Ivan Luque, u233837
Guillem Martínez, u232673
G1

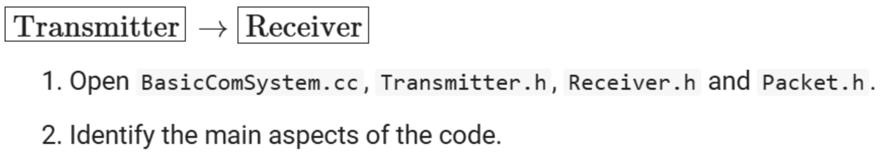
Lab 1 - A Basic Communication System

This is the first lab of Networks Engineering and mainly consisted of getting familiar with COST, a simulation framework in C++.

Exercises:

---

1. **Identify the main parts of the code:**



$\boxed{\text{Transmitter}} \rightarrow \boxed{\text{Receiver}}$

1. Open `BasicComSystem.cc`, `Transmitter.h`, `Receiver.h` and `Packet.h`.

2. Identify the main aspects of the code.

- **BasicComSystem.cc:**
  - The first part of the code does all of the #includes.
  - The next part is a component called BasicComSystem which is set to be public. What follows is a small section where we assign the methods: Setup(), Stop() and Start() all as void which we will be looking at later.
  - Immediately after that we also redefine Transmitter as tx and Receiver as rx.

```
C/C++
component BasicComSystem: public CostSimEng{
        public:
                void Setup();
                void Stop();
                void Start();
        public:
                Transmitter tx;
                Receiver rx;
};
```

  - Next up, we look at the methods mentioned before starting with Setup() where we simply connect tx and rx with tx being the out and rx being the in.

```
C/C++
void BasicComSystem:: Setup(){
```

Ivan Luque, u233837
Guillem Martínez, u232673
G1

```
        connect tx.out,rx.in;
};
```

- The other two methods (Start() and Stop()) simply contain prints indicating the beginning and the end of the process.
- Finally, we have the main which simply sets a seed for the RNG, a stop time for the simulation, and runs the simulation.

```
C/C++
int main(int argc, char *argv[]){
        BasicComSystem test;
        test.Seed = 666;
        test.StopTime(1);
        test.Setup();
        test.Run(); // The simulation starts!
        return(0);
};
```

- **Packet.h:**
    - This file simply defines a structure named packet wich contains: a length, a sequence number, an integer that is zero if the packet contains no errors and a transmission time.

```
C/C++
struct Packet{
        int L; // packet length
        int seq; // sequence number
        int errors;
        double send_time; // transmission time
};
```

- **Transmitter.h:**
    - The code starts by defining a max sequence rate as 1024.
    - Right after that, we create the component Transmitter which is again set to be public and starts by defining the import, the outport, a timer for the transmission delay of the packet and a constructor method which simply connects the inter_packet_timer to the packet received in the import.

```
C/C++
#define MAXSEQ 1024
component Transmitter: public TypeII{
	public:
		// Ports
		inport void new_packet(trigger_t& t);
		outport void out(Packet &packet);
		// Timer
		Timer <trigger_t> inter_packet_timer;
		// Connection
		Transmitter(){
			connect inter_packet_timer.to_component,new_packet;
		}
```

- Next, we create the variables for the length and sequence number for our new packet as well as another variable for the packet rate.
- And immediately after we also assign the methods: Setup(), Stop() and Start() all as void which we will be looking at later. In this case, the Setup() and Stop() methods are completely empty whilst the Start() method contains two lines of code that set the timer for the packet and set the sequence number to zero

```
C/C++
void Transmitter:: Start(){
	inter_packet_timer.Set(Exponential(1/packet_rate));
	seq = 0;
};
```

- The final part for the file is the main method for the Transmitter file which simply sets all of the variables for the packet that we just created, sends the packet, does a print confirming the sending of the packet and increases the sequence number.

- **Receiver.h:**
  - The receiver.h file starts by creating the component Receiver which is again set to be public and starts by defining the import and two variables to keep track of the packets received and the packets received with errors
  - Also, just like in the other cases, we also define the methods: Setup(), Start() and Stop().

```
C/C++
component Receiver: public TypeII{
	public:
		inport void in(Packet &packet);
	public: // Statistics
		long int num_packets_rx;
		long int num_packets_rx_with_errors;
	public:
		void Setup();
		void Start();
		void Stop();
};
```

- As for what the methods contain, the Setup() method is empty,the Stop() method simply contains prints informing about the number of packets with errors and the Start() sets all of the relevant variables to zero before execution.
- Finally, the last thing in this file is the main method for the file which calculates the amount of packets with and without errors.

2. **Implement the new component called channel and modify BasicComSystem to accept inputs, and investigate how the simulation time affects the measured error probability:**

- **Channel.h:**
    - The file new file that we created called Channel.h starts similar to the Transmitter.h and Receiver.h where we begin by creating a public component called Channel, where we will set the import, the outport, a variable that represents the probability of error on any packet and the methods: Setup(), Start() and Stop().

```
C/C++
component Channel: public TypeII {
	public: //Ports
		inport void in(Packet &packet);
		outport void out(Packet &packet);
		double Pe; //Prob Error
	public:
	void Setup();
	void Start();
	void Stop();
};
```

- In this case, all of the three methods are completely empty so there is no code to talk about there.
- Finally, the file ends with the main function which takes care of rolling a random number and comparing it with the probability of error in order to randomly assign packets with errors.

```C/C++
void Channel:: in(Packet &packet){
      if ((double)rand() / RAND_MAX <= Pe){
            packet.errors = 1;
            }
      out(packet);
};
```

- **BasicComSystem.cc (Changes):**
    - The main changes are:
        - Adding an extra #include for the channel.h file.
        - We create the two new variables: rate and pe which will keep track of the inputs for the packet rate and the probability of error
        - Changing the connection from transmitter to receiver to go from transmitter to channel and from channel to receiver.
        - We set variable rate from transmitter to be equal to the rate that we just created and we also set the variable pe from channel to be equal to the pe that we just created
        - We create a new variable called dur which will correspond to the input for the total runtime of the code.
        - We set the three variables dur, rate and pe to be equal to the input.
        - we change the test.stoptime to have a stop time equal to dur.

- **Investigate how the simulation time affects the measured error probability:**
    - As the simulation time increases our measured error probability approaches our input error probability because we are getting more sample data to do the calculation.