Distributed Systems
Master of Science in Engineering in Computer Science

AA 2018/2019

MULTI-HOP BROADCAST IN PRESENCE OF BYZANTINE PROCESSES

Distributed system = ABSTRACTION

modeling a set of spatially separate entities, each of these with a certain computational power, that are able to communicate and to coordinate among themselves for reaching a common goal.

Defined by a set of assumptions

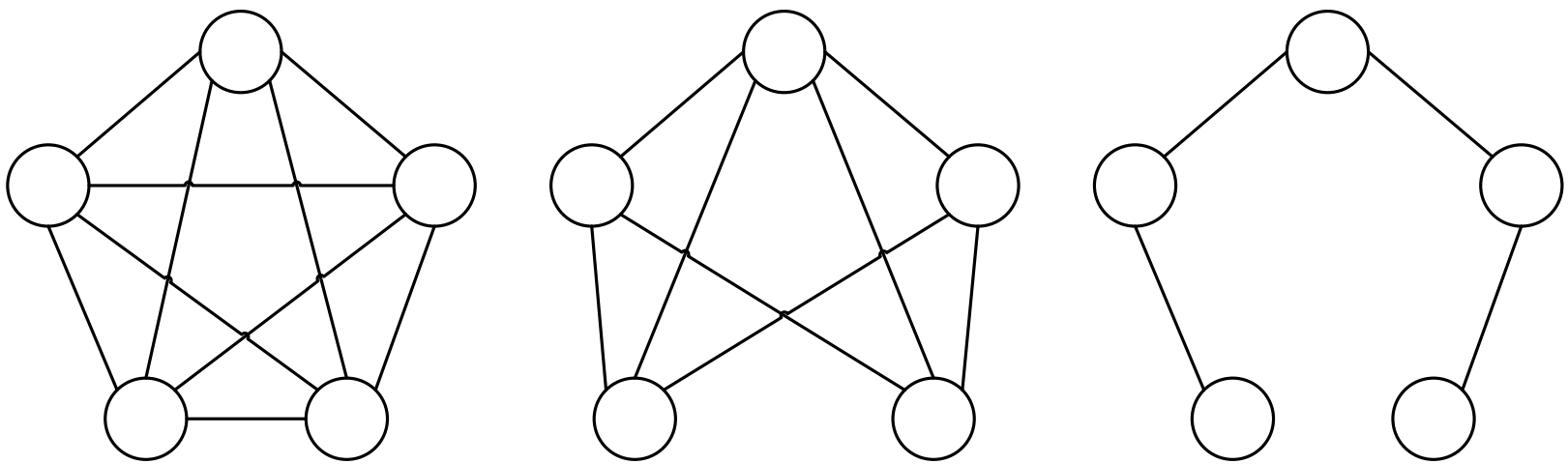
Process Assumptions: All correct, Crash Failures, Crash with

Recovery Failures, Byzantine Failures etc.

Link Assumptions: Fair-loss, Perfect Link etc.

+ Further Assumptions

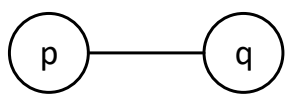
Communication Network (or Network Topology): how processes are interconnected between each other (set of links), defines the set of processes which can *directly* exchange messages.

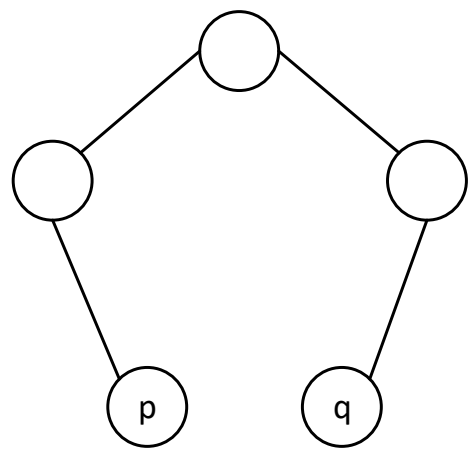


Perfect Point-to-Point Link:

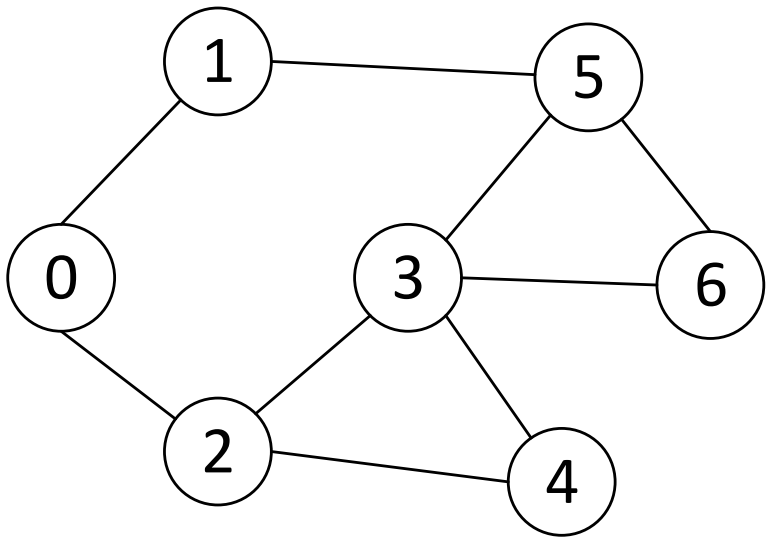
- *Reliable delivery*: If a correct process p sends a message m to a correct process q, then q eventually delivers m.
- *No duplication*
- *No creation*

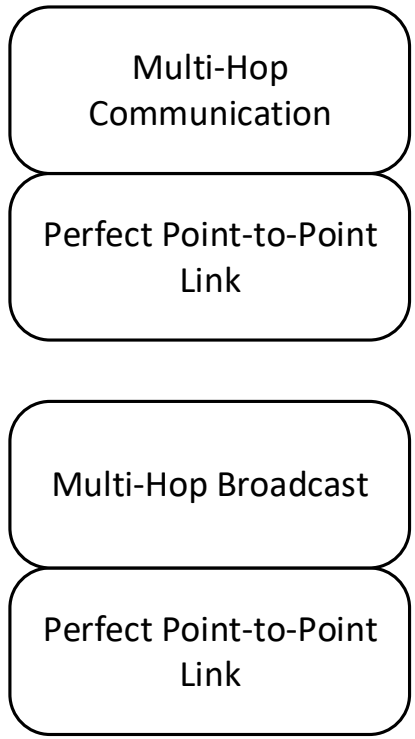
Can processes p and q exchange messages?





multi-hop: a message may cross multiple hop to reach its destination

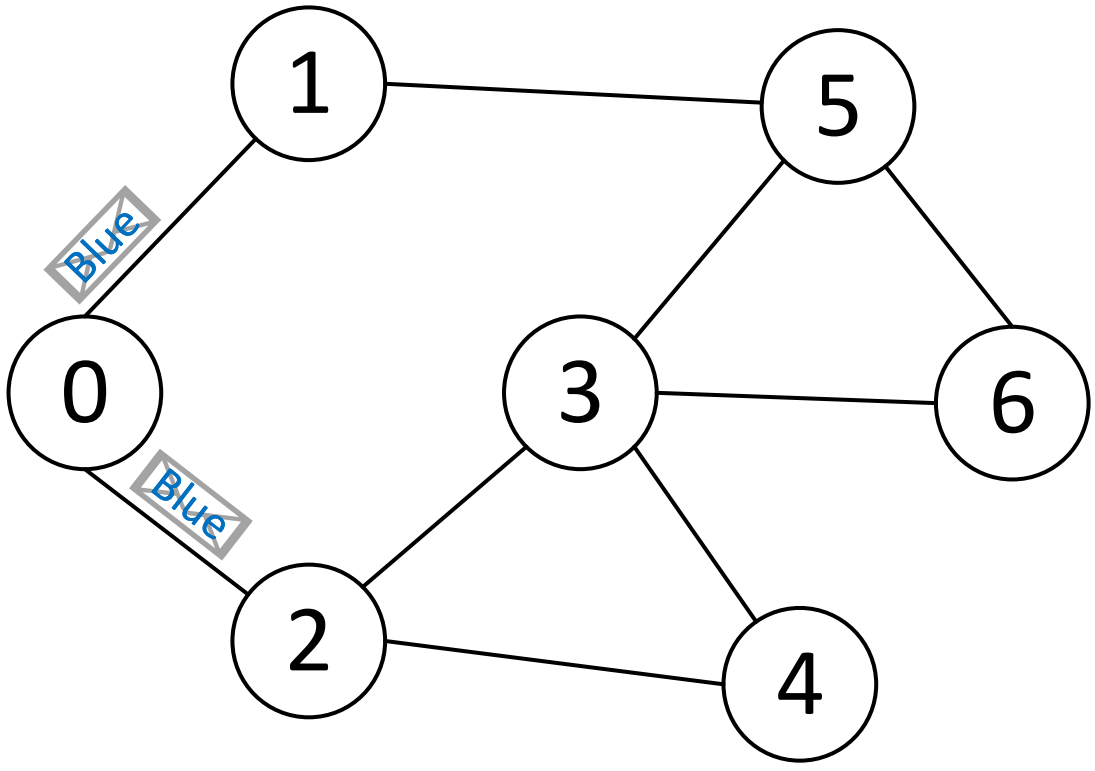




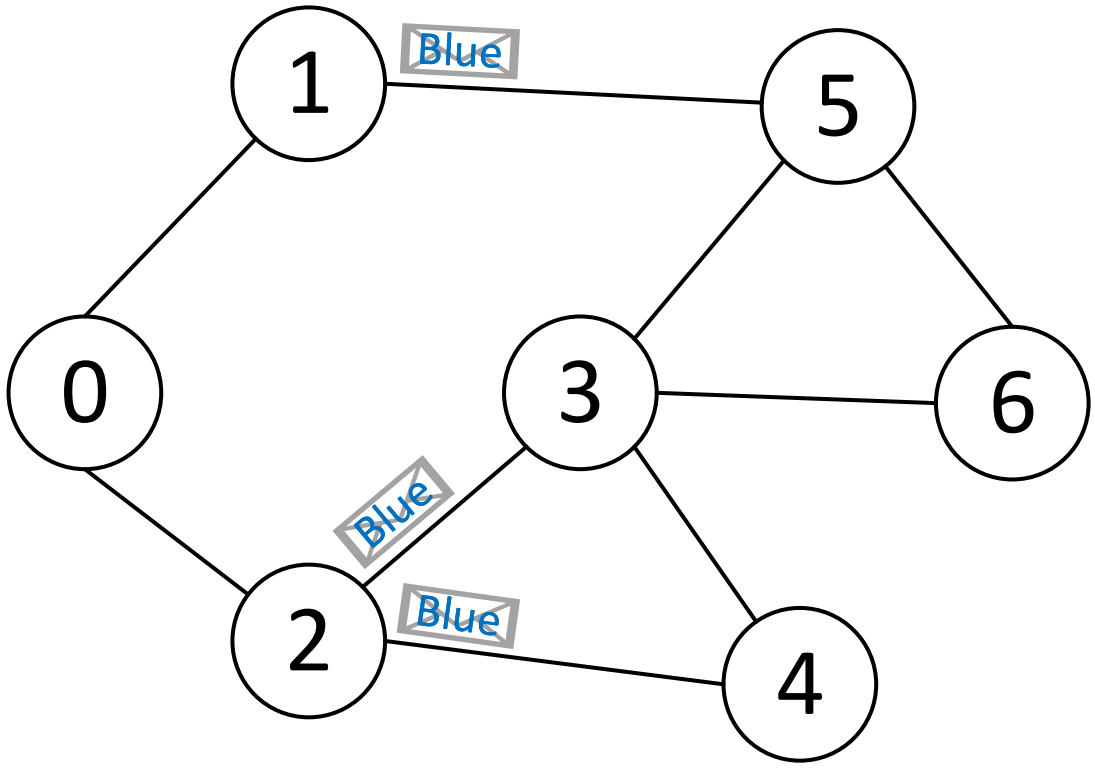
Lots of solutions for Distributed Systems are implicitly designed for **Complete Network**

Not Complete Network:
Perfect Point-to-Point Link $\rightarrow$ **Reliable Communication, Reliable Broadcast**

