Università di Pisa

Master Of Science in Computer Engineering

Performance Evaluation of Computer Systems and Networks

Satellite Communications

Project members:
Dario Bandecchi
Francesco De Lucchini
Niccolò Mulè

Academic Year 2024/2025

Contents

1 Introduction

1.1 Problem Description

1.1.1 Communication system

Consider a communication system with N terminals that receive packets from a ground station via a satellite, which acts as a simple relay node. Each terminal has its own dedicated FIFO queue at the ground station, with packets of size S bytes arriving at each queue every T seconds. S and T are uniformly and exponentially distributed random variables, respectively.

Transmissions from the ground station occur in time slots of fixed and constant duration of 80 ms, organized in the following way:

- First, each terminal randomly selects a coding rate (the maximum data rate supported) from the set `{L3, L2, L1, R, H1, H2, H3}`, and reports it to the ground station
- Then, the ground station composes a frame of M blocks by scheduling packets from the terminals' queues, as described in Section 1.1.2
- Finally, the composed frame is sent back to the terminals

1.1.2 Scheduling algorithm

The ground station serves the terminals according to a **maximum-coding-rate policy**, i.e., by sorting them based on their reported coding rate and starting from the highest one (`H3`). When a terminal is considered for service, the ground station tries to empty the terminal's queue before considering the next one. Each block is assigned the coding rate of the terminal who owns the first packet scheduled inside of it, and that, according to the following table, determines the block's capacity:

Coding rate	`L3`	`L2`	`L1`	`R`	`H1`	`H2`	`H3`
Capacity (bytes)	$904/M$	$1356/M$	$1808/M$	$2260/M$	$2712/M$	$3164/M$	$3616/M$

A block can carry packets for multiple terminals, as long as the coding rate of such terminals is greater or equal than the coding rate of the block, and a packet can be segmented into multiple blocks, as long as it completely fits in a single frame.

1.2 Objectives

The objective of this project is to study the effectiveness of a maximum-coding-rate scheduling algorithm in a satellite-based communication system.

1.3 Key Performance Indicators

The effectiveness of the proposed scheduling algorithm is studied through the analysis of the following key performance indicators:

- The **throughput**, computed as the number of bits sent out by the ground station per unit of time
- The **mean delay**, measured by each terminal as the average elapsed time between the arrival of a packet at the ground station and its arrival at the terminal itself
- The **mean frame utilization**, defined as the average of the frame utilizations, i.e., the total size of the scheduled packets divided by the frame capacity (which is given by the sum of the capacities of the single blocks)

2 Modeling

2.1 General Assumptions

Considering the objective of this study, and considering the chosen key performance indicators, the system can, without loss of generality, be simplified in the following way:

- All the transmissions to and from the satellite are instantaneous
- The processing times (e.g., the time it takes the ground station to create a frame) are negligible
- The queues of the ground station are infinite

2.2 Preliminary validation

A preliminary validation of the model is carried out by analyzing the assumptions made so far:

- The propagation times over the satellite link are assumed to be instantaneous because they are not relevant to the objective and key performance indicators of this study. In fact, even if the satellite link was modeled to have a propagation time, it would have been a somewhat constant value, given by the average distance from the satellite divided by the speed of light. This would have not affected the throughput or the frame utilization, and it would have only shifted the delays of the packets by, more or less, the same value. Therefore, it is safe to assume the propagation times over the satellite link to be instantaneous. Similar reasoning applies to the processing times
- Infinite queues allow to focus on the core system dynamics, without having to handle edge cases (e.g., dropped packets). Moreover, in a real system, queues would be sized in such way that the probability of dropping a packet under normal operating conditions is “small enough”, therefore, it is reasonable to assume this probability to be negligible in the model

2.3 Parameters

The system is characterized by the following parameters, which are calibrated once and then kept fixed in all of the simulations:

- **minPacketSize** and **maxPacketSize**, the extremes of the uniform distribution representing the size of the packets
- **meanPacketInterarrivalTime**, the mean of the exponential distribution representing the inter-arrival time of the packets at each queue of the ground station

2.4 Factors

The performance of the system is studied as the following factors vary:

- $\boldsymbol{N}$[1], the number of terminals
- $\boldsymbol{M}$[1], the number of blocks per frame
- The distribution of the **coding rates** of the terminals

[1] To improve the readability of the code, $\boldsymbol{N}$ and $\boldsymbol{M}$ are called **terminalCount** and **blocksPerFrame**, respectively. From now on, these terms will be used interchangeably.

3 Implementation

3.1 Modules

The simulator is developed using OMNeT++, a C++ discrete event simulation framework, and is composed by the following modules[2]:

- A **GroundStation** compound module (hence, with no active behavior), composed by:
  - An array of **PacketGenerator** modules, one for each *Terminal*, which are responsible for generating the ***`Packets`*** destined to their corresponding *Terminal* (i.e., the *i*-th *PacketGenerator* generates the ***`Packets`*** destined to the *i*-th *Terminal*).

    More specifically, each *PacketGenerator*:

    1. Extracts an inter-arrival time and uses it to set a timer
    2. When the timer ticks, extracts a size, builds a ***`Packet`*** with that size, sends it to the *PacketScheduler*, and then repeats step 1

    Since the extractions are always alternated, the two random variables (the size and the inter-arrival time) are independent "by design", and, therefore, each *PacketGenerator* module only requires one global random number generator (RNG)
  - A **PacketScheduler** module, which carries out the following tasks:
    1. Collects incoming ***`Packets`*** from the *PacketGenerators* and stores them in the corresponding queue
    2. Receives the ***`CodingRatePackets`*** from the *Terminals* and keeps count of how many have been received so far in the current communication slot
    3. Once all the ***`CodingRatePackets`*** have been received, builds a ***`Frame`*** according to the algorithm described in Section 1.1.2 and sends it to the *Satellite*
    4. Finally, computes and emits the throughput and the frame utilization
- A **Satellite** module, which simply acts as a relay node:
  - Whenever it receives a ***`CodingRatePacket`*** from a *Terminal*, it forwards it to the *GroundStation*
  - Whenever it receives a ***`Frame`*** from the *GroundStation*, it broadcasts it to the *Terminals*
- An array of *terminalCount* **Terminals**, which are responsible for initiating the communication slot (i.e., sending out their ***`CodingRatePacket`*** every 80 ms) and computing the delays (if a ***`Packet`*** is segmented, its delay is emitted only when the last segment is fetched). In order for the coding rates to be independent, each *Terminal* must be linked to a different global RNG. Considering also the global RNGs required by the *PacketGenerators*, the simulator counts a total of $2 \cdot terminalCount$ global RNGs
- An **Oracle** (virtual) module, which has no active behavior or connections, and is only used to maintain the positions of the ***`Packets`*** within a ***`Frame`***, allowing the *Terminals* to efficiently retrieve their own ***`Packets`*** without having to loop through all the blocks in the ***`Frame`***

[2] To improve the readability of this section (and, more in general, of this whole chapter), the following convention is used: *Modules* are written in PascalCase italics, while ***`CustomMessages`*** are written in PascalCase monospace italics

3.2 Connections

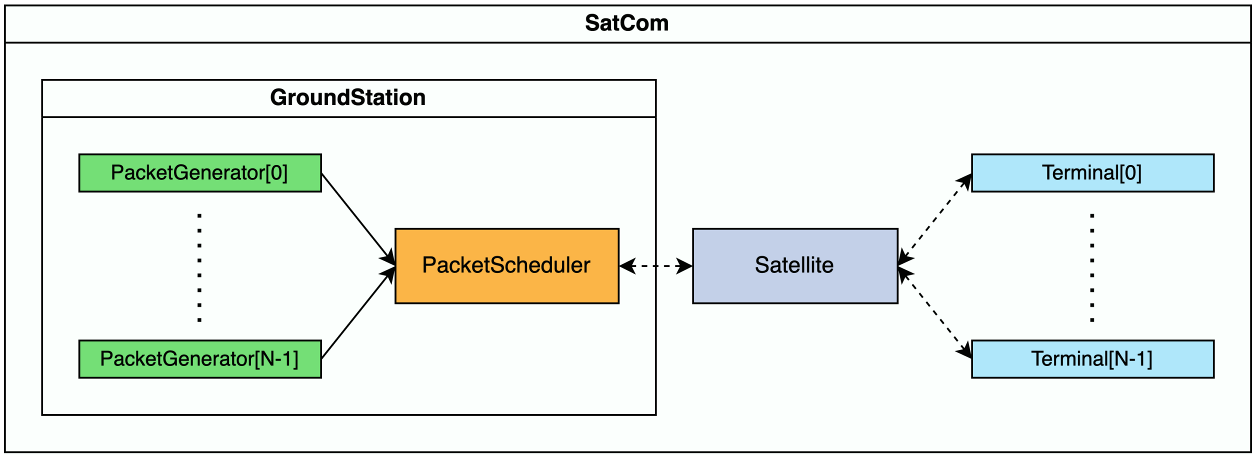

Figure 1: OMNeT++ network design. Dashed arrows indicate direct (wireless) connections.

