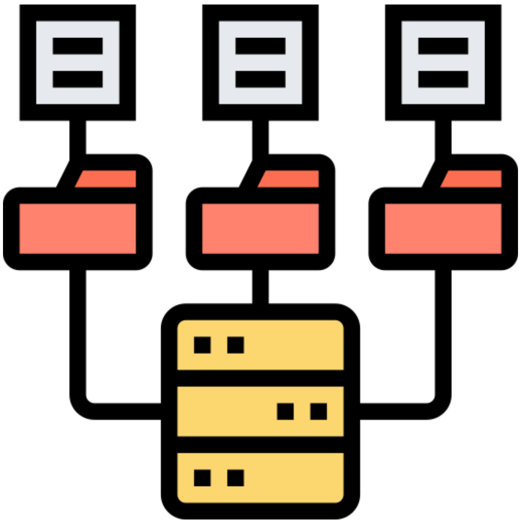


# Dependable Distributed Systems

## Professor: S. Bonomi

A.Y. 2022/23

## Table of Contents

## List of Figures

# List of Tables

# 1 Modeling Distributed Systems

## 1.1 Modeling Processes and their interactions

## 1.2 Specification in terms of Safety and Liveness Property

## 1.3 Modeling Failures

## 1.4 Timing Assumptions

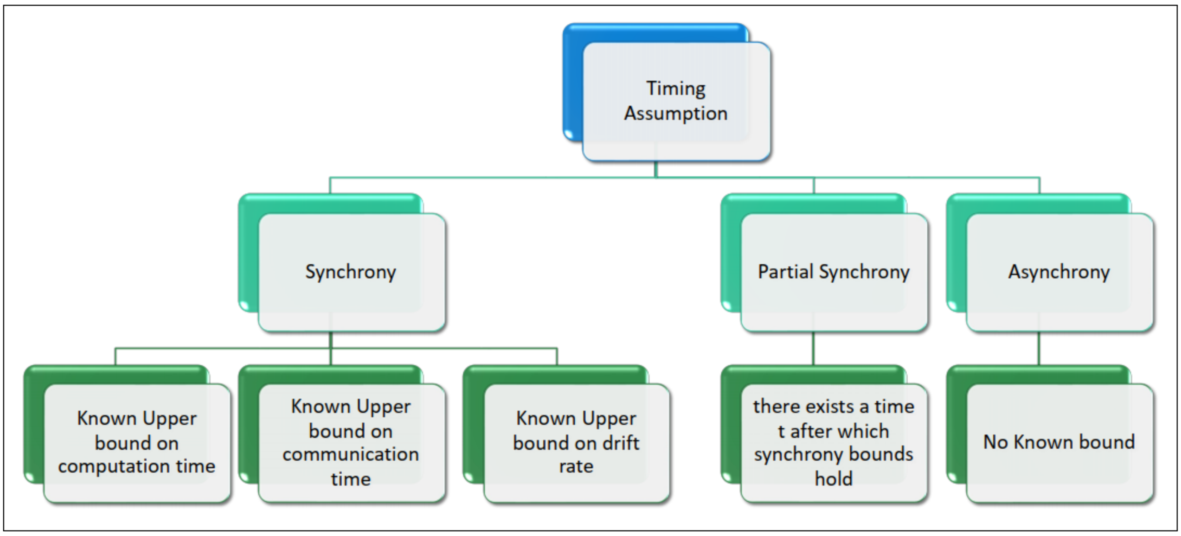

Timing Assumption
Synchrony
Partial Synchrony
Asynchrony
Known Upper bound on computation time
Known Upper bound on communication time
Known Upper bound on drift rate
there exists a time t after which synchrony bounds hold
No Known bound

Figure 1: Summary on Timing Assumptions

## 1.5 Abstracting Communication

The abstraction of a *link* is used to represent the network components of the distributed system.

Every pair of processes is connected by a bidirectional link, a topology that provides full connectivity among the processes.

