# Data Link Layer Protocol Simulator User Guide

Emmanuel Nataf

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

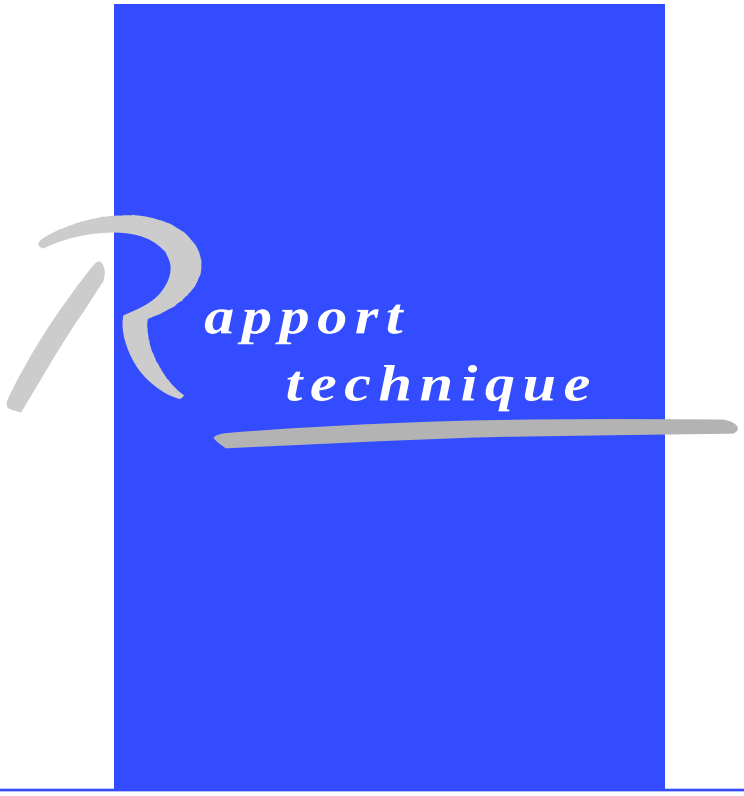
# *Data Link Layer Protocol Simulator*
# *User guide*

Emmanuel Nataf

**N° 6417**

Janvier 2008

Thème COM




ISSN 0249-0803 ISRN INRIA/RT--6417--FR+ENG

# Data Link Layer Protocol Simulator
# User guide

Emmanuel Nataf*

Thème COM — Systèmes communicants
Équipe-Projet Madynes

Rapport technique n° 6417 — Janvier 2008 — 15 pages

**Abstract:** The data link layer protocol simulator allows to write protocol and to see them in execution. This guide describes a step by step approach from simple information communication without any control to windows protocols. A protocol oriented java API is defined to write protocol algorithms.

**Key-words:** network, protocol, data link, simulator, java

* Maître de conférence - Université Nancy2

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

# Simulateur de Protocoles de la Couche Liaison de Données

**Résumé :** Le simulateur de protocoles de la couche liaison de donnée permet de programmer des protocoles et de les voir s'animer afin de les tester. Ce guide décrit une progression pédagogique allant du transfert d'information sans contrôle aux protocoles fenêtres. Une interface de programmation java est dfinie afin de permettre l'criture des algorithmes des protocoles.

**Mots-clés :** réseau, protocole, liaison de donnée, simulateur, java

# 1 Introduction

Data link protocols are used to transmit data between two network nodes (computers, switches, router, ...). In the following we consider one data sender node and one data receiver node. In between there is the network with this statement:

> *Any data sended to the network could be lost.*

Data link protocols are created to work against this hypothesis. They should ensure that :

1. receiver has well receive all data sended;
2. sender is acknowledged of data reception.

In the following, we will start from simple data transmission without any control until complex protocols that encompass theses constraints.

## 1.1 Simulator use

The simulator is written with java language and the `simullc.jar` file contains all needed classes, especially graphical ones.