3.3 Custom Messages

The simulator utilizes the following custom messages:

- **GenericPacket**: Represents a generic packet in the communication system. It extends the `cPacket` class already available in OMNeT++ by adding a `terminalId` integer field, which specifies the packet's destination *Terminal*. It extends the `cPacket` class and not the `cMessage` class because the first one already comes with an integer field representing the packet's size.
- **CodingRatePacket**: Extends the *`GenericPacket`* class described before by adding a `codingRate` enumeration field. It is only used by the *Terminals* to communicate their coding rates to the *GroundStation*
- **Frame**: Extends the `cPacket` class by adding an array of $blocksPerFrame$ `Blocks`. Again, it extends the `cPacket` class and not the `cMessage` class because of the presence of the size field, which simplifies the computation of the throughput. A `Block` is an extension of the `cObject` class equipped with a `codingRate` enumeration field, two integer fields, `maxBytes` and `usedBytes`, and a *`GenericPacket*`* array field. Since the *Satellite* duplicates the *`Frame`* multiple times during the broadcasting process, an array of pointers to *`GenericPackets`* is preferred (only the pointers are copied); of course, each *Terminal* is responsible for de-allocating (only) its own *`GenericPackets`*

4 Verification

4.1 Debugging

The simulator takes advantage of C++'s **conditional compilation** (`#ifdef MACRO ... #endif`) to provide users with a modular debugging environment. While this approach may not be considered best practice in bigger projects, for smaller ones, it offers a quick and straightforward way of selectively enabling not only debugging statements but entire portions of code.

The implemented debugging features are defined in the **makefrag** file, OMNeT++'s intended way of extending the makefile. One in particular deserves attention and it is the `DEBUG_RNGS` flag, which enables detailed debugging statements regarding everything that is random in the simulation. Specifically, each time a module extracts a random variable, it outputs the total number of calls made so far to the global random number generator linked to the module. This comes in handy to check that random variables which are assumed to be independent actually come from independent RNG streams.

An example of the output provided by this macro, along with a few comments that explain the context, is the following:

```
# First, each packetGenerator extracts the arrival time of the first packet
DEBUG: [0.000]> Extracted nextArrivalTime for terminal 0 | Total RNG calls: 1
DEBUG: [0.000]> Extracted nextArrivalTime for terminal 1 | Total RNG calls: 1

# Then, each time a timer ticks, the packetGenerator module who set it
#  extracts a size for the incoming packet and the arrival time of the
#  next one
DEBUG: [0.023]> Extracted byteLength for terminal 1 | Total RNG calls: 2
DEBUG: [0.023]> Extracted nextArrivalTime for terminal 1 | Total RNG calls: 3
DEBUG: [0.024]> Extracted byteLength for terminal 1 | Total RNG calls: 4
DEBUG: [0.024]> Extracted nextArrivalTime for terminal 1 | Total RNG calls: 5
DEBUG: [0.032]> Extracted byteLength for terminal 0 | Total RNG calls: 2
DEBUG: [0.032]> Extracted nextArrivalTime for terminal 0 | Total RNG calls: 3

# Finally, at the beginning of each communication slot (the first
#  one starts at 80ms), the terminals extract their coding rates
DEBUG: [0.080]> Extracted codingRate for terminal 0 | Total RNG calls: 1
DEBUG: [0.080]> Extracted codingRate for terminal 1 | Total RNG calls: 1
```

From this example it is clear that, as explained in Section 3.1, each packet generator and each terminal is linked to a different global RNG, which makes all the inter-arrival times, the packet sizes, and the coding rates independent random variables, as expected.

4.2 Runtime error handling

OMNeT++'s `cRuntimeError` class is used to stop a simulation whenever an obviously impossible logical condition verifies. Examples of such conditions are:

- In `packetScheduler.cc`

```
if (frame->getByteLength() > getMaxTheoreticalFrameBytes())
{
    throw cRuntimeError(this, "The size of the current frame (%lld) is greater than the maximum theoretical size (%d)", frame->getByteLength(), getMaxTheoreticalFrameBytes());
}
```

- In `CodingRatePacket.msg`

```
switch (codingRate)
{
    case L3:
        ...
    case H3:
        ...
    default:
        throw omnetpp::cRuntimeError("The specified coding rate (%d) does not exist", codingRate);
}
```

- In `terminal.cc`

```
if (packet->getTerminalId() != id)
{
    throw cRuntimeError(this, "Terminal %d was allowed to read a packet destined to terminal %d", id, packet->getTerminalId());
}
```

- In `satellite.cc`

```
if (msg->isName("codingRatePacket"))
{
    /* Forward the message to the ground station */
    ...
}
else if (msg->isName("frame"))
{
    /* Broadcast the message to the terminals */
    ...
}
else
{
    throw cRuntimeError(this, "The satellite cannot handle the received message: %s. Supported types are: \"codingRatePacket\", \"frame\"", msg->getName());
}
```

4.3 Memory analysis and code profiling

The code was analyzed for memory leaks using **Valgrind** in a memory-aggressive scenario, i.e., one where the system receives more packets than it is able to handle (and so queues fill up over time). An example of such scenario is the following, where the mean flow of incoming packets, 400 Kbps (10 queues, receiving, on average, packets of $(20 + 380)/2 \cdot 8$ bits every 40 ms), is higher than the **maximum theoretical throughput**, 361.6 Kbps (a full frame of `H3` blocks, i.e., $3616 \cdot 8$ bits, sent every 80 ms):

