ECE428 MT1 cheat-sheet

Latency: delay (time taken) for a bit transmission

Bandwidth: Total amount of info. transmitted over the channel per unit time

clock skew / Clock drift rate)

Asynchronous system:

No bound on process execution speeds, message passing delay, clock drift rate.

Periodic ping and ack (2 msg per T) time out $\Delta_1 = 2$(max delay)

Periodic heartbeats (1 msg per T) $T + \Delta_2$, Δ_2=(Max - min) delay

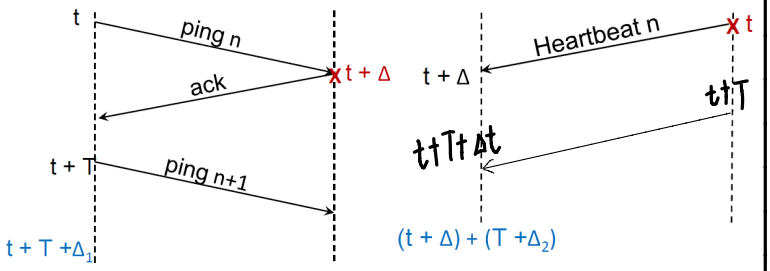
Failure Detection: Completeness (eventually) ① Centralized ② Ring hb ③ all-to-all

Accuracy (failure → the crashed process)

Worst case failure detection time: T_{fd}

Ping-ack: $T + \Delta_1 - \Delta$ Heartbeats: $T + \Delta_2 + \Delta$

T: Time period Δ: network delay / Time between two process



T↓, T_{fd}↓, B↑; Δ_1, Δ_2↑, accuracy↑, T_{fd}↑

Communication Omission: process / channel crash → failed msg transmission

Arbitrary (Byzantine) failure: executing wrongly → wrong msg

Time failure: Timing guarantee are not meet fail-stop

Sync systems have bound on max drift rate.

Clocks in a system externally synchronized within bound D. ⇒ internally sync within a bound of 2D.

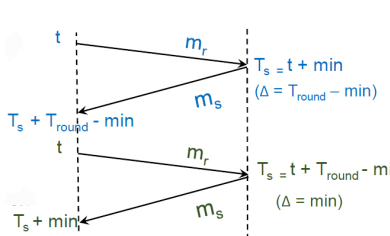
Sync:

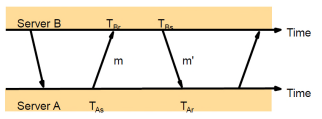
$T_c = T_s + (\min + \max)\text{delay}/2$ ⇒ skew(client, sever)

Cristian Algorithm: RTT $\leq$ (max - min)_delay/2

ASync: $T_c = T_s + (T_{round}/2)$ ⇒ skew $\leq (T_{round}/2) - \min\_delay$

Estimates Δ as RTT/2





- t and t': actual transmission times for m and m'(unknown)
- o: true offset of clock at B relative to clock at A (unknown)
- o_i: estimate of actual offset between the two clocks
- d_i: estimate of accuracy of o_i ; total transmission times for m and m'. $d_i = t + t'$

$T_{Br} = T_{As} + t + o$

$T_{Ar} = T_{Bs} + t' - o$

$o = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}) + (t' - t))/2$

$o_i = ((T_{Br} - T_{As}) - (T_{Ar} - T_{Bs}))/2$

$o = o_i + (t' - t)/2$

$d_i = t + t' = (T_{Br} - T_{As}) + (T_{Ar} - T_{Bs})$

$(o_i - d_i/2) \leq o \leq (o_i + d_i/2)$ given $t, t' \geq 0$

Cannot handle faulty t servers

Berkeley Algorithm: Average all local t and send back offset (Support internal sync.)

Network Time Protocol (NTP)

Hierarchical structure for scalability

Multiple lower strata servers for robustness (LAN)

Authentication mechanisms for security

Statical techniques for better accuracy (Procedure-call & symmetric mode)

Handed-before: (HB) $e \to e'$

$e \to_i e'$: e happened before e', as observed by p_i

$a \to e$ and $e \to a \Rightarrow a \| e$, a and e are concurrent

Lamport's Logical clock HB ✓

$e \to e' \Rightarrow L(e) < L(e')$ $L(e) < L(e') \Rightarrow e \to e'$ or $e \| e'$