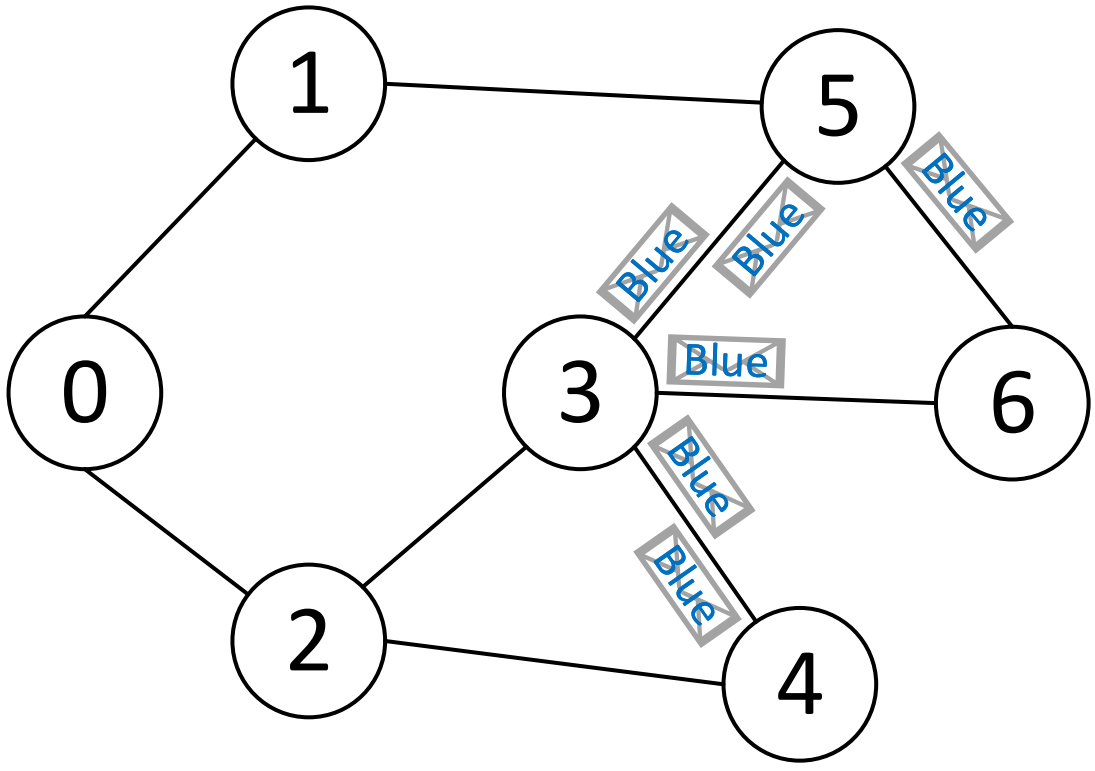
Reliable = Reliable Delivery



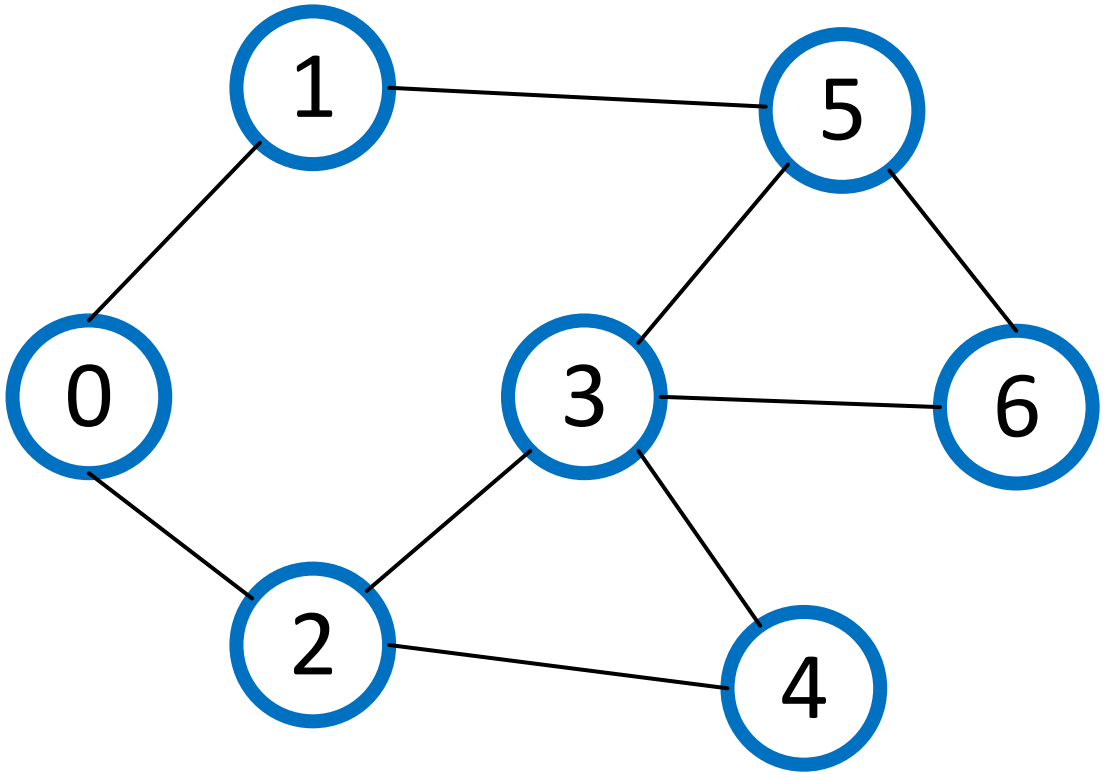
flooding



flooding



flooding


1
5
0
3
6
2
4

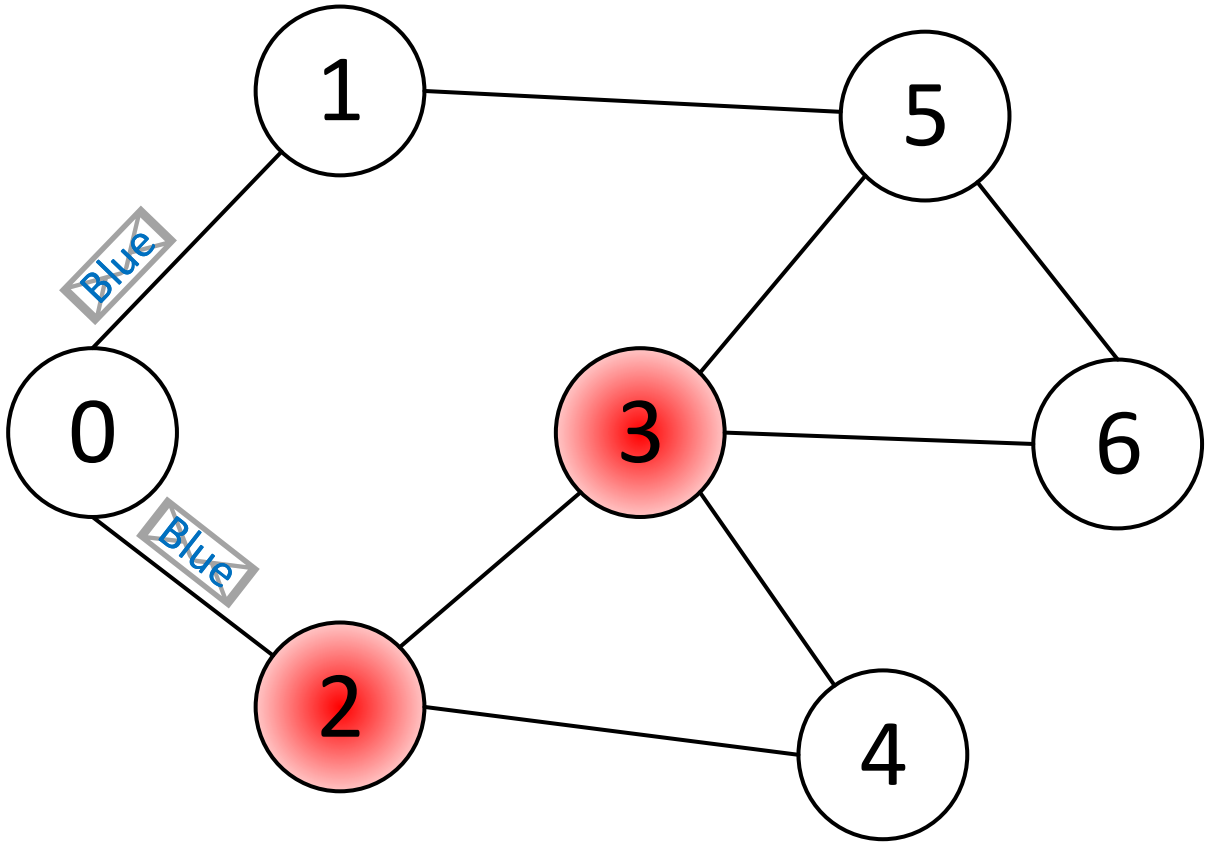
Information spreading

Flooding, naive solution

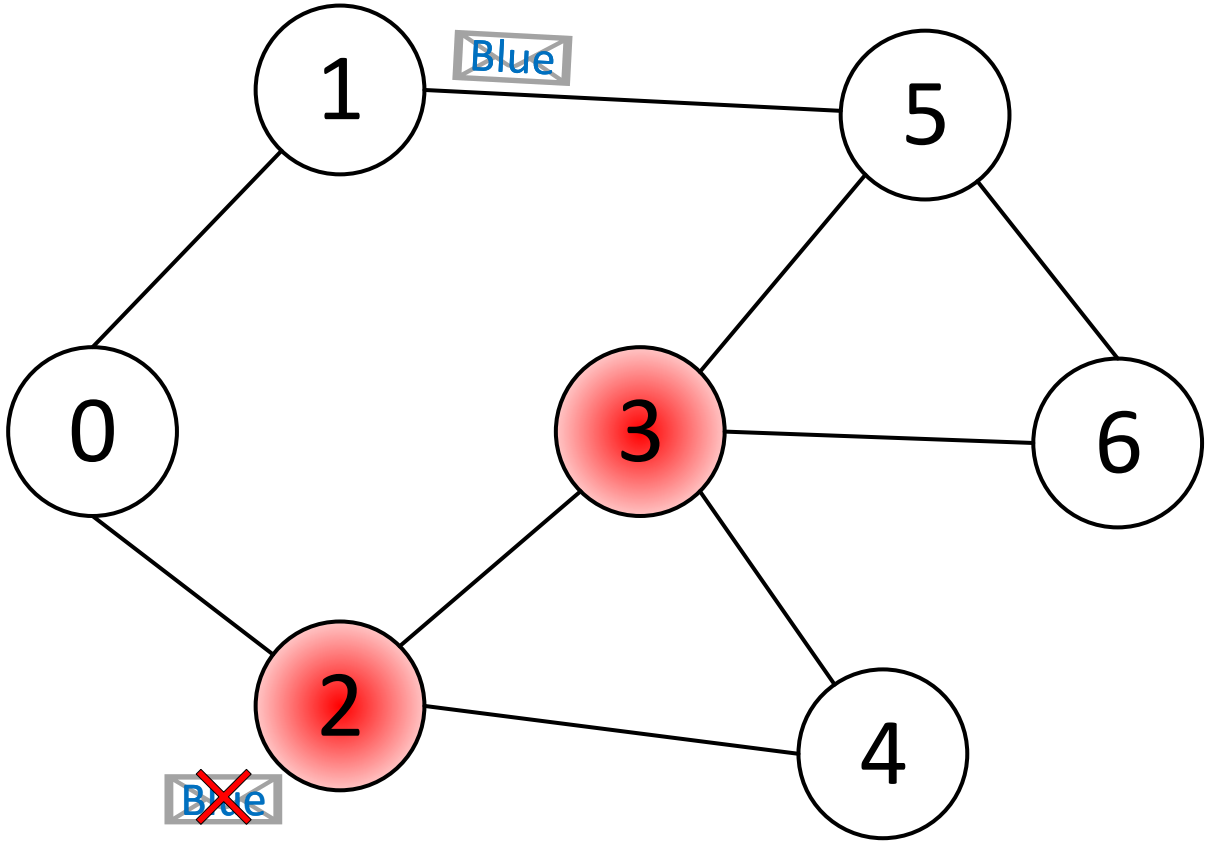
Routing (*when possible*), to save messages (spanning tree)

Gossip, probabilistic, to save messages

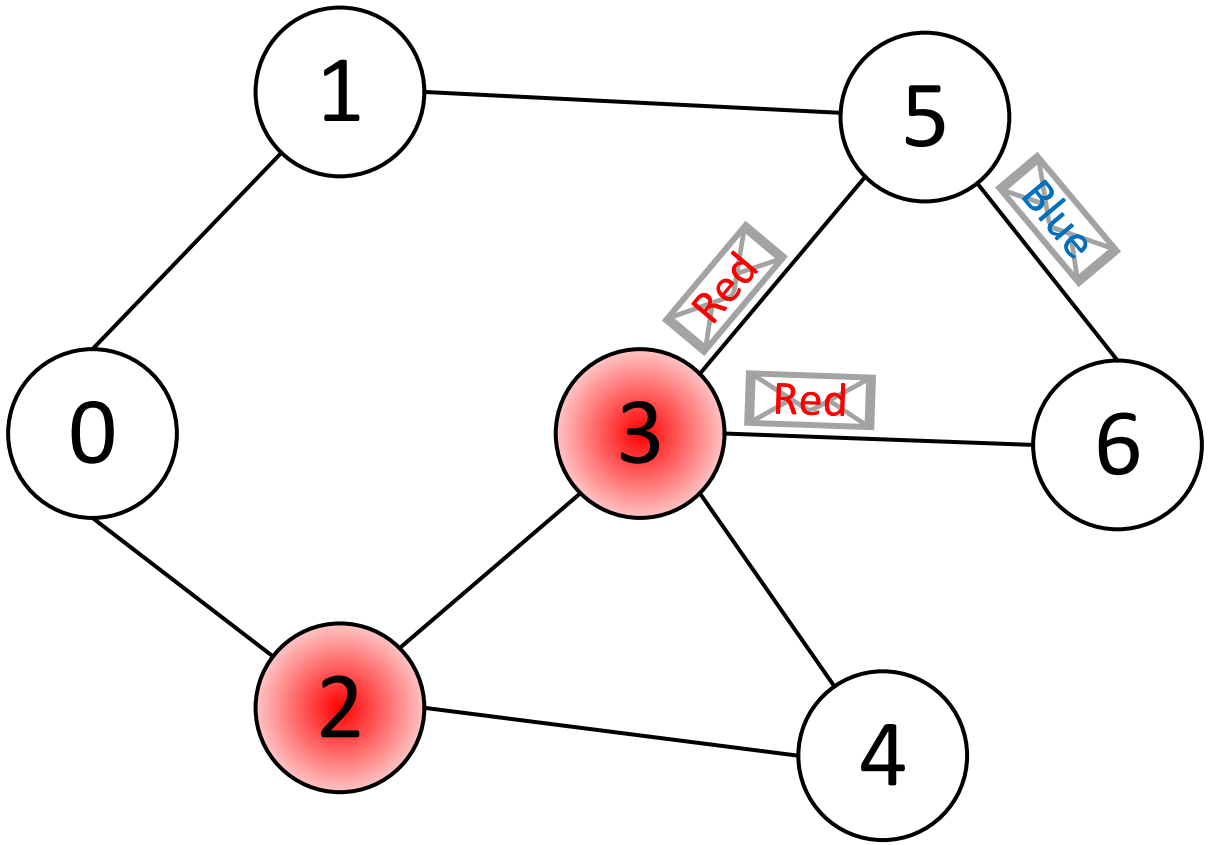
faulty nodes: Byzantine failures



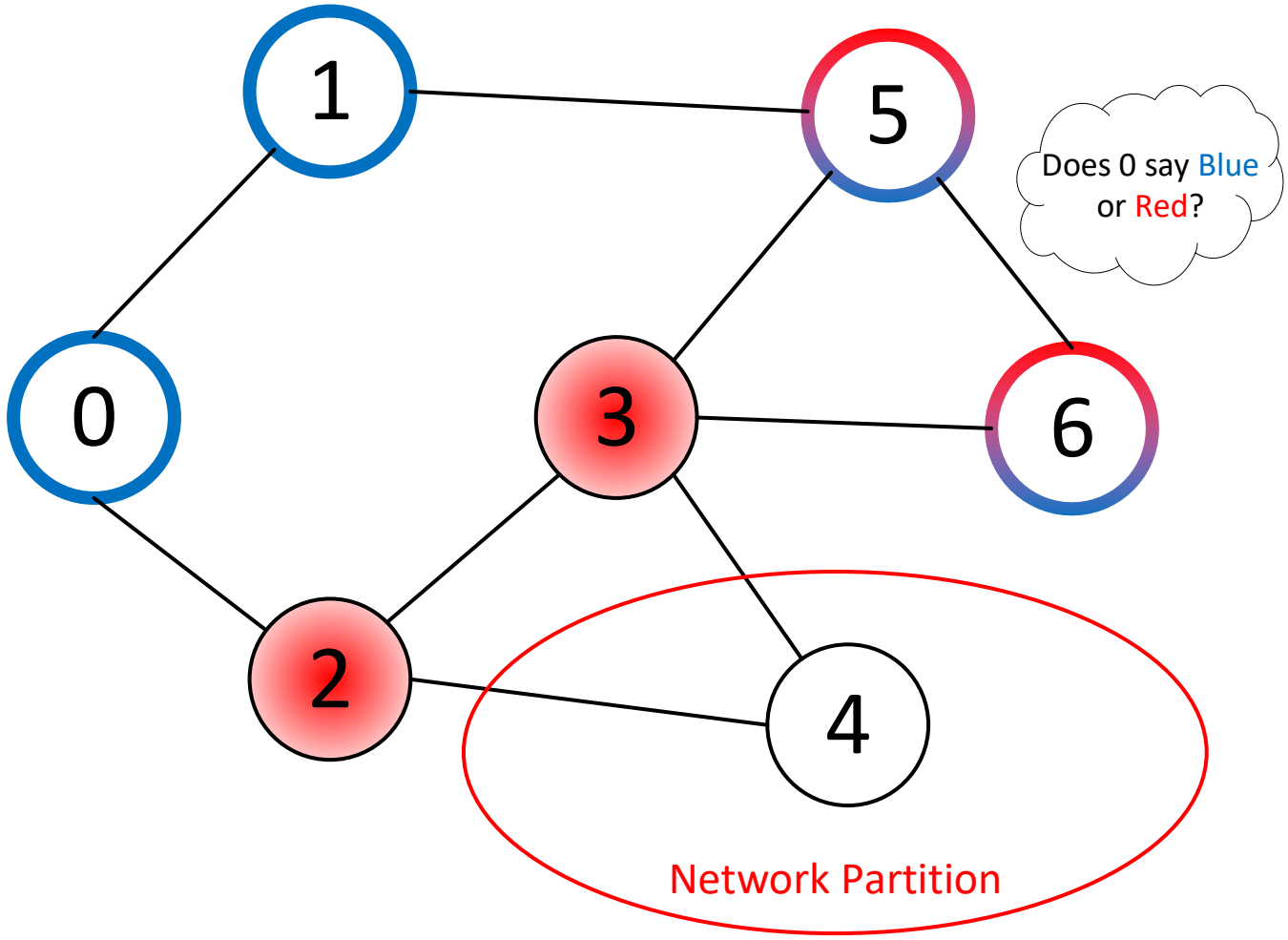
faulty nodes: Byzantine failures



faulty nodes: Byzantine failures



faulty nodes: Byzantine failures



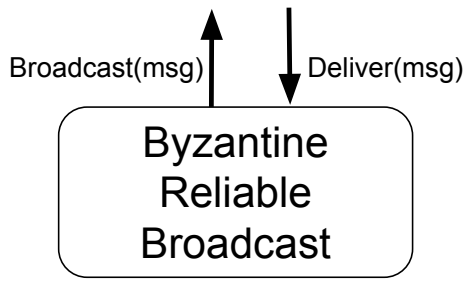
Byzantine Reliable Broadcast Specification

Module:

> **Name:** Byzantine Reliable Broadcast, **instance** *brb*, with source *s*.

Events:

> **Request:** $\langle brb, Broadcast | m \rangle$: Broadcasts a message m to all processes. Executed only by process s.
>
> **Indication:** $\langle brb, Deliver | p, m \rangle$: Delivers a message m broadcast by p.



Properties:

> **RB1:** *Safety*: If some correct process delivers a message m with source p and process p is correct, then m was previously broadcast by p.
>
> **RB2:** *Liveness*: If a correct process p broadcasts a message m, then every correct process eventually delivers m.

Consistency and *Totality* of Byzantine Reliable Broadcast for complete network are not considered.

Clarifications...

Best Effort Broadcast (BEB)

Complete Communication Network, **Crash Failures**

Properties:

- Validity;
- No duplication;
- No creation.

Byzantine Reliable Broadcast (BRB) [1]

Complete Communication Network, Byzantine Failures

Properties:

- Validity;
- No duplication;
- Integrity;
- **Consistency**;
- **Totality**.

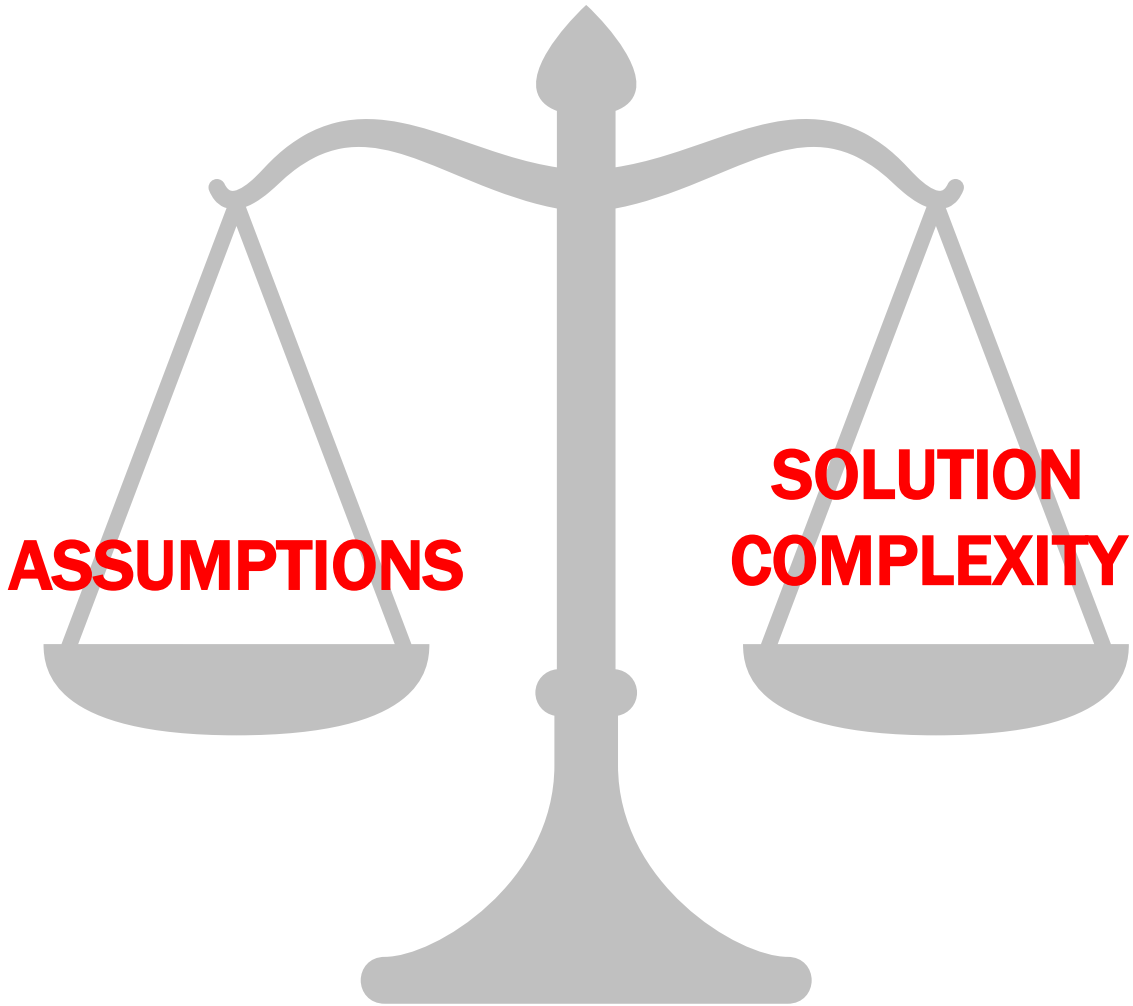
Byzantine Reliable Broadcast (BRB)

Multi-Hop Communication Network, Byzantine Failures, **Correct Source**

Properties:

- Safety (= Integrity);
- Liveness (= Validity).

[1] in the literature it is usually named with Byzantine Agreement

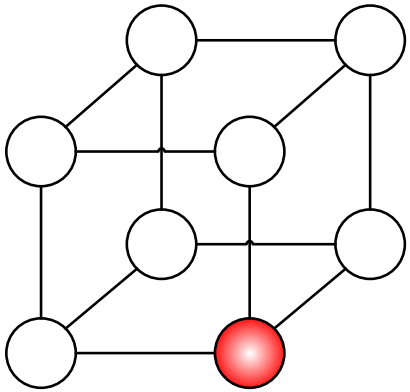


scale By David (en:The Noun Project) [CC BY 3.0 (https://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

Failure Assumptions

Byzantine Failures

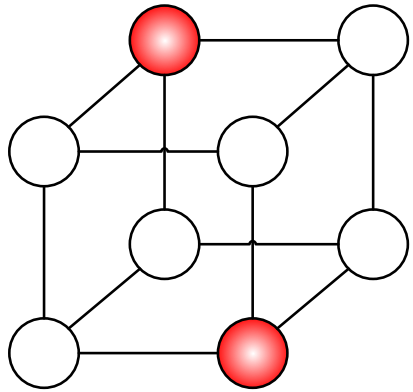
Globally Bounded



f=1

up to f faulty processes arbitrarily spread over the system

Locally Bounded

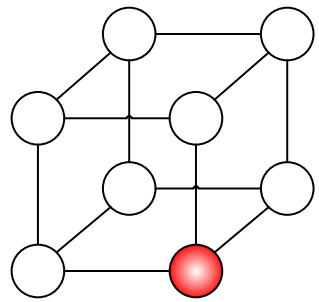


f=1

up to f faulty processes in the neighborhood of every process

Specific Spatial Distribution, Probabilistic Distribution, etc.

