V : \lim_{t\to\infty} (h_v(t).l_v(t)) = speed$

# FCSA [Yildirim and Kantarci, 2013]



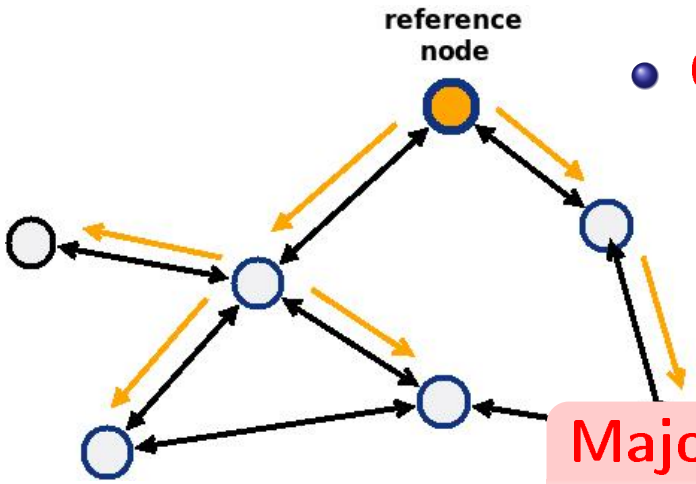

- **Clock Speed Agreement**
  - $l_v(t^+) = \frac{l_v(t) + \sum_{u \in \mathcal{N}_v} \frac{h_u}{h_v}(t).l_u(t)}{|\mathcal{N}_v|+1}$
  - $\forall v \in V : \lim_{t \to \infty} (h_v(t).l_v(t)) = speed$

**Major Drawback**

Which neighbors to keep track / **discard**?

- Memory constraints - **density** - **connectivity**

# FCSA [Yildirim and Kantarci, 2013]

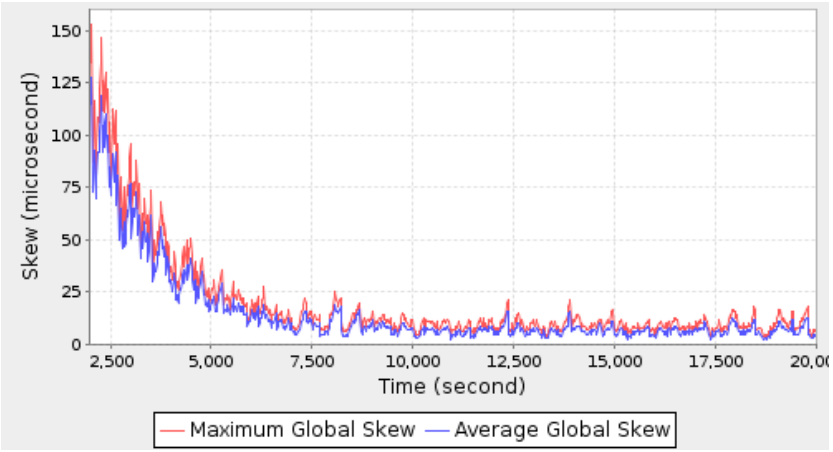


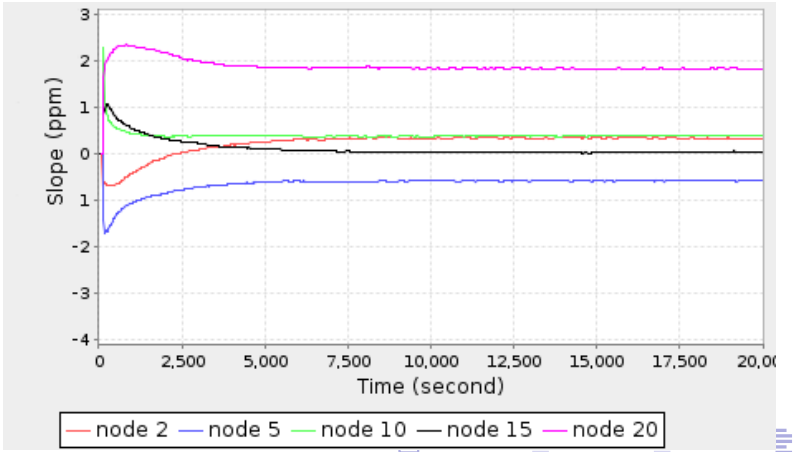

- **Clock Speed Agreement**
  - $l_v(t^+) = \frac{l_v(t) + \sum_{u \in \mathcal{N}_v} \frac{h_u}{h_v}(t).l_u(t)}{|\mathcal{N}_v|+1}$
  - $\forall v \in V : \lim_{t \to \infty} (h_v(t).l_v(t)) = speed$

**Major Drawback**

Which neighbors to keep track / **discard**?

