EPCC-SS97-11

Intersim-Java - A Modular Internetwork Simulator

Hernani Mario Ferreira Fontes Pedroso

Abstract

The growth in network traffic, especially in the Internet, is enormous and is leading to the collapse of some links. The size of the computer networks, their heterogeneity and their intrinsic dynamic behaviour makes them difficult to understand. Simulations are used to investigate the way networks respond to the requests of their users. A network structure is composed by several entities, the diversity of these ranges from the traffic links, through the protocols that inter-operate over the traffic links to the applications that are communicating, each one as a different task. The number of possible combinations of connecting the several entities provide the existence of a wide range of complex network structures. Instead of trying to simulate the network as a whole, a modular simulator uses separated simulations of each component and connects them to form the required structure. This approach enables the creation or modification of a structure by reusing the already existence simulated components. Such modular simulation models tend to require more detail in their implementations, this makes the use of HPC necessary for serious experiments.

Contents

1 Introduction

The development of computer networks has led to several new ways to communicate (Electronic mail, video-conference, WWW among others). The use of these kinds of communications has been growing continously and now they have an important role in academia, industry and commerce. One good example is the World Wide Web which has achieved unforseen popularity in last few years. With this rapid growth of the Internet and its increasing importance to today's world, the problem of making the most efficient use of the available bandwidth becomes more acute.

In the early days of computer networks each company or institution created their own standard and with the almost chaotic growth of networks some became *de facto* standards. Although institutions like IEEE [2] and ANSI [1] defined some formal standards, they were basically following the already existent *de facto* standards. Many standards (formal or not) impose their own unique topological constrains. In addition the number of nodes (computers, routers or other devices that use the network) can vary and can be changed easily. All these different types of networks connected together create a large and highly unstructured topology, with the dynamic nature of their operations, makes congestion problems hard to understand.

The Transmission Control Protocol (TCP) [4] is a protocol that is situated in the transport layer of the OSI model from ISO [5], which is the layer that makes the interface between the network layer and the session layer. TCP implements a reliable communication between both of the intervening ones; this characteristic is the major reason for its popularity since most of the network protocols and transport protocols do not guarantee delivery or the correct order of the packets. This protocol was originally implemented above the Internet Protocol (IP) [3], though it is possible to find implementations above other protocols. The IP protocol is the most widely-used network protocol; often its name is used to refer to the world wide computer networks, although the communications are accomplished through several different protocols. TCP and IP protocols are so commonly used together that usually they are referenced as one: TCP/IP.

Researchers are resorting to simulation techniques to better understand the network traffic in order to be able to improve the existing technology and address the increasing demand of bandwidth. The purpose of this project was to create a software package to create network simulations easily. Since the TCP protocol is the most widely-used transport protocol and is responsible for most of the Internet traffic, another simulation package was built especially to simulate this protocol. The TCP simulation package should be very flexible in order to be able to simulate new features of the protocol or simulate new versions. Such a modular simulator requires more computational resources than a monolithic simulation. Thus the use of HPC might be required for bigger and more detailed experiments.

The reader should understand that the aim of this report is to describe the network simulation packages and the reasons for the choices that were made. Although the reader will find some information about computer simulation and computer networks throughout the report they should have some knowledge in both fields to fully understand this project and the results achieved.

2 Background

The number of hosts connected to the Internet has increased almost exponentially. It is estimated that at the end of 1997, the Internet will have more than 30 million hosts connected together, compared to 10 million in January of 1996. Following this extremely rapid increase in number of hosts is the evolution of computer applications with more interactive interfaces that increasingly use more multimedia features and the growing amount of information that can be found in the Internet (especially in the WWW). These new conditions impose a huge amount traffic, e.g. statistics of JANET [6] about traffic exchange between UK and USA in April of 1997 show that were exchanged 6.7 Terabytes of information. This huge amount of traffic that networks have to deal with is imposing problems on the existing networks structures. Most of the network links, especially international and intercontinental links, are overloaded with traffic and have very poor throughput. Since upgrading the links is very expensive, the research community is seeking new solutions to accomplish the demand for increased bandwidth of network applications.