Globally Bounded Failure Model



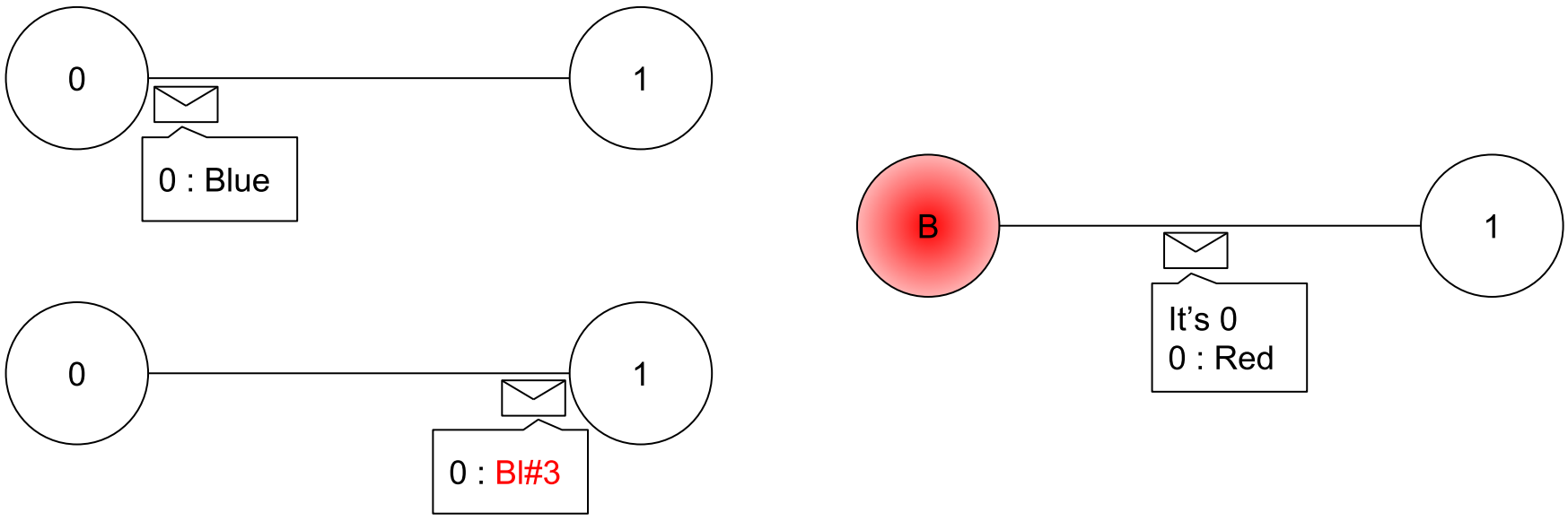
f=1

System Model

- n processes (each one with an unique identifier);
- not complete communication network;
- processes can be correct or Byzantine faulty;
- up to f processes can be faulty (*globally bounded failures*);
- processes have no global knowledge (except the value of f);
- Authenticated Perfect channels.

Safety $\Longrightarrow$ Authenticated Perfect Channels

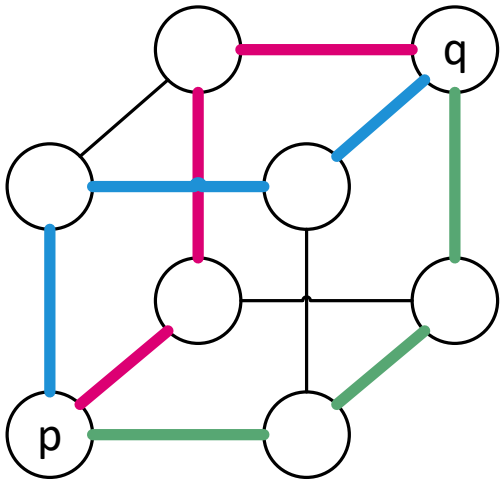
- *Reliable Delivery*;
- *No duplication*;
- ***No creation***;
- ***Authenticity***.



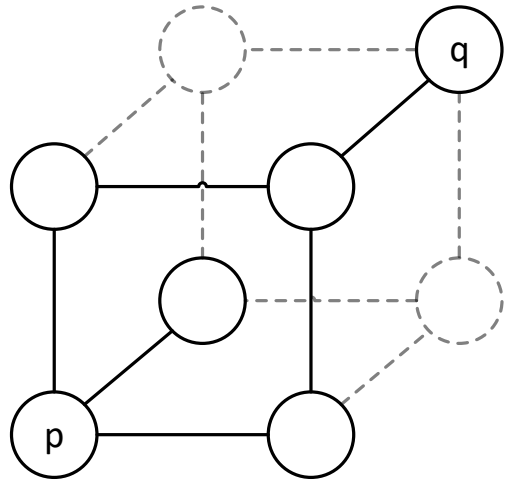
Graph Theory remind - Vertex Connectivity

A graph is k-**connected** if and only if it contains k independent paths between any two vertexes.

Menger Theorem - ***Vertex Cut*** *VS* ***Disjoint Paths***: Let $G = (V, E)$ be a graph and $p, q \subseteq V$. Then the minimum number of vertexes separating p from q in G is equal to the maximum number of disjoint $p - q$ paths in G.



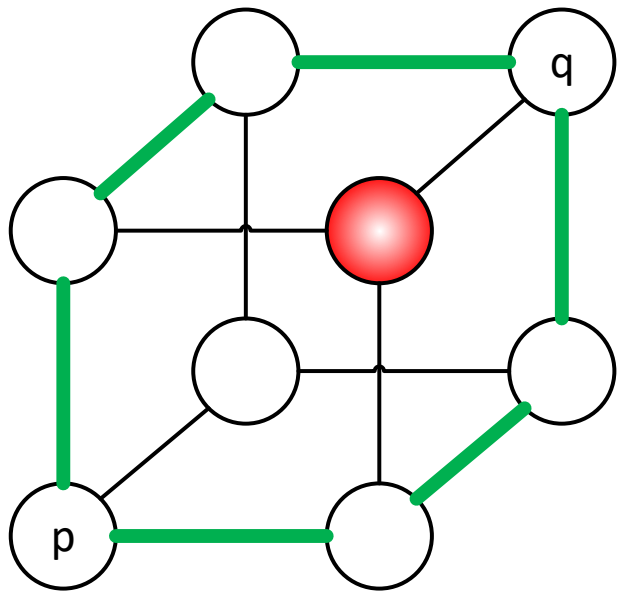
Disjoint Paths



Min Cut

There are at most f Byzantine processes spread over the systems

$\Longrightarrow$ **if a message comes from $f+1$ disjoint paths it can be safely accepted.**



How does a process verify that a message arrived from $f+1$ disjoint paths?

Dolev's Algorithm [Dol81]

Idea: leverage the authenticated channels to **collect the ID's of the processes traversed** by a messege

Message format

*Traversed_Processes is a set data structure.

Dolev Algorithm [Dol81]

Propagation algorithm

- the source process s sends the message msg to all of its neighbors, $msg := \langle s, m, \emptyset \rangle$
- a correct process p saves and relays a message msg sent by a neighbor q to all of other neighbors not included in $traversed\_processes$, appending to it the id of the sender q, namely it multicasts
  $msg := \langle s, m, traversed\_processes \cup \{q\} \rangle$.

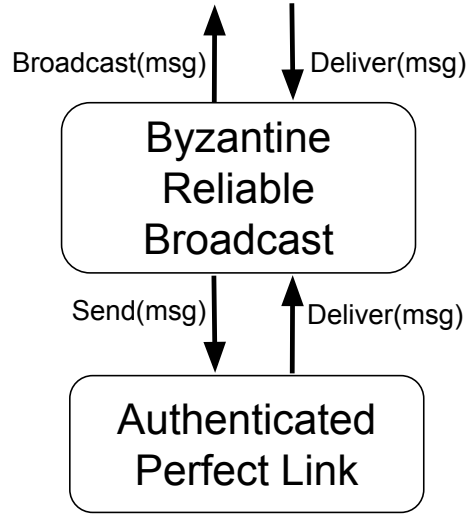
Verification algorithm

- If a process receives copies of msg carrying the same m and s where it is possible to identify $f + 1$ disjoint paths among $traversed\_processes$, then m is delivered by the process.

Dolev Algorithm Implementation

Dolev Byzantine Reliable Broadcast

Implements:
ByzantineReliableBroadcast, **instance** *brb*, with sender *s*.
Uses:
AuthPerfectPointToPointLinks, **instance** *al*.
Asynchronous System

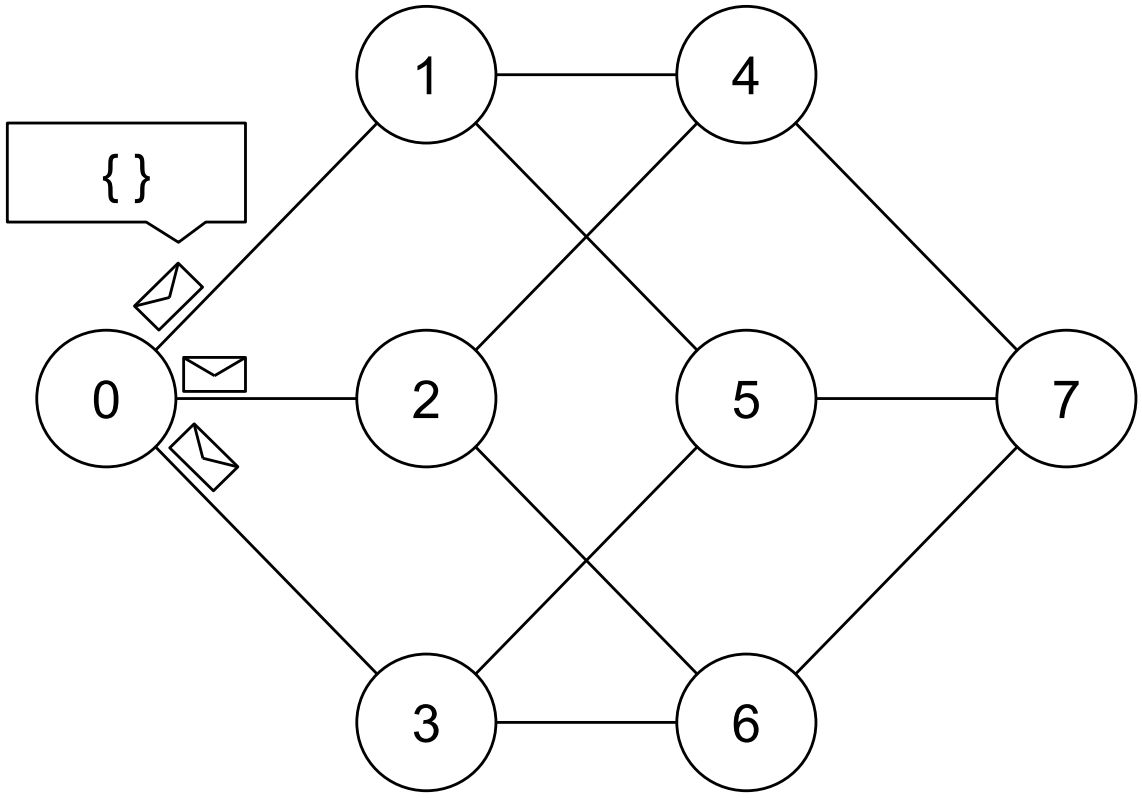


upon event ⟨ *brb*, *Init* ⟩ **do**
    received := Ø
    delivered := Ø

upon event ⟨ *brb*, *Broadcast* | ⟨*p*,*m*⟩ ⟩ **do**
    forall *q* ∈ $Neighbors_p$ **do**
        trigger ⟨ *al*, *Send* | *q*, [⟨*p*,*m*⟩ ,Ø] ⟩;

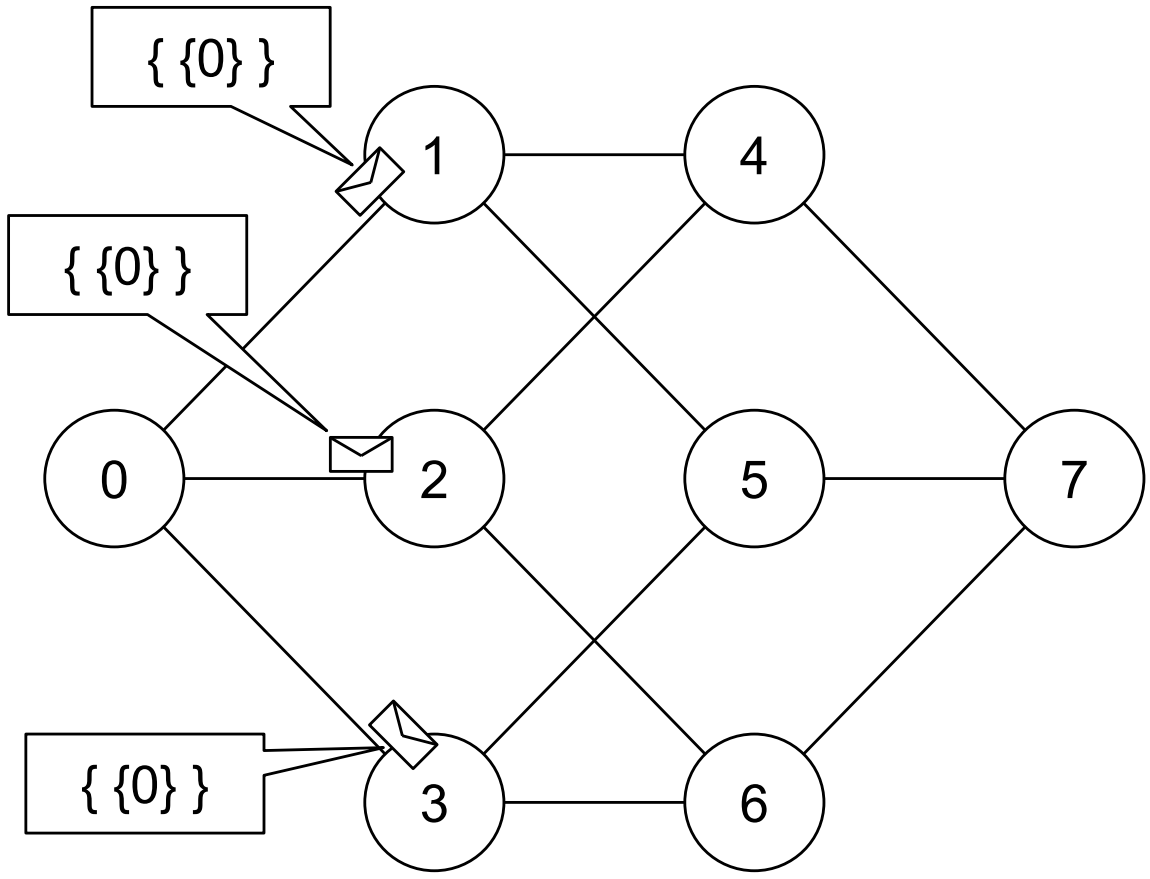
upon event ⟨ *al*, *Deliver* | $\hat{q}$, [⟨*s*,*m*⟩ , *tp*] ⟩ **do**
    new_tp := *tp* ∪ $\hat{q}$
    if $\hat{q}$ == *s* **then**
        trigger ⟨ *brb*, *Deliver* | ⟨*s*,*m*⟩ ⟩;
        delivered := *delivered* ∪ ⟨*s*,*m*⟩
    else
        received[⟨*s*,*m*⟩] := *received*[⟨*s*,*m*⟩] ∪ *new_tp*
        if ⟨*s*,*m*⟩ ∉ *delivered* ***and check_disjoint_paths***(*f+1*, *received*[⟨*s*,*m*⟩]) == *True* **then**
            trigger ⟨ *brb*, *Deliver* | ⟨*s*,*m*⟩ ⟩;
            delivered := *delivered* ∪ ⟨*s*,*m*⟩
    forall *q* ∈ $Neighbors_p$ **such that** *q* ∉ *new_tp* **do**
        trigger ⟨ *al*, *Send* | *q*, [⟨*s*,*m*⟩ , *new_tp*] ⟩;