- Memory constraints - **density** - **connectivity**



# Outline

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^{\star}$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^\star$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$
  - proposes a ***value*** $v_t \in AVT_{ss}$ to its **environment**: $v_t = avt.value(t)$ at any time t

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^\star$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$
  - proposes a ***value*** $v_t \in AVT_{ss}$ to its **environment**: $v_t = avt.value(t)$ at any time t
- ***Without knowing the correct value*** $v^\star$

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^\star$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$
  - proposes a ***value*** $v_t \in AVT_{ss}$ to its **environment**: $v_t = avt.value(t)$ at any time t
- ***Without knowing the correct value*** $v^\star$
  - the environment has to **determine** if $v^\star > v_t$, $v^\star < v_t$ or $v^\star = v_t$

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^\star$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$
  - proposes a ***value*** $v_t \in AVT_{ss}$ to its **environment**: $v_t = avt.value(t)$ at any time t
- ***Without knowing the correct value*** $v^\star$
  - the environment has to **determine** if $v^\star > v_t$, $v^\star < v_t$ or $v^\star = v_t$
  - the environment sends ***feedbacks*** of the form $avt.adjust(f_{v_t} \in \mathscr{F})$

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^\star$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$
  - proposes a ***value*** $v_t \in AVT_{ss}$ to its **environment**: $v_t = avt.value(t)$ at any time t
- ***Without knowing the correct value*** $v^\star$
  - the environment has to **determine** if $v^\star > v_t$, $v^\star < v_t$ or $v^\star = v_t$
  - the environment sends ***feedbacks*** of the form $avt.adjust(f_{v_t} \in \mathscr{F})$
    - ⋆ **increasing** $v_t$ ($f \uparrow$), **decreasing** $v_t$ ($f \downarrow$) or informing that $v_t$ is **good** ($f \approx$).

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^\star$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$
  - proposes a ***value*** $v_t \in AVT_{ss}$ to its **environment**: $v_t = avt.value(t)$ at any time t
- ***Without knowing the correct value*** $v^\star$
  - the environment has to **determine** if $v^\star > v_t$, $v^\star < v_t$ or $v^\star = v_t$
  - the environment sends ***feedbacks*** of the form $avt.adjust(f_{v_t} \in \mathscr{F})$
    - ⋆ **increasing** $v_t$ ($f \uparrow$), **decreasing** $v_t$ ($f \downarrow$) or informing that $v_t$ is **good** ($f \approx$).
- In our case, the environment of an *avt* is its **sensor node**

# The Method of Adaptive Value Tracking [Lemouzy et al., 2011]

- **Adaptive Value Tracker** *avt*
  - finds and tracks a ***dynamic*** value $v^\star$ in a given search space $AVT_{ss} = [v_{min}, v_{max}] \subset \mathbb{R}$
  - proposes a ***value*** $v_t \in AVT_{ss}$ to its **environment**: $v_t = avt.value(t)$ at any time t
- ***Without knowing the correct value*** $v^\star$
  - the environment has to **determine** if $v^\star > v_t$, $v^\star < v_t$ or $v^\star = v_t$
  - the environment sends ***feedbacks*** of the form $avt.adjust(f_{v_t} \in \mathscr{F})$
    - ⋆ **increasing** $v_t$ ($f \uparrow$), **decreasing** $v_t$ ($f \downarrow$) or informing that $v_t$ is **good** ($f \approx$).
- In our case, the environment of an *avt* is its **sensor node**
  - It is the responsibility of this node to send **correct feedbacks** to its *avt*

# Inside AVT

## Next step proposed value

$$v_{t+1} = \begin{cases} v_t + \Delta_{t+1}, & f_{v_t} = f \uparrow \\ v_t - \Delta_{t+1}, & f_{v_t} = f \downarrow \\ v_t, & f_{v_t} = f \approx \end{cases}$$

where $\Delta_{t+1} \in [\Delta_{min}, \Delta_{max}] \subset (0, |v_{max} - v_{min}|]$ is the ***adjustment step***.

