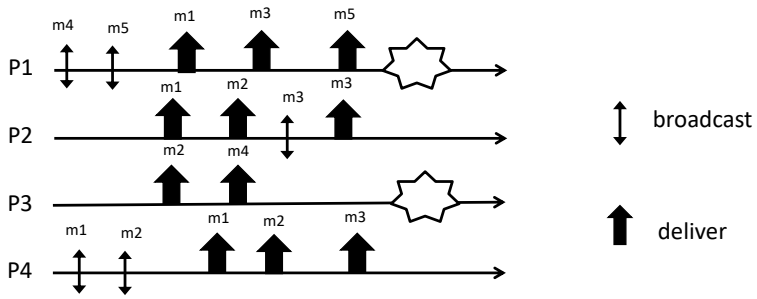
```
upon event new_next (p)
  IF leader > p AND self > p
    leader = self
    trigger leader (leader)
    next = p
    wait = TRUE        → Wait all confirm new leader
    trigger PP2PSend (NEW_LEADER, leader) to next
  else
    next = p
    trigger PP2PSend (NEW_LEADER, leader)

upon event PP2PDeliver (NEW_LEADER, )
    IF leader = (
      IF wait = TRUE
          WAIT = FALSE
          <Stop resending m, i'm new leader>
      else
        <Discard message, it is already recived the messages>
    else
      leader = (
      trigger leader (leader)
      trigger PP2Psend (NEW_LEADER, leader) to next
```

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 23 – Exercises
November 23th, 2022
(Estimated time to complete all exercises: 3 hours)

Ex 1: Consider the execution depicted in the Figure

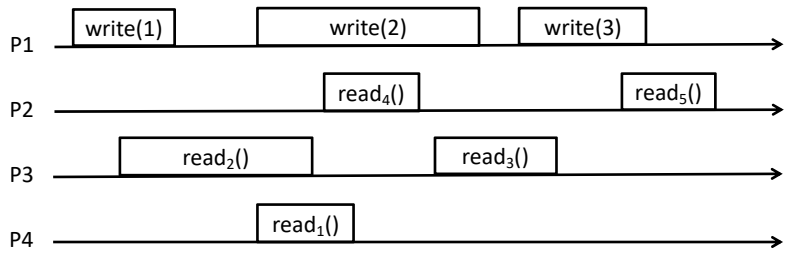


Answer to the following questions:

1. Which is the strongest TO specification satisfied by the proposed run? Motivate your answer.
2. Does the proposed execution satisfy Causal order Broadcast, FIFO Order Broadcast or none of them?
3. Modify the execution in order to satisfy TO(UA, WUTO) but not TO(UA, SUTO).
4. Modify the execution in order to satisfy TO(NUA, WNUTO) but not TO(UA, WNUTO).

NOTE: In order to solve point 3 and point 4 you can only add messages and/or failures.

Ex 2: Consider the execution depicted in the following figure and answer the questions



1. Define <u>ALL</u> the values that can be returned by read operations (Rx) assuming the run refers to a regular register.

2. Define ALL the values that can be returned by read operations (Rx) assuming the run refers to an atomic register.
3. Let us assume that values retuned by read operations are as follow: $read_1() \rightarrow 2$, $read_2() \rightarrow 2$, $read_3() \rightarrow 3$, $read_4() \rightarrow 1$, $read_5() \rightarrow 3$. Is the run depicted in the Figure linearizable?

Ex 3: Consider the algorithm shown in the Figure

```
upon event ⟨ Init ⟩ do
    delivered := ∅; pending := ∅; correct := Π;
    forall m do ack[m] := ∅;

upon event ⟨ urb, Broadcast | m ⟩ do
    pending := pending ∪ {(self, m)};
    trigger ⟨ beb, Broadcast | [DATA, self, m] ⟩;

upon event ⟨ beb, Deliver | p, [DATA, s, m] ⟩ do
    ack[m] := ack[m] ∪ {p};
    if (s, m)/∈ pending then
        pending := pending ∪ {(s, m)};
        trigger ⟨ beb, Broadcast | [DATA, s, m] ⟩;
```

```
upon event ⟨◊P,Suspect |p⟩do
    correct := correct \ {p};

upon event ⟨◊P,Restore |p⟩do
    correct := correct ∪ {p};

function candeliver(m) returns Boolean is
    return (correct ⊆ ack[m]);

upon exists (s, m) ∈ pending such that
candeliver(m) do
    delivered := delivered ∪ {m};
    trigger ⟨ urb, Deliver | s, m ⟩;
```

Assuming that the algorithm is using a Best Effort Broadcast primitive and an Eventually Perfect Failure Detector ◊P discuss if the following properties are satisfied or not and motivate your answer

- *Validity*: If a correct process p broadcasts a message m, then p eventually delivers m.
- *No duplication*: No message is delivered more than once.
- *No creation*: If a process delivers a message m with sender s, then m was previously broadcast by process s.
- *Uniform agreement*: If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.

Ex 4: Consider a distributed system constituted by *n* processes $\prod=\{p_1, p_2 \ldots p_N\}$ with unique identifiers that exchange messages through perfect point-to-point links and are structured in a ring topology (i.e., each process p_i can exchange messages only with processes $p_{(i+1) \bmod N}$ and stores its identifier in a local variable `next`).
Each process p_i knows the initial number of processes in the system (i.e., every process p_i knows the value of *N*).

1. Assuming that processes are not going to fail, write the pseudo-code of an algorithm that implements a (1, N) regular register.

Ex.5: Answer true or false to the following claims providing a motivation:

1. The performance of a system can be analyzed independently from its load.
2. Let us assume a single service with a single class of requests (i.e. a single workload component). If we are under the stability condition ($\lambda < \mu$) then the expected response time of the system is independent from the arrival pattern.
3. The workload parameters that mostly influence the performance of system are the arrival pattern and the service demands.
4. The time needed to run a simulation is upper-bounded by the simulated time
5. The reliability function R(t) associated to a component may increase after the restoration of a component

Ex.6: A service is provided through 3 types of components (e.g. database, webserver, and an application), referred with A, B and C. At least one instance of each component must be available to provide the service. One only instance is available for components A and C, whereas two instances of component B are available. Assuming that all failures and restorations are independent among them, that the failures rates of the three components A, B and C are respectively 0.2 , 0.5 and 0.3 faults per day, that the mean time to repair the components A, B, and C are respectively 2h, 3h, and 1.5h, is the service available at least 98% of the time? If not, is the availability target met deploying an additional instance of one of the components?