Dolev Algorithm Graphical Example



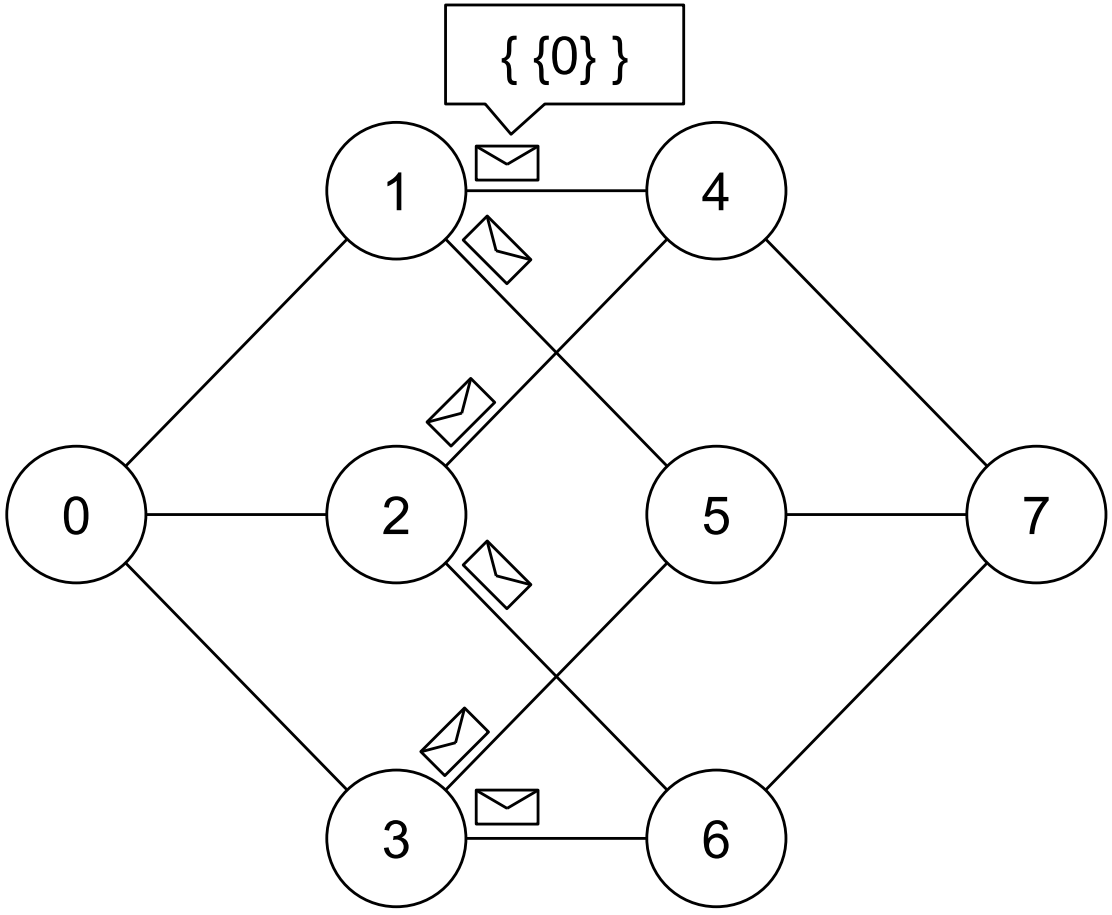
$f = 1$

Dolev Algorithm Graphical Example



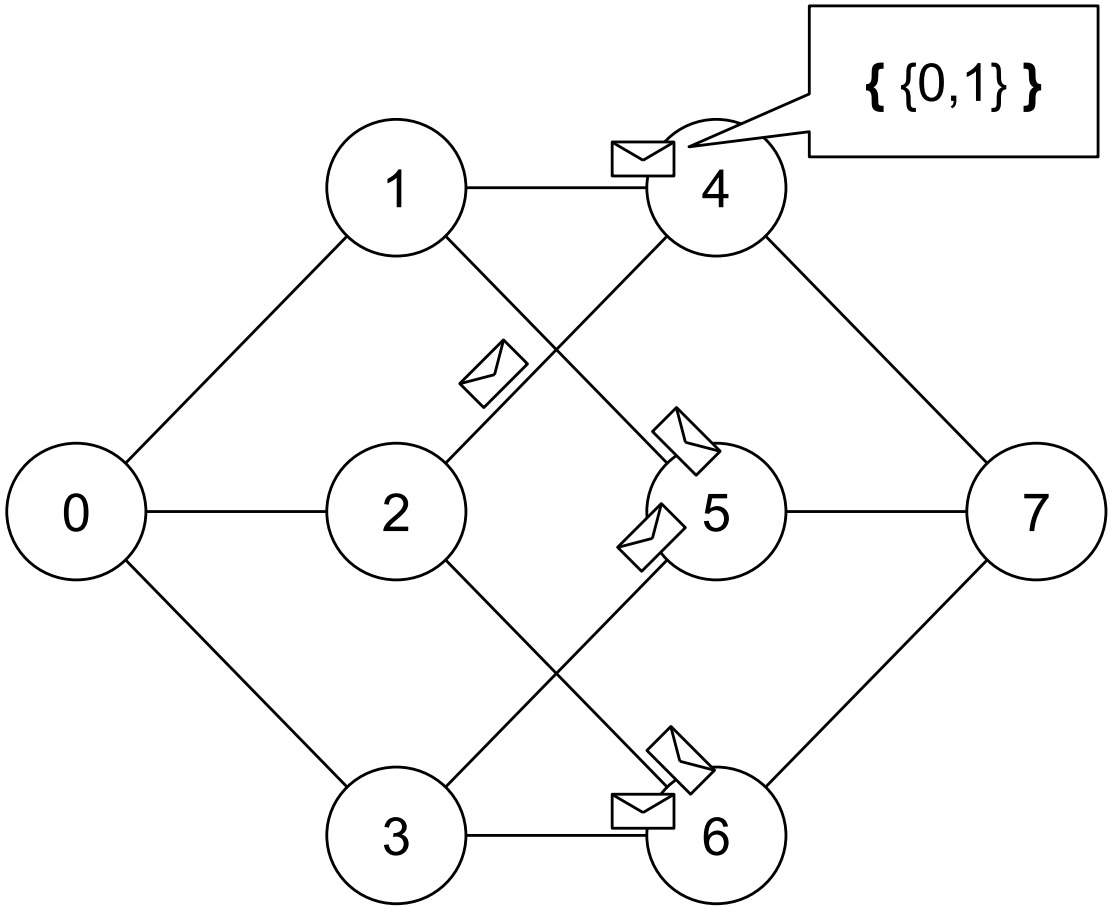
$$f = 1$$

Dolev Algorithm Graphical Example



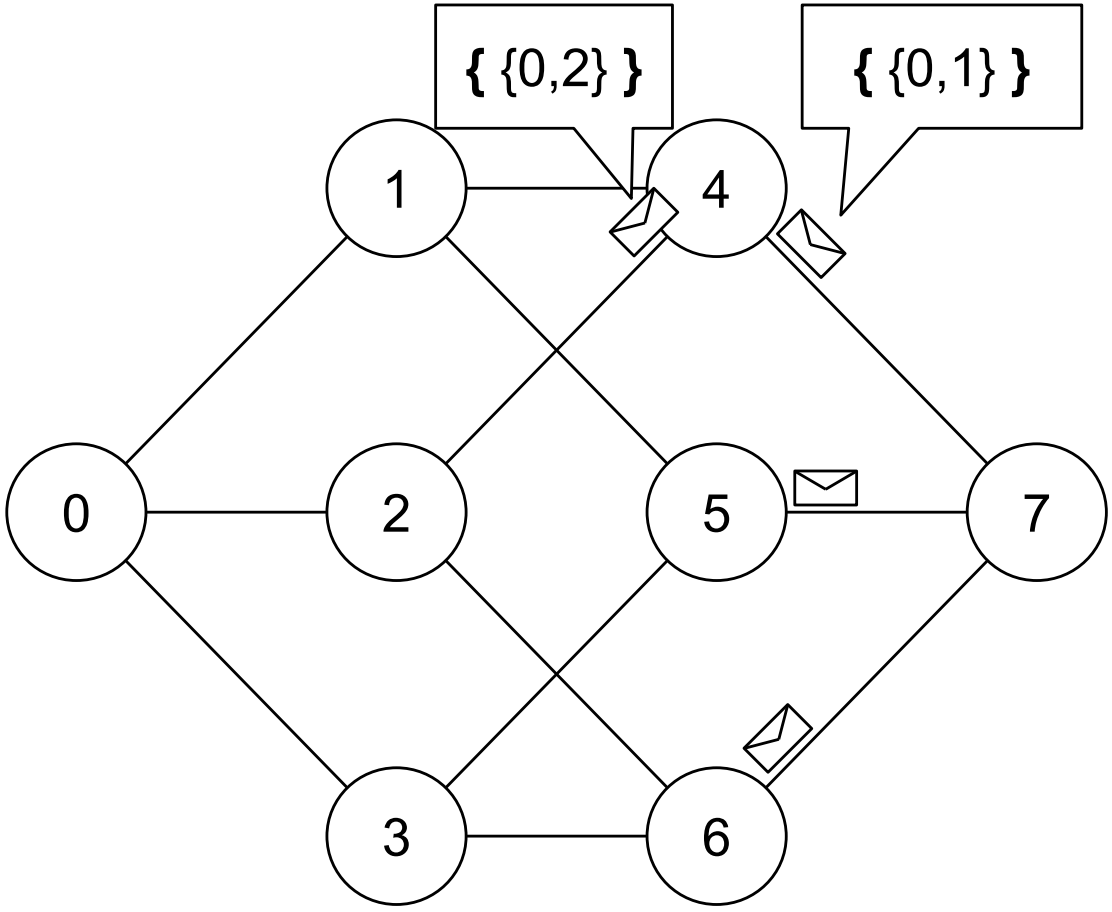
$f = 1$

Dolev Algorithm Graphical Example



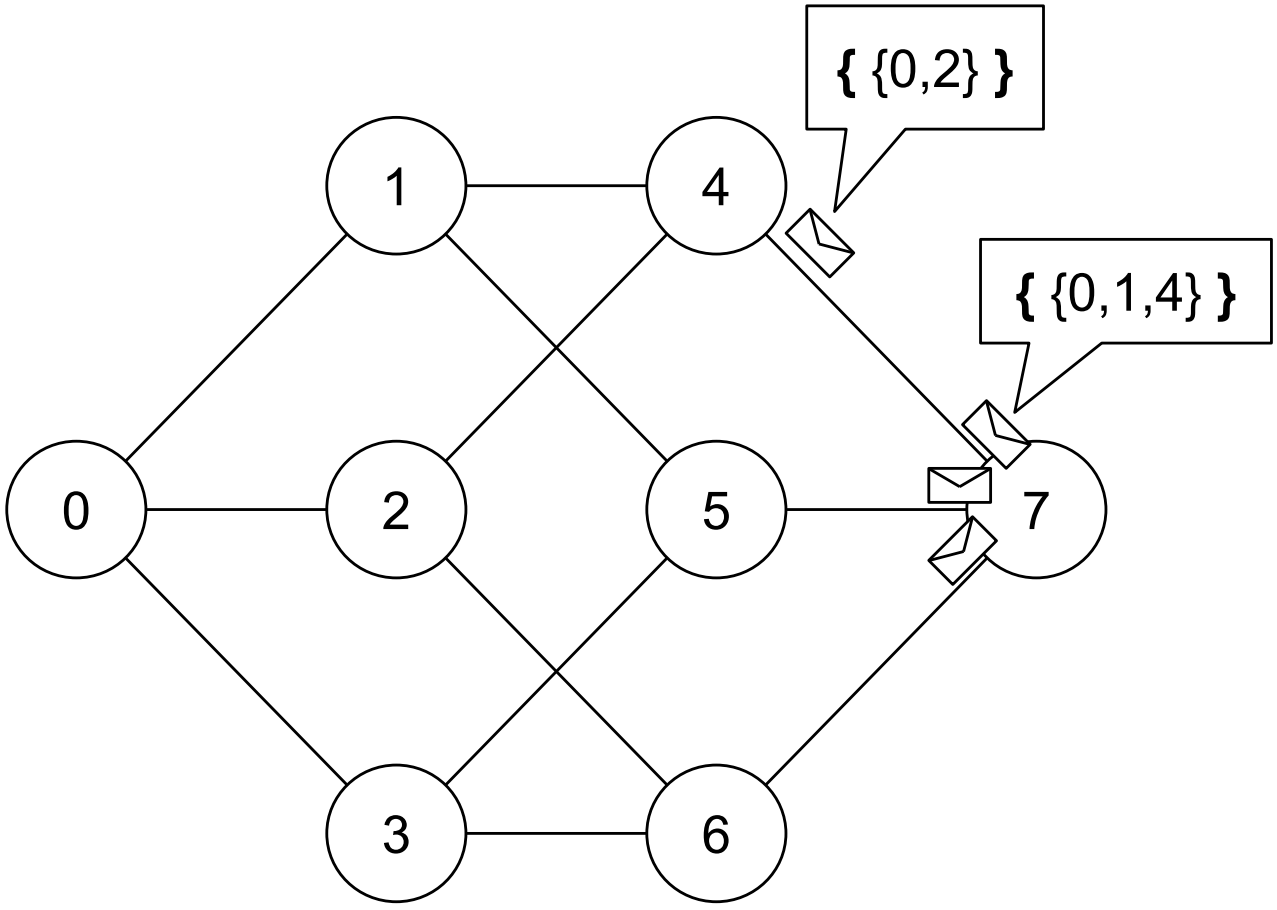
$f = 1$

Dolev Algorithm Graphical Example



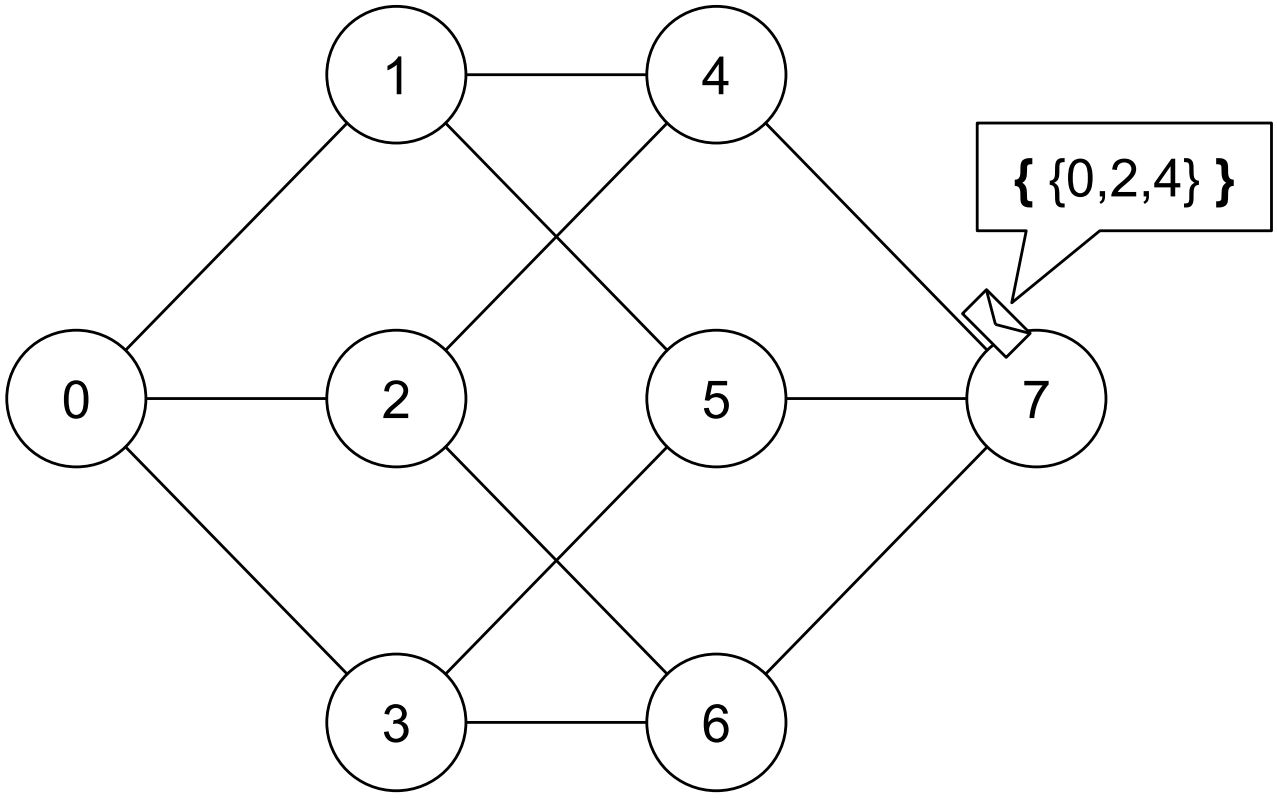
$f = 1$

Dolev Algorithm Graphical Example



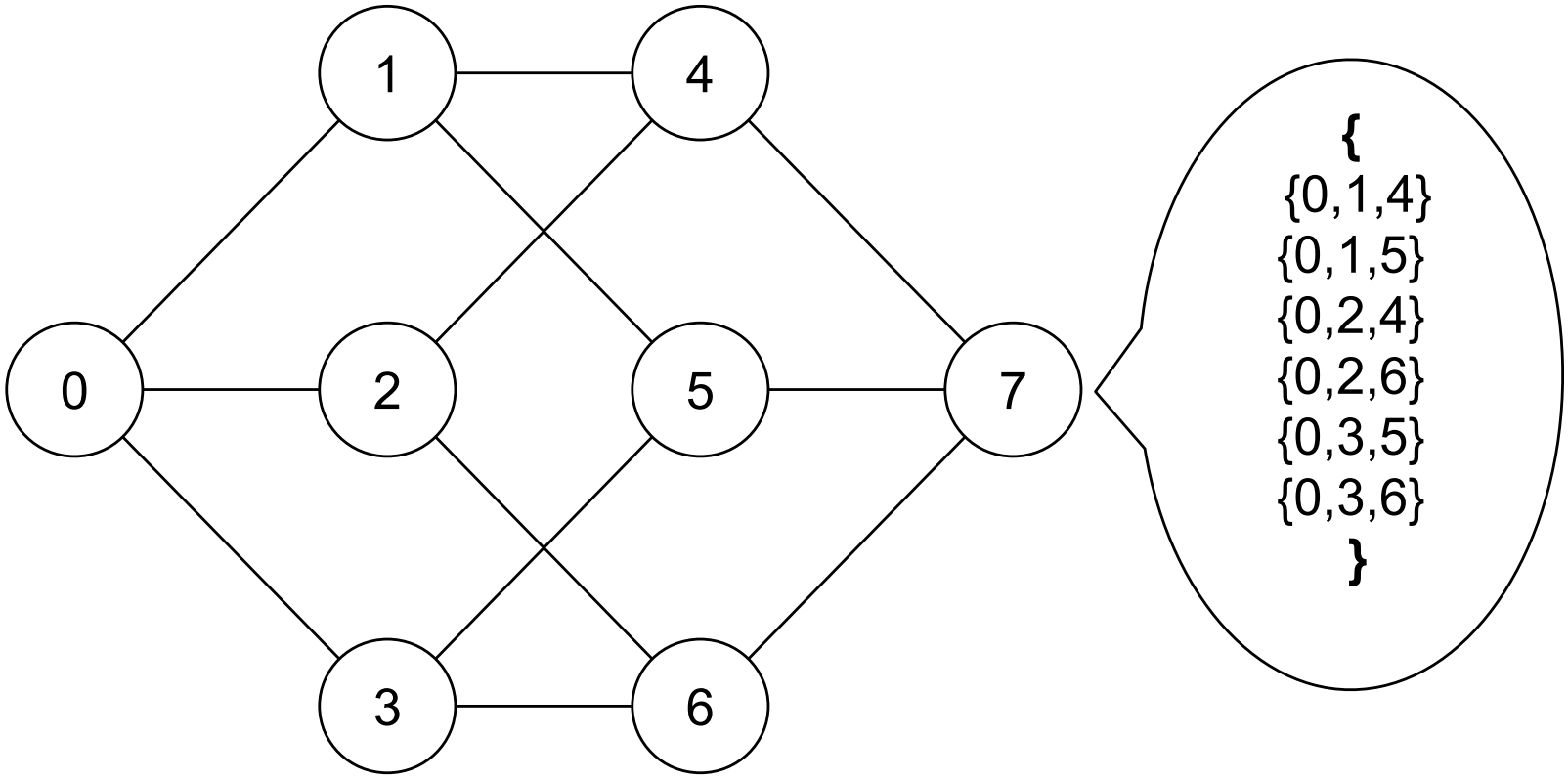
$f = 1$

Dolev Algorithm Graphical Example



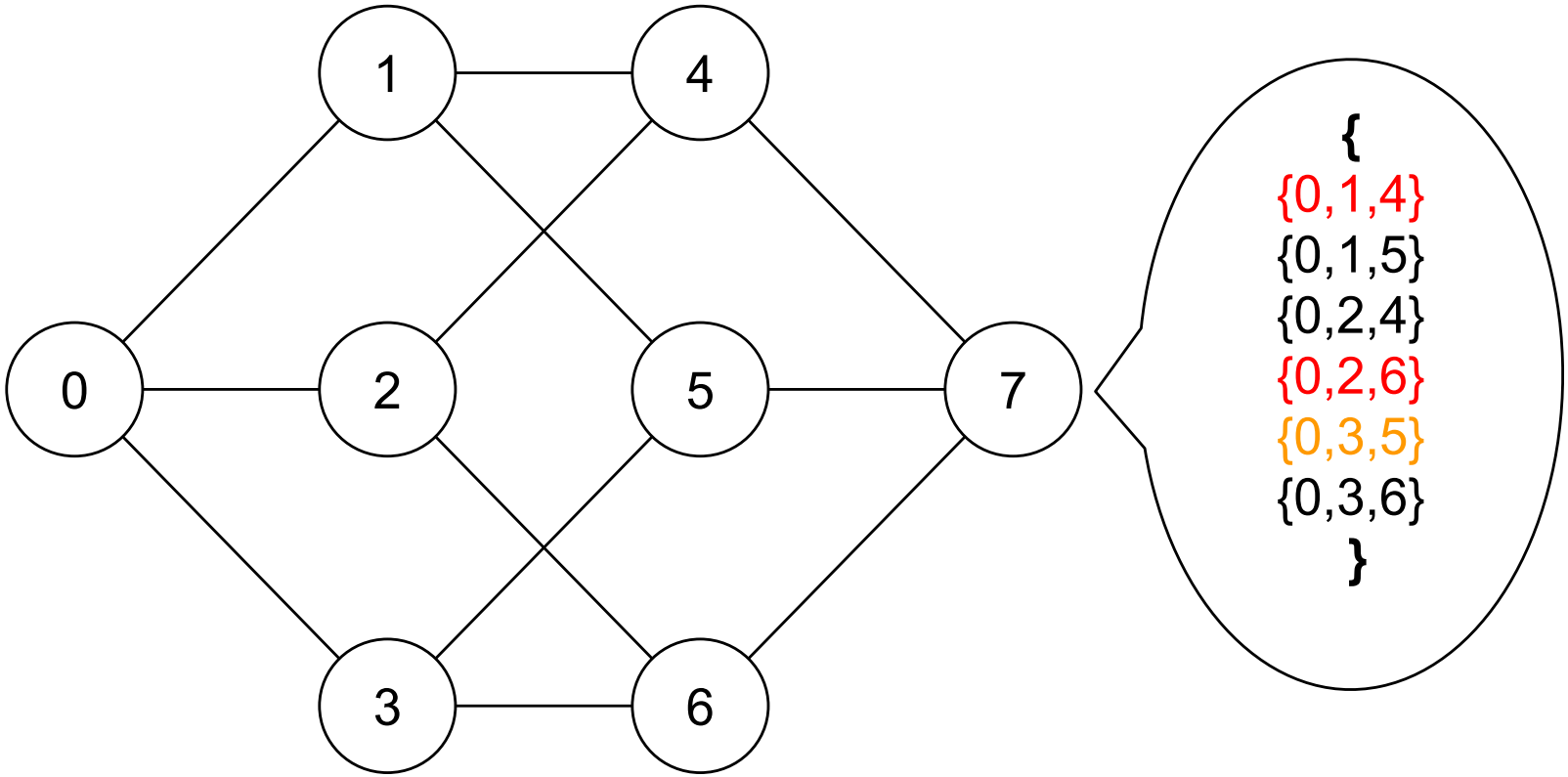
$f = 1$

Dolev Algorithm Graphical Example



$f = 1$

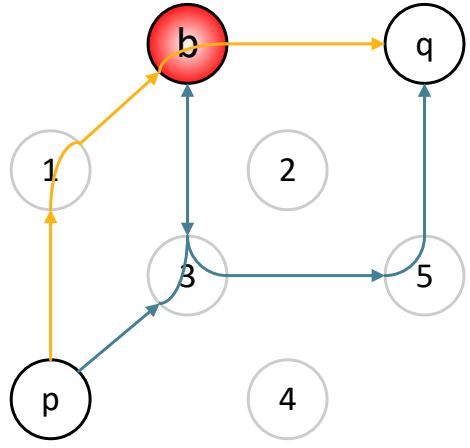
Dolev Algorithm Graphical Example



$f = 1$

Correctness - Safety

All the messages generated by a Byzantine process are labeled with its ID

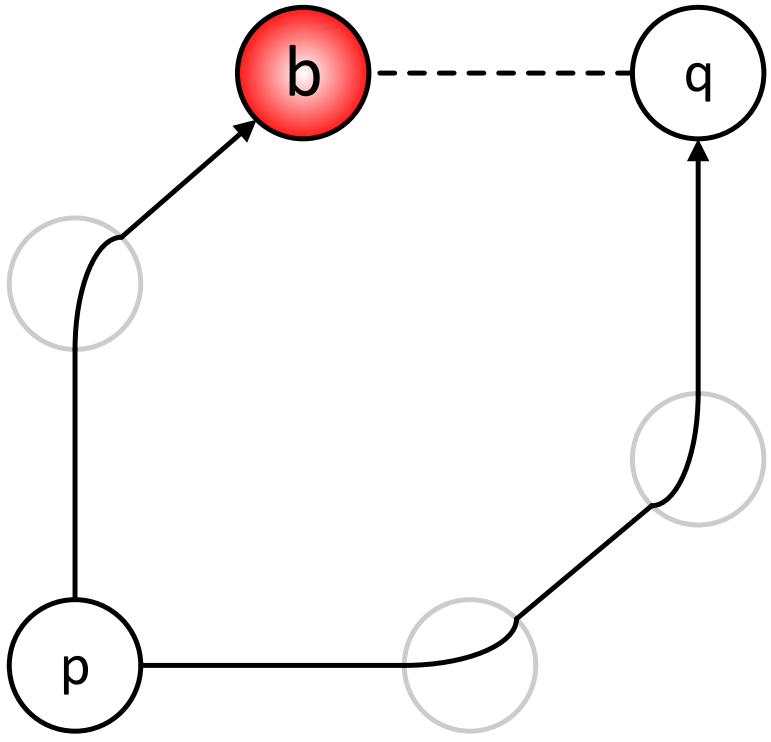


[0,1,b], [p,3,b,5]

$\Longrightarrow$ it is not possible for it to generate Traversed_Processes with minimum cut (or max disjoint) greater than f.

$\Longrightarrow$ Dolev algorithm enforces *safety*

Correctness - Liveness



It depends on the network topology

Correctness Necessary and Sufficient condition [Dol81]

The Byzantine reliable broadcast can be achieved in a static network G composed of n processes considering at most f Byzantine processes ***if and only if the vertex connectivity*** [2] ***of G is at least 2f + 1.***

[2] the vertex connectivity of a graph can be polynomially computed.

Dolev algorithm Analysis

Message Complexity: **Exponential in the number of processes** (considering only correct processes)

Delivery Complexity: Solve an **NP-Complete problem**
(Min-Cut $\Rightarrow$ Minimum Hitting Set,
Maximum Disjoint Paths $\Rightarrow$ Maximum Set Packing)

$\Rightarrow$ **Not practically employable**

Is it possible to do better?

Constraining all processes

More messages a process receives $\Longrightarrow$ Bigger input for our NP-Complete problem

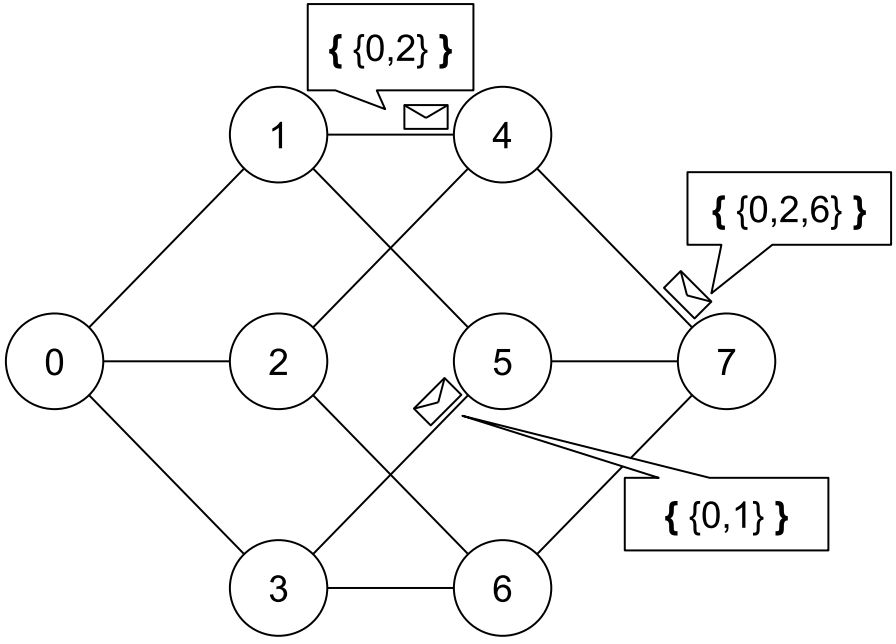
A **Byzantine** process may start flooding the network with lots of messages $\Longrightarrow$ **Denial-of-Service = No Liveness**

Possible solution: restrict the capability of every process

Bounded Channel Capacity: every process can send a bounded number of messages in a time window

Improving Dolev's algorithm

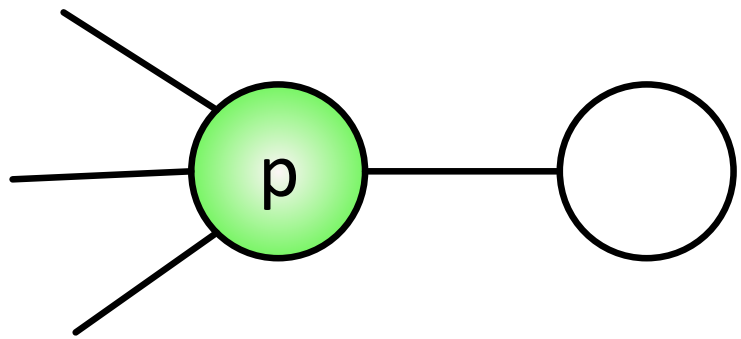
The Dolev algorithm "blindly" relays any message to any process not already included in *Traversed_Processes*



