Distributed Algorithms 2025-2026

Project introduction

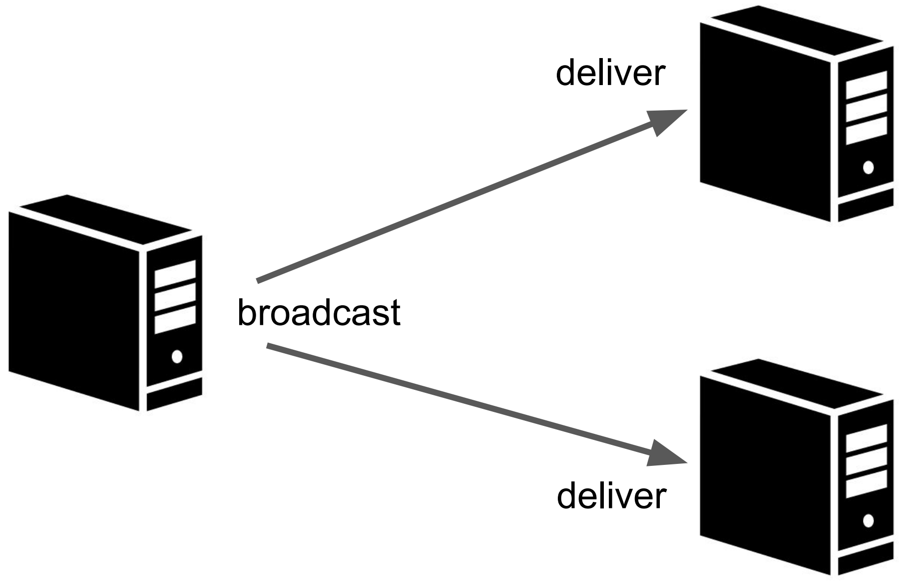
Student Assistants:
Davit Darbinyan, Marcin Wojnarowski

Teaching Assistants:
Beatrice Shokry, Rishi Sharma, Diana Petrescu

Introduction to broadcast

Goal: Ensure reliable message delivery to all participants.

Informally:



Complexities in broadcast

- *Asynchrony*:
  No bounds on messages and process execution delays

- *Failures*:
  Processes fail by crashing.

  They stop executing instructions/actions after the crash.

- *Communication*:
  Messages can be lost, delayed, reordered, duplicated.

How computers communicate today: TCP vs UDP

TCP	*UDP*	*Comment*
Connection-oriented	Connectionless	UDP does not establish a connection before sending data.
Reliable stream	Unreliable datagram	UDP is unreliable, it does not provide guaranteed delivery and a datagram packet may be lost in transit.
Flow control	No control	With UDP, packets arrive in a continuous stream or they are dropped.
Ordered	Unordered	TCP does ordering and sequencing to guarantee that packets sent from a server will be delivered to the client in the same order they were sent. On the other hand, UDP sends packets in any order.

Source: https://www.privateinternetaccess.com/blog/tcp-vs-udp-understanding-the-difference/

Goal of the project

Implement certain building blocks necessary for distributed systems:

Perfect Links is the 1st deliverable:

- Deadline: October 19 2025, 23:59

FIFO broadcast is the 2nd deliverable:

- Deadline: November 23 2025, 23:59

Lattice Agreement is the 3rd deliverable:

- Deadline: December 21 2025, 23:59

The theory behind the first two deliverables are introduced in the class.

Project requirements: Basics

- Allowed languages:
  - C11 and/or C++17
  - Java 11
- Compilation method:
  - CMake for C/C++
  - Maven for Java
  - We provide a template for both. **Using the template is mandatory!**
- Allowed 3rd party libraries: **None**

Project requirements: Interface

Your deliverables should support the following command line arguments:

Usage: `./run.sh --id ID --hosts HOSTS --output OUTPUT CONFIG`

Where:

- **ID** is the id of the process.
- **HOSTS** is the path to a file that contains information about every process in the system.
- **OUTPUT** specifies the path to a text file where a process stores its output.
- **CONFIG** is the path to a file that contains specific information required from the deliverable (e.g. how many messages to broadcast)

We provide source code that does the parsing for you!

Project requirements: Messages

- You are allowed to use only UDP packets in their most basic form:
  - Point to point (no broadcast)
- You are not allowed to use third party libraries
  - Everything must be implemented on top of UDP packets
- Application messages are numbered sequentially at each process
  - They start from number **1** and can go up to **MAX_INT (i.e., 2^31 - 1)**,
  - By default, their payload is only the sequence number.
  - We limit the number of messages per same packet to 8.
- Broadcasting processes have ids from **1** to n, where n will be no more than **128**

Project requirements: Signal Handlers

A process that receives a SIGTERM or SIGINT signal must immediately stop its execution (signals are used to simulate process crashes).