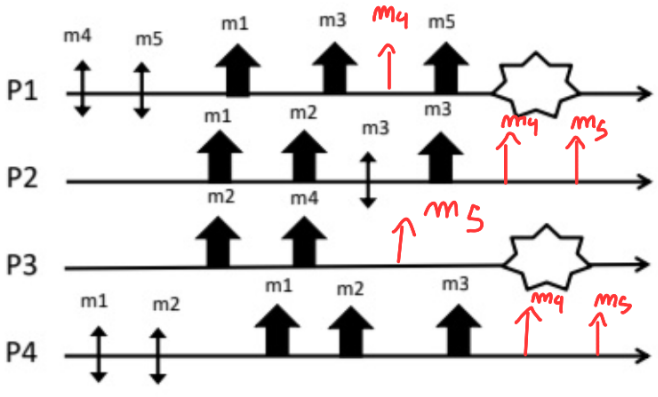
Ex.7: Let us assume a probabilistic version of the Eager Reliable Broadcast protocol in which every process retransmits a rb-delivered message only once with probability 0.5. The rb-broadcast operation is periodically triggered, and the calls follows an exponential distribution with average time between consecutive calls of 10 seconds. Assuming a distributed system of 30 processes and no processing delay, what is the minimum in-channel capacity per process (in terms of message per seconds) to ensure that the expected time for a rb-delivery is below 0.5 seconds?

Exercise:

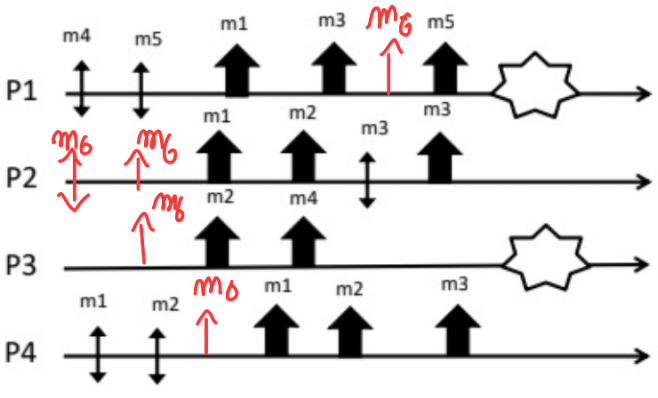
1. We don't have UA because P_1 a faulty process have a set of delivery that is not a subset of the corect one, for m_5.
We have WUTO because the order of pair is respected but not SUTO because we have holes in delivery :
TO(NUA, WUTO)

2. For Fifo we have $m_4 \rightarrow m_5$, $m_1 \rightarrow m_2$
For causal order also $m_1 \rightarrow m_2$ and $m_2 \rightarrow m_3$
Causal order and FIFO broadcast are satisfied because the order in delivery is satisfied in all corect processes.

3.



4.



Exercise 2

1. $R_1: \{1,2\}, R_2 = \{0,1,2\}, R_3 = \{1,2,3\}, R_4 = \{1,2\}, R_5 = \{2,3\}$
2. $R_2: \{0,1,2\}$, R_4 depend on R_2 if $R_2 = 2$ then $R_4 = 2$ else $R_4 = \{1,2\}$, R_1 is concurrent with R_2 and R_4 then can return 1, 2 indipendently from the other 2, R_3 depend on the value before if R_1 or R_4 or R_2 return 2 then $R_3 = \{2,3\}$ else $\{1,2,3\}$, R_5 depend on R_3 if $R_3 = \{3\}$ then $R_5 = \{3\}$ else $\{2,3\}$.
3. $R_1 = 2, R_2 = 2, R_3 = 3, R_4 = 1, R_5 = 3$ is not linearizable because R_2 and R_4 are not concurrent R_4 return 1 after $R_2 = 2$, violating linearizability.

Exercise 3

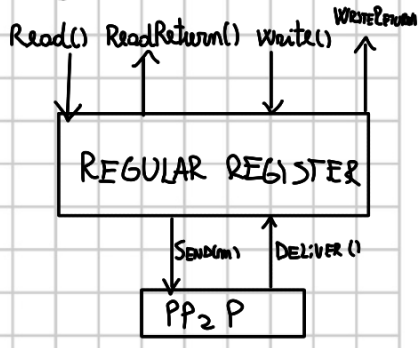
VALIDITY: is satisfied because when we broadcast a message p we add it in pending, and wait that the ack associated to that message is equal to the number of correct processes. We have two case we wrongly suspect a correct process and deliver the message or we recive all acks and deliver, but in any case we deliver p.

No duplication: is not guarantee because we use pending and don't delete any messages from it, if we wrongly suspect a process, we could deliver the message without an ack, but when restore the process and arrive his ack the candeliver(p) check is true, and given that the message is in pending, we will deliver it again.

No creation: also is guarantee we deliver only messages in pending that by the algorithm have only messages previously broadcasted.
Uniform agreement: is violated because if for example we wrongly suspect all other processes and in that period we broadcast a message p, we deliver after recive the delivery of Beb, but when we restore the other corrects processes we don't retrasmitt again p then we have a delivery that other correct processes don't have.

Exercise 4

n processes $\Pi = \{P_1, P_2, \ldots, P_N\}$, PP2P link, each P_i have next = $P_{(i+1) \bmod N}$, No failure.
We have only a writer for example P_1 that send the value and wait for return for do the writeReturn, the other can read their local value at any time.



implementation: Regular Register (1,n), RR
Used: Perfect Point to Point link, PP2P Link

init:

```
val = ⊥
next = P(i+1) mod N
```

upon event < RR, Read > do

```
trigger < RR, ReadReturn | val >
```

```
upon event < RR, Write, v > do
    val = v
    trigger PP2PSend ( WRITE, val ) to next

upon event PP2PDeliver ( WRITE, v ) from next
    IF val >= v do
        trigger writeReturn <val>
    else
        val = v
        trigger PP2PSend ( WRITE, val ) to next
```

When we want read, we read local value and do readReturn. The writer update value, and send the value in the ring, when return the value do the write return, each reader simply update value and send to next.

Exercise 5

1. False, the performance of a system depend heavily on the characteristics of its load, because the expected performances depend on the characteristic of the inputs, we must define very well the workload.
2. λ is not enough to characterize the workload of a sistem, the response time depend on the arrival pattern => False
3. True, they are fundamental, because they characterize the arrival patter, given the information about queueing. The service demands describe the demand on the system.

The service time per completition is important but it is part of service demands, and the response time is not part of workload

4. False, generally there is no relationship between simulated time and the time needed to run a simulation on the computer, the simulation clock is different from the clock of computer.

5. False, Reliability doesn't depend on recovery.

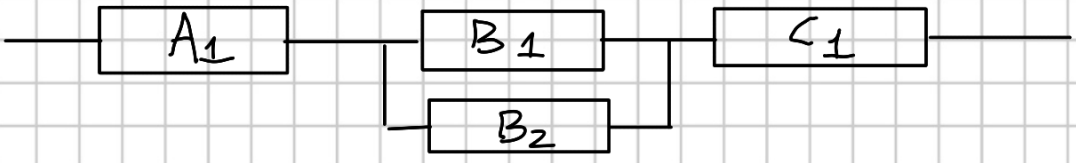
Exercise 6

3 components A, B, C. At least one istance for each variable for provide the services. A, C one istance, B two istances.