It is possible to save messages

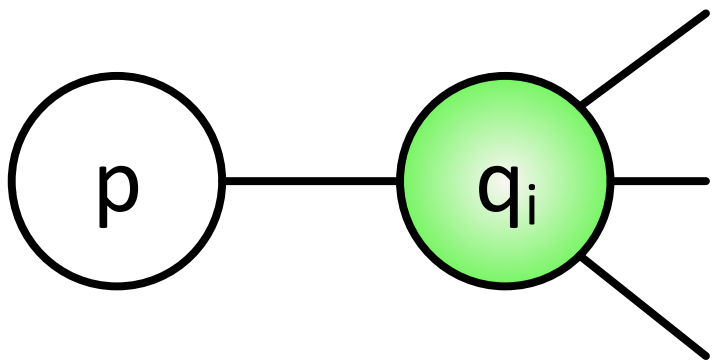
Optimization 1

If a process p has delivered a message *msg*, than p can relay *msg* with an empty *Traversed_Processes*, i.e. $msg := \langle s, m, \emptyset \rangle$ without affecting the safety property.



Optimization 2

A process p has not to relay messages carrying m to processes q_i that have already delivered m.



Optimized Dolev Algorithm Implementation [BFT18]

Optimized Dolev Byzantine Reliable Broadcast

Implements:
ByzantineReliableBroadcast, **instance** *brb*, with sender *s*.
Uses:
AuthPerfectPointToPointLinks, **instance** *al*.
Synchronous System

upon event ⟨ *brb*, *Init* ⟩ **do**
    received := Ø
    delivered := Ø
    neight_del := Ø

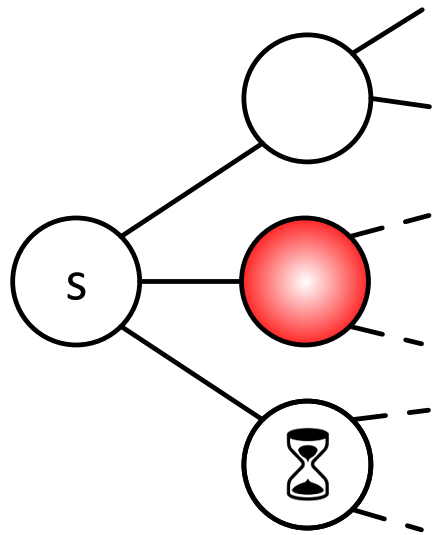
upon event ⟨ *brb*, *Broadcast* | ⟨*p*,*m*⟩ ⟩ **do**
    forall *q* ∈ $Neighbors_p$ **do**
        trigger ⟨ *al*, *Send* | *q*, [⟨*p*,*m*⟩ ,Ø] ⟩;

upon event ⟨ *al*, *Deliver* | $\hat{q}$, [⟨*s*,*m*⟩ , *tp*] ⟩ **do**
    if $\hat{q}$ == *s* **then**
        trigger ⟨ *brb*, *Deliver* | ⟨*s*,*m*⟩ ⟩;
        delivered := *delivered* ∪ ⟨*s*,*m*⟩
        received[⟨*s*,*m*⟩] := *received*[⟨*s*,*m*⟩] ∪ Ø
    else
        if *tp* = Ø **then**
            neight_del[⟨*s*,*m*⟩] := *neight_del*[⟨*s*,*m*⟩] ∪ $\hat{q}$
        new_tp := *tp* ∪ $\hat{q}$
        received[⟨*s*,*m*⟩] := *received*[⟨*s*,*m*⟩] ∪ *new_tp*
        if ⟨*s*,*m*⟩ ∉ *delivered* **and** ***check_disjoint_paths***(*f*+1, *received*[⟨*s*,*m*⟩]) == *True* **then**
            trigger ⟨ *brb*, *Deliver* | ⟨*s*,*m*⟩ ⟩;
            delivered := *delivered* ∪ ⟨*s*,*m*⟩
            received[⟨*s*,*m*⟩].***clear()***
            received[⟨*s*,*m*⟩] := *received*[⟨*s*,*m*⟩] ∪ Ø

upon exists ⟨*s*,*m*⟩ **in** *received* **do**
    [*s*,*m*,*tp*] := *received*[⟨*s*,*m*⟩].***random_pop()***
    forall *q* ∈ $Neighbors_p$ - *neight_del*[⟨*s*,*m*⟩] **such that** *q* ∉ *tp* **and** *q* != *s* **do**
        trigger ⟨ *al*, *Send* | *q*, [⟨*s*,*m*⟩ , *tp*] ⟩;

Are those optimizations effective?

On **asynchronous systems**, in the worst case scenario, the **message complexity** is still **exponential**.

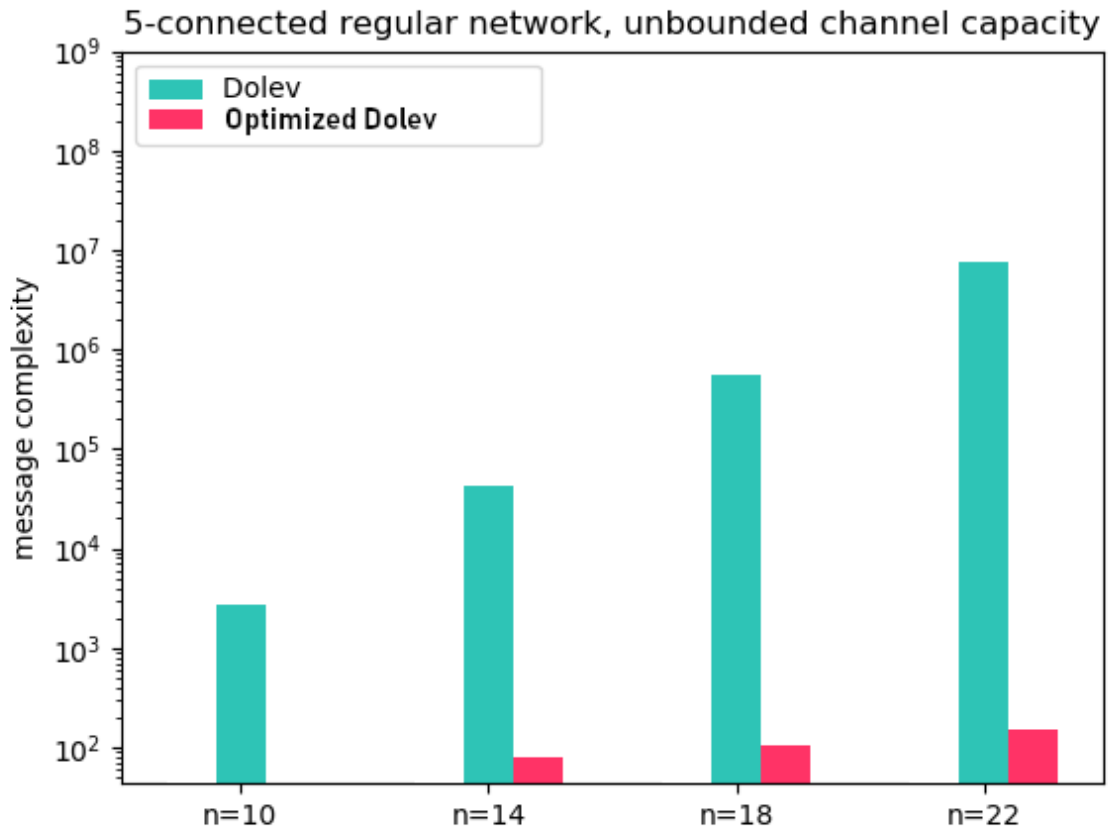


$f = 1$

hourglass by Viktor Fedyuk (Tim P) from the Noun Project

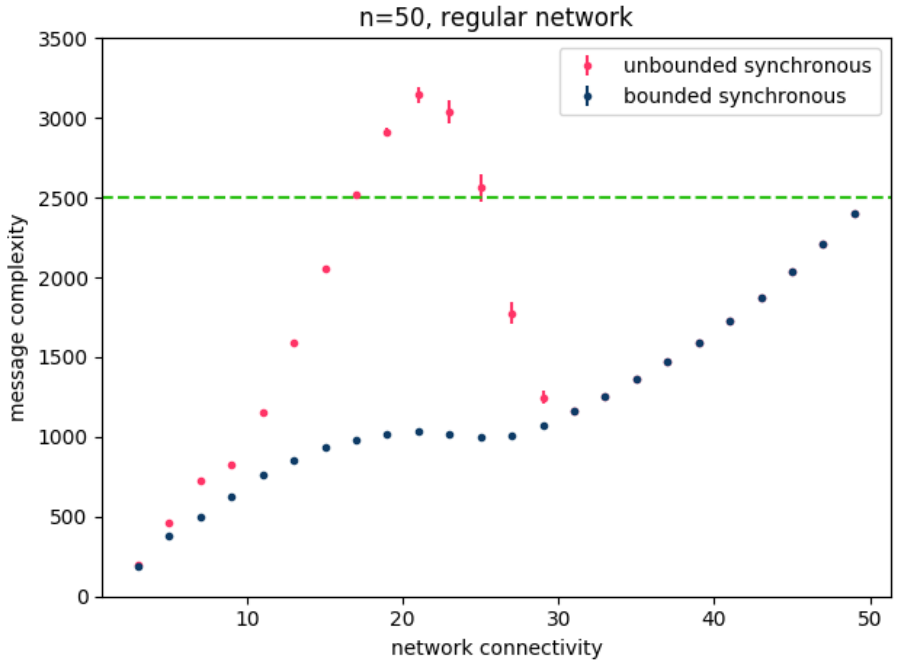
Synchronous System

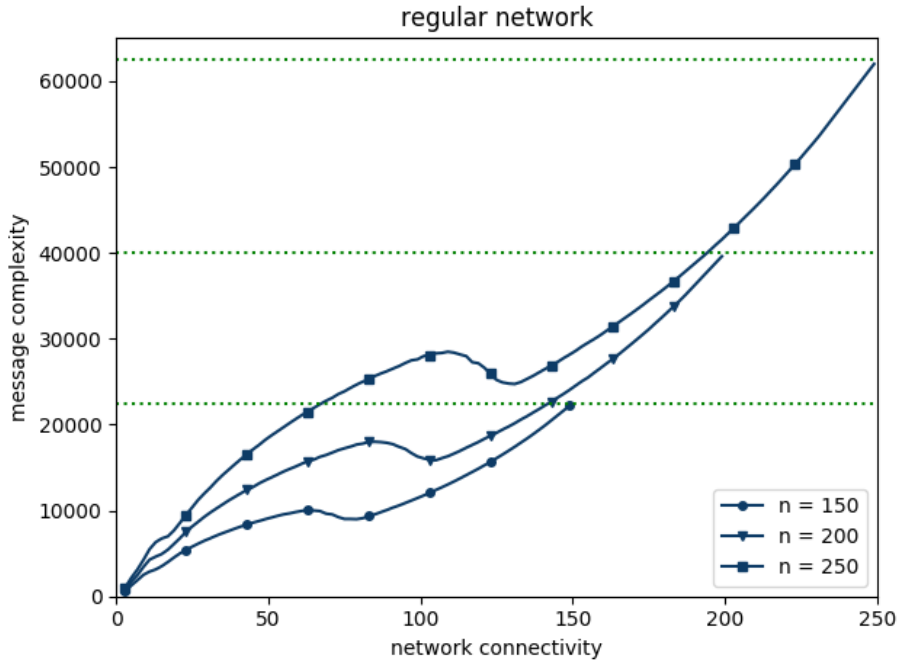
In **Synchronous Systems** with specific topologies (k-regular networks) it has been show that they are **very effective**