This means that:

- After receiving the signal, the process must not send or handle any received network packets.
- The process is only allowed to write to the output log file after one of the signals is received.
- You can assume that at most a minority (e.g., 1 out of 3; 2 out of 5; 4 out of 10, ...) of processes may crash in an execution.

Project requirements: Output format

- The output of a process is a text file.
  - You specify the location of the output file using the **--output** argument.
  - The files contains events. Each event is represented by one line of the file:

E.g. the output of process k may contain:

b 1

b 2

$\boxed{\text{b 3}}$

d 2 1

$\boxed{\text{d 4 2}}$

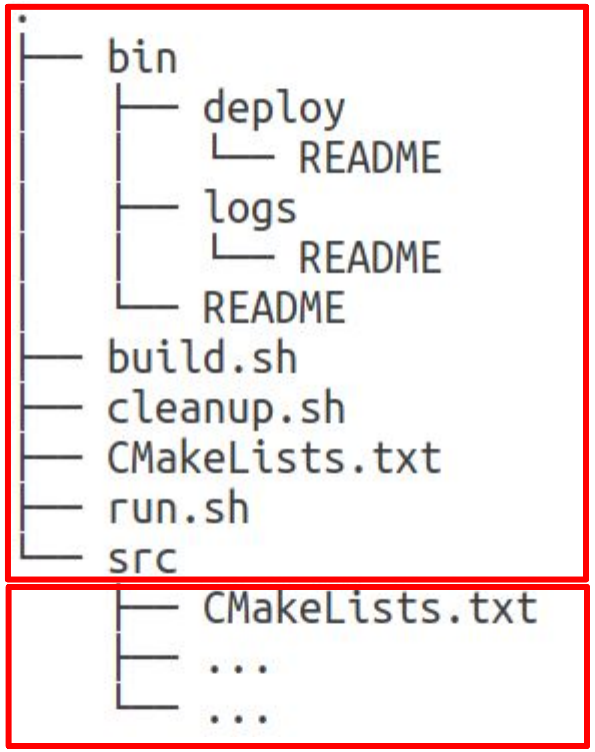
b 4

Process k broadcast a message with sequence number 3

Process k delivered a message with sequence number 2 from process 4

C/C++ Project Template

Deliver the project in a .zip file that has the following structure:



```
.
├── bin
│   ├── deploy
│   │   └── README
│   ├── logs
│   │   └── README
│   └── README
├── build.sh
├── cleanup.sh
├── CMakeLists.txt
├── run.sh
└── src
    ├── CMakeLists.txt
    ├── ...
    └── ...
```

Do not edit this section!
Your binary should not create make use of directories “deploy” and/or “logs”.
Run:

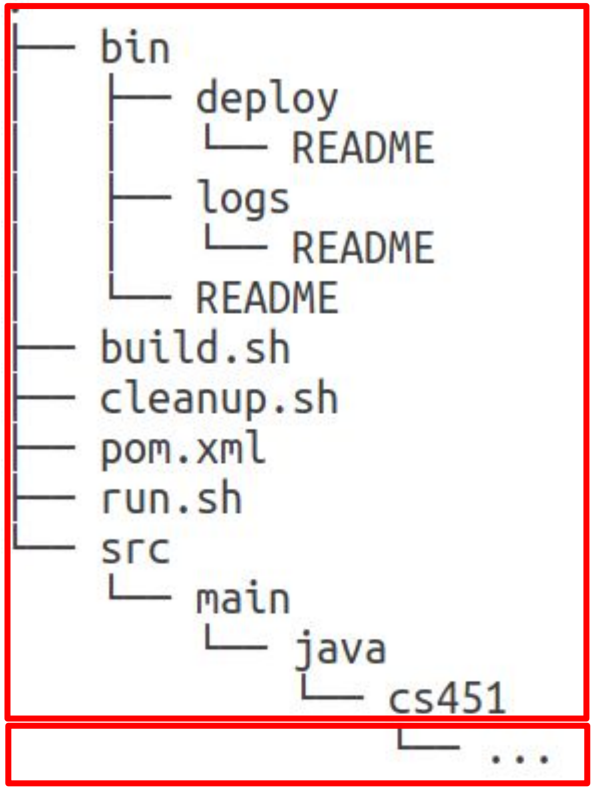
- *build.sh* to **compile** the code
- *run.sh <args>* to **run** the code
- *cleanup.sh* to **delete build artifacts** (use when submitting your project)

You can arrange your project files as you see fit.