There are two java files for each protocol, one is called `Sender`*NameOfTheProtocol*`.java` and the other `Receiver`*NameOfTheProtocol*`.java` (ex. `SenderNoProtocol.java`). Sender and Receiver protocols are implemented in these files. In order to test sender and receiver they need to be compiled with the graphical interface (one can use the files `compile.bat` or `compile.sh`. To launch simulation one should use the file `launch.bat` or `launch.sh` like :

```
launch <ProtocolName> [-m "sended message" ] [-d timeout in second]
                                             [-b window size]
```

for example

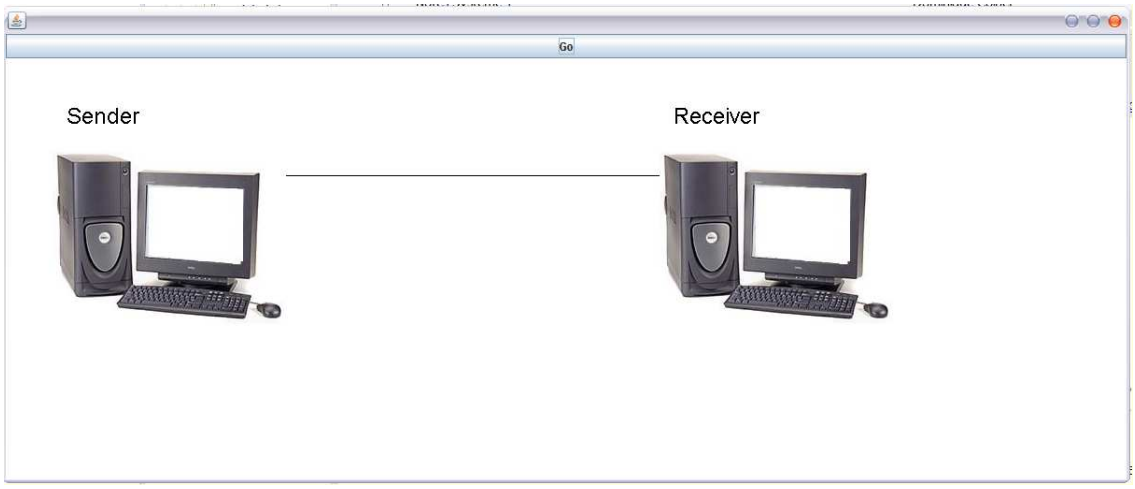
```
launch AnticipationWindow -m "anticipation window"  -d 15 -b 12
```

`<ProtocolName>` should be :

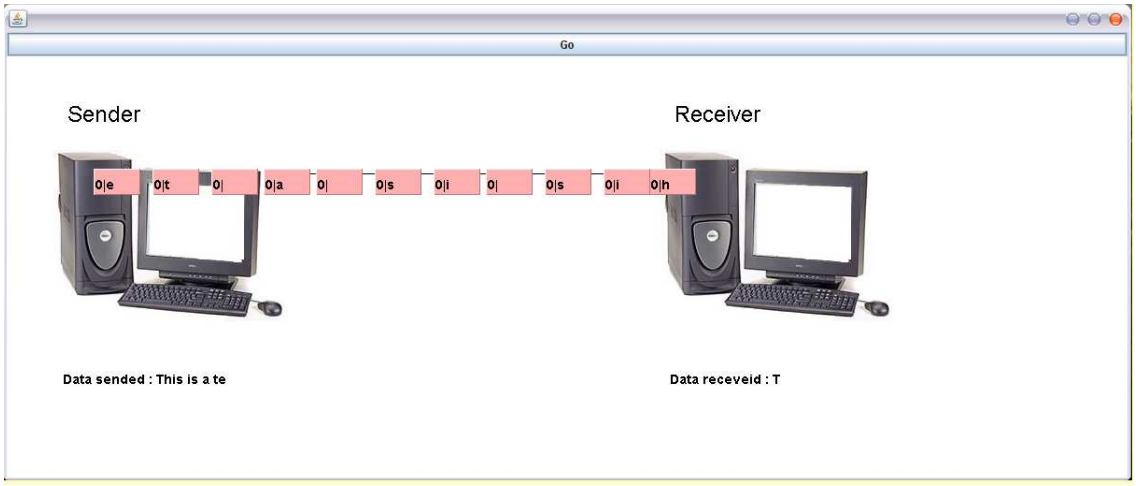
- `NoProtocol`
- `BitAlternate`
- `StopAndWait`
- `SlidingWindow`
- `AnticipationWindow`

Other arguments are optional and default values are defined ("Test Message", 10 s. timeout and a window size of 8 trames). Timeout and window parameters are unusable for `NoProtocol` and `BitAlternate`. The window size is only used in `SlidingWindow` and `AnticipationWindow`. Window size is necessary an even number (the simulator adjust if an odd number is given).

At execution time, the simulator shows the following panel :




Sender and receiver start once the `Go` button is pressed. Frames are visualized on the link between hosts. An animation moves frame from Sender to Receiver.




Any frame could be lost by the network. One can lost a frame with one mouse click on a moving frame that will clear it.