Although the IP and TCP protocols, respectively, dominate the network and transport layers at the physical level a wide range of protocols exists. Physical protocols are divided in to several categories: LANs (Local Area Networks), MANs (Medium Area Networks) and WANs (Wide Area Networks). In each category several competing protocols exist in the market. The existence of all these different protocols is making more difficult to understand the network problems and where improvements should be introduced. The simulation of computer networks vary considerably with the protocols that are used, and sometimes several types of topologies have to be experimented on with the same set of protocols. To simulate networks with their different protocols and topologies easily a modular simulation package is needed.

Since not all network level protocols guarantee the correct delivery of data, when a new network application is created the program designer has to decide between using a dependable communication protocol and implementing in the application itself the communication guarantees. Although the TCP protocol is dependable it introduces an overhead in communications. To create dependability in communications TCP is connection oriented; so to send even a single byte of data from one computer to another at least 7 packets have to be exchanged to manage the connection. UDP (User Datagram Protocol) [7] is a transport protocol that does not guarantee delivery, order of delivery or non-appearance of duplicated data in the destination of the communication, but is faster then TCP. The choice between TCP and UDP depends normally on the nature of the communications needed, some times the overhead of TCP does not pay-off against the dependability of the communication. Applications that need to transfer big amounts of data usually use TCP; passing to the protocol the responsibility of delivering the correct data to the destination in the correct order.

In 1986 Jacobson [8] analysed the first of the Internet 'congestion collapses'. He discovered that TCP was responsible for some of the congestion problems, and it could be improved to deal with them. Since then several improvements were introduced in the protocol [9, 10, 11] and now congestion control mechanisms are considered one of the strengths of the TCP protocol. As Brakmo et al. [12] demonstrated with the new version of TCP, known as TCP Vegas, new improvements can still be introduced in the TCP protocol in order to increase its ability to avoid congestion in the networks. A modular simulator of the protocol can help researchers to seek new features and improvements test them and compare the results with the performance of the original versions.

Since one of the proposed characteristics of the simulators created throughout this project is modularity, the use of a modular language was required. The language chosen was Java [13], which is intrinsically modular. Choosing Java to be the programming language, the simulator also benefits from language portability and the potential for remote execution through applets.

Networks and network protocols can be modelled as discrete event-driven system, since creating a simulation engine was not part of this project Simjava [14] was used. It is a discrete event simulation library written in Java that was developed in the Department of Computer Science of the University of Edinburgh. An attractive part of Simjava is its support for animated models of simulations.

Although Simjava is written in Java it was based on the HASE++ library [15] and porting the applications from Simjava to HASE++ is a easy task. HASE++ is a parallel simulation library written in C++ which enables the users to run more simulations more efficiently.

3 Introduction to the Simjava Package

Simjava is a process based discrete event simulation package written in Java. The package is basically a Java port of HASE++ [15] which, itself, is based in the Jade's SIM++ [16], but with animation facilities. Also using the properties of Java the Simjava package is able to run remote simulations through a Web browsing.

A Simjava simulation is a group of entities connected together by ports and communicate with each other by sending and receiving event objects.

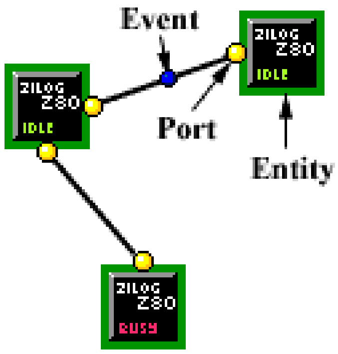


Figure 1: Illustration of a Simjava simulation.

A central system class controls all the threads, advances the simulation time, and delivers the events. The progress of the simulation is recorded through trace messages produced by the entities, and saved in a file.

Each entity in the simulation is a instance of an object of one class that extends the `Sim_entity` class. To create a simulation with Simjava the user must first implement the entities (their behaviour) that will form part of the simulation. When all the entities classes are implemented the user must create a main class with the `main()` method. Inside that method the user initialise the simulation (`Sim_system.initialise()`) and add the instances of the entities (`Sim_system.add(entity)`). Once the entities are added their ports must be linked (`Sim_system.link_ports()`) to communicate. Then the method `Sim_system.run()` starts the simulation.