Do not change the name of the executable in the CMakeLists.txt

Java Project Template

Deliver the project in a .zip file that has the following structure:



```
├── bin
│   ├── deploy
│   │   └── README
│   ├── logs
│   │   └── README
│   └── README
├── build.sh
├── cleanup.sh
├── pom.xml
├── run.sh
└── src
    └── main
        └── java
            └── cs451
                └── ...
```

Do not edit this section!
Your binary should not create make use of directories “deploy” and/or “logs”.
Run:

- *build.sh* to **compile** the code
- *run.sh <args>* to **run** the code
- *cleanup.sh* to **delete build artifacts** (use when submitting your project)

Note: Do not edit pom.xml, i.e. do not use 3rd libraries in maven!

You can arrange your project files as you see fit.

Compilation environment

- All submitted projects will be tested using Ubuntu 18.04 (x86_64) with:
    - **gcc/g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0**
    - **cmake version 3.10.2**
    - **OpenJDK Runtime Environment (build 11.0.8+10-post-Ubuntu-0ubuntu118.04.1)**
    - **Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)**
- Projects that fail to compile with the above will not be considered!

Runtime Environment

When evaluating your submission, we use a fixed set of resources, namely:

- You are given 2 CPU cores,
- You a maximum 4 GiB RAM.
- You are allowed to spawn a maximum of 8 threads per process.

Consider these limitations when developing your project.

Projects that are not well engineered may fail some tests due to them.

How to develop my project

1. Using the Virtual Machine (VM):
   a. It already includes the exact versions of the build tools mentioned previously.
   b. Students with limited programming knowledge are advised to use it.
   c. Get the VM Image from here.
2. Ignoring all recommendations:
   a. E.g., not using the versions mentioned.
   b. Do it at your own risk. Remember, if a project fails to compile in the mentioned setup, it will not be considered!

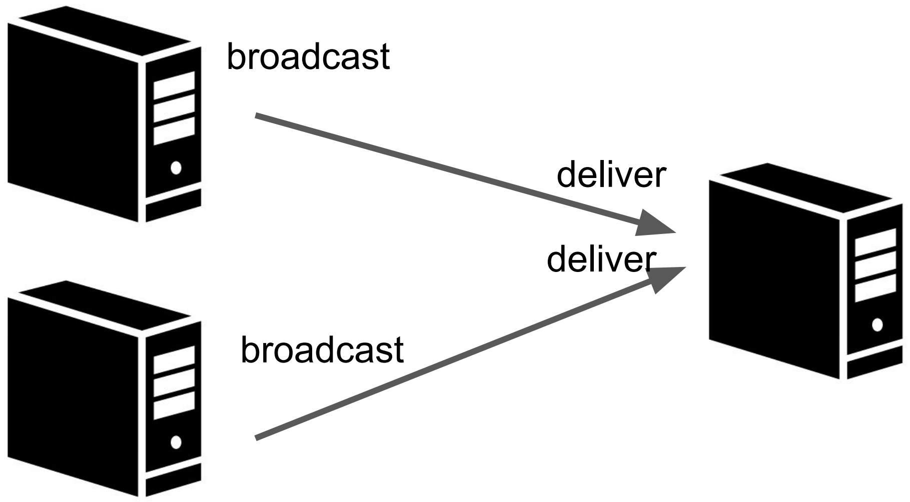
Submission 1: Perfect Links

- 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 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.

Submission 1: Perfect Links

Config file: m i

- m is how many messages each sender process should send
- i is the receiver process
- E.g. 10 1



Submission 1: Example of output files

Proc 1

```
d 2 3
d 2 4
d 2 1
d 2 2
d 2 7
d 2 8
d 2 5
d 3 4
d 2 6
d 3 3
d 2 9
d 2 10
d 3 2
d 3 1
d 3 8
d 3 7
d 3 6
d 3 5
d 3 10
d 3 9
```

Proc 2

```
b 1
b 2
b 3
b 4
b 5
b 6
b 7
b 8
b 9
b 10
```

Proc 3

```
b 1
b 2
b 3
b 4
b 5
b 6
b 7
b 8
b 9
b 10
```

Project Evaluation (1/2)

- The project accounts for 30% of the final grade:
  - The 1st deliverable (Perfect Links) is 20% of the project grade,
  - The 2nd deliverable (FIFO Broadcast) is 40% of the project grade,
  - The 3rd deliverable (Lattice Agreement) is 40% of the project grade.
- The project evaluation consists of two parts:
  - Correctness tests,
  - Performance tests

Project Evaluation (2/2)

- Correctness tests under:
  - **Network issues**: packet delay, packet loss, packet reordering, **no packet corruption**
  - **Process issues**: slow processes, crashed processes
- Performance tests under:
  - Varying number of broadcasting processes,
  - Absence of process crashes/delays

Implementation steps

$$\begin{tikzcd}
\text{Perfect Links} \arrow[d] \\
\text{Uniform Reliable Broadcast} \arrow[d] \\
\text{FIFO Broadcast} \arrow[d] \\
\text{Lattice Agreement}
\end{tikzcd}$$

Devise a retransmission strategy to deal with network issues
***Tip**: Test using only two processes*

Pick the appropriate algorithm from class
***Tip:** For performance, try to keep your payload small*

Focus on correctness first!
Try to improve performance later.