## 2 No protocol link

This first protocol only sends and receives one message without take care of frame lost. The figure 1 shows that a message is sended char by char. At the end of the message a dedicated char : `#` is added to signal the end of the transmission.

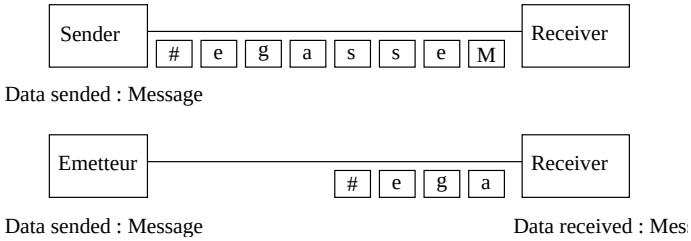



Figure 1: No Protocol link

## 2.1 Java API

In order to write this simple message transmission inside the simulator the following API must be used :

| | |
|---|---|
| `Frame readData()` | Each call of this method create a Frame instance with one char inside. The method manages an internal pointer to read the message from the first char to the last. When the end of the message is reached, any call of this methode creates a Frame with the `#` char inside. |
| `void sendFrame(Frame t)` | Send the Frame on the wire. |
| `Frame receiveFrame()` | Stop the calling thread until a Frame is received. This method return the received Frame. |
| `char getData(Frame)` | Return the char inside the Frame. |
| `void writeData(Frame)` | Print the char inside the Frame on the simulator panel. The printing position is automatically incremented and there is no erasing. This method is used to show which data are sended or received. |
| `void display(String mes)` | Print an information message under the writeData printing (police size and color are different). When called several times, the last display call clear the string of the preceding display call. |

## 2.2 Exercice

Write sender and receiver protocol with this API and `SenderNoProtocol`, `ReceiverNoProtocol` java files. The sender must write any data sended but the last `#` and display an end of transmission message. The receiver must write any data received and also display an end of transmission message.

# 3 Bit Alternate Protocol

With the lack of control of the precedent exercice, any frame could be lost without detection from the sender or the receiver. The Bit Alternate protocol adds an extra information inside each frame. This information is either the 0 or 1 value. The figure 2 shows an example of message transmission.

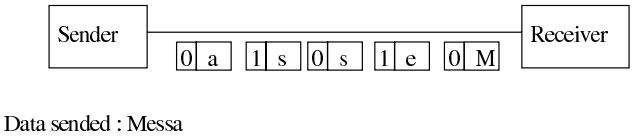



Figure 2: Bit Alternate Protocol

## 3.1 Java API

New methods are needed to write the Bit Alternate protocol :

| | |
|---|---|
| void markFrame(Frame t, int b) | Set the integer parameter inside the Frame. |
| int extractFrameNumber(Frame t) | Extract the integer from the given Frame. |

## 3.2 Exercices :

1. Write sender and receiver for the Bit Alternate protocol. Receiver must signal any lost of frame.
2. In which case frames could be lost without receiver detection ? How improve such issue ?
3. One need to have messages with the `#` inside.
   (a) What is the trouble ?
   (b) Use the escape char `\`[1] that will be added before sending the frame with the `#` when this is not the end of the message. The sender could use the following method :

   | | |
   |---|---|
   | boolean isEnded() | return true if the last char of the message has been already read. |

   The sender should print any data sended (escape char also) and the receiver will only print chars of the initial message (without escape char).
   (c) Test with following messages :

```
One # inside
#box#
One \ inside
the \# = end
The end ?\#
```

[1] Be careful that the \ char has an interpretation too in java

(d) What could happen if a frame with the escape char is lost ?

# 4 Stop And Wait Protocol

The Stop And Wait protocol is safe because each time the sender sends a frame it waits for an acknowledge frame from the receiver. The receiver must sends such acknowledge frame for each received frame. It also uses the bit alternate protocol.

The figure 3 details some steps of this protocol. The sender waits after sending 'M' frame. The receiver sends an acknowledge frame with the 'A' char when it receives the 'M'. The number of the frame and its acknowledge frame must be the same.