Concrete examples of such architectures are illustrated in (Figure 2) include the use of (a) a fully connected mesh, (b) a broadcast medium, (c) a ring, (d) a mesh of links interconnected with bridges

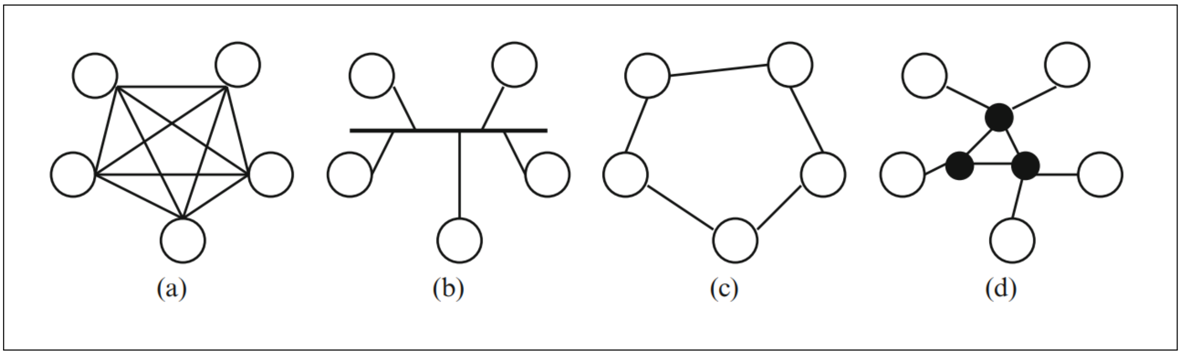


Figure 2: The link abstraction and different instances

### 1.5.1 Abstracting Link Failures

Here we will introduce the following link abstractions considering processes faults:

- Fair-Loss Links
- Stubborn Links
- Perfect Links
- Logged Perfect Links
- Authenticated Perfect Links

### 1.5.2 Fair-Loss Links

The **weakest** variant of the link abstraction.

**Module 2.1:** Interface and properties of fair-loss point-to-point links

**Module:**

**Name:** FairLossPointToPointLinks, **instance** *fll*.

**Events:**

**Request:** ⟨ *fll*, *Send* | $q$, $m$ ⟩: Requests to send message $m$ to process $q$.

**Indication:** ⟨ *fll*, *Deliver* | $p$, $m$ ⟩: Delivers message $m$ sent by process $p$.

**Properties:**

**FLL1:** *Fair-loss:* If a correct process $p$ infinitely often sends a message $m$ to a correct process $q$, then $q$ delivers $m$ an infinite number of times.

**FLL2:** *Finite duplication:* If a correct process $p$ sends a message $m$ a finite number of times to process $q$, then $m$ cannot be delivered an infinite number of times by $q$.

**FLL3:** *No creation:* If some process $q$ delivers a message $m$ with sender $p$, then $m$ was previously sent to $q$ by process $p$.

Figure 3: Interface of fair-loss point-to-point links

### 1.5.3 Stubborn Links

The stubborn delivery property causes every message sent over the link to be delivered at the receiver an unbounded number of times.