Parameter/Factor	**Value**
sim-time-limit	80 s
repeat	30
codingRateDistribution	Uniform
terminalCount	10
blocksPerFrame	10
minPacketSize	20 bytes
maxPacketSize	380 bytes
meanPacketInterarrivalTime	40 ms

```
==5943== Using Valgrind-3.22.0 and LibVEX
==5943==
==5943== HEAP SUMMARY:
==5943==     in use at exit: 16,768 bytes in 7 blocks
==5943==   total heap usage:
==5943==        91,106,704 allocs,
==5943==        91,106,697 frees,
==5943==        6,716,264,373 bytes allocated
==5943==
==5943== LEAK SUMMARY:
==5943==    definitely lost: 0 bytes in 0 blocks
==5943==    indirectly lost: 0 bytes in 0 blocks
==5943==      possibly lost: 0 bytes in 0 blocks
==5943==    still reachable: 16,768 bytes in 7 blocks
==5943==         suppressed: 0 bytes in 0 blocks
==5943==
==5943== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

The "still reachable" category within Valgrind's leak report refers to blocks of memory that were not freed, but could have been freed if the programmer had wanted to, because the program was still correctly keeping track of their pointers. Those memory blocks are likely to be data structures managed by OMNeT++'s simulation kernel, as their size remains constant when the simulation factors change. Consequently, these results provide compelling evidence that the code is free of memory leaks.

Moreover, using **KCachegrind** to analyze **Callgrind**'s profiling data did not reveal any critical function (i.e., a function that takes much longer than others to complete).

4.4 Behavioral tests

The behavior of the system is now tested in simple scenarios that can be manually checked for correctness, starting off with a deterministic test.

4.4.1 Deterministic test

Consider the following scenario:

Parameter/Factor	**Value**
sim-time-limit	800 s
codingRate (deterministic)	`R`
terminalCount	1
blocksPerFrame	1
minPacketSize	1 byte
maxPacketSize	1 byte
packetInterarrivalTime (deterministic)	1 ms

Packets of fixed size of 1 byte are scheduled to arrive at the (only) queue every 1 ms, consequently, every communication slot, a total of 80 packets will arrive. Given that the (only) terminal always has a coding rate of `R`, which corresponds to a block capacity of 2260 bytes, all the packets always fit in the first frame sent after their arrival, therefore, we expect to measure the following statistics:

- Throughput: 8 Kbps (i.e., 1 queue, receiving packets of $1 \cdot 8$ bits, every 1 ms)
- Mean frame utilization: $80/2260 = 3.540\,\%$
- Mean delay:

$$\frac{(79 + 78 + ... + 1 + 0)}{80} = \frac{79 \cdot (79 + 1)/2}{80} = \frac{79 \cdot 80}{2 \cdot 80} = \frac{79}{2} = 39.5 \text{ ms}$$

  Where, in the first sum, 79 ms is the delay of the first packet, which arrives 1 ms after the previous communication slot, and 0 ms is the delay of the last packet, which arrives right on time to be immediately sent to the terminal

The results of the simulation are exactly the ones we expected, therefore, we can move on and introduce randomness in our tests.

4.4.2 Introducing randomness

Consider the same scenario as the one used in Section 4.4.1, with the only changes being that the inter-arrival times of the packets are now exponentially distributed with mean 1 ms, and the coding rates are now uniformly distributed:

Parameter/Factor	**Value**
sim-time-limit	800 s
repeat	30
codingRateDistribution	Uniform
terminalCount	1
blocksPerFrame	1
minPacketSize	1 byte
maxPacketSize	1 byte
meanPacketInterarrivalTime	1 ms

On average, we expect the throughput to remain the same, however, different considerations must be made when it comes to the mean frame utilization and the mean delay:

- Since, in this case, the size and capacity of a frame are independent random variables, we expect the mean frame utilization to be:

$$\mathrm{E}\left[\frac{\text{Size}}{\text{Capacity}}\right] = \mathrm{E}\left[\text{Size}\right] \cdot \mathrm{E}\left[\frac{1}{\text{Capacity}}\right] = 80 \cdot \frac{1}{7}\left(\frac{1}{904} + ... + \frac{1}{3616}\right) = 4.344\,\%$$

- As the simulation time approaches infinity, we expect packets to arrive at every possible offset between a communication slot and the one after, in an uniformly distributed way[3]. Therefore, given that, in this particular scenario, packets always fit in the first frame sent after their arrival, we expect the delays to be uniformly distributed between 0 and 80 ms, with, of course, a mean of 40 ms

After running 30 independent simulations, the averaged results, considering 99 % confidence intervals, are the following:

- Throughput: $8.000 \pm 0.003 = [\,7.997, 8.004\,]$ Kbps
- Mean frame utilization: $4.344 \pm 0.010 = [\,4.334, 4.354\,]\,\%$
- Mean delay: $39.997 \pm 0.014 = [\,39.983, 40.010\,]$ ms

Since the predicted results lie inside of the confidence intervals, we can say with 99 % confidence that the system behaves as expected in this particular scenario.

4.4.3 Continuity test

Considering the same scenario as the one used in Section 4.4.2, since there is only one terminal and all the packets always fit in the first frame after their arrival, we expect that:

- Having one or an arbitrary number of blocks per frame does not change anything at all

[3]Qualitatively, we measured this to be true as long as the simulation duration is at least in the order of 10^4 times the mean packet inter-arrival time.

- Whatever the distribution of the coding rates, both the throughput and the delays stay exactly the same (other than an uniform, we tested this using a binomial with parameters $n = 6$, $p = 0.5$)
- Slightly changing the duration of the communication slot only changes the mean delay to exactly half of the new communication slot duration, as explained in Section 4.4.2

All of the previous statements have been confirmed to be true with the simulator.

4.4.4 Degeneracy test

Considering, again, the scenario described in Section 4.4.2, if we set the size of the packets to 0 bytes, then we should observe: a throughput of 0 bps, a mean frame utilization of 0 %, and the same exact delays. That is precisely what happens in the simulation.