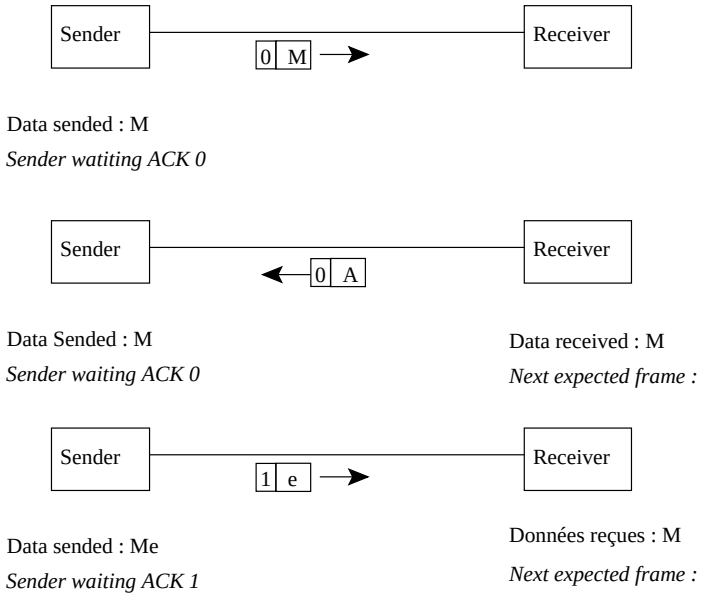



Figure 3: Stop and Wait Protocol

The frame number is usefull when a frame is lost. The figure 4 shows all possibilities (from a to g) . First, if the sended frame is lost, the sender will send again the same frame after a while. Such event is generally called a *TIMEOUT*. If the acknowledge frame is lost then the sender sends again the frame and the receiver must return an acknowledge but not accept the frame as part of the message because it knows that it has already received the frame thanks to the frame number.

## 4.1 Java API

There is a Frame constructor need for the receiver API :

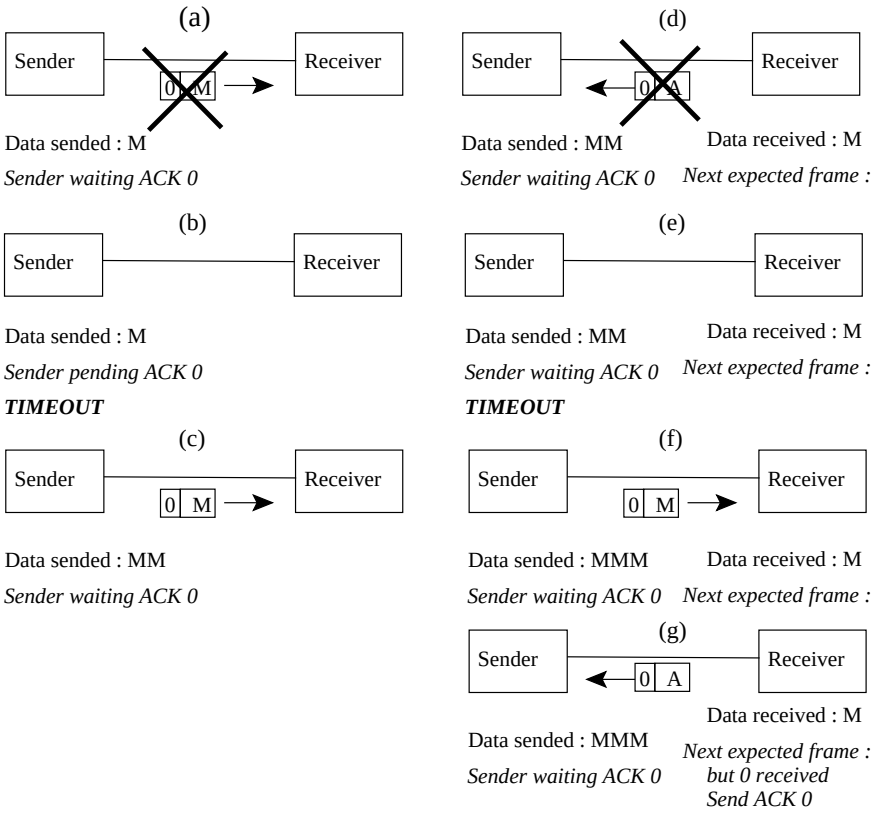



Figure 4: Stop and Wait protocol and frame lost

| | |
|---|---|
| `Frame Frame(char data)` | To create an acknowledge frame (usually with the 'A' char inside). |

There is a new method for the sender API :

| | |
|---|---|
| `Frame receiveFrame(int delay)` | Stop the calling thread until a frame is received or the delay time is reached. The delay is a number of seconds (or use the `delay` attribute). |

The use of this method is twofold because either the frame is received before delay time either it is not. The way to use this method is the following :

```
try{
   Frame t = receiveFrame(delay);

   //... frame is received before delay seconds


}catch(TimeoutException t){

  // ... no frame is received after delay seconds

}
```

### 4.2 Exercice :

1. Write the Stop And Wait protocol. Sender should print information if timeout occurs and receiver should display an error message when it receives a frame without the expected frame number.