$\lambda_A = 0.2$, $\lambda_B = 0.5$, $\lambda_C = 0.3$. $MTTR_A = 2h$, $MTTR_B = 3h$, $MTTR_C = 1.5h$

Service is variable at least 98% of the time? we achive it adding one istance to A or B or C?

Because service is guarantee also if only one istance is variable we have this situation:



$$A_{serial} = \prod A_i = A_A \cdot A_B \cdot A_C$$

$$A_{parallel} \simeq 1 - \prod (1 - A_i) \Rightarrow A_B = 1 - (1 - A_{B_1})(1 - A_{B_2})$$

$$\lambda = \frac{1}{MTBF} \Rightarrow MTBF = \frac{1}{\lambda}, \quad A_i = \frac{MTBF}{MTBF + MTTR}$$

$$MTBF_A = \frac{1}{\underset{\text{fot day}}{0,2}} \frac{1}{\underset{\text{hours}}{24}} = 120h \qquad MTBF_B = \frac{1}{0.5} \frac{1}{24} = 48h$$

$$MTBF_C = \frac{1}{0.3} \frac{1}{24} = 80h$$

$$A_A = \frac{120}{2+120}\frac{k}{k} = 0,983$$

$$A_{B_1} = A_{B_2} = \frac{48}{48+3} = 0,94$$

$$A_B = 1 - (1-0,94)^2 = 0,9964$$

$$A_C = \frac{80}{1,5+80} = 0,981$$

$$A_{TOT} = A_A \cdot A_B \cdot A_C = 0,96 \quad < 0,98$$

No with this inputs not satisfied 98%

Try to optimize the bottleneck C with two istances:

$$A_C = 1 - (1-0,981)^2 = 0,999$$

$$A_{TOT} = A_A \cdot A_B \cdot A_{C\,new} = 0,98$$

We achive objective adding a istance to C.

Exercise 7

Probabilistic Eager Reliable Broadcast, rb-delivered retrasmitted once with probability 0.5. rb-broadcast is triggered every 10 seconds, 30 processes and no delay. message per second capacity per process what is the minimum capacity for it to ensure Expected time for a rb-delivery is less then 0.5 seconds?

We use a queue model with:

$$R = \text{EXPECTED TIME TO DELIVER} = \frac{1}{\mu_i - 30 \cdot \frac{1}{10} \cdot \frac{1}{2}} < 0.5$$

30 ↓ PROCESSES

$\frac{1}{2}$ ↳ PROBABILITIES

$$\frac{1}{\mu_i - \frac{3}{2}} < \frac{1}{2} \Rightarrow \frac{2}{2\mu - 3} < \frac{1}{2}$$

$$2\mu - 3 > 4 \Rightarrow \mu > \frac{7}{2} = 3.5 \text{ messages/sec}$$

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2022/2023

Lecture 27 – Exercises
November 30th, 2022

Ex 1: Let us consider a distributed system composed of N processes executing the algorithm reported in figure

Algorithm 3.12: Broadcast with Sequence Number

Implements:
FIFOReliableBroadcast, **instance** *frb*.

Uses:
ReliableBroadcast, **instance** *rb*.

upon event ⟨ *frb*, *Init* ⟩ **do**
lsn := 0;
pending := ∅;
next := $[1]^N$;

upon event ⟨ *frb*, *Broadcast* | m ⟩ **do**
lsn := *lsn* + 1;
trigger ⟨ *rb*, *Broadcast* | [DATA, *self*, m, *lsn*] ⟩;

upon event ⟨ *rb*, *Deliver* | p, [DATA, s, m, sn] ⟩ **do**
pending := *pending* ∪ {(s, m, sn)};
while exists $(s, m', sn') \in$ *pending* such that $sn' =$ *next*[s] **do**
next[s] := *next*[s] + 1;
pending := *pending* \ {(s, m', sn')};
trigger ⟨ *frb*, *Deliver* | s, m' ⟩;

Let us assume that

1. up to f processes may be Byzantine faulty and
2. A Byzantine process is not able to compromise the underline Reliable Broadcast primitive (i.e., when a Byzantine process sends a message through the rbBroadcast interface, the message will be reliably delivered to every correct process).

For each of the following properties, discuss if it can be guaranteed when f=1 and motivate your answer (also by using examples)

- *Validity:* If a correct process *p* broadcasts a message *m*, then *p* eventually delivers *m*.
- *No duplication:* No message is delivered more than once.
- *No creation*: If a process delivers a message *m* with sender *s*, then m was previously broadcast by process *s*.
- *Agreement:* If a message *m* is delivered by some correct process, then *m* is eventually delivered by every correct process.
- *FIFO delivery:* If some process broadcasts message *m1* before it broadcasts message *m2*, then no correct process delivers *m2* unless it has already delivered *m1*.

Ex 2: Let us consider a distributed system composed of N processes executing the algorithm reported in figure

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
$correct := \Pi$;
$from[p] := [\emptyset]^N$;

upon event $\langle$ *rb*, *Broadcast* | m $\rangle$ **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
if $m \notin from[s]$ **then**
trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$;
$from[s] := from[s] \cup \{m\}$;
if $s \notin correct$ **then**
trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
$correct := correct \setminus \{p\}$;
forall $m \in from[p]$ **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, p, m] $\rangle$;

Let us assume that up to f processes may be Byzantine faulty with a symmetric behaviour i.e., a Byzantine process can change the content of the message it is going to send, but it cannot send different values to different processes when invoking the bebBroascast (they can lie but in a consistent way).

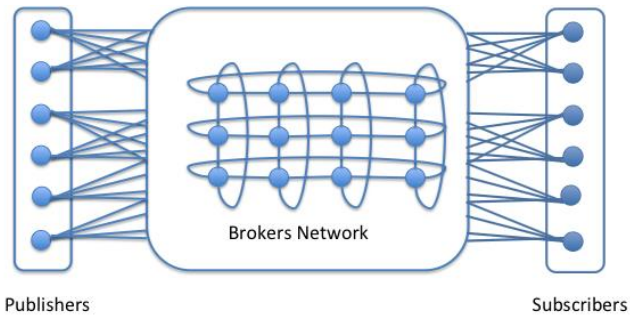
For each of the following properties, discuss if it can be guaranteed when f=1 and motivate your answer (also by using examples)

- *Validity:* If a correct process *p* broadcasts a message *m*, then *p* eventually delivers *m*.
- *No duplication:* No message is delivered more than once.
- *No creation*: If a process delivers a message *m* with sender *s*, then m was previously broadcast by process *s*.
- *Agreement:* If a message *m* is delivered by some correct process, then *m* is eventually delivered by every correct process.

