Lecture 3:
Clock Synchronization

CS 539 / ECE 526
Distributed Algorithms
Announcements

• Problem Set 1 will be out tomorrow
  – One problem set every 2 weeks
  – 2~3 questions
  – Due in 1.5 weeks

• Office hour change: Monday 2-3 pm
  (and after class)
Outline

• Lockstep rounds too strong assumption

• How to enforce lockstep rounds?
  – Today: In synchrony: clock synchronization
  – Next time: In asynchrony: synchronizers
Outline

• Model of clock synchronization

• No drift

• Lower bound

• From clock sync to lockstep rounds

• With drift
Hardware Clocks

• Each process equipped with a hardware clock

• We wish they were perfectly synchronized
  – As if a shared global clock

• Unfortunately, unrealistic assumption …
Hardware Clocks

• Skew: clock value differences at a given time
  – $HC_i(t) = t + b_i$
  – Then, skew is $|b_i - b_j|$

• Drift: clock speed differences
  – $HC_i(t) = a_i \times t + b_i$
  – Then, drift is $a_i / a_j$
Adjusted Clocks

• Each process equipped with a hardware clock
  – … whose reading may be far apart

• Adjusted clock: \( \text{AC}_i(t) = \text{HC}_i(t) + \text{adj}_i(t) \)
  – May omit \((t)\) when clear

• Clock synchronization: how to set \(\text{adj}_i(t)\)
such that skew is reduced to a small value
Clock Synchronization

• Complete graph (can be relaxed)

• Bounded message delay within \([d, D]\)
  – More general than usual where \(d = 0\)

• Bounded drift
  – We will start with zero drift

• No failure
Crucial Remark

• Synchrony = bounded delay + bounded drift
  – First lecture oversimplified

• If drift is unbounded, even bounded delay can ”appear” unbounded

• Clock synchronization only possible under synchrony (will prove this today)
Outline

• Model of clock synchronization

• No drift

• Lower bound

• From clock sync to lockstep rounds

• With drift
Zero Drift, Two Processes

• With 0 drift, synchronize once, good forever
• Simplest case: just two processes
• Proc 1 simply uses its hardware clock
  – $AC_1(t) = HC_1(t)$ ( $adj_1(t) = 0$ )
• Proc 1 sends a clock reading to Proc 2
• How should Proc 2 adjust its clock?

Proc 1

Proc 2
Zero Drift, Two Processes

• Proc 1 sets $AC_1(t) = HC_1(t)$
• Proc 1 sends a clock reading 4:17
• Suppose msg delay ranges from $d=1$ to $D=5$
• Proc estimate current $HC_1$ to be $4:17 + 3$
  – Assume the msg took median delay (minimize error)
• Proc 2 sets $AC_2$ to 4:20 (to try to match $HC_1$)
  – Suppose Proc 2 received the msg at local clock 5:42
  – Then, it sets $adj_2 = -1:22$
Zero Drift, Two Processes

• Proc 1 sets $AC_1(t) = HC_1(t)$
• Proc 1 sends $R = HC_1(t_1)$ at time $t_1$
• Proc 2 receives $R$ at local clock $HC_2(t_2)$
  – Estimate $HC_1(t_2) \approx R + (d+D)/2$
• Proc 2 sets $AC_2(t_2)$ to estimated $HC_1(t_2)$
  – $adj_2 = AC_2(t_2) - HC_2(t_2) = R + (d+D)/2 - HC_2(t_2)$
Zero Drift, Two Processes

• Skew Achieved?
• If msg delay is indeed median, perfect
• If msg delay is d or D, max skew
  – $D - \frac{d+D}{2} = \frac{d+D}{2} - d = \frac{D-d}{2}$
  – I.e., half of uncertainty  (Uncertainty $U = D-d$)
  – May be “obvious” but need a proper proof

\[
\begin{align*}
\text{Proc 1} & \quad \text{Proc 2} \\
\hline
R = HC_1(t_1) & 
\end{align*}
\]
Zero Drift, Two Processes

- $\text{AC}_1(t) = \text{HC}_1(t)$
- $\text{AC}_2(t) = \text{HC}_2(t) + \text{HC}(t_1) + (d+D)/2 - \text{HC}_2(t_2)$
- Let $\delta$ be the actual msg delay
- $\text{HC}_1(t_2) = \text{HC}_1(t_1) + \delta$
- Skew = $\text{HC}_2(t) - \text{HC}_1(t) + \text{HC}_1(t_1) - \text{HC}_2(t_2) + (d+D)/2$
  
  $= \text{HC}_2(t) - \text{HC}_1(t) + \text{HC}_1(t_2) - \text{HC}_2(t_2) + (d+D)/2 - \delta$
  
  $= (d+D)/2 - \delta$  (no drift)
  
  $\leq (D-d)/2$  (max error in delay estimation)

Proc 1

Proc 2

$\text{R} = \text{HC}_1(t_1)$
Zero Drift, Two Processes

• Skew achieved?
• If msg delay is indeed median, perfect
• If msg delay is d or D, max skew U/2

• Can we do better than U/2?
• No! Impossible to clock sync to less than U/2
Lower Bound for Two Processes

• Impossible to clock sync to less than U/2
  – Proof: consider an algo that syncs within E
  – Suppose all 1→2 msgs incur delay d, all 2→1 msgs D

\[ AC_1 - E \leq AC_2 \leq AC_1 + E \]

---

**Diagram:**

- **Proc 1:**
  - Time: 0, 1, 2, 3, 4
  - Messages exchanged:
    - 0→1
    - 1→2
    - 2→3
    - 3→4