Vector Timestamps

$e \to e'$ iff $V(e) < V(e')$ $e \| e'$ iff $(V \not< V'$ and $V' \not< V)$

Global State (Global Snapshot)

For a process p_i, history $(p_i) = h_i = \langle e_i^0, e_i^1 \ldots \rangle$ prefix his p_i^k

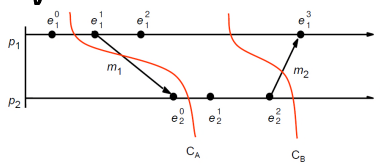
s_i^k: p_i's state immediately after kth event. $h_i^k = \langle e_i^0, e_i^1 \ldots e_i^k \rangle$

Global history $H = \cup_i (h_i)$ Global state $S = \cup_i (s_i)$

a cut $C \subseteq H = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_n^{c_n}$

the frontier of $C = \{e_i^{c_i}, i = 1, 2, \ldots, n\}$

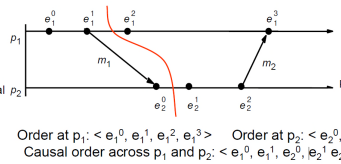
global state S correspondings to cut $C = \cup_i (s_i^{c_i})$



C_A: $< e_1^0, e_2^0 >$
Frontier of C_A: $\{e_1^0, e_2^0\}$

C_B: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1 e_2^2 >$
Frontier of C_B: $\{e_1^2, e_2^2\}$

Cut Inconsistent cut. Consistent cut.



Order at p_1: $< e_1^0, e_1^1, e_1^2, e_1^3 >$ Order at p_2: $< e_2^0, e_2^1, e_2^2 >$

Causal order across p_1 and p_2: $< e_1^0, e_1^1, e_2^0, e_2^1 e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1 e_2^2, e_1^3 >$

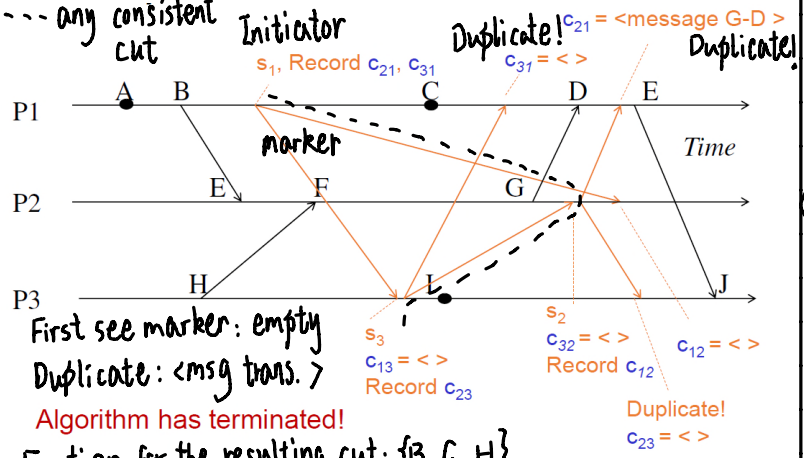
Linearization $< e_1^0, e_1^1, e_2^0 | e_2^1, e_1^2, e_2^2, e_1^3 >$

Consistent iff $\forall e \in C$, if $f \to e$, then $f \in C$

Global state S consistent iff corresponding to Cons. cut.

Chandy-Lamport Algorithm: pair communication FIFO

--- any consistent cut



First see marker: empty
Duplicate: <msg trans. >
Algorithm has terminated!
Frontier for the resulting cut: {B, G, H}

Execution Lattice: Each path represents a linearization

Liveness: iff ① liveness ($P(S_0)$) ≡ ∀L ∈ linearizations from S_0. L passes through a S_L & $P(S_L)$ = True
② For all linearizations starting from S_0, P is True for some state S_L reachable from S_0. i.e. terminate; fail detect complete

Safety: iff ① safety ($P(S_0)$) ≡ ∀s reachable from S_0, P(S) = true ② For all states S reachable from S_0, P(S) is True

---

Unicast: one to one; Broadcast: one to all; Multicast: any one to a group

Basic (B-Multicast): each in unicast.

Reliable (R-Multicast):
Validity and Agreement ensure liveness: All correct P (itself m) lothers agree on m) deliver msg m.