2. What will happen if the network is broken (any frame are received)? Program the sender in order to prevent an infinite loop of send and timeout.

## 5 Sliding window protocol

A limitation of the Stop And Wait protocol is waiting time after each sended frame, especially if transmission are on long link. The main principle of the Sliding Window protocol is to send several frames and, after, wait their acknowledge.

An example on the figure 5 shows the interest of this protocol. Sender sends several frames and starts waiting each time. The receiver sends an acknowledge frame for any received frame. The network bandwith is better used and the message transmission time faster than with the Stop And Wait Protocol.

What are the differences of this protocol between the Stop and Wait? First, it needs to have more than one bit to count frames and second, it needs to store each frame that are sended but no yet acknowledged. This is to be able to send again a lost frame (or its acknowledge). In the following we use eight values (from 0 to 7, see figure 6) to count frames. A circular buffer is used to store sended frames and when this buffer is full, the sender should wait before sended more frames. The receiver should ackwoldege each received frame with the same frame number. This acknowledge allows the sender to release a slot

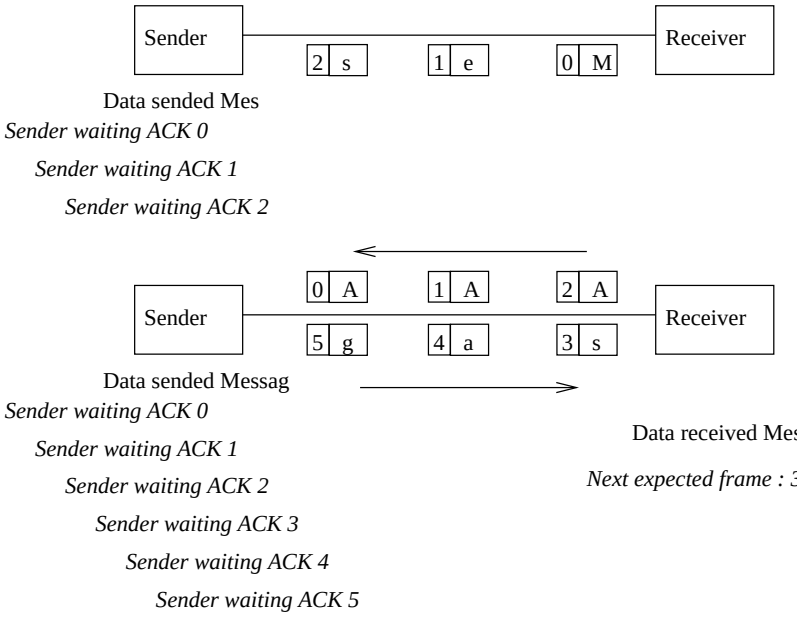



Figure 5: Protocole *Sliding Window*

in its circular buffer. So it could send one other new frame. The receiver must acknowledge any frame but must only accept the expected frame and so it must know the sender buffer size and the starting frame number.

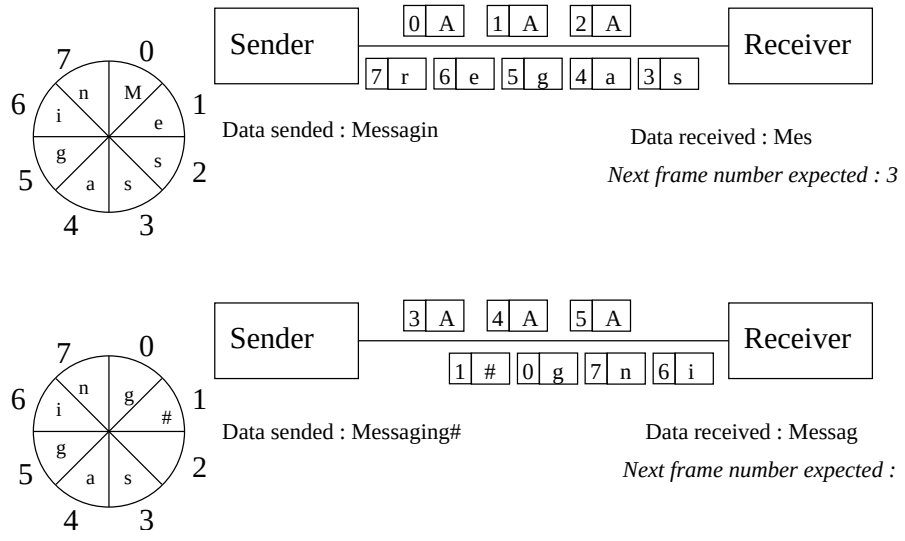



Figure 6: Circular buffer of the Sliding Window protocol

This protocol il safe against frame lost as the Stop And Wait protocol. A frame lost leads to a timeout and so sending of the same frame. An acknowledge

lost will do the same and, as the receiver knows which frame number it waits, there will be not possible to have two identical frames accepted.