More information can be found in:
http://www.dcs.ed.ac.uk/home/hase/simjava/simjava-1.2/doc/simjava_guide/index.html

3.1 Interception of Future Events

Currently an entity can only access past events. Enabling entities to get events that only "appear" in future simulation time may not have an obvious use. But it is a significant help when working with timeouts.

A timeout can be set by creating an event for itself to occur at the appropriate future timein the timeout time. Once the event arrives it triggers the timeout. Frequently times once the timeout is set it can become superfluous if the monitored action eventually succeeds. In this case the applications must inactive the timeout or ignore it on its arrival. To ignore it the application need to be testing all the arriving timeouts to see if they are valid or not. The alternative is to allow the application to remove the timeout when the target action is completed and all arriving timeouts are true timeouts. The second option is easier to implement and faster, since both the simulation engine and the application are saving processing time. The simulation engine does not need to process useless events and the application does not need to keep track or test all the timeouts.

The current Simjava package is able only to get at events that already have reached by the entity (deferred queue). To get future events from the future queue it was necessary to implement new methods. In the analysis of the code it was noticed that the deferred queue and the future queue were different instances of the same class `Evqueue`, which means they have the same functionalities. So, to get events from the future queue the proceedings are the same as to get them from the deferred queue. Some changes were then introduced to the simulation engine in order to enable it to get the future events. This changes were reported to Fred Howell so he can introduce them in future releases of the package. Two classes were changed, `Sim_system` and `Sim_entity`. No changes in the existent code were performed, only new methods were introduced. A better understanding of the prototype of the new methods are presented next.

Methods in the `Sim_entity` class:

```
public void sim_select_future(Sim_predicate p,Sim_event ev)
```

Extract the first event waiting for this entity on the future queue, matched by the predicate p.

```
public void sim_putback_future(Sim_event ev)
```

Put an event back on the future queue.

```
public int sim_waiting_future(Sim_predicate p)
```

Count how many events matching a predicate are waiting for this entity on the future queue.

Methods in the `Sim_system` class:

```
static public void select_future(int, Sim_predicate)
```

Basically it is equal to the `select(int,Sim_predicate)` method but works on the future queue instead of the deferred queue.

```
static public void putback_future(Sim_event)
```

Equal to the `putback(Sim_event)` method but works on the future queue instead of the deferred queue.

```
static public int waiting_future(int, Sim_predicate)
```

Similar to the `waiting(int,Sim_predicate)` method but works on the future queue instead of the deferred queue.

4 Intersim-Java Simulator

The implementation of a general network simulator was one of the objectives of this project. The simulator should be as modular as possible to enable the user to created a wide range of network types.

This chapter will provide an explanation of the difficulties found and options taken. The structure of the simulator; the interface of the package will be explained, and finally some brief examples will be presented.

4.1 Difficulties in Simulate Networks

Simulating networks is not a trivial task, mostly due to the their great heterogeneity. It is very difficult to implement a simulator able to simulate the wide variety of existing networks.

The structure of the networks may vary significantly from example to example. Although there exists a set of common structures (see Figure 2) configurations can vary a great deal, for example in the number of nodes and connections.