Another quick degeneracy test is checking that the maximum theoretical throughput (361.60 Kbps) is indeed reachable under optimal conditions, which are the following ones:

- The coding rate of any terminal is always the maximum one, `H3`
- Packets are one byte big, so they can fit anywhere and always fill the frame
- There always are as many packets to send as the ones needed to fill a frame

Under such conditions, achieved by manually assigning `H3` to every terminal and using a very low mean packet inter-arrival time (e.g., 0.01 ms), we correctly measured the maximum theoretical throughput (and, of course, a frame utilization of 100 %), hence, the system behaves as expected.

Note that, in this very peculiar case, even though there is only one terminal, having one or an arbitrary number of blocks per frame <u>does</u> affect the throughput. This is because of the truncation error introduced when calculating the capacity of the blocks: if, under the optimal conditions stated before, we set the number of blocks in a frame to 100, then, the capacity of each block is $36.16 \approx 36$ bytes, leading to a throughput of 360.00 Kbps.

4.4.5 Consistency test

Considering, one last time, the same scenario as the one used in Section 4.4.2, we expect to measure, one average, twice the throughput (16 Kbps), twice the mean frame utilization (8.687 %), and the same mean delay (40 ms), if any of the following changes are made:

- Double the size of the packets (from 1 byte to 2 bytes)
- Half the mean packet inter-arrival time (from 1 ms to 0.5 ms)

If, instead, we double the number of terminals (from 1 to 2), the complexity of the system skyrockets, and manually verifying the results becomes way harder; nevertheless, we tried anyway to make sense of them. After running 30 independent simulations, the averaged results, considering 99 % confidence intervals (not visible due to the scale), are the ones reported in Figure 2.

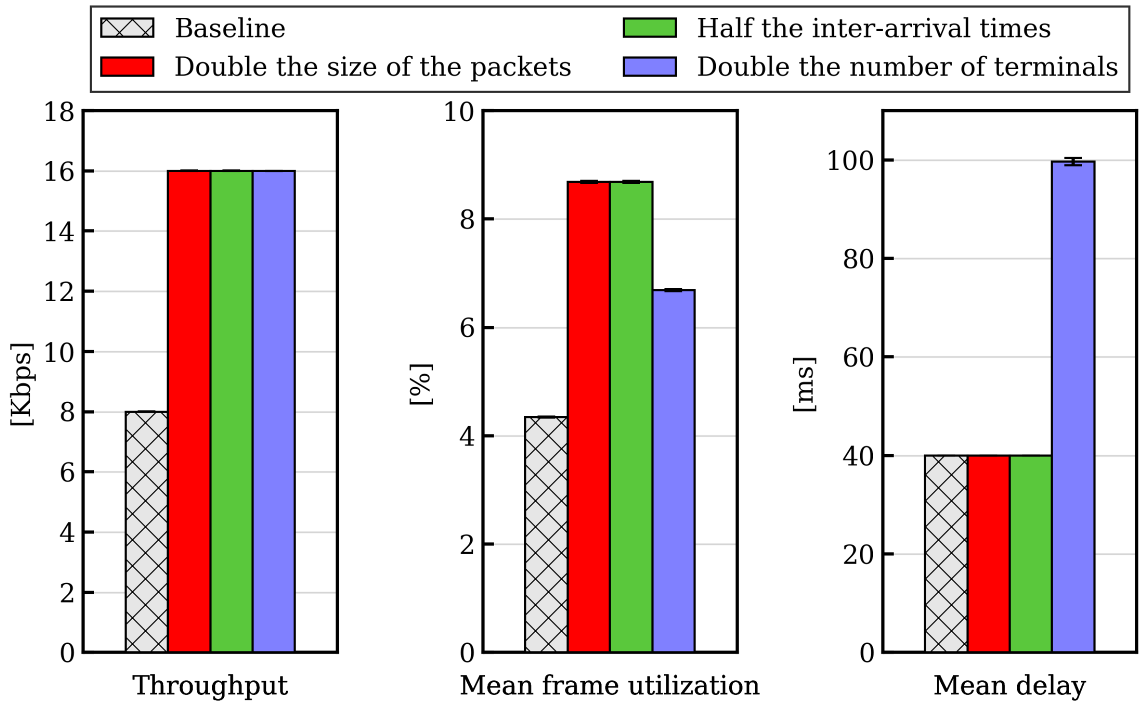

Figure 2: Consistency test

One can immediately observe that when doubling the size of the packets and when cutting in half the inter-arrival times, the system behaves exactly as expected, moreover, there is no reason to believe that by doubling the number of terminals the throughput should be anything other than 16 Kbps.

When it comes to the mean delay[4], intuitively, it makes sense that with still one block but two terminals it gets higher. This is because the two terminals compete for that single block, and, when they lose (i.e., they extract a coding rate strictly lower than the other terminal), their packets will have to wait (at least) an entire communication slot.

A mathematical explanation for both the apparently odd values of mean frame utilization and mean delay obtained when doubling the number of terminals can be found in Section 9.

[4]In case of two terminals, we take, for each repetition, the mean delay measured by the first one (which, in case of uniformly distributed coding rates, is representative of all terminals)

5 Calibration

In this section, we outline the tests that were conducted in order to calibrate the parameters of the simulator, which include the ones listed in Section 2.3 as well as the warm-up time, the simulation time, and the parameters for the distribution of the coding rates.

5.1 Packet inter-arrival times and sizes

We calibrated the parameters of the simulation in the following way:

Parameter	**Value**
minPacketSize	20 bytes
maxPacketSize	80 bytes
meanPacketInterarrivalTime	40 ms

Intuitively, these values make sense because:

- The mean packet inter-arrival time is not wildly different from the duration of a communication slot
- The minimum packet size is at least a few bytes (in the real world, packets always have a fixed minimum size given by their headers)
- The maximum packet size is not bigger than, more or less, a hundred bytes (otherwise, given the table in Section 1.1.2, it would have been too hard to schedule)

5.2 Coding rate distributions

As explained in Section 2.4, among the other things, we want to study the performance of the system as the distribution of the coding rates vary. The distributions that we chose to analyze are:

- A (discretized) **uniform** distribution with parameters $(0, 7)$, which is always useful as a baseline benchmark, although it is quite unrealistic to encounter in a real system, since it would imply that very high (or low) values of coding rate (hence, SNR) are as frequent as average ones
- A **binomial** distribution, chosen in such way that the mean coding rate of different terminals are sensibly different, which might represent a network where terminals are fixed and, therefore, their SNR is biased by their geographical location (e.g., presence of obstacles, distance from the satellite, ...)
- A (discretized) **normal** distribution with parameters $(\mu = 3.5, \sigma = 1)$, which could model a real-world scenario where most of the time terminals have average performance, but occasionally either very good or poor

Specifically, when using the binomial distribution, to achieve the fact that the mean coding rate of different terminals are sensibly different, each terminal employs a slightly different binomial distribution, with parameters $n = 6$ (the number of possible coding rates, including 0), and

$$p = \frac{\texttt{terminalId} + 1}{\texttt{terminalCount} + 1},$$

where the +1 terms ensure that no terminal has $p = 0$ (always `L3`) or $p = 1$ (always `H3`). Figure 3 shows an example of this when $terminalCount = 3$.

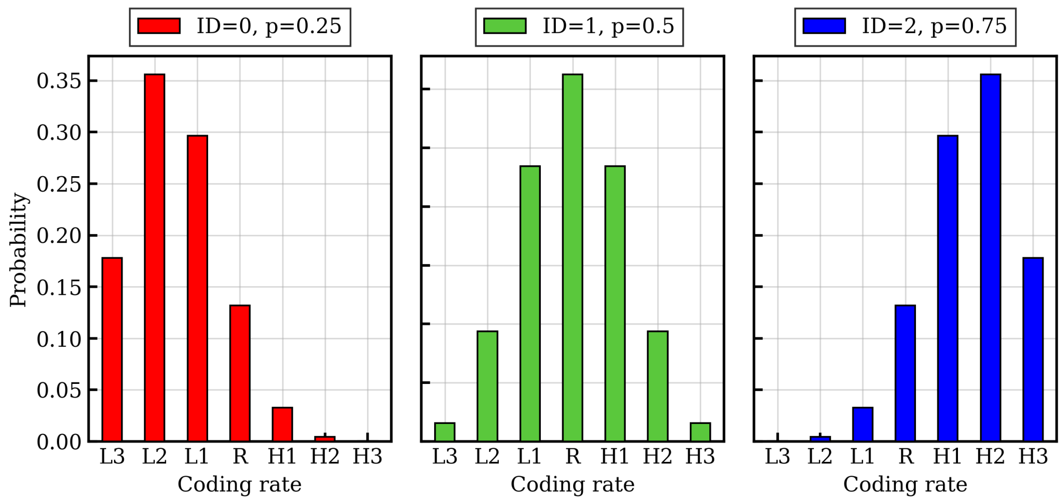

Figure 3: Binomial distributions of the coding rates when $terminalCount = 3$

5.3 Warm-up period and simulation time limit

Considering the scenario described in the table below, which uses the parameters calibrated in Section 5.1 and represents the system under normal operating conditions, we estimated the warm-up period by observing the time average of the throughput over 10 independent repetitions, as shown in Figure 4.