However if the sender use all its buffer slots then the lost of the last acknowledge leads to a mistake. The figure 7.a shows such a case. As it was not acknowledged the sender send again this frame (7.b). When it comes to the receiver it will accept it as it care the good number 0 and finally displays the message "MessaginM". A solution to avoid it will be to not full the sender buffer but have always at least one free slot.

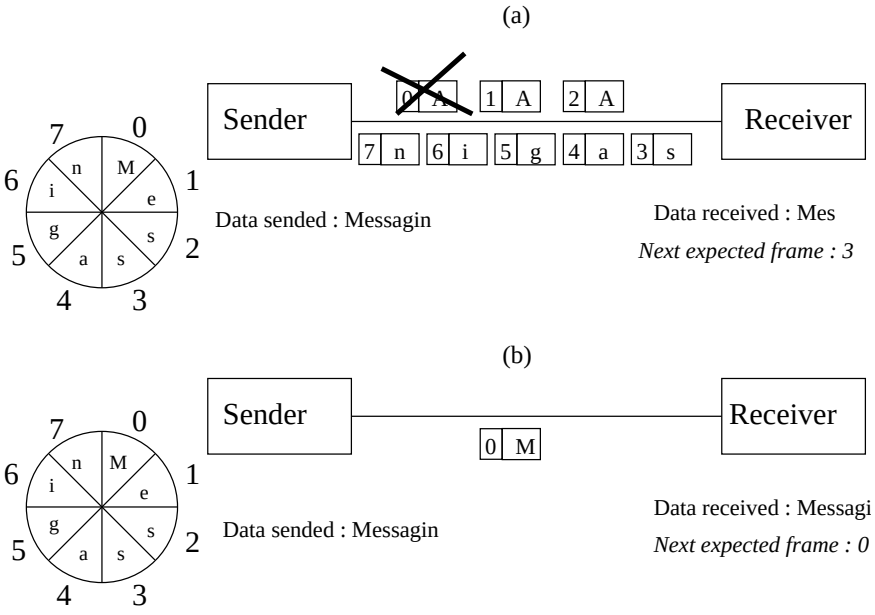



Figure 7: *Sliding Window* protocol limitation

## 5.1 Java API

A Buffer class is needed to program the Sliding Window protocol. The figure 8 shows that a Buffer has two pointers b_inf and b_sup (for the inferior and superior bounds). Copies of sended but not yet acknowledged frames are stored in the buffer from the b_inf position and a copy of the next new frame to send will be put at the b_sup position. In the figure the buffer starts from an initial state to the sending of two frames and the reception of their acknowledge.

Following methods are to use with the buffer (there is no need to explicitely create a Buffer as there is already one in the Sender class) :

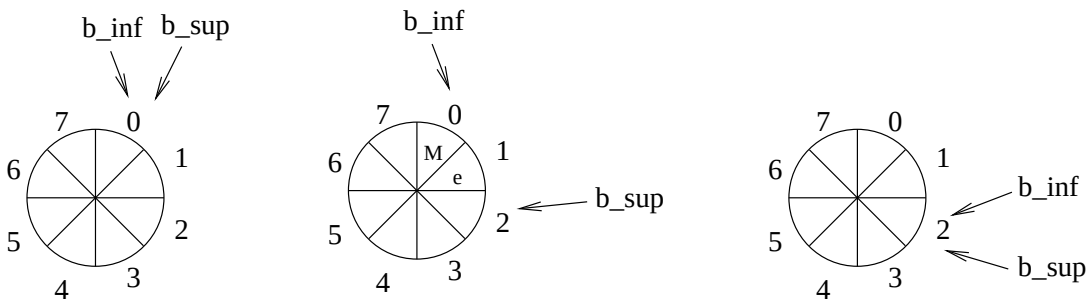



Figure 8: Operation on *buffer*

| | |
|---|---|
| `int getInf()` | Return the inferior bound of the Buffer. |
| `int getSup()` | Return the superior bound of the Buffer. |
| `void incInf()` | Add one to b_inf modulo the Buffer size. |
| `void incSup()` | Add one to b_sup modulo the Buffer size. |
| `void receiveFrame(int delay, Frame t)` | Stop the calling thread until an acknowledge Frame of the given Frame is received or if a timeout occurs (after delay seconds). |
| `Frame getFrameAt(int i)` | Get the Frame at the given position in the Buffer. |
| `void ackFrame(int i)` | Remove the Frame at the given position in the Buffer. |
| `void putFrameAt(Frame t, int i)` | Put the Frame at the given position in the Buffer. |
| `void waitFreePlaces(int i)` | Stop the calling thread until there is at least as much free buffer places than the given integer. |