Stronger Assumptions,

Synchronous System





Quadratic Message Complexity in the number of processes

Routed Network

Routed Network: **fixed routes between every pair of processes**

The source broadcasts a message along $2f + 1$ disjoint routes. Any other process relays a message only if it respects the planned routes.

Correctness Necessary and Sufficient condition: $2f + 1$ vertex connected network.

$\Longrightarrow$ **Quadratic Message Complexity** (every edge is traversed once)

$\Longrightarrow$ **Linear Delivery Complexity** (counting the copies of a message)

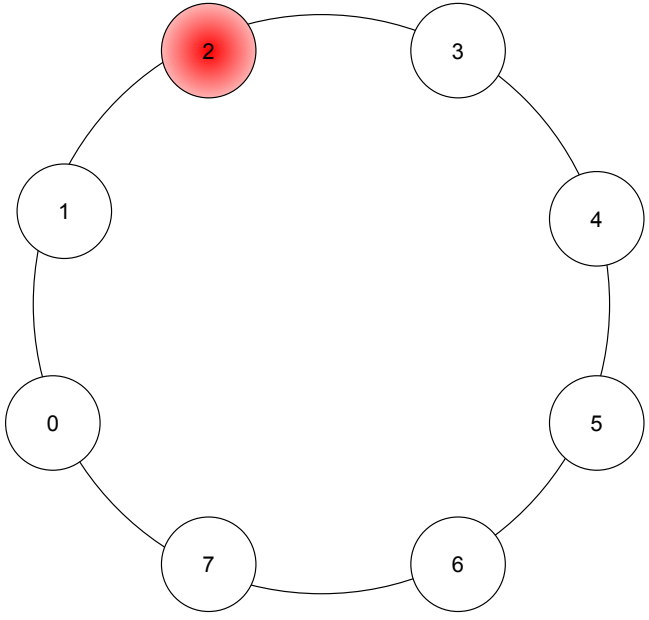
Digital Signatures

The source digitally signs a message to broadcast

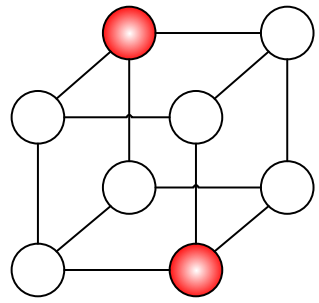
Any correct process just relays the received messages

Correctness Necessary and Sufficient condition: **f+1 vertex connected network**.

The third party providing the cryptographic keys is a **single point of failure**.



Locally Bounded Failure Model



f=1

System Model

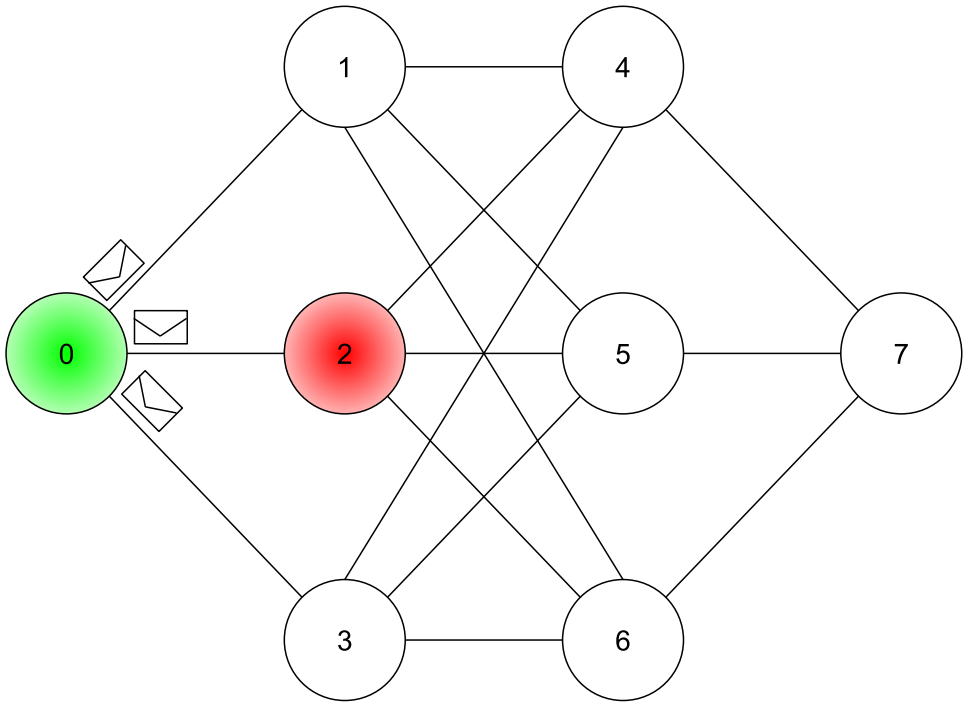
- n processes (each one with an unique identifier);
- not complete communication network;
- processes can be correct or Byzantine faulty;
- up to f processes can be faulty in the neighborhood of every process (*locally bounded failures*);
- processes have no global knowledge (except the value of f);
- Authenticated Perfect channels

CPA Algorithm [PP05]

Idea: leverage the authenticated channels + at most f failures in every neighborhood $\Longrightarrow$ a process **waits for $f+1$ copies of the same message** to deliver.

Message Format

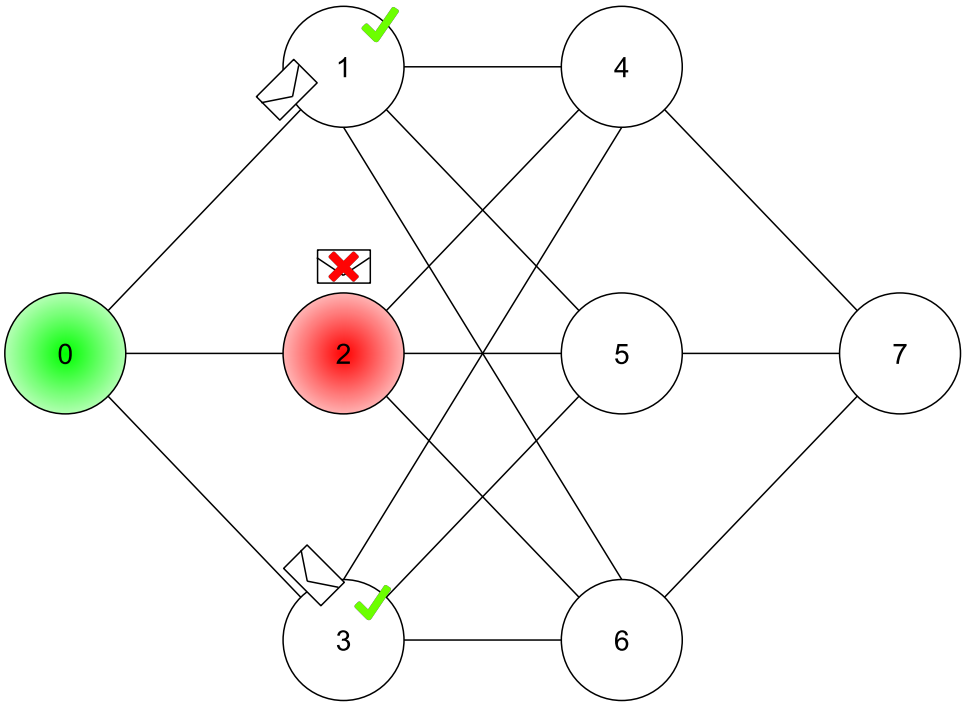
CPA Algorithm [PP05]



f = 1

- **the source broadcasts the message**;
- a neighbor of the source directly accepts and relays the message;
- a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message.

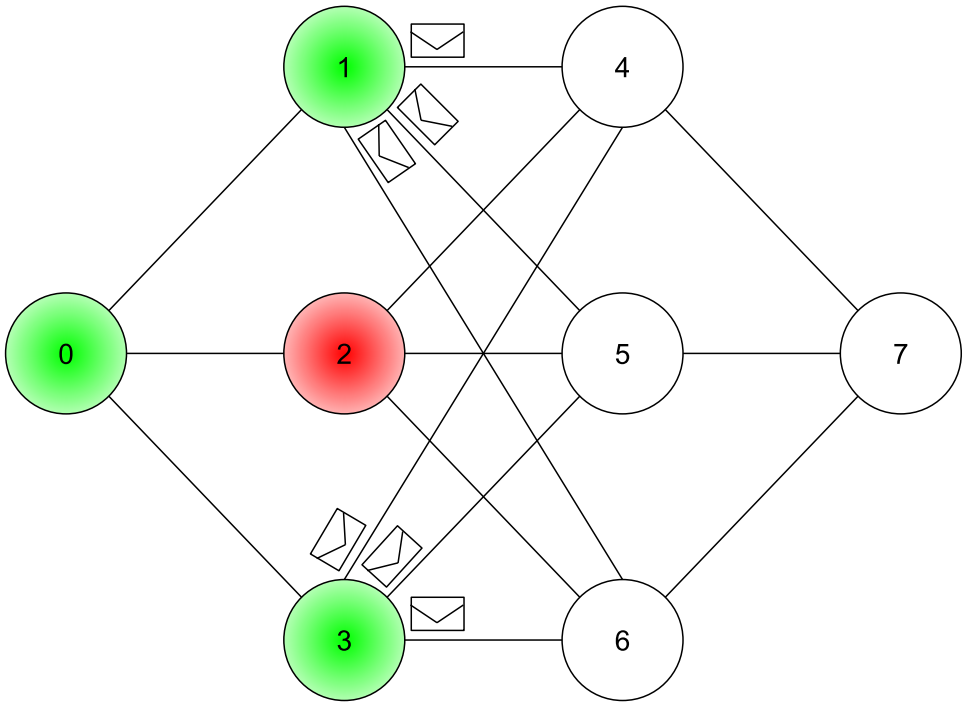
CPA Algorithm [PP05]



f = 1

- the source broadcasts the message;
- **a neighbor of the source directly accepts and relays the message**;
- a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message.

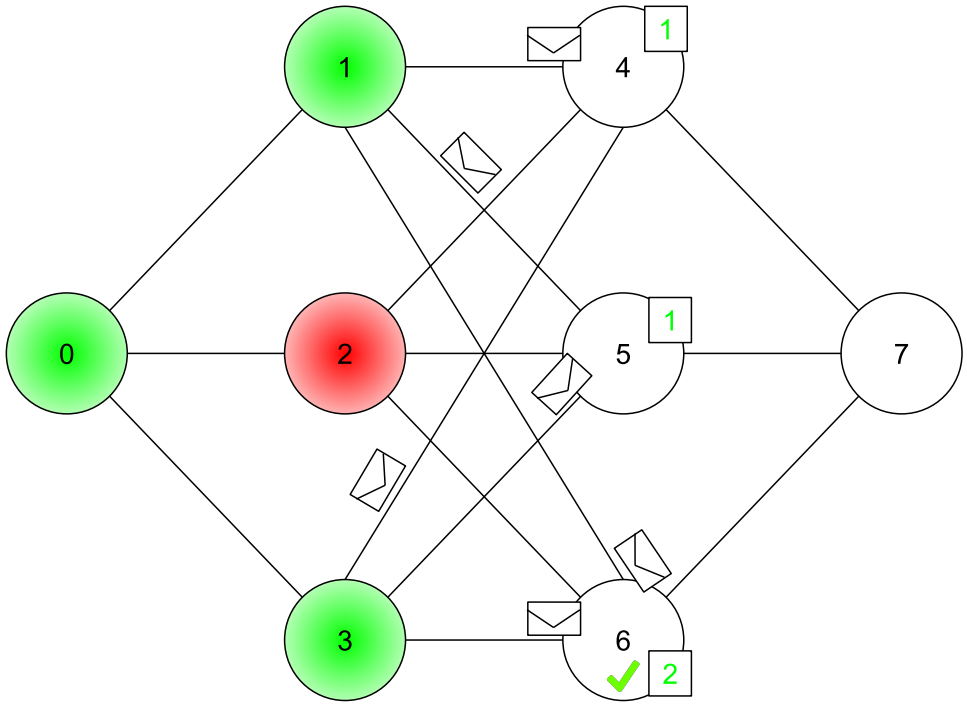
CPA Algorithm [PP05]



f = 1

- the source broadcasts the message;
- **a neighbor of the source directly accepts and relays the message**;
- a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message.

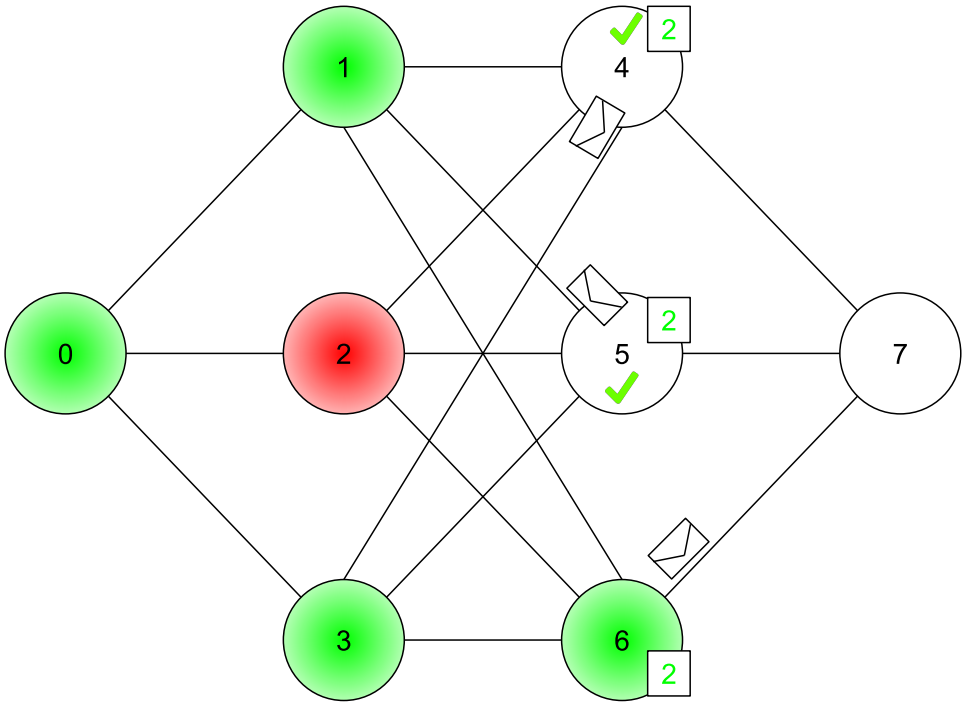
CPA Algorithm [PP05]



f = 1

- the source broadcasts the message;
- a neighbor of the source directly accepts and relays the message;
- **a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message**.

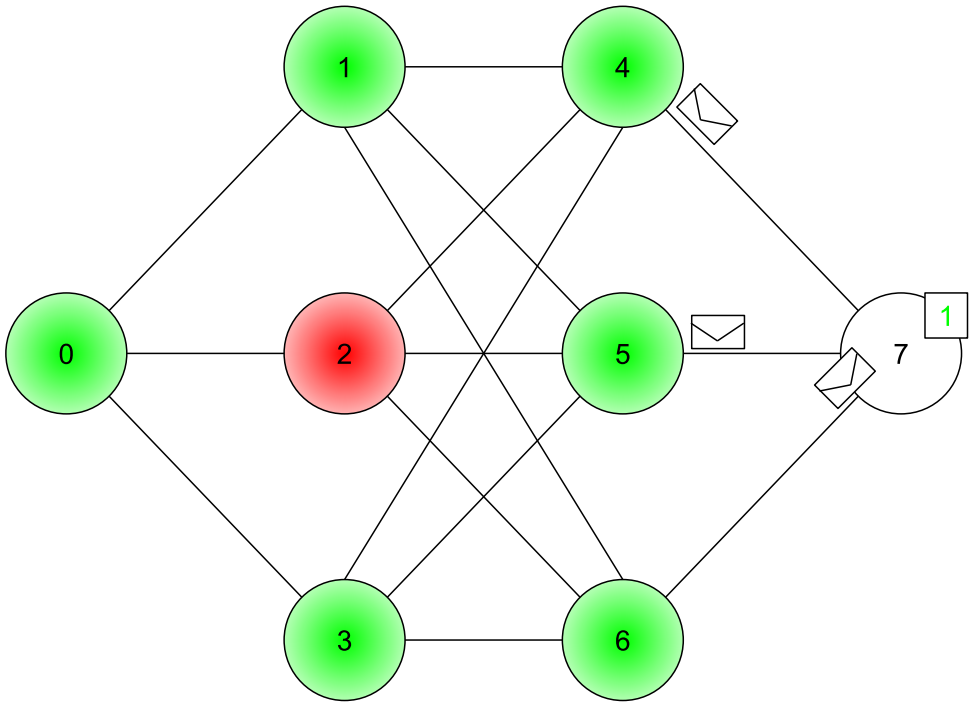
CPA Algorithm [PP05]



f = 1

- the source broadcasts the message;
- a neighbor of the source directly accepts and relays the message;
- **a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message**.