# Inside AVT

## Next step proposed value

$$v_{t+1} = \begin{cases} v_t + \Delta_{t+1}, & f_{v_t} = f \uparrow \\ v_t - \Delta_{t+1}, & f_{v_t} = f \downarrow \\ v_t, & f_{v_t} = f \approx \end{cases}$$

where $\Delta_{t+1} \in [\Delta_{min}, \Delta_{max}] \subset (0, |v_{max} - v_{min}|]$ is the ***adjustment step***.

1. Successive feedbacks of ***same direction***: $v_t$ is *far away* from $v^\star$, hence $\Delta_{t+1} = \Delta_t \cdot \lambda_{incr}$.

# Inside AVT

## Next step proposed value

$$v_{t+1} = \begin{cases} v_t + \Delta_{t+1}, & f_{v_t} = f \uparrow \\ v_t - \Delta_{t+1}, & f_{v_t} = f \downarrow \\ v_t, & f_{v_t} = f \approx \end{cases}$$

where $\Delta_{t+1} \in [\Delta_{min}, \Delta_{max}] \subset (0, |v_{max} - v_{min}|]$ is the ***adjustment step***.

1. Successive feedbacks of ***same direction***: $v_t$ is *far away* from $v^\star$, hence $\Delta_{t+1} = \Delta_t \cdot \lambda_{incr}$.
2. Successive feedbacks of ***opposite directions***: $v_t$ is *oscillating* around $v^\star$, hence $\Delta_{t+1} = \Delta_t \cdot \lambda_{decr}$