- **Proc 2:**
  - Time: 0, 1, 2, 3, 4
  - Messages exchanged:
    - 0→1
    - 1→2
    - 2→3
    - 3→4
    - ...
Lower Bound for Two Processes

• Impossible to clock sync to less than U/2
  – Proof: consider an algo that syncs within E
  – Suppose all 1→2 msgs incur delay d, all 2→1 msgs D
  – “Spring forward” Proc 1 hardware clock by U = D − d
Lower Bound for Two Processes

- Impossible to clock sync to less than $U/2$
  - Proof: consider an algo that syncs within $E$
  - Suppose all $1 \rightarrow 2$ msgs incur delay $d$, all $2 \rightarrow 1$ msgs $D$
  - “Spring forward” Proc 1 hardware clock by $U = D - d$
Lower Bound for Two Processes

• Impossible to clock sync to less than U/2
  – Proof: consider an algo that syncs within E
  – Suppose all 1→2 msgs incur delay d, all 2→1 msgs D
  – “Spring forward” Proc 1 hardware clock by U = D - d
  – 1→2 msgs incur delay D, 2→1 msgs incur d
Lower Bound for Two Processes

- Indistinguishable to both processes
  - Hence, apply same adj in the two situations
  - $AC_2' = AC_2$ $AC_1' = AC_1 + U$
- Both are legal executions (respect msg delay bounds)
  - $AC_2 \leq AC_1 + E$ $AC_1' \leq AC_2' + E$

![Diagram of two processes](image)
Lower Bound for Two Processes

- $AC_2' = AC_2$  $AC_1' = AC_1 + U$
- $AC_2 \leq AC_1 + E$  $AC_1' \leq AC_2' + E$
  - $AC_1 + U \leq AC_2 + E$
    - $\leq (AC_1 + E) + E$
  - $E \geq U/2$

![Diagram showing the lower bound for two processes]

Proc 1'

Proc 2'

Proc 1

Proc 2
Zero Drift, Many Processes

- With 0 drift, synchronize once, good forever
- Two processes: sync within $U/2$, best possible
- Many processes: want $|AC_i - AC_j| \leq E$ for all $i, j$
  - Simple algo exists for sync within $U$

- Let one proc be reference, and every process runs 2-proc algo with reference
  - Max skew $\leq U/2 + U/2$ (triangle inequality)

- Can we do better?
Lower Bound for \( n \) Processes

- Impossible to clock sync to less than \( U(1-1/n) \)
  - Proof: consider an algo that syncs within \( E \)
  - Suppose all “downward” msgs incur delay \( d \), and all “upward” msgs incur delay \( D \)
Lower Bound for $n$ Processes

- Lemma: $AC_i \leq AC_{i+1} - U + E$
  - “Spring forward” processes 1 through $i$
  - Switch downward and upward delays
Lower Bound for $n$ Processes

- **Lemma:** $AC_i \leq AC_{i+1} - U + E$
  - Indistinguishable: $AC_{i+1}' = AC_{i+1}$, $AC_i' = AC_i + U$
  - Clock sync algo: $AC_i' \leq AC_{i+1}' + E$

\[\begin{array}{cccccc}
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
\end{array}\]
Lower Bound for $n$ Processes

- Lemma: $AC_i \leq AC_{i+1} - U + E$
- $AC_n - E \leq AC_1$
- $AC_1 \leq AC_2 - U + E$
  $\leq AC_3 - 2U + 2E$
  ...
  $\leq AC_n - (n-1)U + (n-1)E$
- $(n-1)U \leq nE \Rightarrow E \geq U(1-1/n)$
Lower Bound for Clock Sync

• Impossible to clock sync to less than $U(1 - 1/n)$
  – Might as well use the simple algo to sync to $U$
  – Does not tolerate reference failure (topic for later)

• Impossible to clock sync under asynchrony
  – Essentially, $U$ is infinite
Outline

• Model of clock synchronization

• No drift

• Lower bound

• From clock sync to lockstep rounds

• With drift
Enforce Lockstep Rounds

- Simple algo to sync within U
- Make each round $U + D$
  - "Dragging" processes’ msgs still considered in time
  - “Rushing” processes’ msgs need to be buffered
  - Make it 2D if $d = 0$
Outline

• Model of clock synchronization
• No drift
• Lower bound
• From clock sync to lockstep rounds
• With drift
Clock Sync with Drift

- Drift must be bounded, otherwise == async
  \[
  \frac{HC_i(t_2) - HC_i(t_1)}{HC_j(t_2) - HC_j(t_1)} \leq 1 + r
  \]

- Idea: sync periodically, every T
  - Immediately after one sync, skew is at most U
  - After T, drift by at most rT
  - Skew at the end of a period is at most U + rT
Lockstep with Drift

• Make each round $U + rT + D$ and sync every $T$
• One subtlety: time skipping
Lockstep with Drift

• Make each round $U + rT + D$ and sync every $T$

• One subtlety: time skipping
  – Proc 2 changes from dragging to rushing
  – Proc 2 “misses” the beginning of yellow round
Lockstep with Drift

• Make each round $U + rT + D$ and sync every $T$

• One subtlety: time/round skipping

• Solution: add buffer time at the end of each period during which rounds do not advance
Summary

• Algorithm to sync clocks within U
  – U/2 for two processes, best possible
  – Almost optimal due to U(1-1/n) lower bound
  – Periodic sync to handle skew

• Can now enforce the **lockstep** abstraction using longer rounds