Ex 3: Consider a distributed system constituted by *n* processes $\prod=\{p_1, p_2 \ldots p_n\}$ with unique identifiers that exchange messages through perfect point-to-point links and are arranged in a unidirectional ring (i.e., each process p_i can exchange messages only with process $p_{(i+1) \bmod n}$ and stores its identifier in a local variable `next`).
Each process p_i knows the initial number of processes in the system (i.e., every process p_i knows the value of *n*).

1. Assuming that processes are not going to fail, write the pseudo-code of an algorithm that implements a consensus primitive
2. Let's now assume that processes may crash and that each correct process has access to a perfect failure detector. Discuss the issues of the implementation provided in point 1 and describe how you should modify the algorithm to make it fault tolerant
3. Considering the algorithm proposed in point 1, discuss which properties may be compromised by the presence of one Byzantine process in the ring.

Ex 4: Let us consider a distributed system composed by publishers, subscribers and brokers. Processes are arranged in a network made as follows and depicted below:



1. Each publisher is connected to k brokers through perfect point-to-point links;
2. Each subscriber is connected to k brokers through perfect point-to-point links;
3. Each broker is connected to k brokers through perfect point-to-point links and the resulting broker network is k-connected [1](4-connected in the example);

Answer to the following questions:

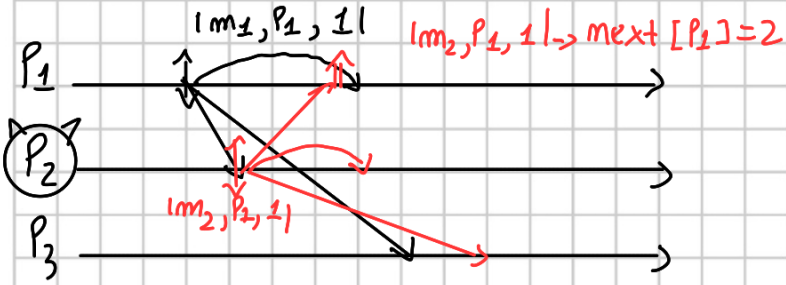
1. Write the pseudo-code of an algorithm (for publisher, subscribers and broker nodes) implementing the subscription-flooding dissemination scheme assuming that processes are not going to fail.
2. Discuss how many crash failures the proposed algorithm can tolerate.
3. Modify the proposed algorithm in order to tolerate f Byzantine processes in the broker network and discuss the relation between f and k.

Exercise 1

N processes, up to F processes may be Byzantine faulty and Byzantine process is not able to compromise the RB primitive. LINK ARE NOT AUTHENTICATED

For F = 1:

Validity: is not guarantee, because if a correct process P_1 send a broadcast a message m, sn = 1, but the byzantine process send a message m_2 with the same sn, like this:

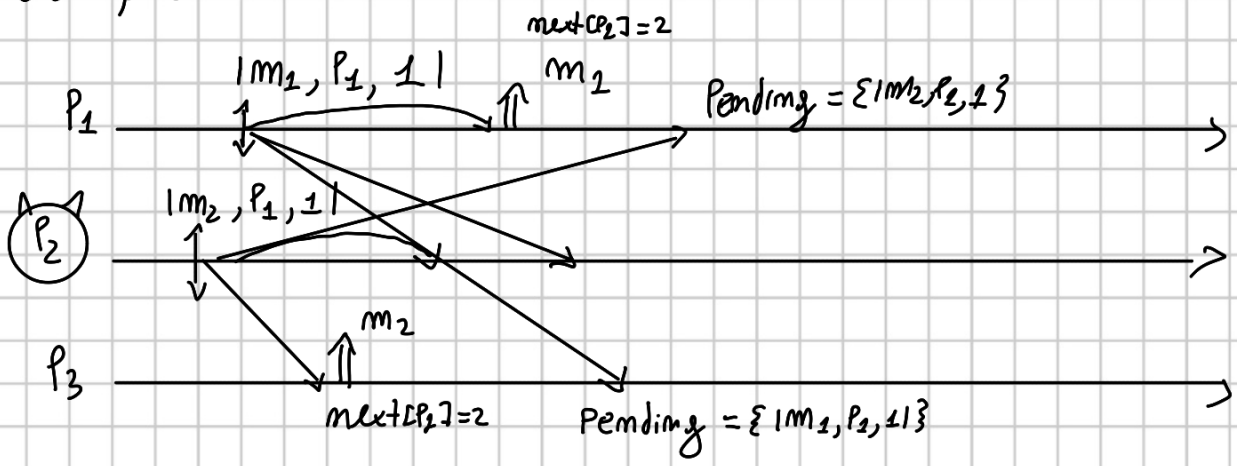


We have that P_1 recive $|m_2, P_1, 1|$, put it in pending and because $sn = 1 = next[P_2]$, he will deliver it, but when arrive $|m_1, P_1, 1|$ the original message will not be delivered because now $next[P_1] = 1$.

No duplication: is not guarantee because like before a process P_1 broadcast a message $|m_1, P_1, 1|$, when the Byzantine process recive it, he can broadcast the message $|m_2, P_1, 2|$ that will be delivered by all correct processes because have the next sequence number, also the original message will be delivered with sn = 1.

No creation: not guarantee, the motivation is exactly the same for validity, byzantine process can impersonate another process and achive the delivery of the message.

Agreement: is not guarantee because we can show this example:



P_1 deliver m_1 but P_3 deliver m_2, violating that must deliver the same set of messages. The messages in pending will not be delivered because for the $SN = 1$.

FIFO delivery: is not satisfied because bysantine proces can broadcast a message m_3 with the same sequence number of m_1, achive the delivery and after m_2 will be delivered without m_1.

Exercise 2

For $f = 1$:

validity: it is satisfied because we have From[p] variable and when we recive a deliver of a broadcast, we check that m is in the list, the bysantine process can only send the same message impersonating the correct process p or another message, but m is always delivered when we check $m \notin$ From[s].

No duplication: is not guarantee because if a process i deliver a message m, if the

byzantine process send the same message but changing the sender, the process i deliver again the message because now from[s] not contain m with that sender.

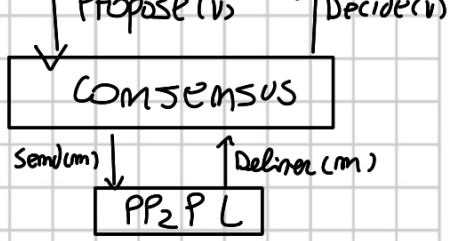
No creation: not guarantee because we don't use authenticated links, the real sender p is not checked, the byzantine process can create false broadcast with fake sender.

Agreement: is guarantee because is true that byzantine can change the content of the message that he send, but have a symmetric behaviour, we achive agreement thanks to the retrasmission of messages of a crashed process.