Traffic Control (1/2)

tc: linux utility for altering the network characteristics (delay, loss, reorder).

How to use:

```
$ sudo tc qdisc add dev lo root netem 2>/dev/null
$ sudo tc qdisc change dev lo root netem delay 200ms 50ms loss 10% 25% reorder 25% 50%
```

What it does:

- Adds packet delay of 200ms ± 50ms [150ms-250ms] that is normally distributed.
- 10% of the packets are lost. Each successive probability depends by 25% on the the last one: P(n) = 0.25 * Prob(n-1) + 0.75 * Random (Emulated burst losses)
- 25% of packets (with a correlation of 50%) will get sent immediately, others will be delayed by the aforementioned delay factor.

More info: https://man7.org/linux/man-pages/man8/tc-netem.8.html

Traffic Control (2/2)

For convenience, **tools/tc.py** applies the TC commands of the previous slide for you.

How to use:
$ python3 tc.py

Good to know:

- The tc rules are applied for as long as this program is running. When this program exits, the tc rules are reset.
- This program applies the following TC configuration:
```
config = {
  'delay': ('200ms', '50ms'),
  'loss': ('10%', '25%'),
  'reordering': ('25%', '50%')
}
```

  Edit tools/tc.py to try different configurations

Testing (1/3)

Instructions to manually run some processes are given in the GitHub page. Manual runs are useful for the early phases of developing and for debugging.

For example

```
# Build the application:
$ ./build.sh

# In first terminal window:
$ ./run.sh --id 1 --hosts ../example/hosts --output ../example/output/1.output
../example/configs/perfect-links.config

# In second terminal window:
$ ./run.sh --id 2 --hosts ../example/hosts --output ../example/output/2.output
../example/configs/perfect-links.config

# In third terminal window:
$ ./run.sh --id 3 --hosts ../example/hosts --output ../example/output/3.output
../example/configs/perfect-links.config

# Wait enough time for all processes to finish processing messages.
# Type Ctrl-C in every terminal window to create the output files.
```

Testing (2/3)

You can also run using the tools/stress.py script.

Usage: ./stress.py MILESTONE -r RUNSCRIPT -l LOGSDIR -p PROCESSES -m MESSAGES

Where:

- **RUNSCRIPT** is the path to *run.sh*. Remember to build your project first!
- **MILESTONE** specifies which deliverable (one of `perfect`, `fifo`, `agreement`) you run with. Some milestones might need additional flags, see the project description for all details.
- **LOGSDIR** is the path to a directory where stdout, stderr and output of each process will be stored. It also stores generated HOSTS and CONFIG files. The directory must exist as it will not be created for you.
- **PROCESSES** specifies the number of processes spawn during validation.
- **MESSAGES** specifies the number of messages each process is broadcasting (for perfect link and fifo broadcast).

Testing (3/3)

You can edit `tools/stress.py` in order to test different scenarios.
Go to the bottom of the file and edit the following:

```
testConfig = {
        'concurrency' : 8,
        'attempts' : 8  ,
        'attemptsDistribution' : {
                'STOP': 0.48,
                'CONT': 0.48,
                'TERM':0.04
        }
}
```

How many threads are interfering with the running processes.

How many successful operations (SIGCONT, SIGSTOP, SIGTERM) each thread will attempt before stopping. Threads stop if a minority of processes has been terminated.

Each thread selects a process randomly and issues one of these operations with the given probability.

Further info about stress.py

- You can use tc.py before running stress.py to change the network configuration.
- You can change the rate at which interfering threads interfere with the processes:
  - Modify this line: `time.sleep(float(random.randint(50, 500)) / 1000.0)` under the `StressTest` class

Measuring performance

We measure performance under no process crashes/delays.

- We are interested in the aggregate throughput:
  - i.e., the combined throughput of all processes.
- Have each process broadcast a large number of messages.
  - The number must be large enough such that no process finishes
- Measure as follows:
  - Start all processes,
  - Wait x minutes,
  - Stop all processes,
  - Take a snapshot.
- Compute the throughput based on the generated logs and the execution time

Submission

For each Milestone, two links will be available on Moodle approximately one week before the corresponding deadline.

- Testing Milestone
  - verify your solution against a limited set of correctness tests. Not all grading tests are included, and performance is not evaluated here.
- Submitting Milestone
  - official submission for offline grading of correctness and performance.