# Inside AVT

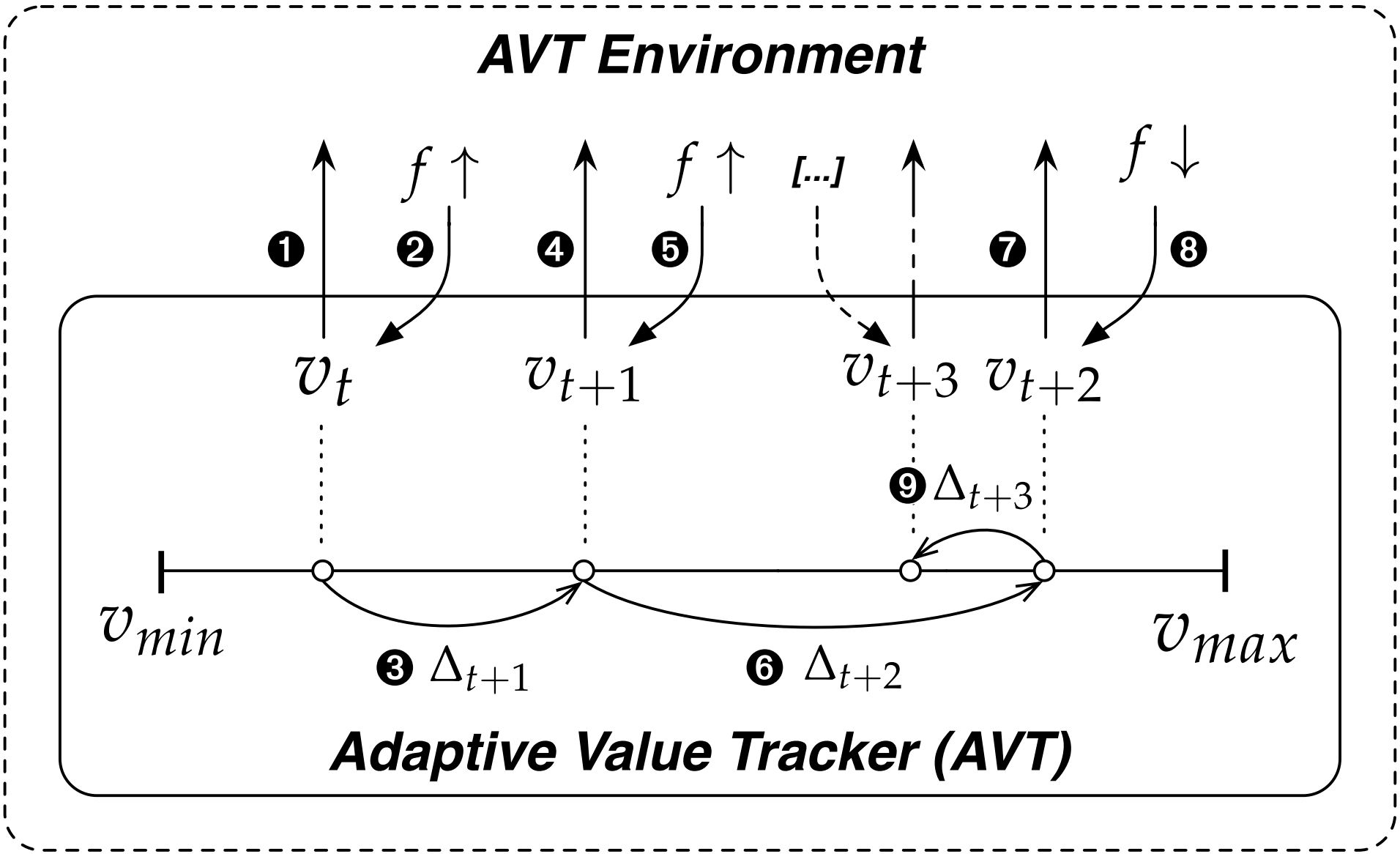
## Next step proposed value

$$v_{t+1} = \begin{cases} v_t + \Delta_{t+1}, & f_{v_t} = f \uparrow \\ v_t - \Delta_{t+1}, & f_{v_t} = f \downarrow \\ v_t, & f_{v_t} = f \approx \end{cases}$$

where $\Delta_{t+1} \in [\Delta_{min}, \Delta_{max}] \subset (0, |v_{max} - v_{min}|]$ is the ***adjustment step***.

1. Successive feedbacks of ***same direction***: $v_t$ is *far away* from $v^\star$, hence $\Delta_{t+1} = \Delta_t \cdot \lambda_{incr}$.
2. Successive feedbacks of ***opposite directions***: $v_t$ is *oscillating* around $v^\star$, hence $\Delta_{t+1} = \Delta_t \cdot \lambda_{decr}$
3. When $f \approx$: $v_t$ has reached an (at least, briefly) ***correct value***, hence $\Delta_{t+1} = \Delta_t \cdot \lambda_{decr}$

# AVT and Environment - Interaction



# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- Upon receiving $< L_v, seq_v >$ such that $seq_u < seq_v$

# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // *calculate clock skew*

# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***

# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***

# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

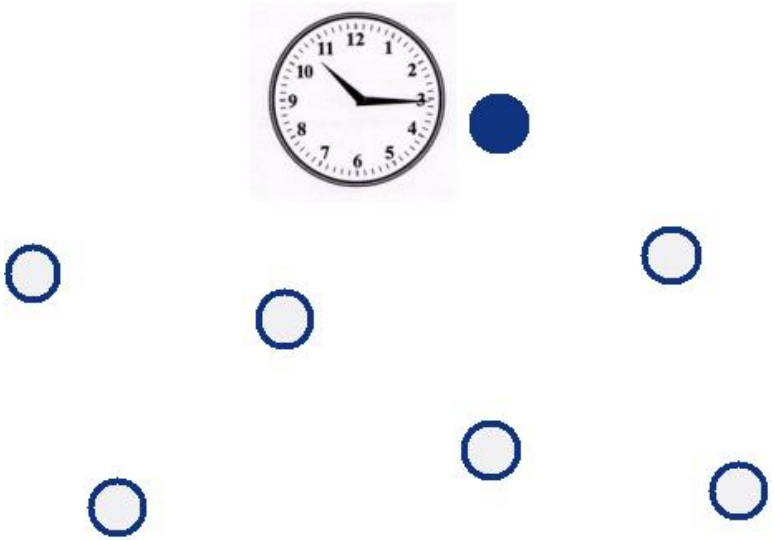
- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***
  **else** $avt_u.adjust(f \approx)$ **endif** // ***no change***

# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***
  **else** $avt_u.adjust(f \approx)$ **endif** // ***no change***
  $L_u \leftarrow L_v$ // ***update offset***