**Module 2.2:** Interface and properties of stubborn point-to-point links

**Module:**

**Name:** StubbornPointToPointLinks, **instance** *sl*.

**Events:**

**Request:** ⟨ *sl*, *Send* | $q$, $m$ ⟩: Requests to send message $m$ to process $q$.

**Indication:** ⟨ *sl*, *Deliver* | $p$, $m$ ⟩: Delivers message $m$ sent by process $p$.

**Properties:**

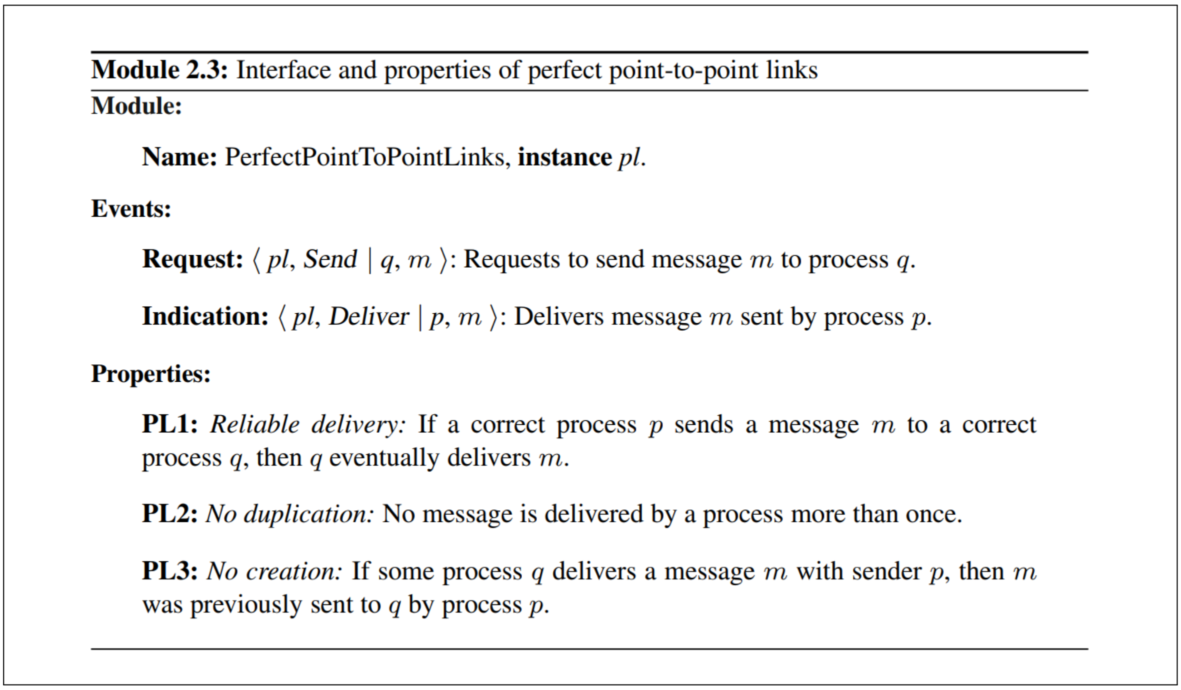
**SL1:** *Stubborn delivery:* If a correct process $p$ sends a message $m$ once to a correct process $q$, then $q$ delivers $m$ an infinite number of times.

**SL2:** *No creation:* If some process $q$ delivers a message $m$ with sender $p$, then $m$ was previously sent to $q$ by process $p$.

Figure 4: Interface of stubborn point-to-point links

### 1.5.4 Perfect Links

With the stubborn links abstraction, it is up to the target process to check whether a given message has already been delivered or not. Adding mechanisms for detecting and suppressing message duplicates, in addition to mechanisms for message retransmission, allows us to build an even higher-level primitive: the **perfect links** abstraction, sometimes also called the *reliable links* abstraction.