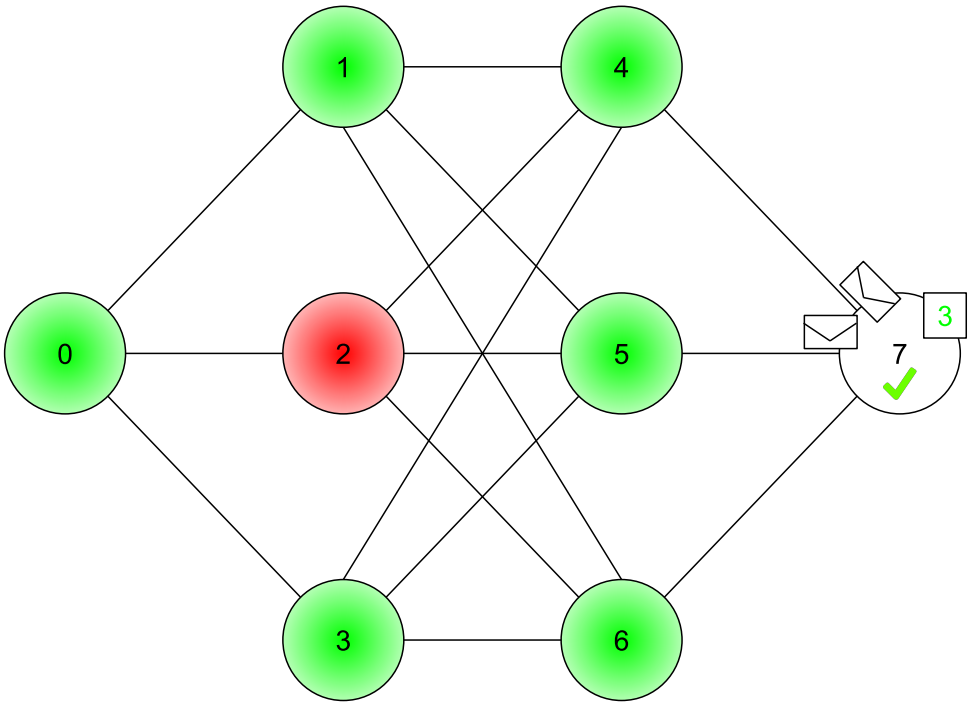
CPA Algorithm [PP05]



f = 1

- the source broadcasts the message;
- a neighbor of the source directly accepts and relays the message;
- **a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message**.

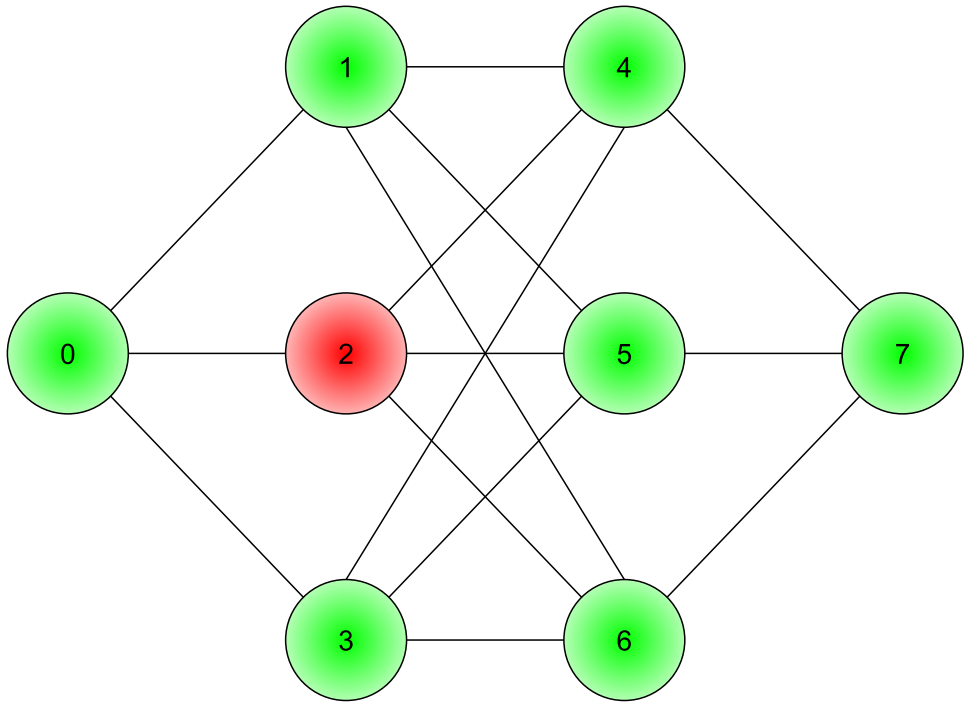
CPA Algorithm [PP05]



f = 1

- the source broadcasts the message;
- a neighbor of the source directly accepts and relays the message;
- **a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message**.

CPA Algorithm [PP05]



f = 1

- the source broadcasts the message;
- a neighbor of the source directly accepts and relays the message;
- **a process that receives the same message from $f + 1$ distinct neighbors accepts and relays the message**.

CPA Algorithm Implementation

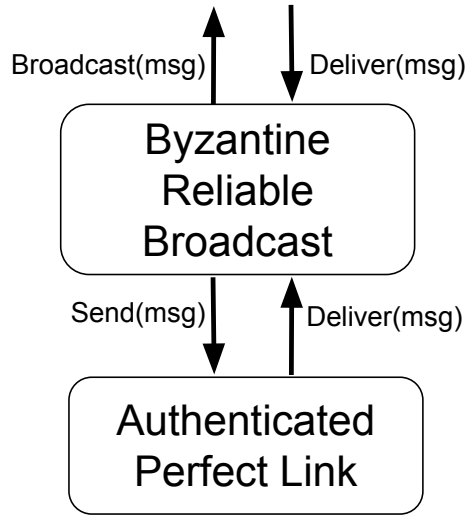
Certified Propagation Algorithm (CPA) Byzantine Reliable Broadcast

Implements:
ByzantineReliableBroadcast, **instance** *brb*, with sender *s*.
Uses:
AuthPerfectPointToPointLinks, **instance** *al*.
Asynchronous System

upon event $\langle\, brb,\ Init \,\rangle$ **do**
received := Ø
delivered := Ø

upon event $\langle\, brb,\ Broadcast \mid \langle p, m \rangle \,\rangle$ **do**
forall $q \in Neighbors_p$ **do**
trigger $\langle\, al,\ Send \mid q,\ \langle p, m \rangle \,\rangle$;

upon event $\langle\, al,\ Deliver \mid \hat{q},\ \langle s, m \rangle \,\rangle$ **such that** $\langle s,m \rangle \notin delivered$ **do**
if $\hat{q} == s$ **then**
trigger $\langle\, brb,\ Deliver \mid \langle s, m \rangle \,\rangle$;
delivered := *delivered* ∪ $\langle s,m \rangle$
else
received[$\langle s,m \rangle$] := *received*[$\langle s,m \rangle$] ∪ *p*
if |*received*[$\langle s,m \rangle$]| > *f* **then**
trigger $\langle\, brb,\ Deliver \mid \langle s, m \rangle \,\rangle$;
delivered := *delivered* ∪ $\langle s,m \rangle$
forall $q \in Neighbors_p$ **do**
trigger $\langle\, al,\ Send \mid q,\ \langle s, m \rangle \,\rangle$;



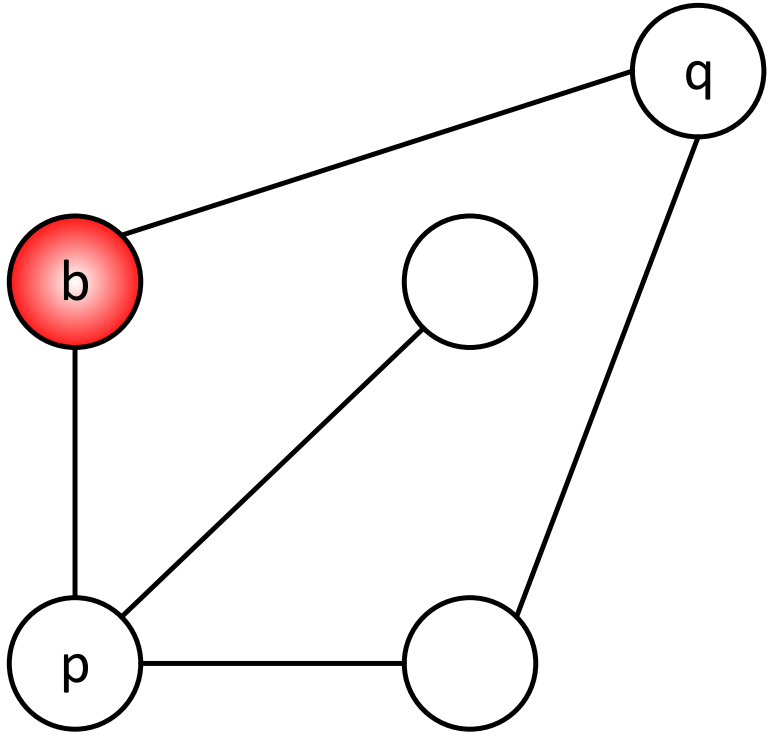
Correctness - Safety

Every process relays a message only if it has been delivered

At most f faulty process are present in the neighborhood

$\Longrightarrow$ CPA algorithm enforces safety

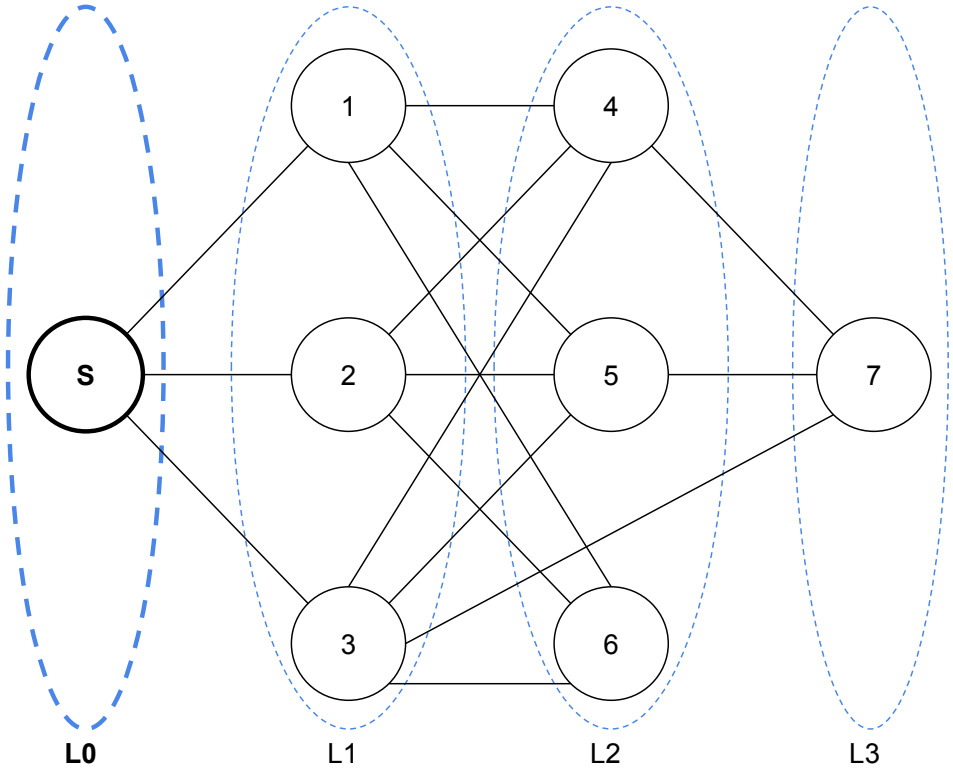
Correctness - Liveness



It depends on the network topology

Minimum K Level Ordering (MKLO) [LPS13]

MKLO = Partition of the nodes in levels

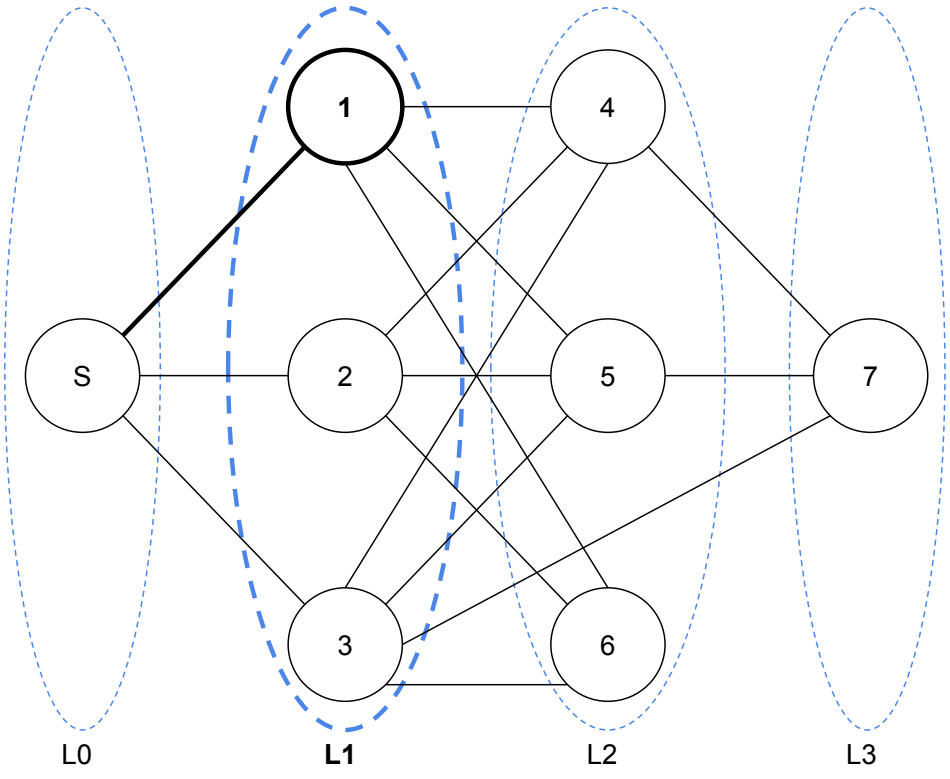


k = 3

- **The source is placed in** $L0$;

Minimum K Level Ordering (MKLO) [LPS13]

MKLO = Partition of the nodes in levels

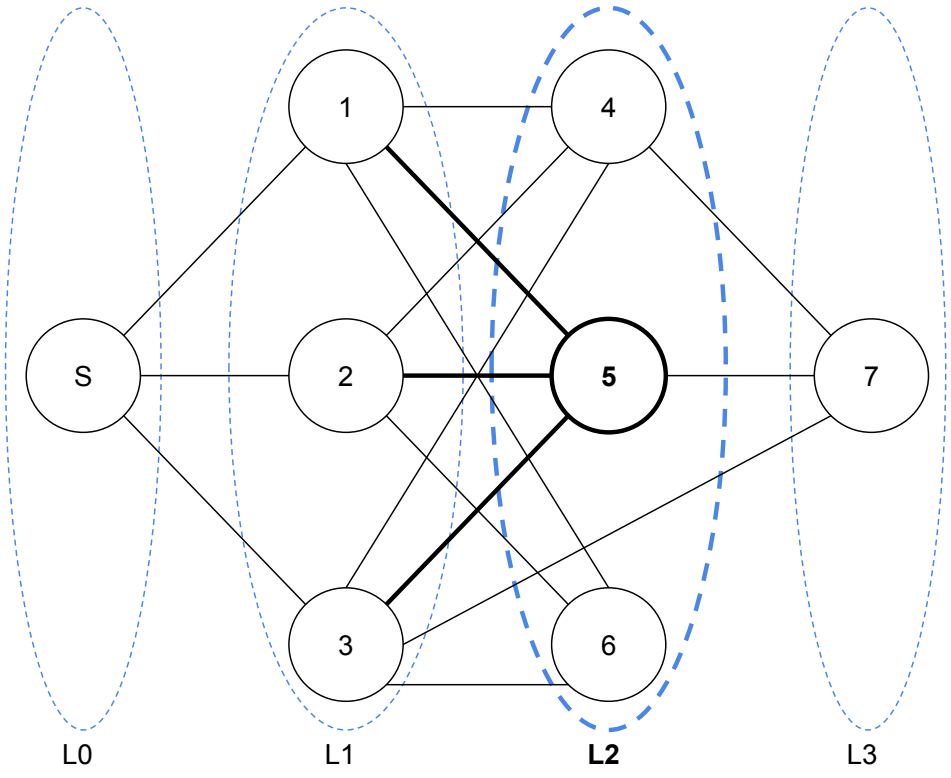


k = 3

- The source is placed in $L0$;
- **The neighbors of the source are placed in level** $L1$;

Minimum K Level Ordering (MKLO) [LPS13]

MKLO = Partition of the nodes in levels

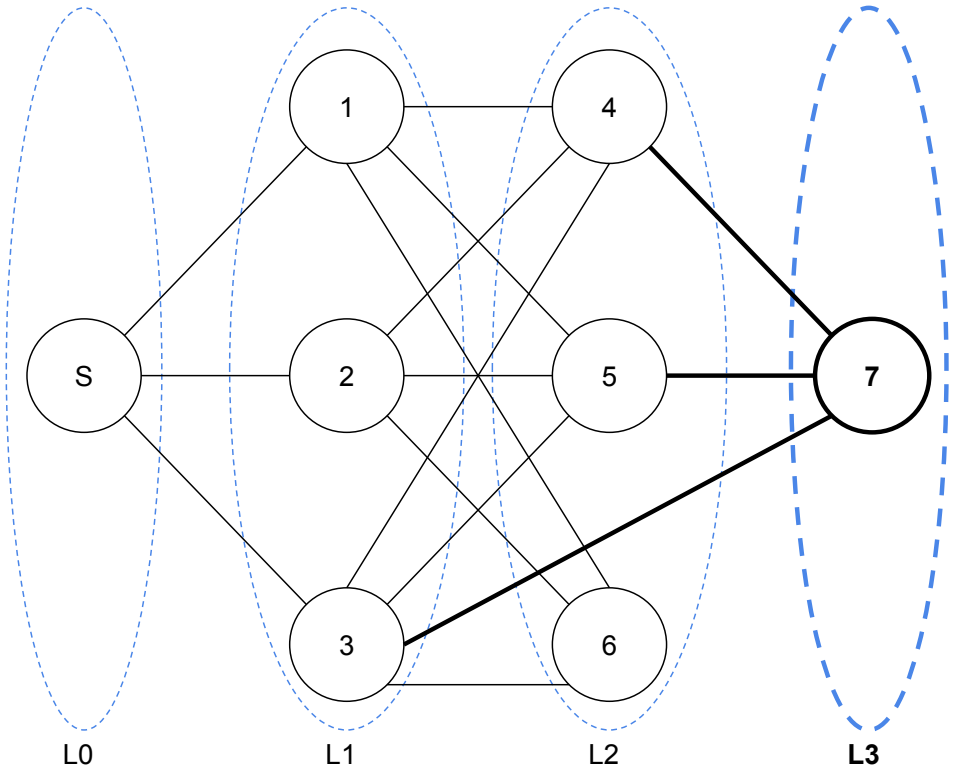


k = 3

- The source is placed in $L0$;
- The neighbors of the source are placed in level $L1$;
- **Any other node is places in the first level such that it has at least k neighbors in the previous levels.**