FIFO ordering: focus on the delivering order of msg sent from one process

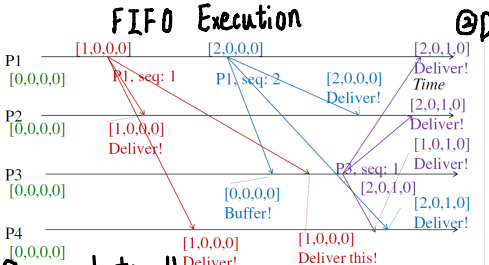
Causal ordering: Lamport's HB rule

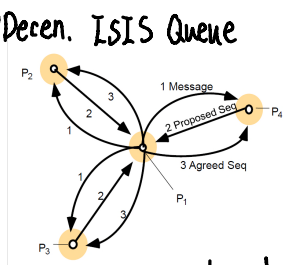
Total ordering (Atomic Broadcast):
Ensure all the processes deliver multicast in the same order

Implement FIFO ordering: sequencer (Vector) (pre-sender) Buffer until FIFO

Implement causal ordering: buffering until causality

- FIFO ordering
  - If a correct process issues multicast(*g*,*m*) and then multicast(*g*,*m'*), then every correct process that delivers *m'* will have already delivered m. M1:1, M1:2...
- Causal ordering
  - If multicast(*g*,*m*) → multicast(*g*,*m'*) then any correct process that delivers *m'* will have already delivered *m*.
  - Note that → counts multicast messages delivered to the application, rather than all network messages. M1:1, M2:1...
- Total ordering
  - If a correct process delivers message *m* before *m'* (independent of sending order), then any other correct process that delivers *m'* will have already delivered *m*. M1:1, M2:1, M3:1, M3:2





① Unicast to all
② Tree-based Multicast
Minimum Spanning Tree
③ Gossip: transmit to random b targets (epidemic multicast)

① Central server based algorithm

Breaking deadlocks (MA in mutual)

- *System of 6 processes {0,1,2,3,4,5}. 0,1,2 want to enter critical section:*
  - V_0= {0, 1, 2}: 0, 2 send reply to 0, but 1 sends reply to 1;
  - V_1= {1, 3, 5}: 1, 3 send reply to 1, but 5 sends reply to 2;
  - V_2= {2, 4, 5}: 4, 5 send reply to 2, but 2 sends reply to 0;
- Suppose (L1, P1) < (L0, P0) < (L2, P2).
- P2 will send fail to itself when it receives its own request after P0.
- P5 will send inquire to P2 when it receives P1's request.
- P2 will send relinquish to V_2. P5 and P4 will set "voted = false". P5 will reply to P1.
- P1 can now enter CS, followed by P0, and then P2.

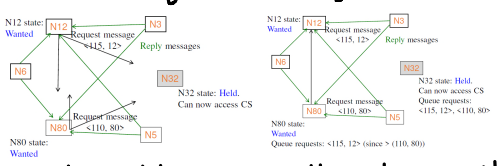
Mutual Exclusion

enter( ), AccessResource( ) (CS), exit( )
i.e. wait(S) critical section Problem i.e. signal(S)

1. Central Server Algorithm (bottleneck on the leader)
Leader keeps a queue of waiting requests from processes that require to access CS, a token that allows holder to access CS. safety ✓ and liveness ✓, request + guarantee
Client delay is focused on each enter. (exit too but ignored)
Synchronization delay is the T between the pre-exit and the new enter. (throughput: measure of the system) release + guarantee
Bandwidth: 2 msg enter + 1 msg exist

2. Ring-based Algorithm
Bandwidth: 1 msg / N msg per enter for the req. P / the system
1 msg exist
Client delay: 0 to N (just receive/send out token) O(N)
Synchronization delay: 1 to (N-1) O(N)
process to enter is: successor predecessor of the Pexit

3. Ricart-Agrawala's Algorithm



Held until receive all replies; reply by FIFO priority no token. Worst-case: smaller T in timestamp.
wait (N-1) P to send Reply
Bandwidth: 2*(N-1) per enter, (N-1) per exist
Client delay: one round-trip time (RRT) O(1)
Synchronization delay: one msg transmission time. O(1)

4. Maekawa's Algorithm: voting sets by only some G members
$\sqrt{N}$ by $\sqrt{N}$ matrix works best. Bandwidth: $2\sqrt{N}-1$ $O(\sqrt{N})$
Client delay: 1 RRT; Synchronization delay: 2 msg trans. T
safety ; Livenessx : A deadlock could occur; Orderx