Parameter/Factor	**Value**
sim-time-limit	80 s
repeat	10
terminalCount	20
codingRateDistribution	Uniform
blocksPerFrame	4
minPacketSize	20 byte
maxPacketSize	80 byte
meanPacketInterarrivalTime	40 ms

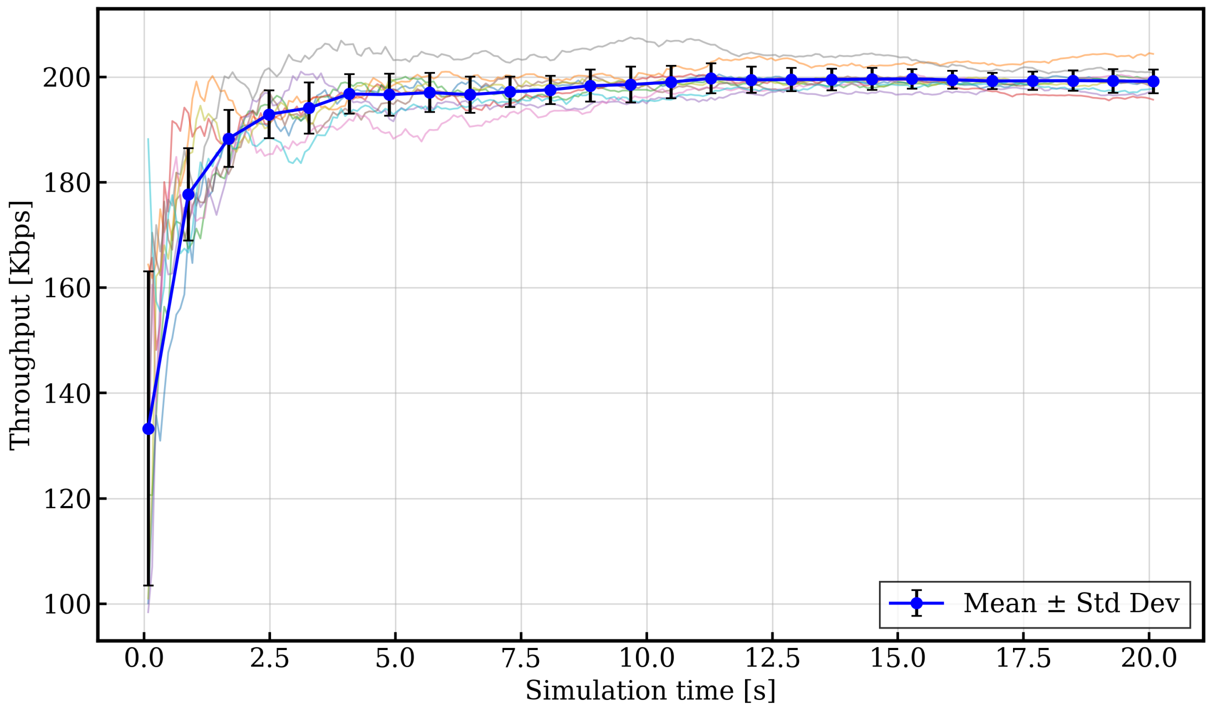

Figure 4: Calibration of the warmup period under normal operating conditions. The thick line represents the mean and standard deviation of a group of 10 independent repetitions (thin lines) in correspondence of the same communication slot, subsampled with a ratio 1:10.

As illustrated in Figure 4, starting at simulation time of approximately **10 seconds**, the standard deviations of the repetitions remain relatively constant, thereby signifying the conclusion of the warm-up phase. Furthermore, it is reasonable to select a simulation time limit which is considerably longer than the warm-up period. Considering also the trade-off with the CPU time required to execute the simulations, **800 seconds** (10^4 communication slots) appears to be an appropriate simulation time limit.

6 Experiment design

Consider N queues receiving, on average, packets of S bytes every T ms, as long as S, T and N are set in such way that the mean flow of incoming packets ($N \cdot S \cdot 8 \cdot 1/T$) does not approach the maximum theoretical throughput (361.6 Kbps), then the delays of the packets stay in the order of a communication slot duration. Instead, if the mean flow of incoming packets approaches or surpasses the maximum theoretical throughput, then the packets starts queuing up, and, thus, the delays grow linearly over time.

When the coding rates are uniformly distributed, this phenomenon starts happening for incoming flows of around $2260 \cdot 8 \cdot 1/0.080 = 226$ Kbps, where 2260 bytes is the mean frame capacity. Figure 5 shows exactly this behavior for $N = 1$, $T = 1$ ms and various values of S: when $S = 25$, the mean flow of incoming packets is $1 \cdot 25 \cdot 8 \cdot 1/0.001 = 200$ Kbps, which is lower than 226 Kbps, and, therefore, the delays do not grow over time, instead, with $S = 30$ and $S = 35$, the mean flow of incoming packets is greater than 226 Kbps, and, consequently, the delays grow linearly over time.

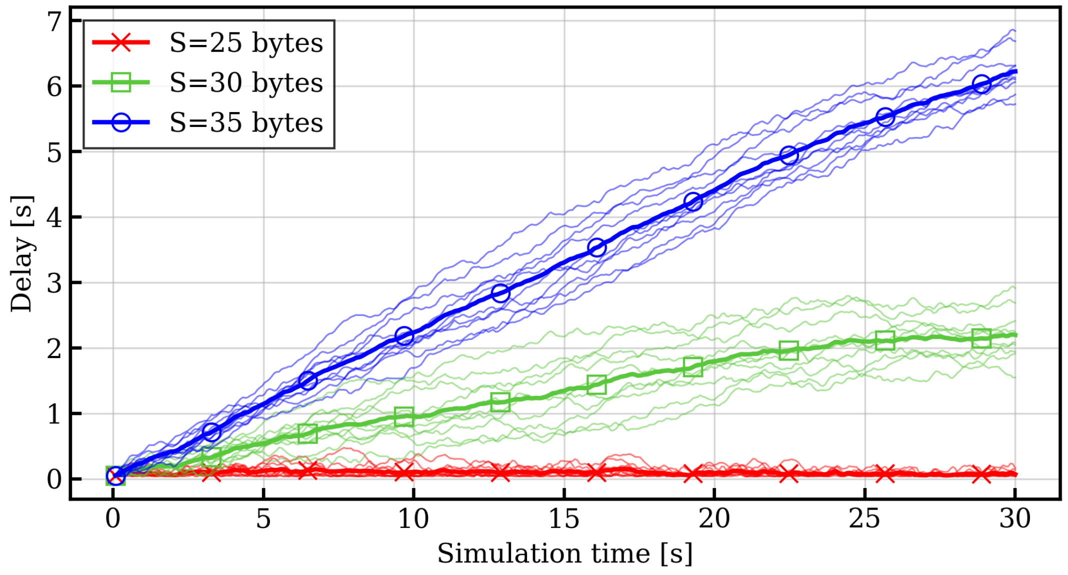

Figure 5: Thin lines are single independent repetitions (10 for each value of S), while thick lines are the mean of the corresponding group of repetitions

Keeping that in mind, in order to obtain a comprehensive understanding of the system's behavior, the experiments should be designed in such way that the mean flow of incoming packets ($N \cdot (80 + 20)/2 \cdot 8 \cdot 1/0.004 = 10\,N$ Kbps) is lower than, in the order of, and greater than the maximum theoretical throughput. Moreover, instead of using constant values, it also makes sense to set the number of blocks per frame as a fraction of the number of terminals.

That said, we assigned the following ranges to the factors of the simulation:

- $\boldsymbol{N} = \{10, 20, 30, 40, 50, 60\}$
- $\boldsymbol{M} = \{20\,\%, 40\,\%, 60\,\%, 80\,\%, 100\,\%\}$ of $\boldsymbol{N}$
- $\boldsymbol{CR} = \{\text{Uniform, Binomial, Normal}\}$

7 Data analysis

All the charts in this section were computed using averaged results obtained by running 30 independent simulations. When not vanishingly little, 99% confidence intervals are shown.

7.1 Analysis of the throughput

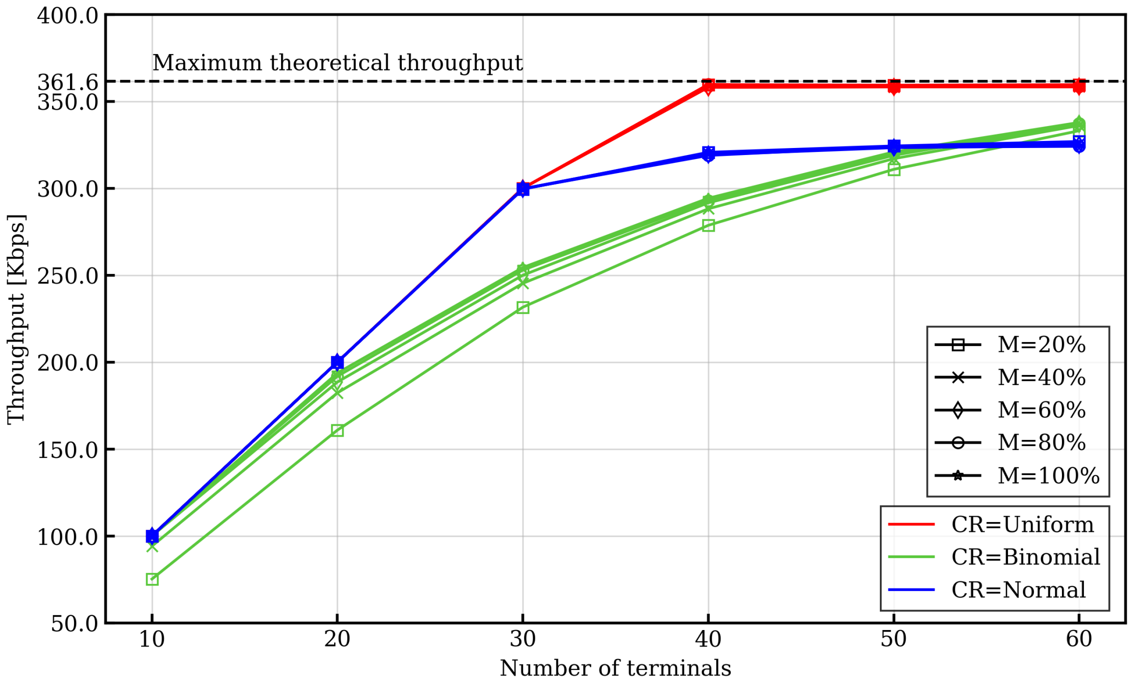

Figure 6: Throughput of the system as the number of terminals, the number of blocks per frame, and the distribution of the coding rates vary.

Figure 6 shows that, as expected, regardless of the number of blocks per frame and the distribution of the coding rates, **the throughput increases as the number of terminals grows large**, up to the maximum theoretical value of 361.6 Kbps. This is because, trivially, the more are the terminals, the more are the packets to send, and also the higher is the probability of scheduling terminals with the highest coding rate.

Note that this is also true when the coding rates are normally distributed, the convergence to the maximum theoretical throughput is slow due the very low probability of extracting the highest coding rate. Oppositely, **when the coding rates are uniformly distributed**, the probability of extracting the highest coding rate is no different than the probability of extracting any other coding rate, hence, **the convergence to the maximum theoretical throughput is faster**. When it comes to the binomial distribution, instead, only a handful of terminals gets consistently scheduled (the ones with an higher parameter p), therefore, the problem is not finding terminals with the highest coding rate, but having enough terminals with the highest coding rate that they alone are able to fill a frame each time.

Furthermore, **as far as the throughput is concerned, varying the number of blocks per frame has an appreciable effect only when the coding rates are binomially distributed**, and only for low-to-average numbers of terminals. This is because, when the coding rates are uniformly or normally distributed, at each communication slot there always are a considerable amount of terminals with the same coding rate, therefore, they most likely can already be scheduled in the same block, and, hence, changing the number of blocks per frame does not yield appreciable results. Instead, when the coding rates are binomially distributed, increasing the number of blocks per frame also gives the low-end terminals a chance of consistently contributing to the throughput.

Generally speaking, regardless of the distribution of the coding rates, for large values of N, the frame is almost always exclusively composed by `H3` blocks. Therefore, only considering the throughput, it would be more convenient set the value of blocks per frame as low as possible, in order to speed up the scheduling process and avoid the block size truncation problems explained in Section 4.4.4.