**Module 2.3:** Interface and properties of perfect point-to-point links

**Module:**

**Name:** PerfectPointToPointLinks, **instance** *pl*.

**Events:**

**Request:** $\langle$ *pl*, *Send* | $q$, $m$ $\rangle$: Requests to send message $m$ to process $q$.

**Indication:** $\langle$ *pl*, *Deliver* | $p$, $m$ $\rangle$: Delivers message $m$ sent by process $p$.

**Properties:**

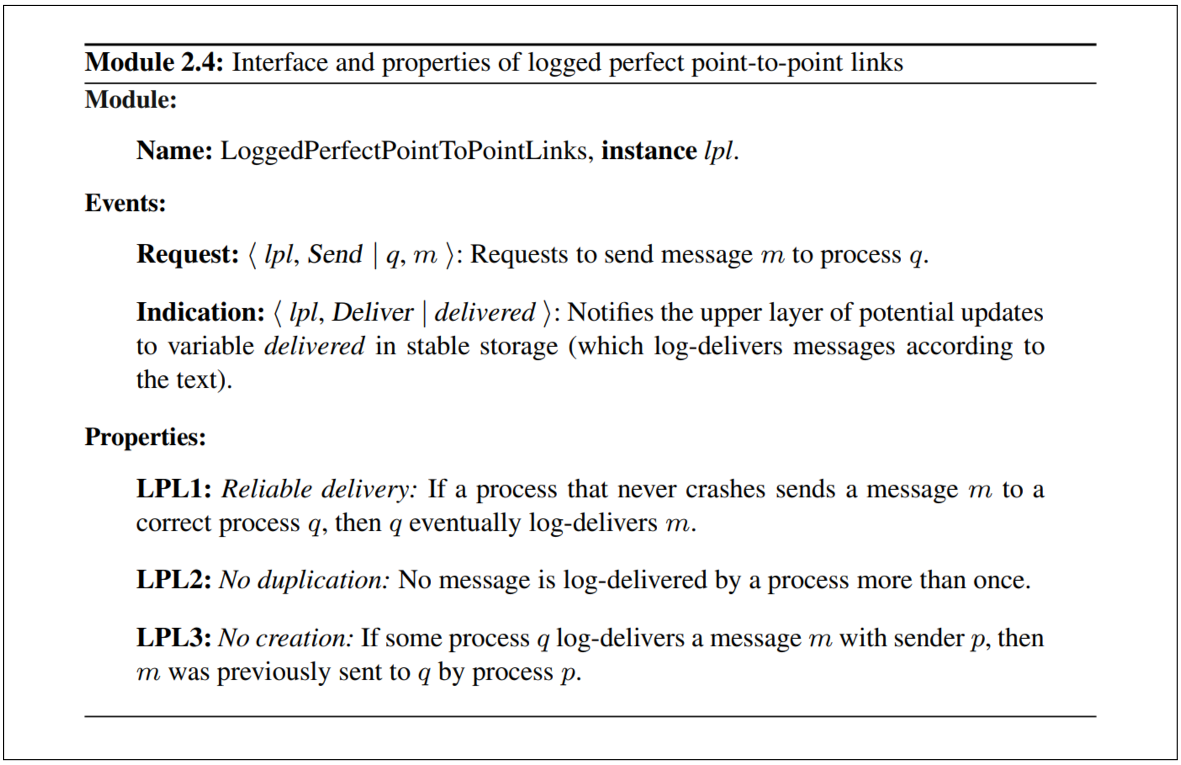
**PL1:** *Reliable delivery:* If a correct process $p$ sends a message $m$ to a correct process $q$, then $q$ eventually delivers $m$.

**PL2:** *No duplication:* No message is delivered by a process more than once.

**PL3:** *No creation:* If some process $q$ delivers a message $m$ with sender $p$, then $m$ was previously sent to $q$ by process $p$.

Figure 5: Interface of perfect point-to-point links

### 1.5.5 Logged Perfect Links



**Module 2.4:** Interface and properties of logged perfect point-to-point links

**Module:**

**Name:** LoggedPerfectPointToPointLinks, **instance** *lpl*.

**Events:**

**Request:** ⟨ *lpl*, *Send* | $q$, $m$ ⟩: Requests to send message $m$ to process $q$.

**Indication:** ⟨ *lpl*, *Deliver* | *delivered* ⟩: Notifies the upper layer of potential updates to variable *delivered* in stable storage (which log-delivers messages according to the text).