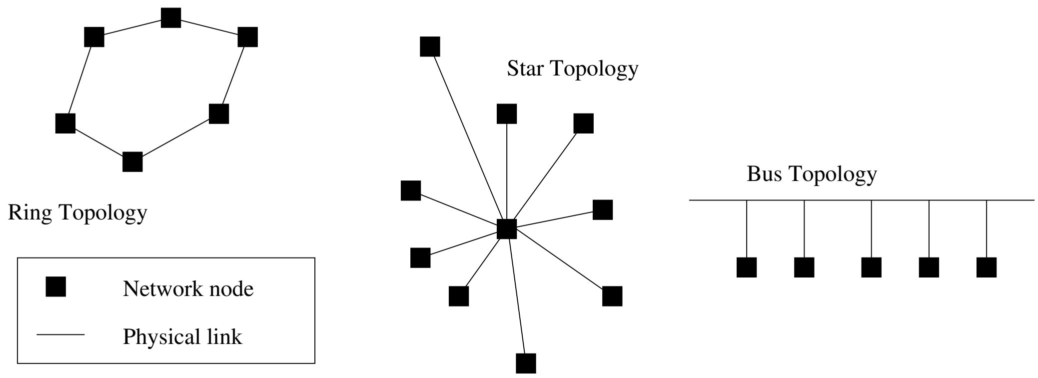


Figure 2: Principal subnet topologies.

If the simulator does not give to the user the freedom to modify built-in structures or create others from scratch, it is restricting the possible number of different simulations that can be done with it. Although it might be more difficult to use, to allow the user to simulate a wide range of networks simulations, simulators must be capable of leting the user to create their own structure.

The nodes have a set of protocols that inter-operate over the links to create a communication. This protocols are divided in layers according to their function. The number of layers that each node has varies significantly according to the type of node (e.g. router, bridge, terminal), type of communication (e.g. point-to-point or network based), and other factors. Figure 3 shows the standard OSI model of ISO which has 7 (seven) layers and other protocol structures that are used every day, showing differences that have to be considered when creating the simulator.

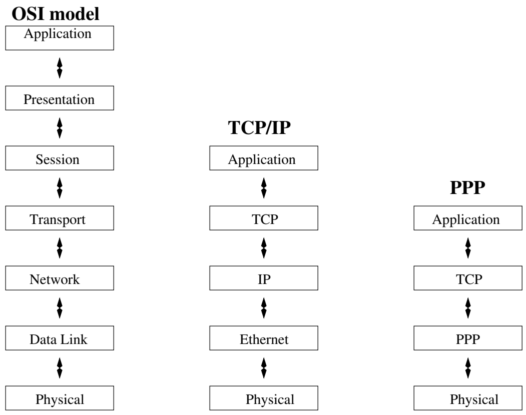


Figure 3: Different number of protocol layers.

But the number of protocols varies not only with the number of layers. More than one protocol may be working in the same layer, specially in the upper ones. If two protocols work in the same layer, although they have the same basic function, they do it in different ways with different characteristics. Figure 4 shows one of these examples.

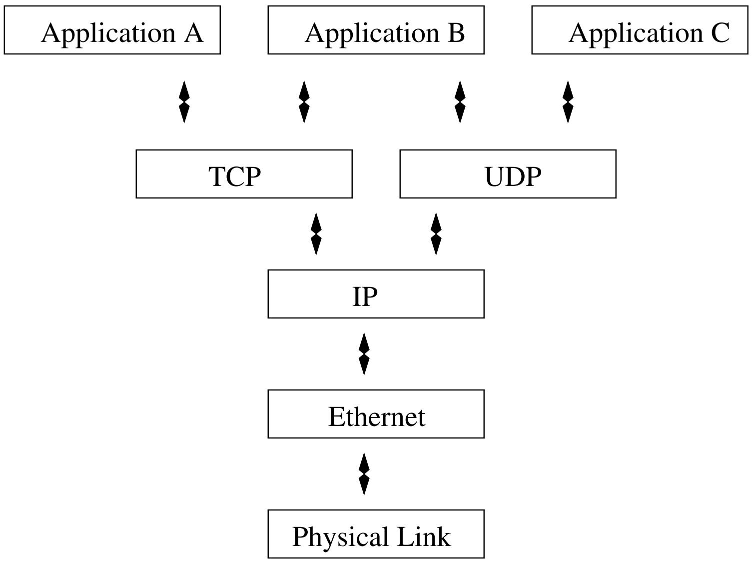


Figure 4: More than one protocol in the same layer.

The problem of restricting the range of possible simulations may not seem very important.