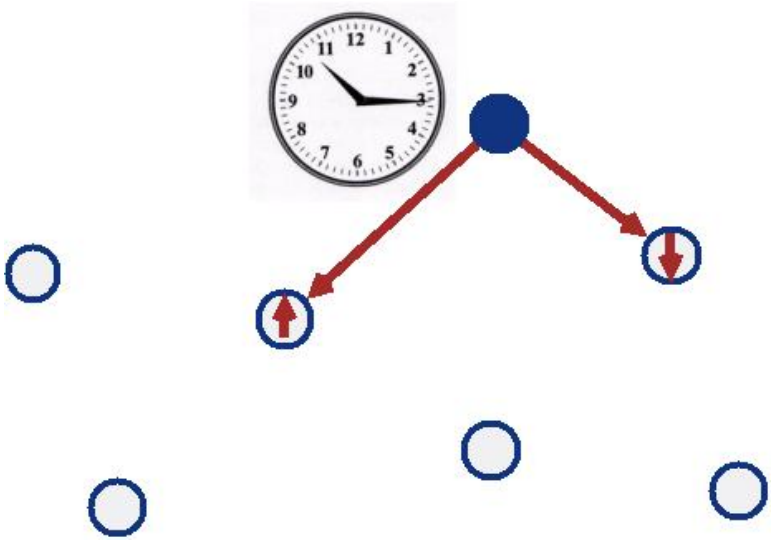
# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***
  **else** $avt_u.adjust(f \approx)$ **endif** // ***no change***
  $L_u \leftarrow L_v$ // ***update offset***



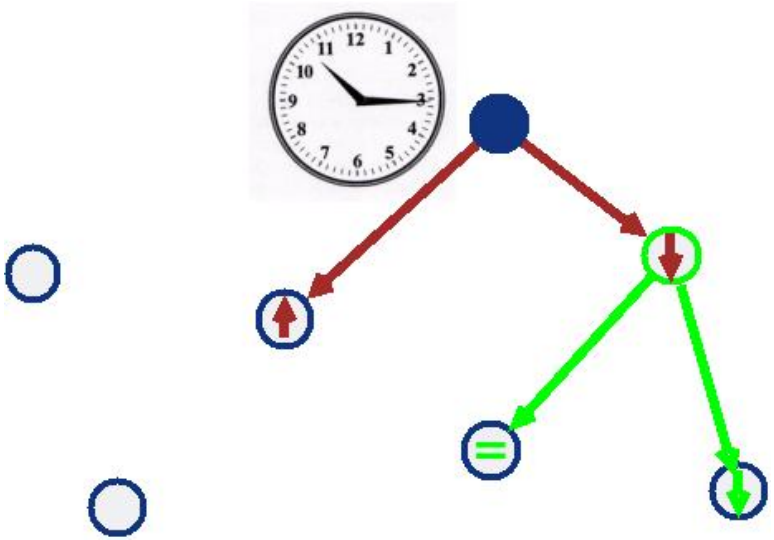
# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***
  **else** $avt_u.adjust(f \approx)$ **endif** // ***no change***
  $L_u \leftarrow L_v$ // ***update offset***



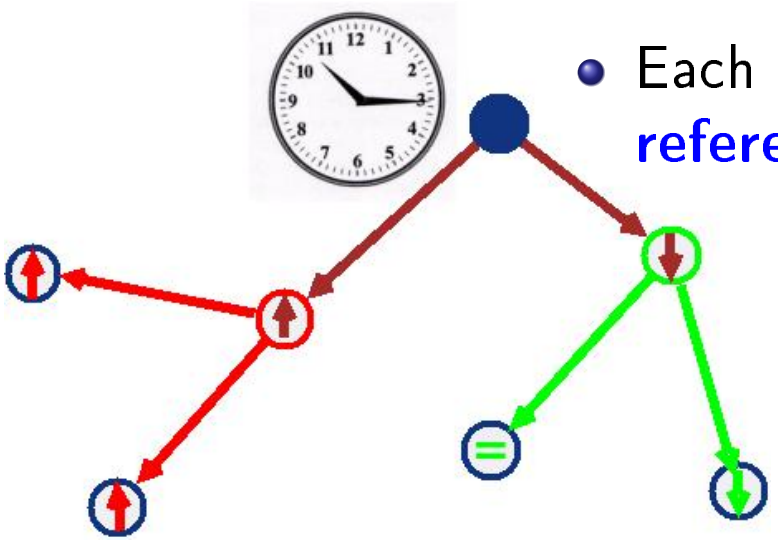
# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***
  **else** $avt_u.adjust(f \approx)$ **endif** // ***no change***
  $L_u \leftarrow L_v$ // ***update offset***



# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

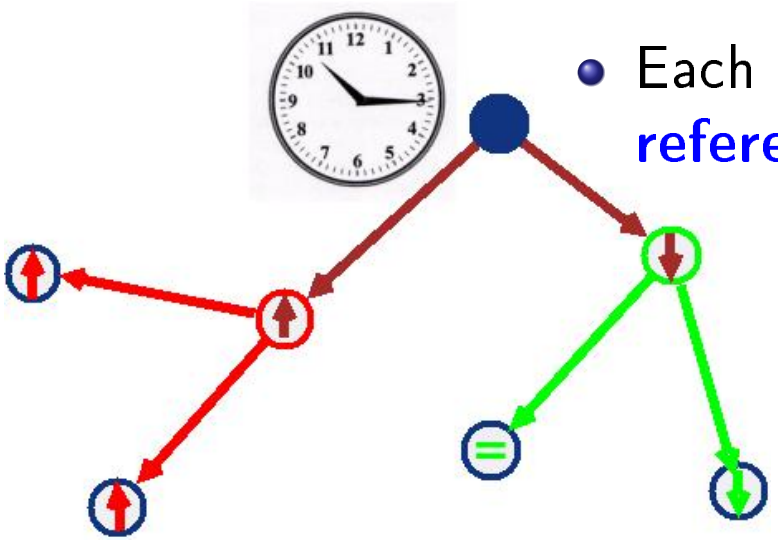
- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***
  **else** $avt_u.adjust(f \approx)$ **endif** // ***no change***
  $L_u \leftarrow L_v$ // ***update offset***



- Each $avt_u$ **finds** the value of the **reference speed** with respect to $H_u$

# Time Synchronization with Adaptive Value Tracking (AVTS) [Yıldırım and Gürcan]

- **Upon receiving** $< L_v, seq_v >$ **such that** $seq_u < seq_v$
  $error \leftarrow L_u - L_v$ // ***calculate clock skew***
  **if** $error > 0$ **then** $avt_u.adjust(f \downarrow)$ // ***slow down***
  **else if** $error < 0$ **then** $avt_u.adjust(f \uparrow)$ // ***speed up***
  **else** $avt_u.adjust(f \approx)$ **endif** // ***no change***
  $L_u \leftarrow L_v$ // ***update offset***



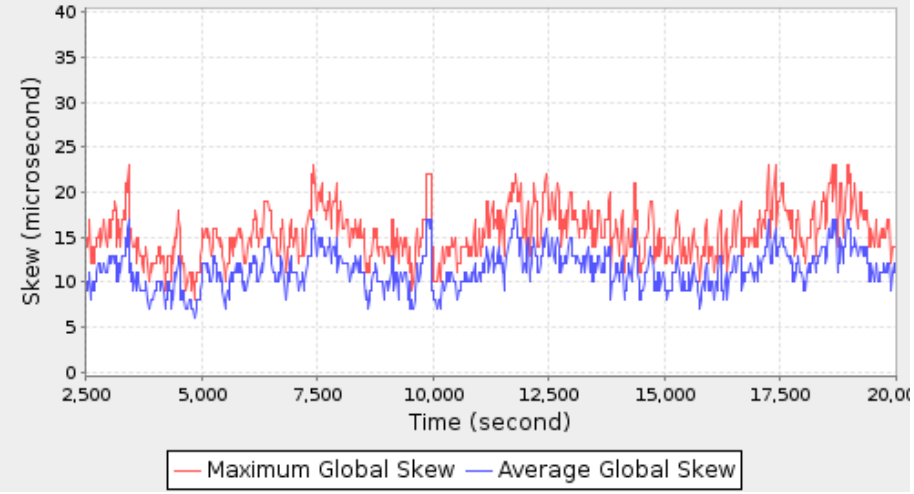
- Each $avt_u$ **finds** the value of the **reference speed** with respect to $H_u$