Inside the simulator a class called `Acknowledge` should be used to wait the reception of an acknowledge. An instance of this class is automatically created each time the `SendFrame(Frame t)` method is used[2]. At one time there could be several instances of this class around one buffer instance. Concurrent access is managed by the buffer.

## 5.2 Exercice :

1. Write sender and receiver for the Sliding Window protocol. There are two parts in the sender. One is for sending frames and filling the buffer and

[2]this method has been overloaded in order to do so

the other one is for acknowledge reception. The receiver could use the `buffer_size` attribute.

2. Be sure two identical frames could not be accepted.

# 6 Anticipation Window Protocol

The Anticipation Window protocol improves the Slinding Window. It allows the receiver to accepts frames that are not exactly the next one expected. If one frame is lost but not the following then it will be more efficient to store these following frames pending the sender sends the missing one.

The figure 9 shows an example of such protocol. As in the Sliding Window, the sender sends several frames and stores a copy of each one in its buffer. However the receiver is not waiting for one frame but for any frame with a number from 0 to 7. It is why the receiver also must have one circulary buffer of the same size than the sender. In the top part of the figure three frames was correctly received and so the receiver is waiting frames number 3 to 2. Suppose now that the frame number 4 is lost. The bottom parts of the figure shows

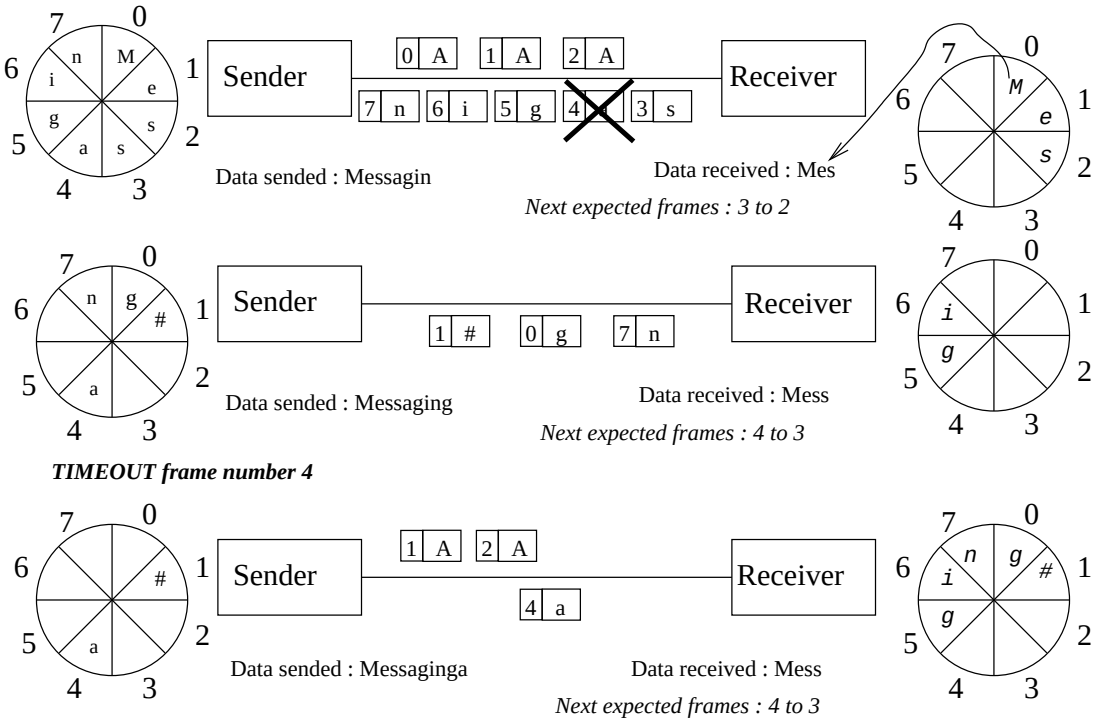



Figure 9: Protocole *Anticipation Window*

what going on. First the sender continues to send frames and release buffer slots upon acknowledge receptions and second the receiver has accepted frames after the number 4 but has kept them in its buffer as it has not received the

frame number 4. When this frame is sended again and (this time) received then all the buffer frames from number 4 to 2 are emptied.