Although in many cases simulations are used to understand existing problems or verify the behaviour of one existing technology, they can also be used to test new protocols or verify the performance and feasibility of a new network structure. In these cases it is unknown which type of structure the user will need in the nodes or in the network. Creating a simulator with any structures is almost impossible due to the existence of a massive number structures and the number of their variable parameters.

Besides the type of network structure the simulation can also vary depending on the aim of the simulation. The user probably wants to simulate the link as an entity if the aim of the simulation is the Fast Ethernet protocol, and not only a delay as it might happen if the aim is the transport protocol. Thus in addition to the already very diverse types of networks and nodes the simulation may also vary according to its goal.

For these reasons it is impossible to create a rigid structure for the simulator; and also the variety of existing types make it impossible to find a unifying structure in the networks. Each time a structure was chosen for the simulator, it raised a series of problems. A modular simulator can be a solutions for must of these problems, though it might be a little more difficult to use.

In a modular simulator, the simulation is constructed with a set of entities connected together to form the desired structure. This entities may be protocols, applications, links or any other components of the network that the user might want to simulate. Until now this section has been focusing the issues of the global network simulation, now that is concluded that the simulation can only be done by connecting together small pieces, the rest of the section will focus on the problems of those.

Protocols are the biggest candidates to be entities. And once again the lack of a standard is a problem. More than one implementation exist for each protocol, and is difficult to find specific information about them. The Requests For Comments (RFCs) are the best source of information about the protocols, although sometimes they can only be considered the core of the current implementations. The documentation of the new features of the latest implementations is also difficult to find and sometimes is abstruse.

Another difficulty in connecting these protocols is to find the way they communicate with each other. During this project some research was made in this subject. The achieved results are not valid to every case, but for most of them. Protocols communicate by writing in memory and signalling the each other. Since the simulation uses events its easy to simulate this process by creating an event for the other entity with an object attached. The biggest problem is to know the number of parameters to pass. This differs from protocol to protocol and with the call that is made. In memory they are placed in continuous memory and the target protocol according to the type of the call knows where each one is. To simulate this, one event was created which can carry an variable number of objects.

4.2 The Structure of the Intersim-Java Package

When development started on this package, a rigid structure was designed for the simulator. The idea was to base the structure on TCP/IP protocols, over any type of network. But after some research and experiments were made this idea was being abandoned. As already has already been explained in the previous section, to use a rigid structure for the simulation is impractical, since the purpose of the project was to create a simulator with a considerable degree of modularity. So instead of having a structure, the simulation was divided in small parts that are joined together to create the simulation. This parts are called Intersim-Java entities (during this section Intersim-Java entities will be referred simple as entities).

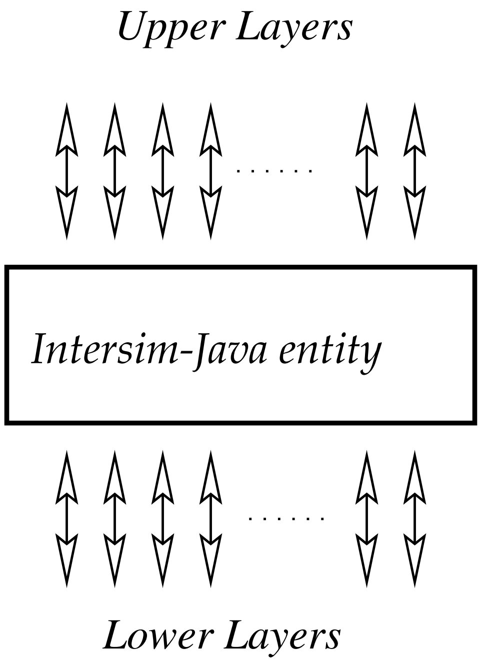


Figure 5: Intersim-Java entity.

Entities may represent any part of the network, a link, a communication protocol, an application or an whole node. Normally it depends on the simulation and its aim or degree of detail and on the behaviour of that part.

Once created these entities are connected together to create the required structure of the simulation. Each entity may have one or more connections with the upper layer entities and lower layer entities. The connections are created at the start of each simulation depending on the simulation. The user can abstract the simulation of that part (the entity) from its neighbours, and create an entity that can be used in several different simulations.