7.2 Analysis of the mean frame utilization

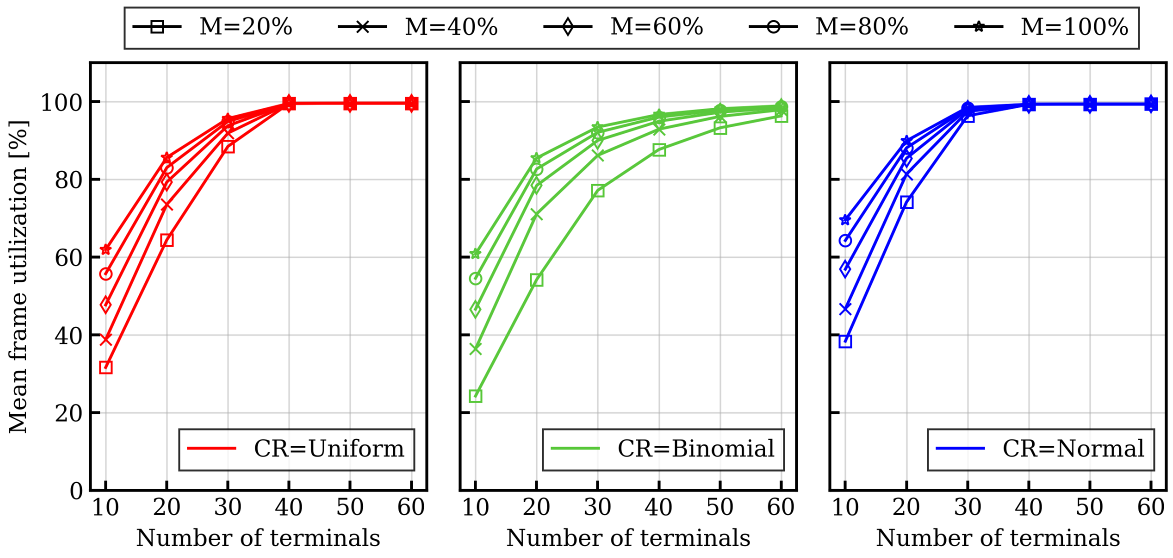

Figure 7: Mean frame utilization as the number of terminals, the number of blocks per frame, and the distribution of the coding rates vary.

The mean frame utilization indicates how well the available resources (i.e., the space in the frame) are used. As expected, **with an higher number blocks per frame there is less wasted space**, and, thus, an higher frame utilization (if $M = 1$, the terminals with the highest coding rate take the whole frame even in they only have a few packets). Intuitively, this phenomenon becomes less and less relevant as the number of terminals gets higher, since more of them get the highest coding rate, and, consequently, can be scheduled anyway in the same block.

Generally speaking, an higher frame utilization also implies an higher throughput, but that's not the full picture. Consider the case of normally distributed coding rates, Figure 7 shows that the mean frame utilization is already indistinguishably close to one for $N = 30$, however, for that same number of terminals, the throughput is nowhere close to the maximum theoretical one. This is because it's true that the frame almost always gets full, but that same frame rarely has the maximum capacity (the probability of extracting a single `H3` terminal is low, let alone the one of scheduling an entire frame of `H3` blocks).

Finally, Figure 7 shows that **when the coding rates are binomially distributed, the mean frame utilization is the worst**. This is because, as already said in Section 7.1, only a handful of terminals gets consistently scheduled, therefore, an higher number of blocks per frame is needed to give the low-end terminals a chance of consistently contributing to the throughput.

7.3 Analysis of the delays

When the coding rates are uniformly or normally distributed, all the terminals contribute equally, therefore, when it comes to measuring statistics, **one terminal is representative of all the others**. Consequently, in the following charts we plot the mean delay measured by the first terminal, averaging the results over 30 independent simulations.

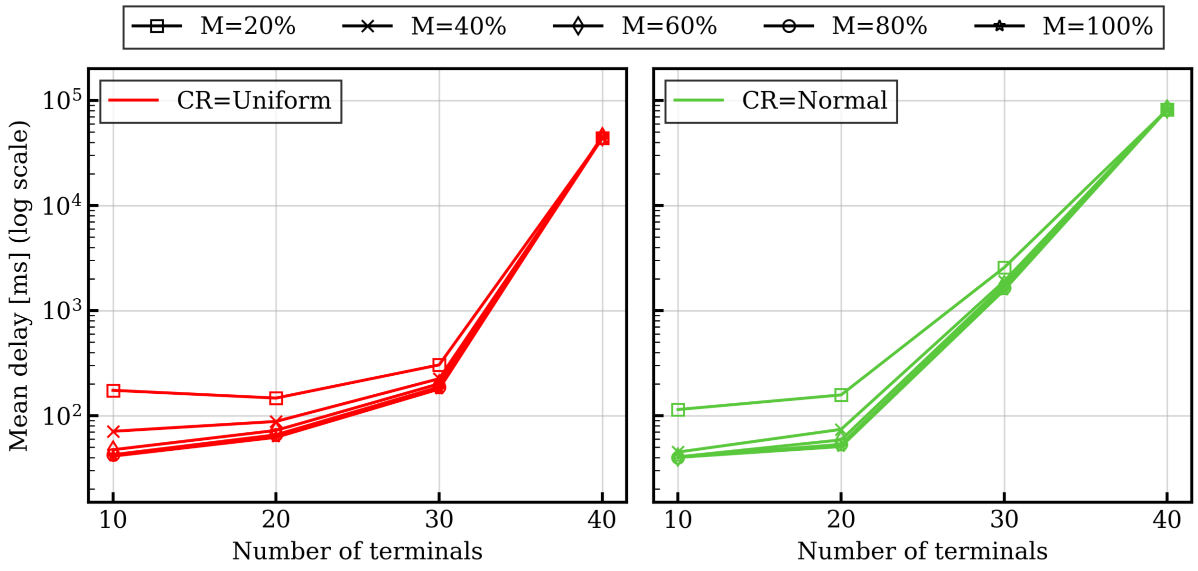

Figure 8: Qualitative plot of the mean delay against the number of terminals

Figure 8 shows that **the mean delays skyrocket** (even in log scale) when the number of terminals gets to around $30 \sim 40$. This is because, for those same numbers of terminals, the mean frame utilization approaches 1, meaning that there is no space left in the frame, consequently, adding more terminals only decreases the chance of getting scheduled, hence, raising the delays.

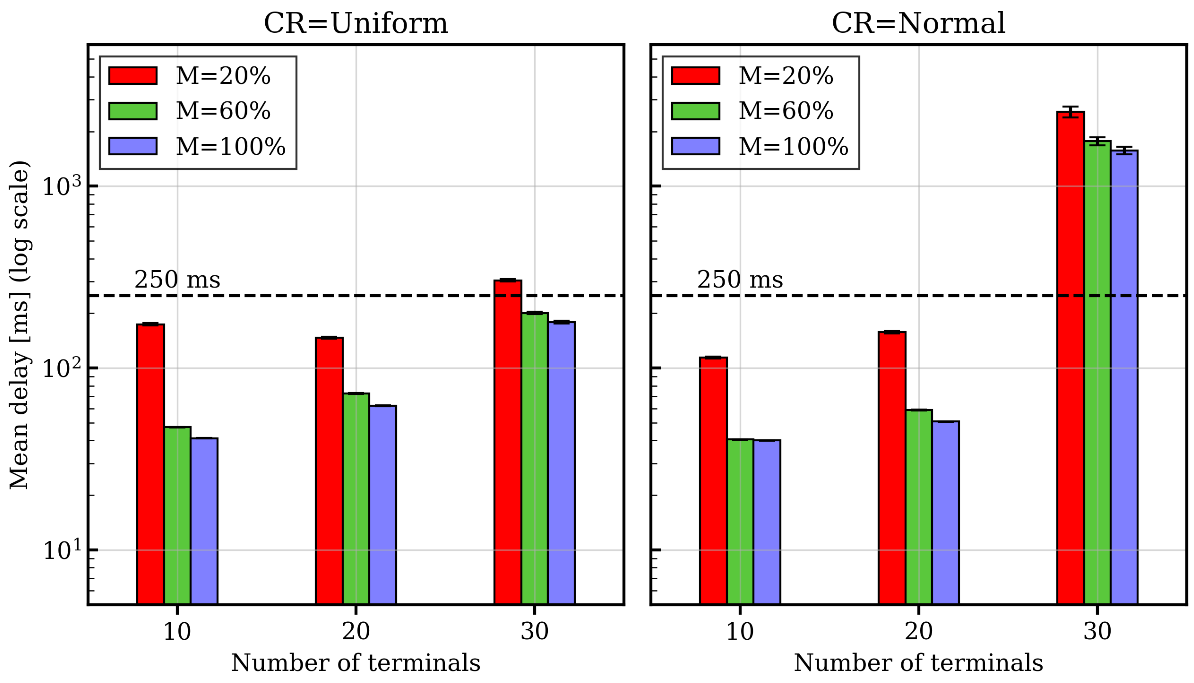

Figure 9: Effect of the number of blocks per frame on the mean delay

When the coding rates are uniformly or normally distributed, **considering an acceptable upper bound of 250 ms for the mean delay**, we can restrict the study up to values of N of around 30, since, as shown in 8, after that the mean delays skyrocket.

As shown in Figure 9, we can say with 99% confidence that **having an higher number of blocks per frame is beneficial to the mean delays**, and it is particularly beneficial for lower values of N, as already explained in Section 7.

For $N = 30$, the mean delay of the terminals with normally distributed coding rates is significantly higher than both the uniformly distributed counterpart and the acceptable upper bound. This is because, as shown in Figure 7, when the coding rates are normally distributed the system reaches the maximum frame utilization sooner and, thus, the mean delays also rise sooner (as explained under Figure 8).

Consider instead the case of binomially distributed coding rates. In such scenario, each terminal employs a slightly different binomial distribution (as described in 5.2), and, consequently, consistently measures a different mean delay. Therefore, **when examining the impact that binomially distributed coding rates have on the mean delays**, it becomes clear that, unlike the case of normally and uniformly distributed coding rates, **a single terminal is no longer representative of all the others**. Consequently, we decided to study the empirical CDF of the mean delays measured by all the terminals in the system, considering the samples coming from 30 independent repetitions.

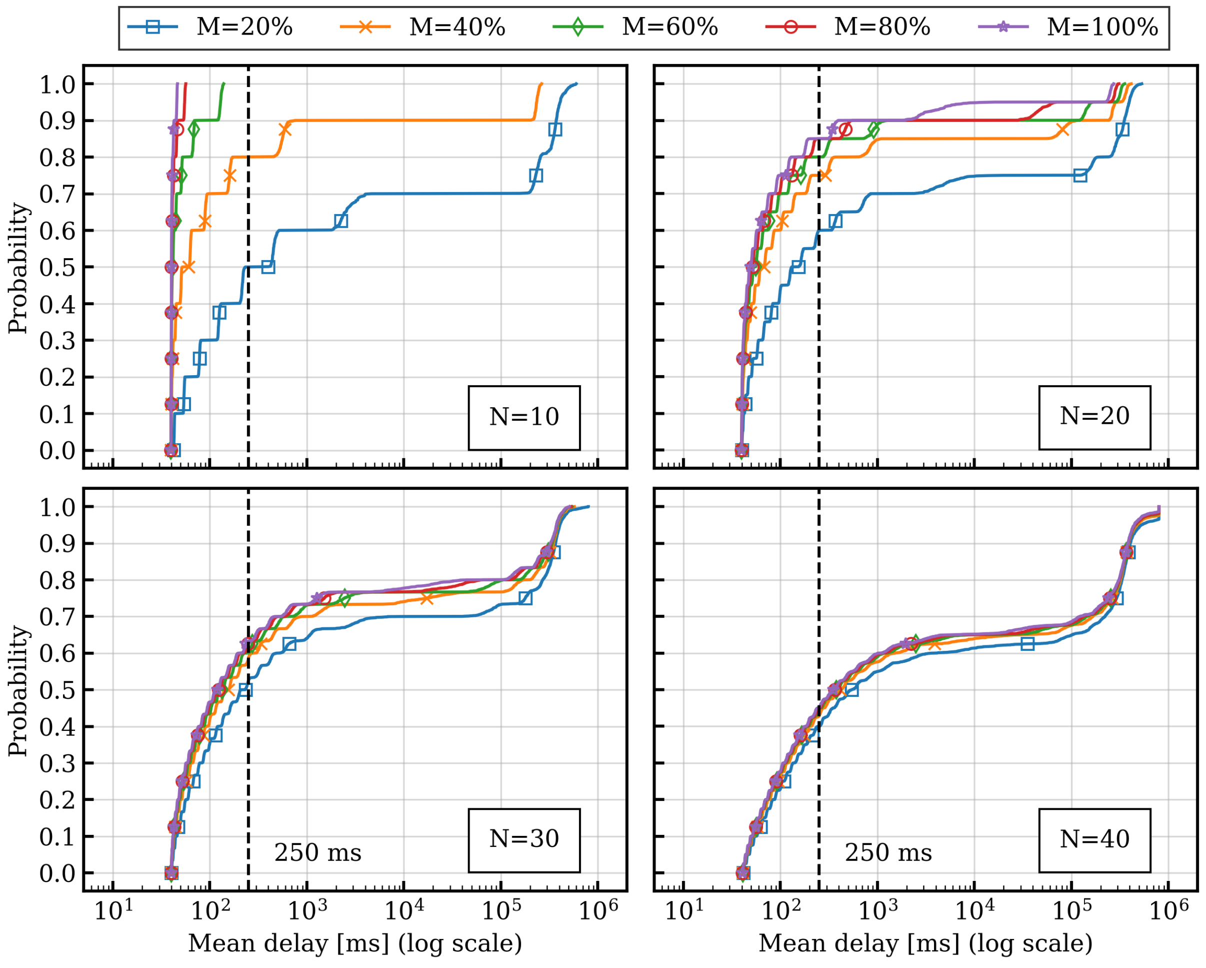

Figure 10: Empirical CDF of the mean delays of the terminals when the coding rates are binomially distributed

The ECDF in Figure 10 illustrates that, **for $N \leq 20$, increasing the number of blocks per frame greatly improves the mean delays of the terminals**, and, therefore, the percentage of terminals that are considered to experience a good quality of service. Once again, this is expected, as a higher number of blocks per frame increases the probability of scheduling terminals with lower coding rates.

For $N \geq 30$, instead, the influence of the number of blocks per frame decreases considerably. This is again expected, as with more terminals, the frame mainly includes `H3` blocks, and the likelihood of scheduling terminals with lower coding rates remains unchanged despite increases in the number of blocks per frame.

The main thing to observe in these charts is that, **unlike the case of normally and uniformly distributed coding rates, the best-performing terminals are always guaranteed to have great quality of service**, while the lowest-performing ones are not guaranteed to get any service at all. Matter of fact, for $N = 40$, the small step at the tail of the distribution is due to the fact that some of the lowest-performing terminals remain unscheduled for the whole simulation. For such terminals, the mean delay is manually set to the simulation time limit.

8 Conclusions

In this project we studied the effectiveness of a maximum-coding-rate scheduling algorithm for a satellite-based communication system, by means of the throughput, the mean packet delay, and the mean frame utilization.

