THE OHIO STATE UNIVERSITY

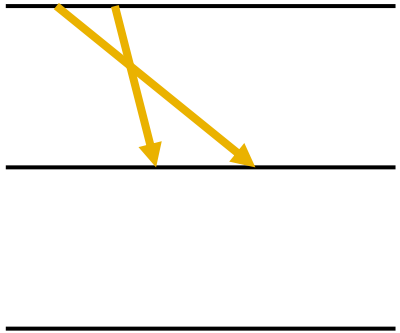
Causal Ordering for Unicast Using Heartbeats

Laine Rumreich

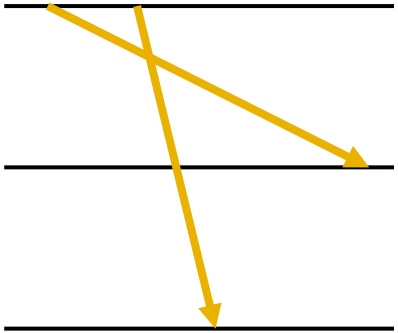
Advisor: Dr. Paul Sivilotti

Definitions

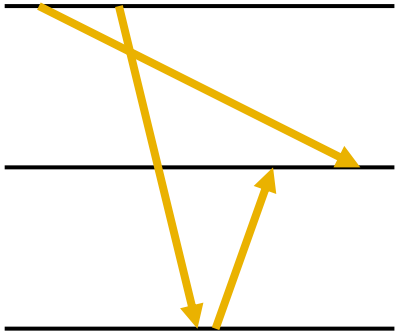
Non-FIFO and Non-Causal Order



FIFO and Causal Order



FIFO but not Causal Order



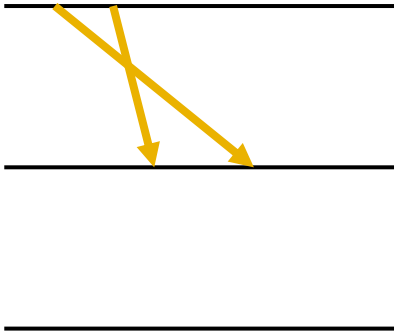
FIFO Ordering

For messages a and b sent from the same process that have the same destination:

If $S_a \rightarrow S_b$,

Then $D_a \rightarrow D_b$

Non-FIFO Order

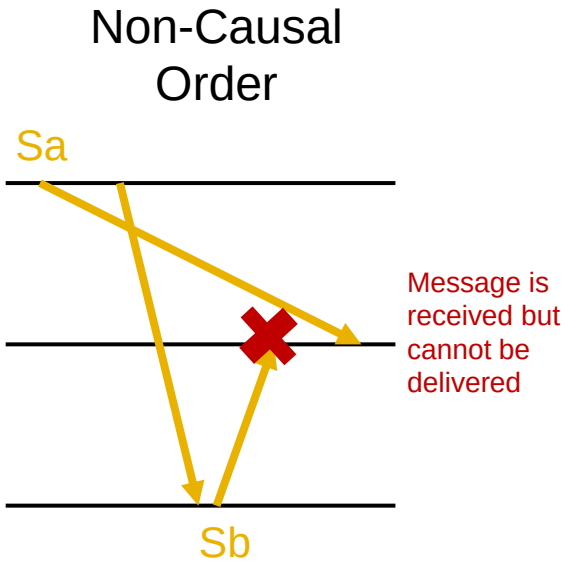


Causal Ordering

For all S_a and S_b that have the same destination:

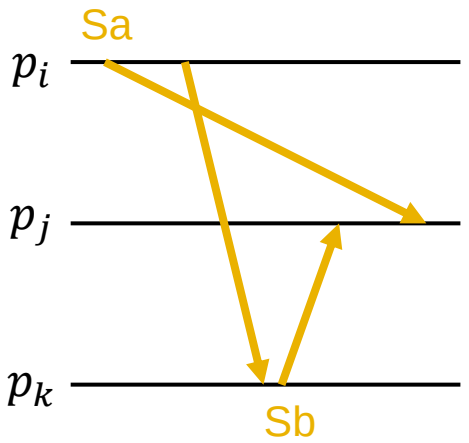
If $S_a \rightarrow S_b$,

Then $D_a \rightarrow D_b$



Problem Statement

- Given a FIFO Channel
- Implement an algorithm to preserve causal ordering for unicast messages
- Assume all processes have a direct connection to every other process



Existing Solution

With each message, send a matrix of values about all messages that have been sent so far. Also keep track of your own delivered messages.

M_i	i	j	k	ℓ
i				
j				
k				
ℓ				

$Delivered_i$

i	
j	
k	
ℓ	

Existing Solution

Each process p_i maintains its own *n* x *n* matrix, M_i, where $M_i[j,k]$ stores the count of the number of messages sent by p_j to p_k as known to p_i

M_i	i	j	k	ℓ
i				
j				
k				
ℓ				

Existing Solution

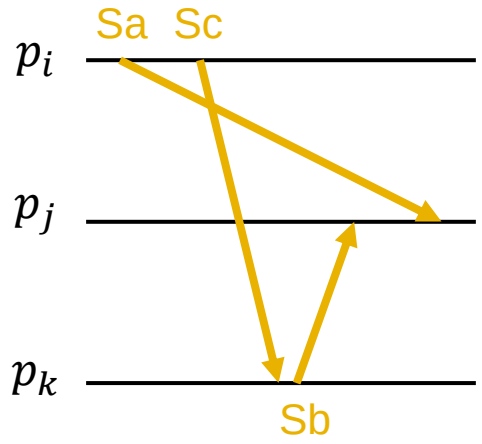
When a message is received by p_j with an attached matrix *M*, it is not delivered until:

$\forall x, M[x, j] \leq Delivered_j[x]$

M_b	i	j	k	ℓ
i		1		
j				
k				
ℓ				

$Delivered_j$

i	0
j	
k	
ℓ	



Existing Solution

After delivery, each value in the local matrix is updated to be the max of the local $\boldsymbol{M_x}$ and the sent $\boldsymbol{M}$

$\boldsymbol{M_x}$	i	j	k	ℓ
i				
j				
k				
ℓ				

Existing Solution

This leads to the potential for attacks if there are byzantine nodes

M_i	i	j	k	ℓ
i				
j				
k				
ℓ				

Proposed Solution

- Broadcast a heartbeat with each message (the message remains unicast)
- Attach a vector to each message with the number of broadcasts received

V_i	i	j	k	ℓ

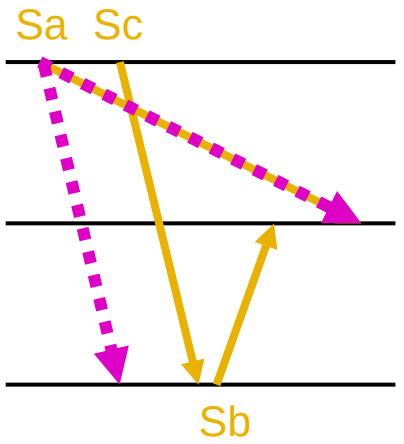
Proposed Solution

- The local vector is only updated upon receipt of a heartbeat directly from the originating process, **not** from other application messages

V_i	i	j	k	ℓ

Proposed Solution

- Send a broadcast heartbeat with each message



Proposed Solution

- Deliver a message when every element of the vector is ≤ that of the local vector
- $E_a \rightarrow E_b$ iff every element of the E_a vector is ≤ that of E_b