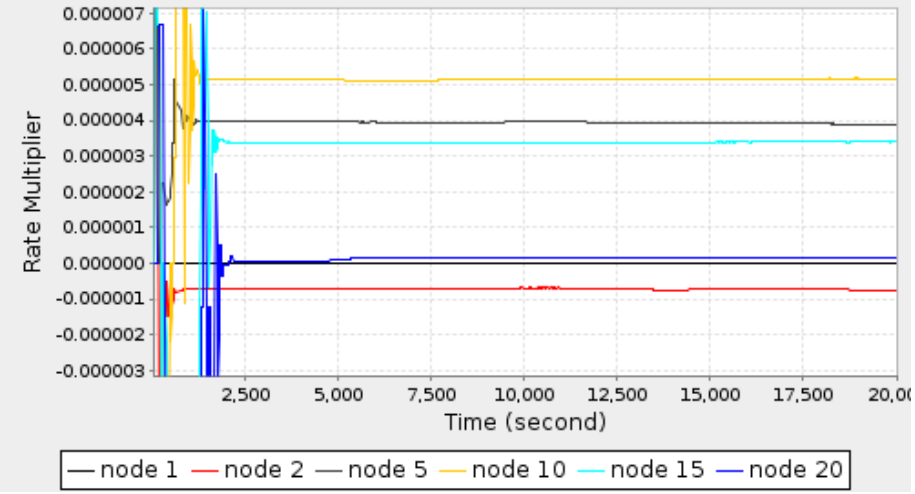
# Experiments with AVTS

- $v^{\star} \in [-10^{-4}, 10^{-4}]$ - $\pm$MICAz 100 ppm, i.e. $10^{-4}$ seconds.
- $\Delta_t \in [10^{-10}, 10^{-5}]$ - precision

# Experiments with AVTS

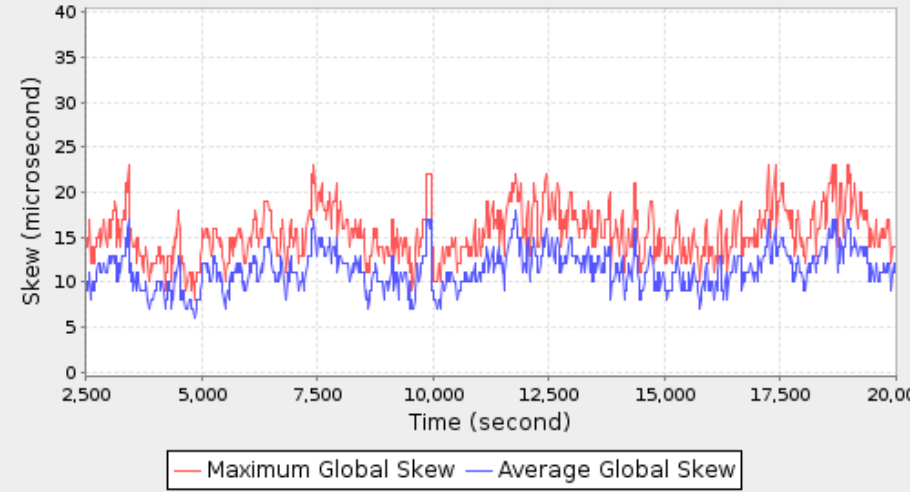
- $v^{\star} \in [-10^{-4}, 10^{-4}]$ - ±MICAz 100 ppm, i.e. $10^{-4}$ seconds.
- $\Delta_t \in [10^{-10}, 10^{-5}]$ - precision