The simulator was developed using OMNeT++, and was thoughtfully validated using: detailed debugging statements, runtime error handling, memory leak analysis, code profiling, and, finally, both intuitive and mathematical proofs for a wide range of behavioral tests.

The experiments, designed to provide a comprehensive understanding of the system, revealed that:

- The throughput increases as the number of terminals grows large, faster when the coding rates are uniformly distributed, and slower when they are binomially or normally distributed
- When the number of terminals is low-to-average, an higher number of blocks per frame generally leads to an higher mean frame utilization, which leads to considerably lower delays
- Considering 250 ms as an acceptable upper bound for the mean delays, when the coding rates are binomially distributed, the best-performing terminals are always guaranteed to have great quality of service, instead, when the coding rates are normally or uniformly distributed, all the terminals get similar delays, and, thus, they can have a great quality of service only when they are few in number

9 Appendix

9.1 Mathematical verification of the mean frame utilization

Consider the scenario introduced in Section 4.4.2, reported in the table below for convenience:

Parameter/Factor	**Value**
sim-time-limit	800 s
repeat	30
codingRateDistribution	Uniform
terminalCount	1
blocksPerFrame	1
minPacketSize	1 byte
maxPacketSize	1 byte
meanPacketInterarrivalTime	1 ms

We observed that, by doubling the number of terminals (from 1 to 2), the mean frame utilization grew from $4.344 \pm 0.010\,\%$ to $6.689 \pm 0.019\,\%$ (99% CI), instead of the intuitive $8.688\,\%$. The reason behind this odd behavior is that with two terminals the random variables representing the size and the capacity of a frame are no longer independent, therefore, it is no longer true that

$$\mathrm{E}\left[\frac{\text{Size}}{\text{Capacity}}\right] = \mathrm{E}\left[\text{Size}\right] \cdot \mathrm{E}\left[\frac{1}{\text{Capacity}}\right]$$

To understand this phenomenon better, consider a simplified version of the proposed scenario, where there are still two terminals, competing for one single block, but only two coding rates, `L3` and `L2`, still uniformly distributed, and deterministic packet inter-arrival times of 1 ms.

To compute the PMF of the frame utilization, we can condition the probabilities to the fact that the (only) block in the frame is scheduled with a specific coding rate:

$$\begin{aligned} \mathrm{P}\{\text{Utilization} = u\} &= \mathrm{P}\{\text{Utilization} = u \mid \texttt{L3}\} \cdot \mathrm{P}\{\texttt{L3}\} + \mathrm{P}\{\text{Utilization} = u \mid \texttt{L2}\} \cdot \mathrm{P}\{\texttt{L2}\} \\ &= \mathrm{P}\{\text{Size} = 904u \mid \texttt{L3}\} \cdot \mathrm{P}\{\texttt{L3}\} + \mathrm{P}\{\text{Size} = 1356u \mid \texttt{L2}\} \cdot \mathrm{P}\{\texttt{L2}\} \end{aligned}$$

In particular, $\mathrm{P}\{\texttt{L3}\}$ is the probability that the (only) block in the frame is scheduled with a coding rate of `L3`. Since that only happens when both of the terminals have a coding rate of `L3` (see Section 1.1.2), and the coding rates are uniformly distributed, then $\mathrm{P}\{\texttt{L3}\} = 1/4$. Consequently, $\mathrm{P}\{\texttt{L2}\} = 3/4$.

The key observation that needs to be made here is that, since the inter-arrival times of the packets are deterministically set to 1 ms, every communication slot each queue receives precisely 80 packets, hence, if only one terminal is scheduled, the other one accumulates packets.

Moreover, a key assumption needed to compute the frame size probabilities is that, whenever a terminal is scheduled, its queue always empties. It is reasonable to believe that assumption to be true because, intuitively, for an `L3` frame, it would require a single terminal to be scheduled around 10 times in a row (even higher that that for an `L2` frame).

Given that the previous assumption is true, we can say that the support of the random variable Size | `L3` is the finite set $\{80k \mid k \in \mathbb{N}^+, 80k \leq 904\} = \{80k \mid k \in \{1, ..., 11\}\}$, and computing $\mathrm{P}\{\text{Size} = s \mid \texttt{L3}\}$ becomes quite straightforward:

- For $k = 1$, $\mathrm{P}\{\text{Size} = 80 \mid \texttt{L3}\} = 0$, because, as stated before, a coding rate of `L3` only happens when both of the terminals are scheduled, therefore, the minimum frame size is 160 bytes
- For $k = 2$, $\mathrm{P}\{\text{Size} = 160 \mid \texttt{L3}\} = 1/2$, because this only happens if also in the previous communication slot the terminals were scheduled together (i.e., either both extracted `L2` or both extracted `L3`)
- For $k = 3$, $\mathrm{P}\{\text{Size} = 240 \mid \texttt{L3}\} = 1/2 \cdot 1/4 + (1/2)^2$, because, looking at the previous communication slots, there are only two ways this could have happened:
  1. A single terminal was scheduled (1/2), then the opposite one (1/4), and finally both (1, implied by the conditioning)
  2. Both of the terminals were scheduled (1/2), then a single one (1/2), then both again (1)
- For $k = 4$, $\mathrm{P}\{\text{Size} = 320 \mid \texttt{L3}\} = 1/2 \cdot (1/4)^2 + (1/2)^2 \cdot 1/4$, because, just like before, there are two ways this could have happened:
  1. A single terminal was scheduled (1/2), then the opposite one, two times, $(1/4)^2$, and finally both (1)
  2. Both of the terminals were scheduled (1/2), then a single one (1/2), then that same one again (1/4), then both (1)

At this point, one can see that

$$\mathrm{P}\{\text{Size} = 80k,\ k \in \{3, ..., 11\} \mid \texttt{L3}\} = \frac{1}{2} \cdot \left(\frac{1}{4}\right)^{k-2} + \left(\frac{1}{2}\right)^2 \cdot \left(\frac{1}{4}\right)^{k-3}$$

therefore,

$$\mathrm{P}\{\text{Size} = 80k \mid \texttt{L3}\} = \begin{cases} 0 & k = 1 \\ 1/2 & k = 2 \\ 3/2 \cdot (1/4)^{k-2} & k \in \{3, ..., 11\} \end{cases}$$

With similar reasoning, one can also find that