Entities can only communicate with entities that are connected to them and this communication takes place by means of events. The entity can send an event to any of its upper or lower layers at any time in the future. Also a broadcast is available to the upper or lower layer. Each event carries a number of objects that is defined when it is created, the receiver must know (or be able to infer) what represent each of the objects. Each entity has a unique name in the simulation which is set when the object instance is created. Also each entity has an identity that is used in the communication between the entities. When the user specify the destination of the event it uses this second identity. So, the identity does not need to be different for each entity, but must be unique within the set of entities connected to an entity. One of the examples of the utility of this identity is in the simulation of TCP/IP. When the IP receives a datagram it has to pass it to the correct entity above it. When receiving the datagram the upper entity which the event has to be sent to is specified there. Then the IP sends the event to the entity with the TCP identity that is connected to it. If each TCP entity has a different identifier for the communication, it would be very difficult to the IP protocol to discover to what entity send the event. Also is impractible to send a different entity name when communicating, a search for the correct name to the destination of that entity is computational expensive (time and memory) if the number of TCP simulation entities is big.

To initiate the simulation the user must create a Java application where the simulation engine is initialised, the entities are *added* to the simulation and connected according to the structure. In this stage the user must also set any necessary parameters of the entities. After everything is set the user must initiate the simulation engine.

Basically, the structure of the Intersim-Java package is similar to what is found in the Simjava package, but hard-wired to the network simulations. Due to its simplicity, it gives a considerable amount of freedom to simulate different types of networks and also to re-use the entities in several simulations.

4.3 The Interface to the Intersim-Java Package

The Intersim-Java package is composed by four classes ISJ_system, ISJ_entity, ISJ_event and ISJ_connection. The user has to know the first three, the last one is only used for internal purpose.

Through the next sections each class is introduced and their most important methods explained.

4.3.1 ISJ_system class

The `ISJ_system` class is the one that manages the simulation system. All the methods of this class are `static`, so there is no need to create an instance of it.

This class has the method to initialise the simulation system (`initialise()`), this method has to be invocate before the user start to constructing the structure of the simulation. The `run` method starts the simulation system, after calling this method the system gets in control of the simulation and works the entities and events has necessary.

To build the structure the user has two methods: `addEntity()` is used to add entities to the simulation system; `connect(entity_nameA, entity_nameB` connects two entities. The order used to add the entities is not important, but they must be known by the system (added) before trying to connect them. The entity `entity_nameA` is connected as a upper layer entity of `entity_nameB`.

`ISJ_system` is mostly a mask of the `eduni.simjava.Sim_system` class of Simjava created by Ross McNab.

4.3.2 ISJ_entity class

To simulate an entity the user must create a class to implement its behaviour. Every class that defines an entity must extend the `ISJ_entity` class and override the `body()` method. Entities are executed by separate threads, each thread starts its execution in the `body()` method.

The entity has several methods to communicate with the upper and lower connected entities. The entity can send an event to the first connected entity of upper and lower layers using the methods `isjSendEventToUpper(event,delay)` and `isjSendEventToLower(event,delay);`

this methods are normally used when it is known that the entity will have one connection in that direction. To send an event to a determined entity: if the entity is in the upper level use `isjSendEventToLower(entity_identity,event,delay)`; if the entity is in the lower level `isjSendEventToUpper(entity_identity,event,delay)`.

The user also has available two methods to send broadcasts events to the connections of the upper or lower layer:
`isjSendBroadcastEventToLower(event,delay)`;
`isjSendBroadcastEventToUpper(event,delay)`.

The `isjPutEventToItself(event,delay)` sets an event to the current entity `delay` time in the future.

To get an arrived event (that already is in the deferred queue) two methods are available. `isjWaitForEvent()` gets the first event from the deferred queue, if the queue is empty waits for the next event. isjGetEvent(eduni.simjava.Sim_predicate) should be the option if the user wants to choose a specific event. Is possible to get a future event, in this case the user has always to specify a predicate (`isjGetFutureEvent(eduni.simjava.Sim_predicate)`).