**Properties:**

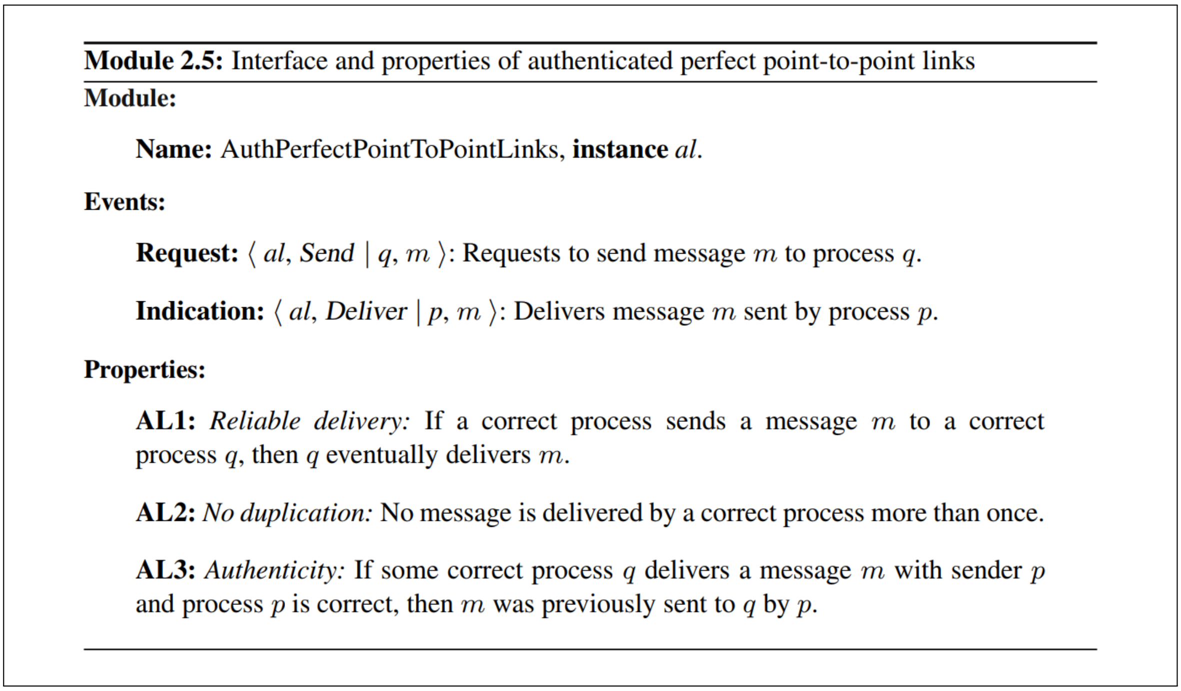
**LPL1:** *Reliable delivery:* If a process that never crashes sends a message $m$ to a correct process $q$, then $q$ eventually log-delivers $m$.

**LPL2:** *No duplication:* No message is log-delivered by a process more than once.

**LPL3:** *No creation:* If some process $q$ log-delivers a message $m$ with sender $p$, then $m$ was previously sent to $q$ by process $p$.

Figure 6: Interface of logged perfect point-to-point links

### 1.5.6 Authenticated Perfect Links



**Module 2.5:** Interface and properties of authenticated perfect point-to-point links

**Module:**

**Name:** AuthPerfectPointToPointLinks, **instance** *al*.

**Events:**

**Request:** ⟨ *al*, *Send* | $q$, $m$ ⟩: Requests to send message $m$ to process $q$.

**Indication:** ⟨ *al*, *Deliver* | $p$, $m$ ⟩: Delivers message $m$ sent by process $p$.

**Properties:**

**AL1:** *Reliable delivery:* If a correct process sends a message $m$ to a correct process $q$, then $q$ eventually delivers $m$.

**AL2:** *No duplication:* No message is delivered by a correct process more than once.

**AL3:** *Authenticity:* If some correct process $q$ delivers a message $m$ with sender $p$ and process $p$ is correct, then $m$ was previously sent to $q$ by $p$.

Figure 7: Interface of Authenticated perfect point-to-poin links

### 1.5.7 Algorithms on Abstracting Links

# 2 Time in Distributed Systems

# 3 Logical Clock

# 4 Distributed Mutual Exclusion

## 4.1 Abstracting Communication

### 4.1.1 Failure Detection Abstraction

### 4.1.2 Perfect Failure Detectors

### 4.1.3 Eventually Perfect Failure Detectors

### 4.1.4 Leader Election

### 4.1.5 Eventual Leader Election

# 5 Broadcast Communications

# 6 Consensus

# 7 Ordered Communications

# 8 Registers

# 9 Software Replication

# 10 Overview on Capacity Planning

# 11 Modeling the Workload of a System

# 12 Building a Performance Model 1

# 13 Building a Performance Model 2

# 14 Dependability Evaluation

# 15 Intro to Experimental Design

# 16 CAP Theorem

# 17 Consistency Criteria for Distributed Shared Memories

# 18 Publish-Subscribe Communication Paradigm

# 19 Overlay Networks

# 20 DLT and Blockchain

# 21 Exercises

*Notice: The exercises are from 2022-2023 academic year.*

# 22 Exams