Exercise 3

n processes $\Pi = \{p_1, \dots, p_n\}$, PP2P links, ring.

1. no failure, implements consensus primitive, we use a token where each process propose a value and after token is full we take the greater value.



implementation: CONSENSUS, C

Uses: PP2P links, PL

init

state = wait

next = $p_{(i+1) \bmod N}$

TT = n

```
    IF self = p0:
      TOKEN = TOKEN ∪ {v}
      trigger PP2PSend(TOKEN) to next

Upon event PP2PDeliver(TOKEN)
    IF size(TOKEN) == n
      decision = max(TOKEN)
      trigger Decide(decision)
      trigger PP2PSend(decision) to next
    else
      TOKEN = TOKEN ∪ {v}
      trigger PP2PSend(TOKEN) to next

Upon event PP2PDeliver(decision)
    If state == WAIT
      state = done
      trigger Decision(decision)
      trigger PP2PSend(decision) to next
    else
      <teach Consensus>
```

2. We have a problem with crashes, messages can be lost and not reach consensus, we must add:

```
init
    pending = ∅
    decision = null
```

state = wait

next = $P_{(i+1) \bmod N}$

$\Pi = n$

IF self = P_0:

TOKEN = TOKEN $\cup \{v\}$ → pendi =

pending = TOKEN ng

trigger PP_2Psend (TOKEN) to next

Upon event PP_2PDeliver (TOKEN)

IF TOKEN == pending:

< discard duplicate token >

else IF size(TOKEN) == Π number of c

decision = max(TOKEN) processes

trigger Decide (decision)

trigger PP_2Psend (decision) to next

else

TOKEN = TOKEN $\cup \{v\}$

pending = TOKEN

trigger PP_2Psend (TOKEN) to next

Upon event PP_2PDeliver (d)

IF state == WAIT AND decision == null

state = done , decision = d

trigger Decision (d)

trigger PP_2Psend (d) to next

else

< reach Consensus > or < duplicate token>

```
upon event <crash(p)>
    Π = Π - 1
    if p == next
        next = next.correct()
        if decision = NULL
            TOKEN = pending
            trigger PP2Psend(TOKEN) to next
        else
            trigger PP2Psend(decision) to next
```

We have add pending where we store the token value we have and decision where we store the decided value. In case of crash we update alive processes and if the next is crashed, we update next and resend the token or the decision. We check in deliver that the message is a duplicate.

3 Taking algorithm 1., we loose termination because the byzantine proces can decide to not resend the token or the decision. We loose validity because byzantine process can change the TOKEN arbitrary and a process p can decide a value V never proposed. Integrity is guarantee because corrects processes have variable wait, we must add that is not Uniform because of Byzantine. Agreement is not guarantee because

if a process decide a value v and the next is the byzantine process, it can send another decision v_1.

Exercise 4 publish, subscribe, notify, unsubscribe

1. In subscription-flooding dissemination scheme, we have that each publisher when want to publish a value, he broadcast the value to all the k brokers which he is connected. The subscribers when want to subscribe send the message to each of the k broker which he is connected. When a broker recive a publish event start a notify event flooding and save the value, but before check if has saved a subscription with that value, and in affermative case send the value to the subscriber. Subscriber get the notify event, and save the value.

implementation: subsctiption-flooding

Uses: PP2P links, PC

init:

delivered = $\emptyset$ Publishet, subsctibet and broker.

neighbours = $\{b_1, \ldots, b_k\}$ Publishet and subsctibet

↳ brokers can have links also with publisher and subscribers

subsctiption = $\emptyset$ subsctibet and broker

(subscription is an array of couples that are composed like (constraint, subscriber) where constraint is a function that take a value and return a boolean)

```
upon event publish ( v )
    if v ∉ delivered :
        delivered = delivered ∪ {v}
        for each n ∈ neighbours :  // send to brokers
            trigger P2PSend ( |NOTIFY|, v ) to n

upon event subscribe ( constraint )
    subscription = subscription ∪ { (constraint, self) }
    for each n ∈ neighbours :  // send to brokers
        trigger P2PSend ( |SUBSCRIPTION|, <constraint, self> ) to n

upon event P2PDeliver ( |SUBSCRIPTION|, < C, p > ) from s
    subscription = subscription ∪ { ( C, p ) }
    ↳ broker save the subscription !

upon event unsubscribe (constraint )
    subscription = subscription / { (constraint, self) }
    for each n ∈ neighbours :
        trigger P2PSend (|UNSUBSCRIPTION|, <constraint, self> ) to n

upon event P2PDeliver(|UNSUBSCRIPTION|, < C, p >) from s
    subscription = subscription / { ( C, p ) }
```

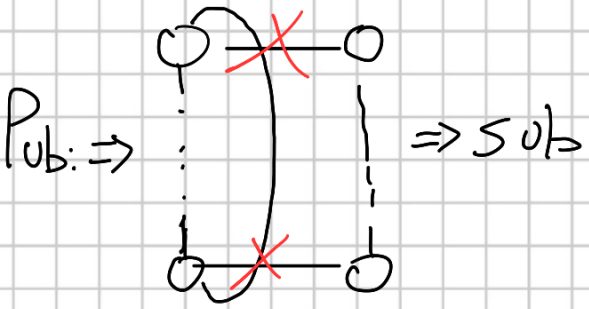
upon event PP2PDeliver (NOTIFY I, V) from P
    if V ∉ delivered do
        for each s ∈ subscription s.t. S.constraint(V)== True
            trigger PP2PSend(PUBLISH I, V) to S.subscriber
        for each n ∈ neighbours s.t. n ≠ p:
            trigger PP2PSend (NOTIFY I, V) to n
                ↳ Flooding of the event.
        delivered = delivered ∪ {V}

upon event PP2PDeliver (PUBLISH I, V)
    delivered = delivered ∪ {V}
        ↳ subscriber save the value

Publisher with publish send his value to the k broker. The subscriber can subiscribe/ Unsubscribe sending the message to the k brokers. The brokers send the value recived to a subscriber if exist a subscription saved and broadcast the value to the K broker near to him.

2. We can tolerate at most K-1 failures, because each time a failure, the neighbours of some brokers decrease, a partition is created, but until K-1 failures is impossible to have that a partition is isolated from an other, but with K failures can happen that a

partition is isolated, and the NOTIFY events can't arrive to the subscribers:



⇒ message doesn't have a path!

At the beginning we have K partition of K brokers each, each partition have K connection to other partitions.