Minimum K Level Ordering (MKLO) [LPS13]

MKLO = Partition of the nodes in levels



k = 3

- The source is placed in $L0$;
- The neighbors of the source are placed in level $L1$;
- **Any other node is places in the first level such that it has at least k neighbors in the previous levels.**

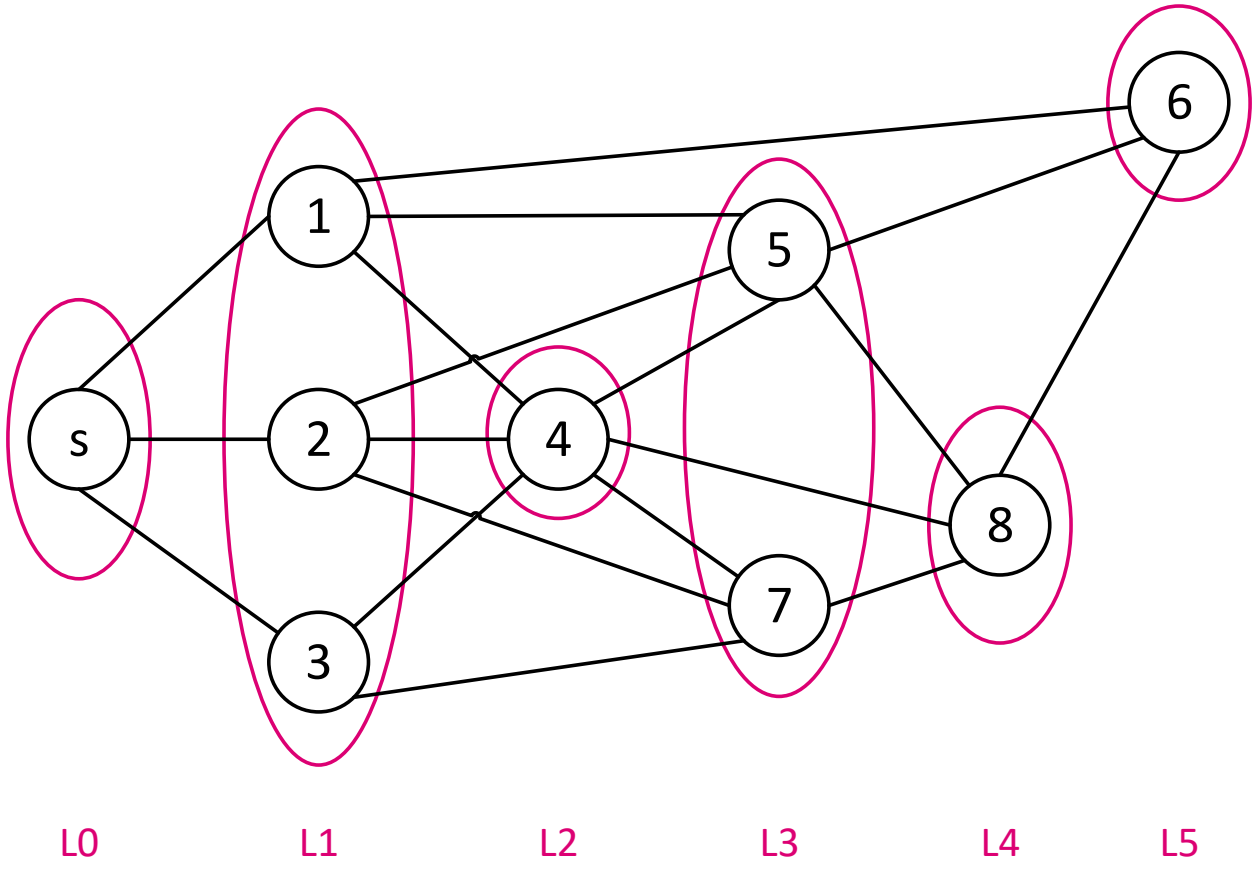
Correctness Necessary and Sufficient condition [LPS13]

Necessary condition: MKLO with $k = f+1$

Sufficient condition: MKLO with $k = 2f+1$

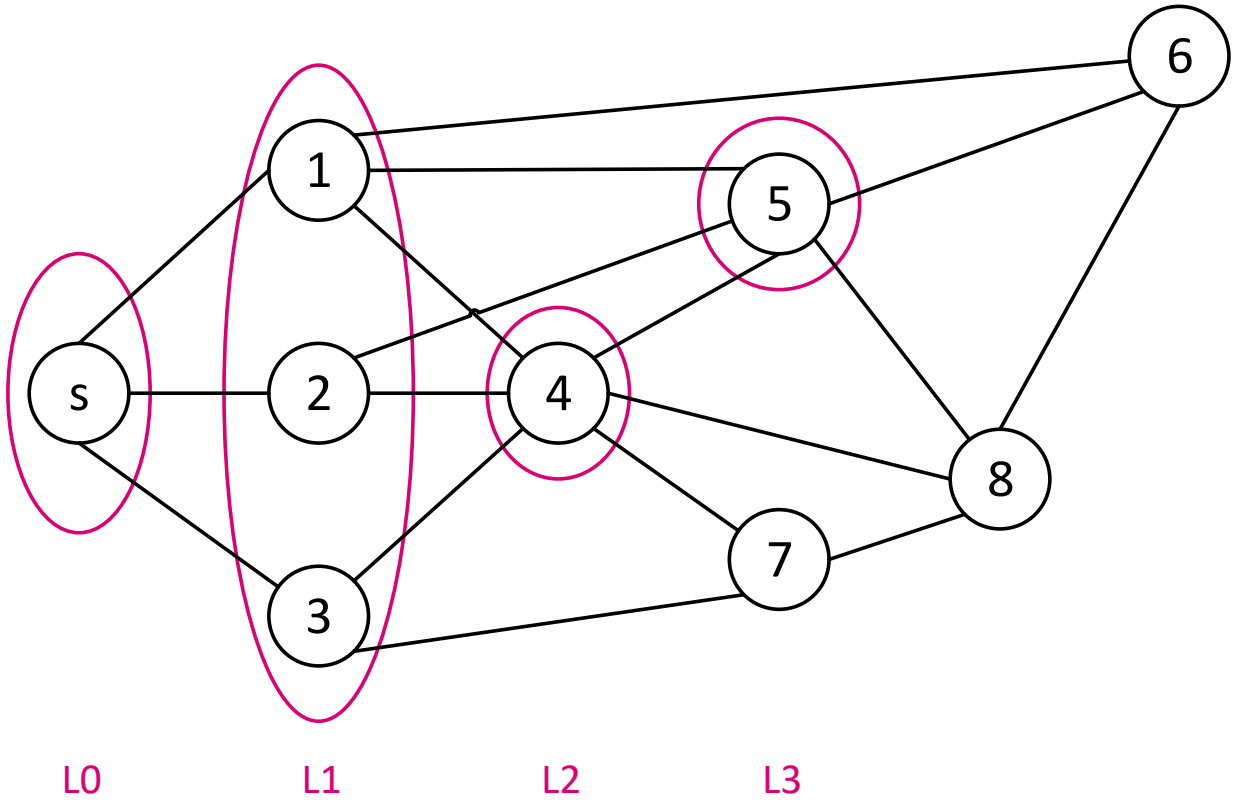
Strict condition: MKLO with $k = f+1$ removing any possible placement of the Byzantine processes (NP-Complete Problem)

MKLO Examples



MKLO with K=3

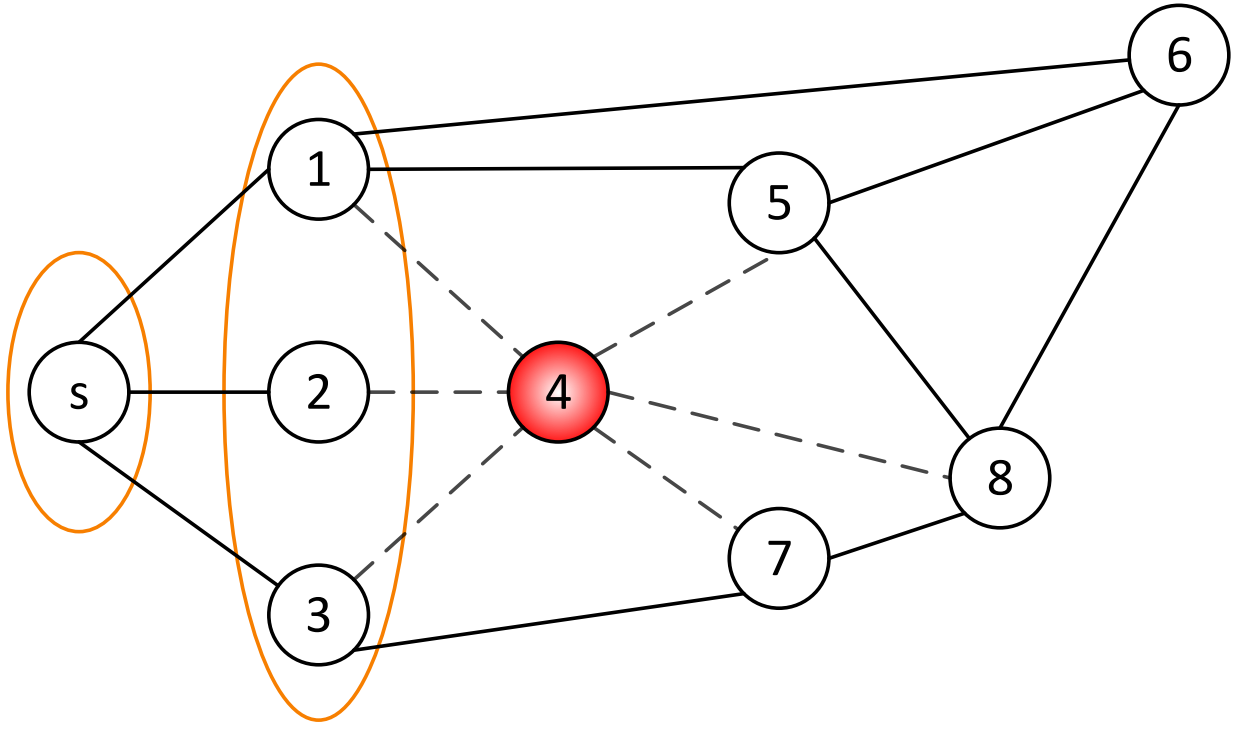
MKLO Examples



MKLO with K=3 is not defined

MKLO with K=2 is defined for every 1-local removal

MKLO Examples



MKLO with K=2 is not defined removing node 4

CPA Algorithm Analysis

Message Complexity: **Quadratic** (every edge is traversed once)

Delivery Complexity: **Linear** (counting the copies of a message)

Knowledge on Topology [PPS17]

If the processes knows the network topology they are able to **tolerate more failures**

BUT

the delivery complexity becomes **exponential**

Sparse Multi-Hop Networks

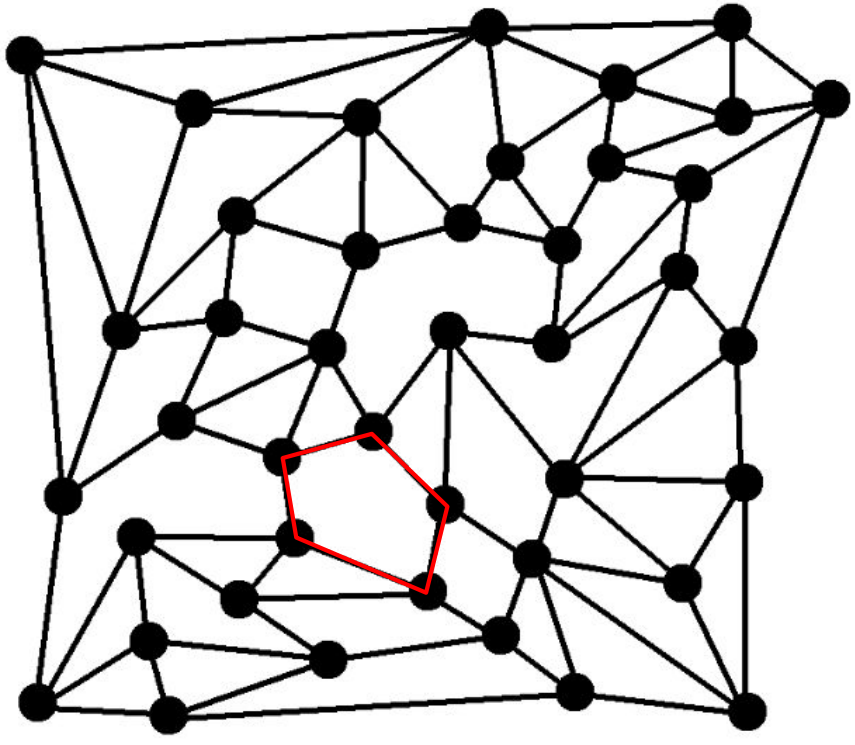
The correctness conditions defined for globally bounded and locally bounded failures require quite **dense graphs**.

Those **correctness conditions** are **strict** $\Longrightarrow$ it is not possible to solve Byzantine Reliable Broadcast if they are not satisfied.

What about **sparse** graphs?

- Consider more constrained assumptions on failures (a.e. constrained placement, distance etc.)
- Consider weaker safety property (namely part of nodes may deliver invalid messages)
- Consider weaker liveness property (namely part of nodes may not deliver valid messages)

Byzantine Reliable Broadcast in Planar Networks [MT13]



4-connected planar graph

Planar graph := $\exists$ bi-dimensional representation where the edges do not cross

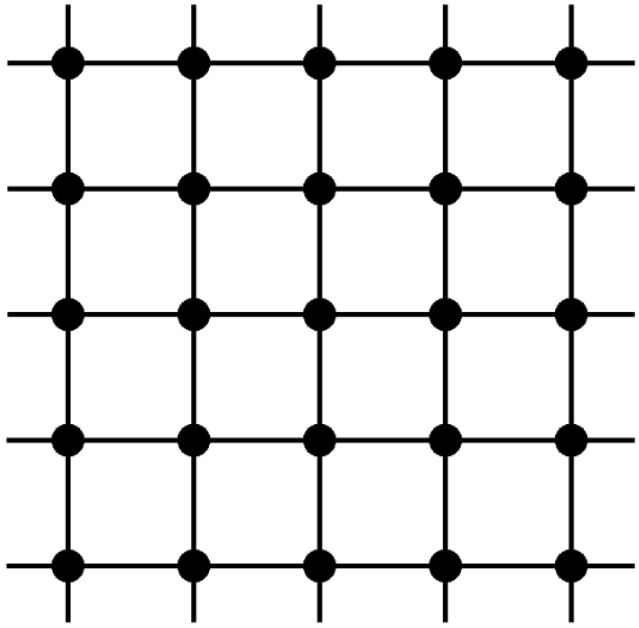
Polygon

Z := max number of edges per polygon

D := min number of nodes between two Byzantine nodes

Byzantine Reliable Broadcast in Planar Networks

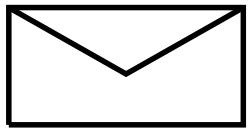
It is possible to achieve Byzantine Reliable Broadcast in a 4-connected planar graph if and only if $D > Z$.



2D torus

Reliable Broadcast on torus when $D > 4$

Byzantine Reliable Broadcast in Planar Networks - Implementation



Message format

- every process saves in $Rec(q)$ the last message received from a neighbors q;
- **the source** s multicasts an information m;
- **the neighbors of the source** wait until they receive m from s, then deliver m and multicast $\langle s, m, \emptyset \rangle$;
- **when** $\langle s, m, S \rangle$ is received from a neighbor q, with $q \notin S$ and $|S| \leq Z - 3$ **then** $Rec(q) := \langle s, m, S \rangle$ and multicast $\langle s, m, S \cup \{q\} \rangle$.
- **when** $\exists\ m, p, q, S$ such that $q \neq p, q \notin S, Rec(q) = \langle s, m, \emptyset \rangle$ and $Rec(p) = \langle s, m, S \rangle$ **then** deliver m, multicast $\langle s, m, \emptyset \rangle$ and stop.

Byzantine Reliable Broadcast in Planar Networks - Analysis

Every process keeps a copy of the last message received from every neighbors $\Longrightarrow$ **linear memory** required on every process

At most 1 Byzantine arbitrarily placed

OR

Spatial condition $D > 4$

References I

[BFT18] Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil.
Multi-hop byzantine reliable broadcast made practical.
In *Dependable Computing (LADC), 2018 Eighth Latin-American Symposium on*, 2018.
URL: `https://hal.archives-ouvertes.fr/hal-01826865`.

[Dol81] Danny Dolev.
Unanimity in an unknown and unreliable environment.
In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 159–168. IEEE Computer Society, 1981.
URL: `https://doi.org/10.1109/SFCS.1981.53`.

[LPS13] Chris Litsas, Aris Pagourtzis, and Dimitris Sakavalas.
A graph parameter that matches the resilience of the certified propagation algorithm.
In Jacek Cichon, Maciej Gebala, and Marek Klonowski, editors, *Ad-hoc, Mobile, and Wireless Network - 12th International Conference, ADHOC-NOW 2013, Wrocław, Poland, July 8-10, 2013. Proceedings*, volume 7960 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2013.
URL: `https://doi.org/10.1007/978-3-642-39247-4_23`.

References II

[MT13] Alexandre Maurer and Sébastien Tixeuil.
On byzantine broadcast in planar graphs.
CoRR, abs/1301.2875, 2013.
URL: http://arxiv.org/abs/1301.2875, arXiv:1301.2875.

[PP05] Andrzej Pelc and David Peleg.
Broadcasting with locally bounded byzantine faults.
Inf. Process. Lett., 93(3):109–115, 2005.
URL: https://doi.org/10.1016/j.ipl.2004.10.007.

[PPS17] Aris Pagourtzis, Giorgos Panagiotakos, and Dimitris Sakavalas.
Reliable broadcast with respect to topology knowledge.
Distributed Computing, 30(2):87–102, 2017.
URL: https://doi.org/10.1007/s00446-016-0279-6.