$$\mathrm{P}\{\text{Size} = 80k \mid \texttt{L2}\} = \begin{cases} 1/2 & k = 1 \\ 7/24 & k = 2 \\ 15/6 \cdot (1/4)^{k-1} & k \in \{3, ..., 16\} \end{cases}$$

Finally, knowing $\text{P}\{\text{Size} = s \mid \texttt{L3}\}$ and $\text{P}\{\text{Size} = s \mid \texttt{L2}\}$ $\forall s$, one can trivially calculate $\text{P}\{\text{Utilization} = u\}$ $\forall u$ as $1/4 \cdot \text{P}\{\text{Size} = 904u \mid \texttt{L3}\} + 3/4 \cdot \text{P}\{\text{Size} = 1356u \mid \texttt{L2}\}$.

Running a simulation for 8000 seconds (10^5 communication slots, enough to consistently observe events that are quite unlikely) yields the empirical CDF shown in Figure 11. Since it perfectly overlaps with the theoretical CDF computed before, as a qualitative check, we can say that the system behaves as expected.

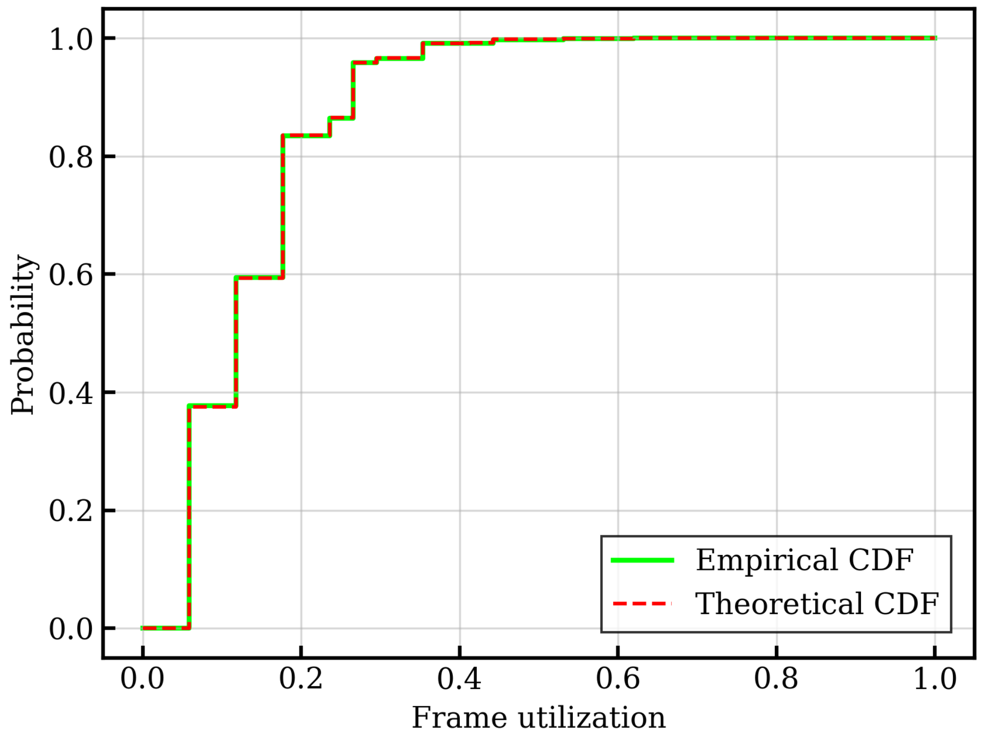

Figure 11: Empirical CDF against theoretical CDF of the frame utilization

9.2 Mathematical verification of the mean delay

Consider the scenario introduced in Section 4.4.2, reported in the table below for convenience:

Parameter/Factor	**Value**
sim-time-limit	800 s
repeat	30
codingRateDistribution	Uniform
terminalCount	1
blocksPerFrame	1
minPacketSize	1 byte
maxPacketSize	1 byte
meanPacketInterarrivalTime	1 ms

We observed that, by doubling the number of terminals (from 1 to 2), the mean delay (taking, in the second case, the average of both terminals) grew from 39.997 ± 0.014 ms to 99.672 ± 0.737 ms (99 % CI).

To understand why, consider a scenario in which the following simplifying assumptions hold:

- The frame always consist of only one block ($M = 1$)
- The coding rates are uniformly distributed
- The mean packet inter-arrival time sufficiently low, compared to the communication slot duration, so that it is seldom that a terminal that could potentially be scheduled has an empty queue (and, therefore, other terminals are scheduled even if they have a lower coding rate)
- Whenever a terminal is scheduled, its queue always empties

In such scenario, which is a more general version of the one reported at the beginning of this section, the computations to estimate the mean delay of the packets become rather straightforward.

Whenever a packet arrives at a queue, in any case, it must wait at least until the start of the next communication slot. Then, if the corresponding terminal is not scheduled immediately, the packet will have to wait an additional $80 \cdot K$ ms, where K is a random variable representing the number of consecutive communication slots in which the terminal is not scheduled. Since at each communication slot the probability of being scheduled is independent from the previous ones, K follows a geometric distribution, and estimating the mean delay $\mathrm{E}[D]$ of the packets boils down to determining the parameter p of K:

$$\mathrm{E}\left[D\right] = \mathrm{E}\left[80 - T_s + 80 \cdot K\right] = 80 - \mathrm{E}\left[T_s\right] + 80 \cdot \mathrm{E}\left[K\right] = 40 + 80 \cdot \mathrm{E}\left[K\right] = 40 + 80\left(\frac{1-p}{p}\right)$$

In particular, T_s is the random variable that represents the within-slot arrival time of the packets. As shown in 4.4.2, under the simplifying conditions stated before, T_s is uniformly distributed between 0 and 80 ms, therefore, $\mathrm{E}\left[T_s\right] = 40$ ms.

The parameter p of the geometric random variable K can be computed by observing that a specific terminal, call it i, can only be scheduled if its coding rate CR_i ($\mathrm{CR}_i \in \mathbb{N}, \mathrm{CR}_i \in [0,6]$) is the highest (or tied for the highest) among the coding rates of all the competing terminals:

$$p = \mathrm{P}\{\text{Terminal } i \text{ being scheduled}\} = \mathrm{P}\{\mathrm{CR}_i \geq \mathrm{CR}_j \ \forall j\}$$

Applying the law of total probability, the former equation can be rewritten as:

$$\begin{aligned} p = \mathrm{P}\{\mathrm{CR}_i \geq \mathrm{CR}_j \ \forall j\} &= \sum_{k=0}^{6} \mathrm{P}\left\{\mathrm{CR}_i \geq \mathrm{CR}_j \, \forall j \mid \mathrm{CR}_i = k\right\} \cdot \mathrm{P}\{\mathrm{CR}_i = k\} \\ &= \frac{1}{7} \cdot \sum_{k=0}^{6} \mathrm{P}\left\{\mathrm{CR}_i \geq \mathrm{CR}_j \, \forall j \mid \mathrm{CR}_i = k\right\} \end{aligned}$$

When $N = 2$, $\mathrm{P}\left\{\mathrm{CR}_i \geq \mathrm{CR}_j \, \forall j \mid \mathrm{CR}_i = k\right\}$ is just the the probability that the coding rate k of terminal i is greater than or equal to the one of terminal j, and that is $(k+1)/7$. Due to independence, the above formula can be generalized for any value of N by simply multiplying the probabilities:

$$p = \frac{1}{7} \cdot \sum_{k=0}^{6} \left(\frac{k+1}{7}\right)^{N-1}$$

Going back to the initial problem, when $N = 2$ we obtain:

$$p = \frac{1}{7} \cdot \sum_{k=0}^{6} \left(\frac{k+1}{7}\right)^{2-1} = \frac{1}{49} \cdot \sum_{k=0}^{6} (k+1) = \frac{28}{49} = \frac{4}{7}$$

therefore,

$$\mathrm{E}[D] = 40 + 80 \left(\frac{1-p}{p}\right) = 40 + \frac{3}{4} \cdot 80 = 100 \text{ ms}$$

which fits perfectly in the confidence interval of the statistic that we wanted to explain (99.672 ± 0.737 ms), consequently, even in this scenario, we can say with 99 % confidence that the system behaves as expected.

Moreover, as the number of terminals N goes to infinity, p converges to $1/7$, therefore, we are also able to say that, under the assumptions stated at the beginning of this section, an upper bound for the mean delay that a terminal can expect to measure, shown in Figure 12, is:

$$\mathrm{E}[D] = 40 + 80 \left(\frac{1-p}{p}\right) = 40 + 6 \cdot 80 = 520 \text{ ms}$$

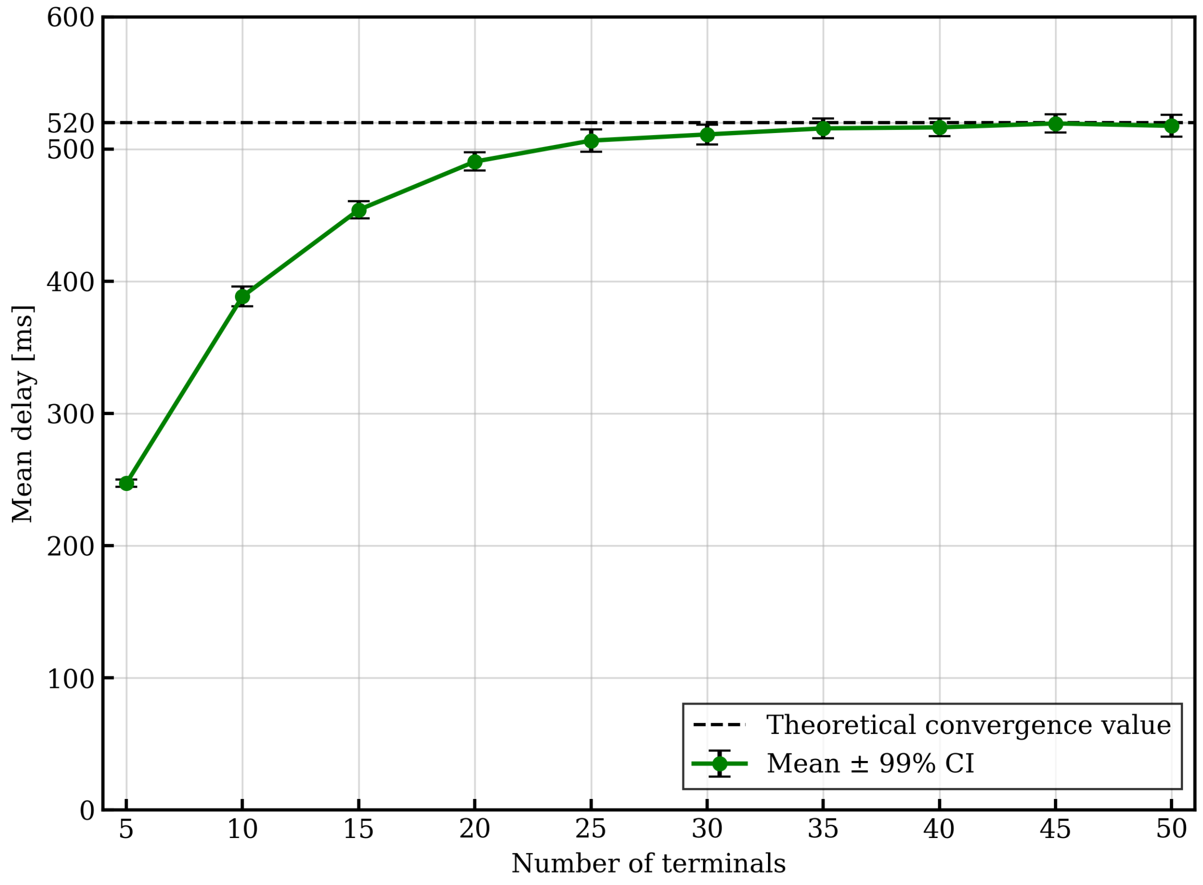

Figure 12: Mean delay that a terminal can expect to measure, as a function of the number of terminals. The tests were conducted in the scenario reported below.

Parameter/Factor	**Value**
sim-time-limit	80 s
repeat	30
codingRateDistribution	Uniform
terminalCount	5 – 50 (step 5)
blocksPerFrame	1
minPacketSize	1 byte
maxPacketSize	1 byte
meanPacketInterarrivalTime	10 ms