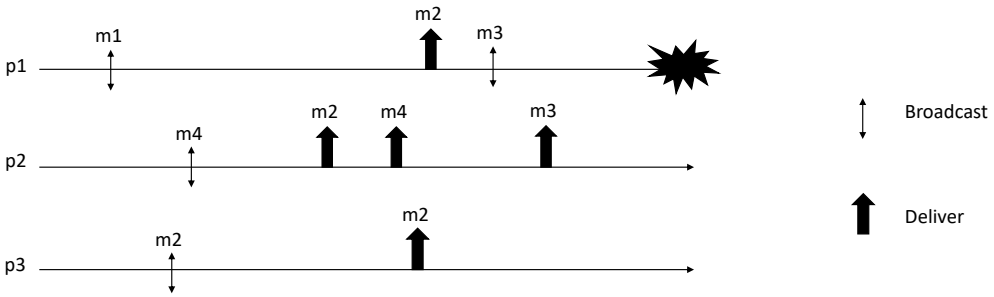
3. For tolerate the Byzantine processes we must use authenticated links, in each event we must check that the sender is equal to the sender in the event. Subscribers must checks the value recived by brokers that is really a value on which is interested and that doesn't already recived, also we can accept a value only when we have recived F+1 events with the same value. It is necessary that K is at least equal to 2F+1 because we must guarantee a reliable communication.

Distributed Systems (9 CFU)
05/07/2022

Family Name_____Name_____Student ID_____

Ex 1: Provide the specification of the (1, N) Regular Register and describe the majority voting algorithm discussed during the lectures.

Ex 2: Consider the message pattern shown in the Figure

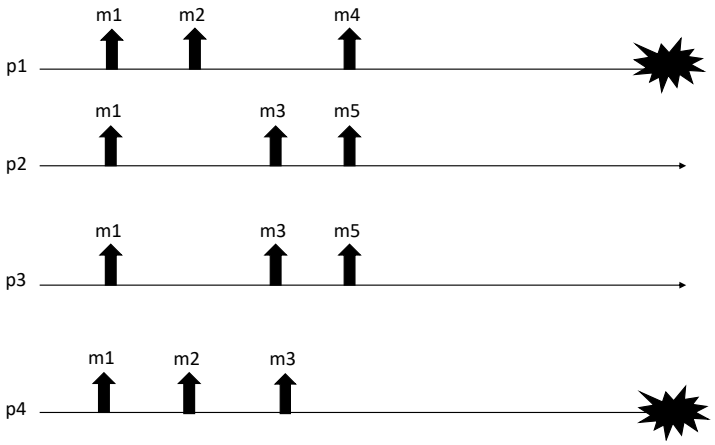


Answer to the following questions:

1. Complete the partial execution in order to obtain a run satisfying Uniform Reliable Broadcast
2. Complete the partial execution in order to obtain a run satisfying Regular Reliable Broadcast but not Uniform Reliable Broadcast
3. Complete the partial execution in order to obtain a run satisfying Best Effort Broadcast but not Regular Reliable Broadcast
4. List **ALL** the possible sequences satisfying both causal order and total order

NOTE: To solve the exercise you can just add deliveries of messages and new broadcast (if needed)

Ex 3: Consider the execution depicted in the Figure



Answer to the following questions:

1. Which is the strongest Total Order specification satisfied by the proposed run? Provide your answer by specifying both the agreement and the ordering property.
2. Modify the run in order to obtain an execution satisfying TO (UA, WUTO) but not TO (UA, SUTO)

3. Modify the run in order to obtain an execution satisfying TO (NUA, WNUTO) but not TO(NUA, WUTO).

NOTE: To solve the exercise you can just add deliveries of messages.

Ex 4: Let us consider a Regular Reliable Broadcast primitive satisfying the following properties:

- *Validity:* If a correct process *p* broadcasts a message *m*, then *p* eventually delivers *m*.
- *No duplication:* No message is delivered more than once.
- *No creation*: If a process delivers a message *m* with sender *s*, then m was previously broadcast by process *s*.
- *Agreement:* If a message *m* is delivered by some correct process, then *m* is eventually delivered by every correct process.

Let us consider a distributed system composed of N processes executing the Eager algorithm (reported in figure)

Algorithm 3.3: Eager Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*.

upon event ⟨ *rb*, *Init* ⟩ **do**
delivered := ∅;

upon event ⟨ *rb*, *Broadcast* | m ⟩ **do**
trigger ⟨ *beb*, *Broadcast* | [DATA, *self*, m] ⟩;

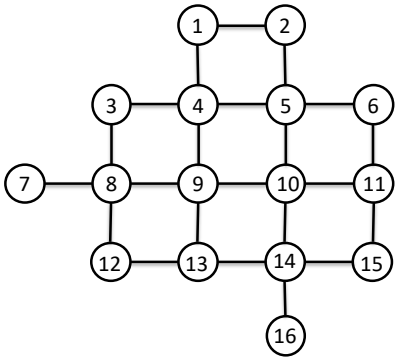
upon event ⟨ *beb*, *Deliver* | p, [DATA, s, m] ⟩ **do**
if $m \notin$ *delivered* **then**
delivered := *delivered* ∪ {m};
trigger ⟨ *rb*, *Deliver* | s, m ⟩;
trigger ⟨ *beb*, *Broadcast* | [DATA, s, m] ⟩;

Answer to the following questions:

1. assuming that up to *f* processes may commit omission failures and no other failures may happen, discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).
1. assuming that up to *f* processes may be Byzantine faulty but constrained to have a symmetric behaviour[1], discuss if the eager algorithm is still able to satisfy the Regular Reliable Broadcast specification (discuss each property individually).

Ex 5: Consider a distributed system composed by *n* processes each one having a unique identifier. Processes communicate by exchanging messages through perfect point-to-point links and are connected through a grid (i.e., each process p_i can exchange messages only with processes located at *nord*, *sud*, *east* and *west* when they exist).
An example of such network is provided in the following figure:

[1] A Byzantine process has a symmetric behaviour if it can change the content of every message it is going to send, but it cannot send different values to different processes when invoking the bebBroascast. Summarizing, it can cheat but it will do it in a consistent way.



Processes are not going to fail, and they initially know only the number of processes in the system N and the identifiers of their neighbors.
Processes in the system must agree on a color assignment satisfying the following specification:

Module

Name: k-Color assignment

Events:

Request: ⟨ ca, Propose | c ⟩: Proposes a color to be adopted.
Indication: ⟨ ca, Decide | c ⟩: Outputs a decided color to be adopted by the process

Properties:

Termination: Every process eventually decides a color.
Validity: If a process decides a color *c*, then *c* was proposed by some process or c = *default*.
Integrity: No process decides twice.
Weak Agreement: If two processes decide c_i and c_j then either $c_i = c_j$ or one of the two is *default*
Color Selection: Let *C* be the set of proposed colors, if $|C| > 0$ then there exists at least a color $c_i \in C$ that is decided by k processes.

Assuming that $1 < k < N$ and that k is known by every process, write the pseudo-code of an algorithm implementing the k-Color assignment primitive.

According to the Italian law 675 of the 31/12/96, I authorize the instructor of the course to publish on the web site of the course results of the exams.