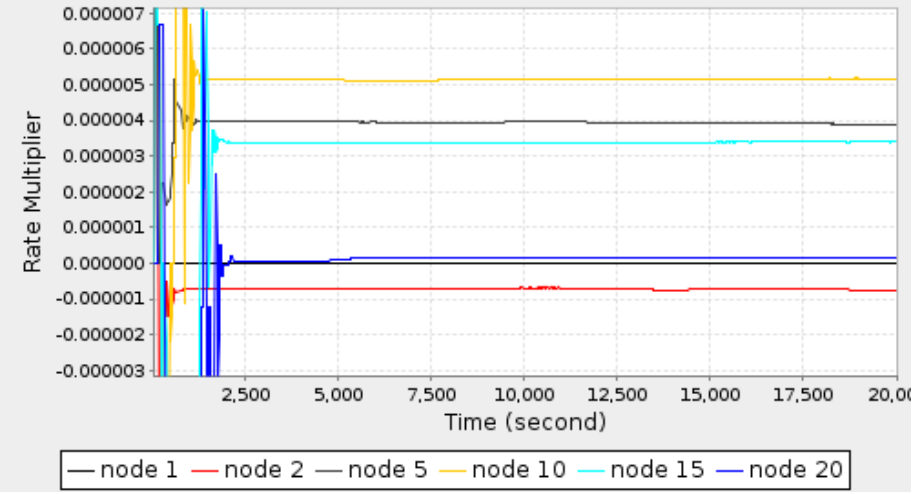


# Experiments with AVTS

- $v^\star \in [-10^{-4}, 10^{-4}]$ - ±MICAz 100 ppm, i.e. $10^{-4}$ seconds.
- $\Delta_t \in [10^{-10}, 10^{-5}]$ - precision




## Memory and Energy Requirements

| | FTSP | PulseSync | FCSA | AVTS |
|---|---|---|---|---|
| **CPU** | $\approx$ 5440 $\mu s$ | $\approx$ 5440 $\mu s$ | 5620 $\mu s$ for $\lvert\mathcal{N}\rvert = 1$ | $\approx$ **175** $\mu s$ |
| **Message Length** | 9 *bytes* | 9 *bytes* | 15 *bytes* | 9 *bytes* |
| **RAM** | 40 *bytes* | 40 *bytes* | 64*$\lvert\mathcal{N}\rvert$ *bytes* | 9 *bytes* |
| **ROM** | 18000 *bytes* | 17856 *bytes* | 20660 *bytes* | **15696** *bytes* |

# Outline

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**
- **considerably smaller code size**

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**
- **considerably smaller code size**
- **No memory storage** to collect time information

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**
- **considerably smaller code size**
- **No memory storage** to collect time information
  - regression table - **no**

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**
- **considerably smaller code size**
- **No memory storage** to collect time information
  - regression table - **no**
  - keeping track of the neighboring nodes - **no**
    - ★ decision of which neighbors to keep track and which ones to discard - **no**

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**
- **considerably smaller code size**
- **No memory storage** to collect time information
  - regression table - **no**
  - keeping track of the neighboring nodes - **no**
    - ★ decision of which neighbors to keep track and which ones to discard - **no**
  - **80% less memory allocation!**

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**
- **considerably smaller code size**
- **No memory storage** to collect time information
  - regression table - **no**
  - keeping track of the neighboring nodes - **no**
    - ★ decision of which neighbors to keep track and which ones to discard - **no**
  - **80% less memory allocation!**
- propagates time information **slowly** - it can also employ rapid flooding
  - eliminating the drawbacks of rapid flooding.

# Conclusions

- AVTS requires **quite a few arithmetic operations**
  - preferable to least-squares - **97% less CPU overhead!**
- **considerably smaller code size**
- **No memory storage** to collect time information
  - regression table - **no**
  - keeping track of the neighboring nodes - **no**
    - ★ decision of which neighbors to keep track and which ones to discard - **no**
  - **80% less memory allocation!**
- propagates time information **slowly** - it can also employ rapid flooding
  - eliminating the drawbacks of rapid flooding.
- quick synchronization compared to agreement

# References


Ferrari, F., Zimmerling, M., Thiele, L., and Saukh, O. (2011).
Efficient network flooding and time synchronization with glossy.
In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 73–84. IEEE.

Lemouzy, S., Camps, V., and Glize, P. (2011).
Principles and properties of a mas learning algorithm: A comparison with standard learning algorithms applied to implicit feedback assessment.
In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '11, pages 228–235, Washington, DC, USA. IEEE Computer Society.

Lenzen, C., Sommer, P., and Wattenhofer, R. (2009).
Optimal Clock Synchronization in Networks.
In *7th ACM Conference on Embedded Networked Sensor Systems (SenSys), Berkeley, California, USA*.

Maróti, M., Kusy, B., Simon, G., and Lédeczi, A. (2004).
The flooding time synchronization protocol.
In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA. ACM.

Wang, Y., He, Y., Mao, X., Liu, Y., Huang, Z., and Li, X. (2012).
Exploiting constructive interference for scalable flooding in wireless networks.
In *INFOCOM, 2012 Proceedings IEEE*, pages 2104–2112. IEEE.

Yildirim, K. S. and Kantarci, A. (2013).
Time synchronization based on slow flooding in wireless sensor networks.
*IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1.