V	i	j	k	ℓ

≤

V_i	i	j	k	ℓ

Pseudocode

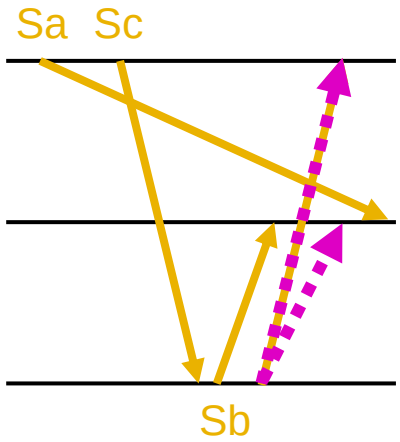
Message Send

```
Send m to p –>
   Broadcast(<1>, U / p)
   Send(<m, heartbeats>, p)
```

Message Receive

```
Received <m, sentHeartbeats> from pi –>
   receivedHBs[pi]++
   receivedList.add(pi, sentHeartbeats, m)
```

Pseudocode



Heartbeat Receive

```
Receive heartbeat from pi –>
    receivedHBs[pi]++

    if(not receivedList.contains(pi))
       heartbeats[Pi]++
```

Message Deliver

```
m : (for all p in U : m.sentHeartbeats[p] <= heartbeats[p]) –>
    Deliver(m)

    heartbeats[p] = MIN(receivedHBs[p],
          (MIN(m' in receivedList from p :: m'.sentHeartbeats[p])))
    receivedList.remove(p, m)
```

Proof

1. Safety
2. Liveness

1. Safety

Assume the following is true

For all $S_i : S_i \rightarrow S_k, D_i \rightarrow D_k$

if the message destinations are the same

To prove: For any next action, the property is still satisfied

1. Message is Sent
2. Message is Delivered

1. Safety

For all $S_i : S_i \rightarrow S_k, D_i \rightarrow D_k$

if the message destinations are the same

1. Message is sent

This action creates an *S* but not a *D* so the invariant remains true

1. Safety

2. Message M_k is delivered (D_k)

For all $S_i : S_i \rightarrow S_k$, $D_i \longrightarrow D_k$

if the message destinations are the same

V_i	i	j	k	ℓ

We want to show that for all such M_i, M_i has been delivered

Case 1: M_i is still in transit

- HB vector for S_i at row i must be ≤ to row i in S_k 's HB vector
- Since the HB is attached to message S_i, it has not been Received
- Since the S_i HB has not been received, the HB vector for the destination process must be less than that of S_i
- HB at destination ≤ $S_i \leq S_k$ so S_k never could have been delivered. Contradiction.

1. Safety

2. Message M_k is delivered (D_k)

For all $S_i : S_i \rightarrow S_k$, $D_i \longrightarrow D_k$

if the message destinations are the same

We want to show that for all such M_i, M_i has been delivered

Case 2: M_i has been received but not yet delivered
Since $S_i \rightarrow S_k$, the vector values of S_i <= S_k for all values. Since the heartbeat vector at the destination was large enough to deliver S_k, it was large enough to deliver S_i and it must have been delivered. Contradiction.

1. Safety

2. Message M_k is delivered (D_k)

For all $S_i : S_i \rightarrow S_k$, $D_i \longrightarrow D_k$

if the message destinations are the same

We want to show that for all such M_i, M_i has been delivered

~~*Case 1: M_i is still in transit*~~

~~*Case 2: M_i has been received but not yet delivered*~~

Case 3: M_i has been delivered

2. Liveness

A message is delayed only if we have not yet received a heartbeat.

Intuition: heartbeats cannot be infinitely delayed because there is always a message coming that is the next one in order

Observations

- Total overhead summed over all messages
  - Message size O(2N*M) improvement upon O(N^2*M)
  - Old: N^2 per application message
  - New: N per application message + (N-1)*1 heartbeats = 2N
- Additional network traffic due to heartbeat broadcasts
  - M application messages + MN heartbeats

Research Questions

- Previous algorithm does not protect against byzantine nodes for unicast
- Consider if there is additional wait time due to heartbeats
- Heartbeat frequency optimizations and tradeoffs