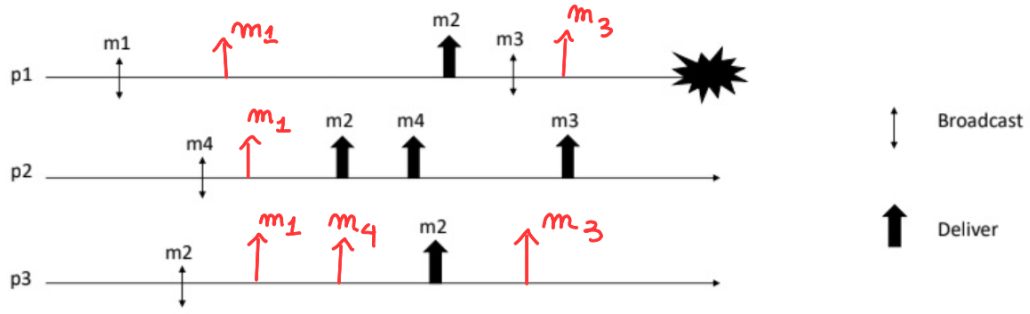
Signature: _____

Exercise 1.

(1, N) Regular Register specification. It has 4 events that are: <rr, Read> that invokes a read operation on the register. <rr, write | v> that invokes a write operation with value v on the register. <rr, Read Return | v> that completes a read operation on the register with return value v. <rr, WriteReturn> that completes a write operation on the register. it must satisfie two properties: termination (if a correct process invokes an operation, then the operation eventually completes) and validity (A read that is not concurrent with a write returns the last value written; a read that is concurrent with a write returns the last value written or the value concurrently written. In the majority voting regular registes protocol we assumed a crash failure model with no perfect failure detector. We assumed N processes whose 1 writer and N readers, with a majority of correct processes. It use PP2P Link and best-effort broadcast. The main idea of the algorithm is that each process locally stores a copy of the current value of the register. Each written value is univocally associated to a timestamp. The writer and the reader processes use a set of witness processes, to track the last value written. Use a quorum, the intersection of any 2 sets of witness processes is not empty for deliver a write or a read. The read return the highest value witness with the current timestamp when reach a quorum of reply and WriteReturn is done when p reach a quorum of acks.

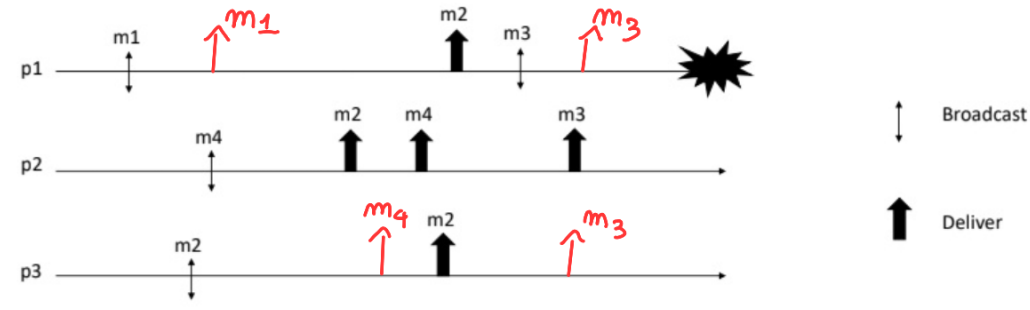
Exetcise 2

1.



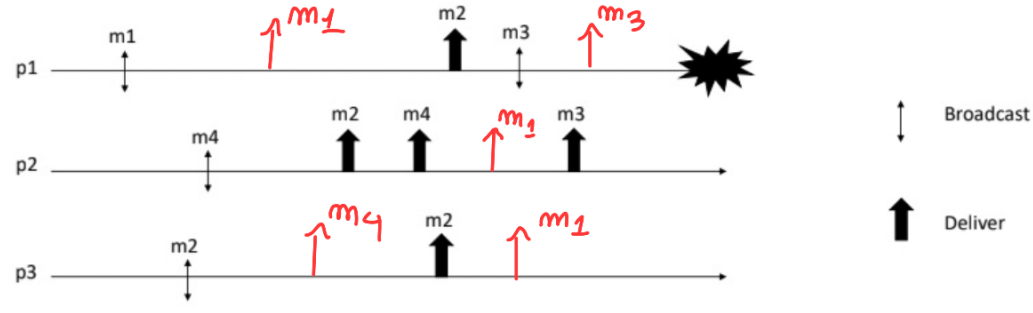
This tum satisfying Uniformm Reliable Broadcast, because cotrect processes have same set of deliveries, the faulty one has a subset.

2.



This tum satisfying Regulat Reliable Broadcast, because cotrect processes have same set of deliveries, but not uniformm because faulty one have delivered m_1 that other don't have.

3.



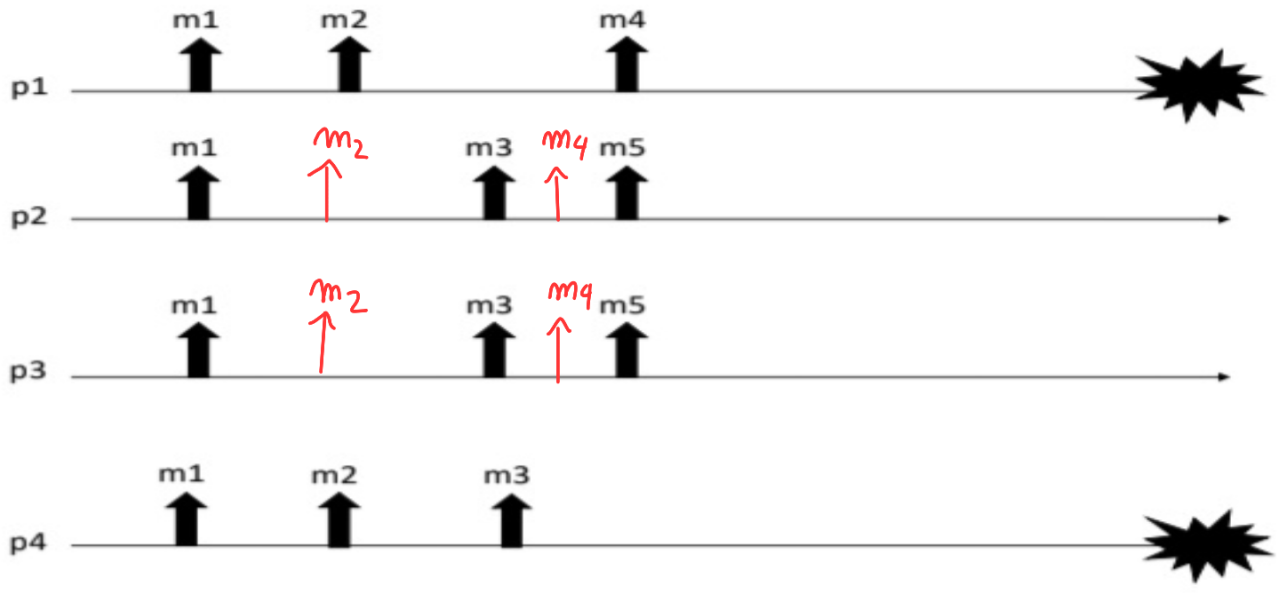
This tum satisfying best effort broadcast because cotrect processes don't have same set of deliveries.