The figure 10 shows a case where a trouble occurs. If the acknowledge number 4 is lost and the sender continues to send frames. The receiver reception window moves from 5 to 4 waiting numbers to 4 to 3 wainting numbers. When timeout occurs at the sender side it sends again the frame number 4 and the receiver could not know that this frame was identical with the previous frame number 4. In order to solve this problem the sender should not send too much

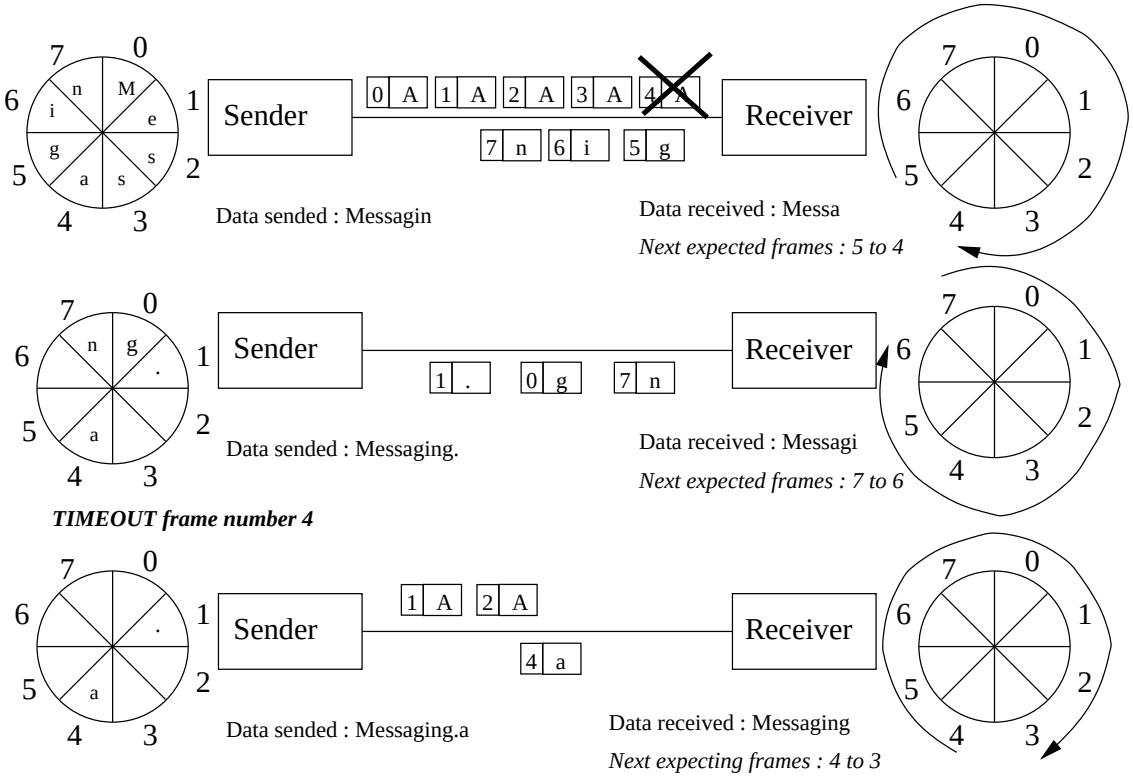



Figure 10: Limite du Protocole *Anticipation Window*

frame in advance. Indeed when the sender has send frames after the number 4, the receiver has moved its reception window until overlaps the number 4. If the sender has limited itself to not send more than three frames after the 4 then the receiver would not accept the second frame number 4 because it was not inside waiting numbers (but just send an aknowledge to release the sender). There will be no trouble if the sender stops its sending after has sended half of the buffer size frames (not acknowedged) in order to not overlap sending and reception windows.

## 6.1 Java API

Following methods are to be used to write the Anticipation Window protocol:

| | |
|---|---|
| `int getBufferSize()` | Return the buffer size. |
| `boolean inWindow(int inf, int sup, int num)` | Check if a frame number is inside the anticipation window. |

## 6.2 Exercice :

1. Write the Anticipation Window protocol.
2. Write the safe version of the Anticipation Window protocol.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifi que de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803