4. For have causal order we must have $m_1 \to m_3$ and $m_2 \to m_3$, For total order because of P_2 we have constraint $m_2 \to m_4 \to m_3$, the sequences that satisfying both are: (m_1, m_2, m_4, m_3), (m_2, m_1, m_4, m_3), (m_2, m_4, m_1, m_3).

Exercise 3

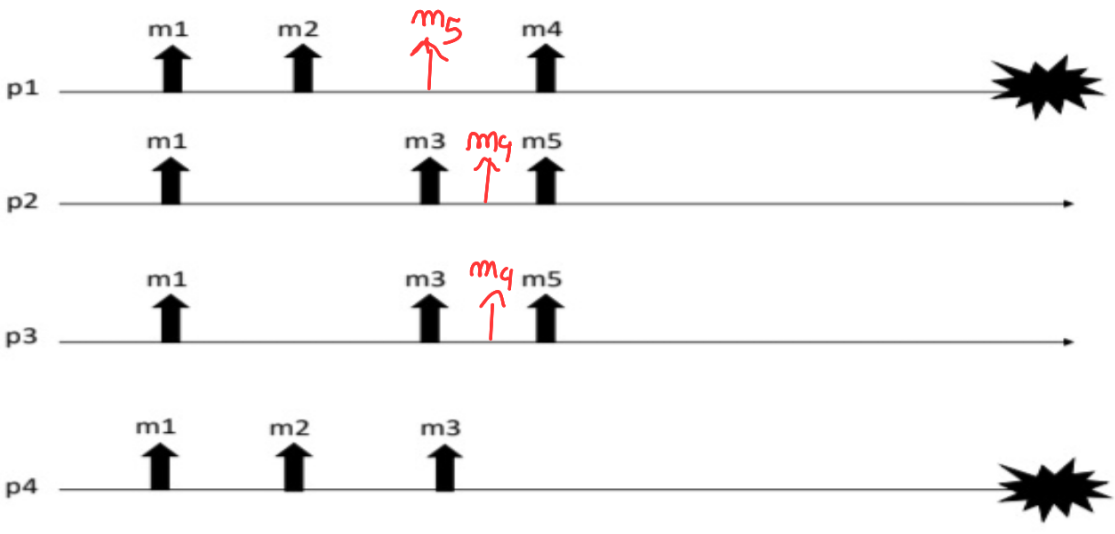
1. This tum doesn't satisfy UA because, P_1 and P_4 that are faulty have not a subset of delivery of the correct processes. It satisfies SUTO because every process respect the constraint of m1°, only pair are respected of other. TO (NUA, W UTO)

2.



it satisfies UA because correct processes have same set of messages, and fauty one a subset. It satisfies WUTO because the order of any pair is respected but not SUTO because P_1 deliver m_4 but not m_3 before

3.



it not satisfies UA because P_1 and P_4 deliver m_2, it doesn't satisfy WUTO because P_2 violate order of pair $m_{c1} \rightarrow m_5$

Exercise 4

1. validity: it is satisfied because if a correct process p broadcast a message m, eventually p recive (p, DATA, S, m) and because m ∉ delivered, he will delivet m. All work because beb use f2p links.

No duplication: it is satisfied because when a process recive a broadcast message m, check that m ∉ delivered, it is impossible to have duplication.

No creation: it is satisfied because if m is delivered, them following the algorithm it must be that s generate a broadcast event with m, the f processes can only do an omission but not generate fake messages.

Agreement: it is satisfied because also if f processes faulty do an omission, when a correct process recive a message do a broadcast, and other correct processes eventually deliver that message

2. validity: it is satisfied because if a correct process p broadcast a message m, eventually p recive (p, DATA, S, m) and because m ∉ delivered, he will delivet m. All work because beb use f2p links.

No duplication: it is not guarantee because a byzantine process can not insert a message delivered in the set delivered, and te deliver the same message when recive again m.

No creation: it is not guarantee because a byzantine process can forge a message m with a fake sender, and because the algorithm don't use authenticate link, a process p will deliver this m.

agreement: it is satisfied thanks to the fact we use PP2P links and every correct process when recive a message m, he will generate a new broadcast of the same message, then at some point every correct process reach the same set of deliveries.

Exercise 5

We have n processes each with an unique identifier. We use PP2P links, using a topology like in the figure, each process know and can comunicate with nord, sud, east and west, if exist. No failures. Each process know only number of processes N. The algorithm that i propose do: is like a consensus protocol. Each process propose a color c_i and disseminate to his neighbours. When a process recive N different propose (each is a pair of (value, identifier)), it will decide for the darkest color (HP: colors are compatable). After he choose a value, given that we want that a color is decided by K processes, a process p check if is id is less or equal to K then generate a decide event with color c, otherwise it will decide default (= 1).

next page pseudo-code:

two events and a passive function.

Init:

np = N (number of processes)

decision = (⊥, FALSE)

propose = ∅ , where we collect proposes

neighbours = {nord, sud, east, west} (some not exist)

K = * is given

Upon event < ca, Propose | c > do

propose = propose ∪ <c, self>

for each p ∈ neighbours do

trigger PP2PSend (|PROPOSE|, <c, self>) to p

Upon event PP2PDeliver (|PROPOSE|, <c, ID>) from s

if <c, ID> ∉ propose do

propose = propose ∪ <c, ID>

for each p ∈ neighbours s.t. p != s do

trigger PP2PSend (|PROPOSE|, <c, self>) to p

else

nothing i have already recived

Upon exist |Propose| == N and decision._2 == FALSE do

chosen = darkest_color (Propose)

return darkest color proposed

If self ≤ K do

decision = (chosen, TRUE)

trigger < ca, decide | chosen>

else

decision = (⊥, TRUE)

trigger < ca, decide | ⊥ >

it satisfies termination, because every process dissemimate his propose and the tecived one to neighbours, given that there are no failures and PP2P links are used, all processes will tecive N different propost and decide a value. It satisfies validity because when a process decide, it decide the darkest color proposed by a process of the default. it satisfies integrity because when a process decide, set decision_2 equal to TRUE and the passive check is no longer satisfied. It satisfied weak agreement because no process decide a value different ffom darkest color of $\perp$, darkest color is the same for all because all processes have same set propose. It satisfie color selection because we construct algorithm in a way that the darkest color is chosen by the first K processes, exist always a color chosen k times.