To produce output two methods may be used: `isjTrace(int,String)` writes a message in the trace file of the simulation; `isjPrint(String)` writes a message in the standard output.

4.3.3 ISJ_event class

All the events are instances of this class. Every event has a tag of `int` type. Also it can carries several objects with it. To add an object to the event's data the user can use one of two methods: `addObjectToData(Object)` and the object will be collocated in the end of the buffer; or `addObjectToData(Object,int)` and the object will be collocated in the `int` position of the buffer (all the objects from that position onwards are shift by one unit). Note that position here is set in number of objects and not in bytes, and it must be smaller then the size of the buffer (`sizeOfData()`).

To get objects from the event the method `getObjectFromData(int)` has to be used. The parameter specifies the position of the wanted object.

The user can get the identity of the entity that sent `getOriginEntity()` and the tag of the event (`getTag()`).

5 The TCP Simulation Entity

Transmission Control Protocol [4] is one of the most used transport protocol in the Internet. It is able to implement a full duplex dependable communication between pairs of processes. Since a big slice of the network data is communicated through it, it became very important to understand how it reacts to the different network situations.

The protocol suffered several changes since its was invented and several implementations exists. In this entity the original specification was simulated, but during the implementation especial care was given to modularity. Thus although the entity simulates the original version, it has the core of all the implementations and can be, with considerable degree ease, be *upgraded* to a newer version.

5.1 The TCP Protocol

TCP (Transmission Control Protocol) is situated in the transport layer of the OSI model. It provides the interface between the applications (e.g. FTP, Telnet) and the network. It also has some special features, like reliable and connection oriented connection oriented communications, that makes it the best choice for most of the applications.

The the data that flows on the TCP connections may be thought of as a stream of bytes. Once the data is passed to the TCP protocol, it has the responsibility to send it (and receive at the other end) in a correct way. This means that it has to recover from segment losses, out-of-order delivery, damage in data during the network trip and duplications of segments. The transmission is made reliable via the use of sequence numbers and acknowledgments. Each byte has an assigned sequence number. The sequence number of the first byte of data to be transmitted is sent in the segment and this is called the segment sequence number. The acknowledgement number is the sequence number of the next expected byte. These numbers are also transported in the segments and are used to inform the receiver of the segment that all the bytes with a lower sequence number were already correctly receive and hence what the next expected byte is.

The TCP protocol can cope with more them one connection at same time, each one has a port identifier, which must be unique in the set of ports of the network node. To provide for unique address within each TCP the port number of the connection is concatenated with the Internet address creating a socket, which is unique throughout all the networks.

To establish a connection the TCP uses the "three-way handshake" (Figure 6). The procedure normally is initiated by one TCP process and responded by another TCP process. If the procedure is initiated by two simultaneous TCP processes it also will work, both send a segment with the `SYN` flag set and is responded with and acknowledgment (`ACK`).

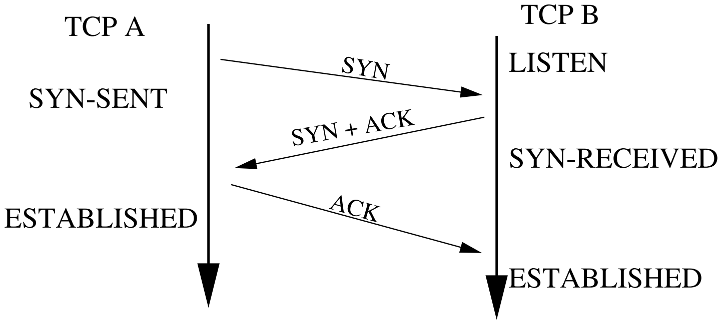


Figure 6: The "three-way handshake" procedure used to open an connection.

Also when closing both TCP's process go through a handshake procedure to close the *stream*.

A connection in its *life* evolves through several states and has several types of events (also called `calls`). Figure 7 presents the state diagram of a TCP connection. Although the propose of this report is not to explain how the TCP manages the connection and its path, the diagram is useful to understanding the reactions of the TCP during the simulation.

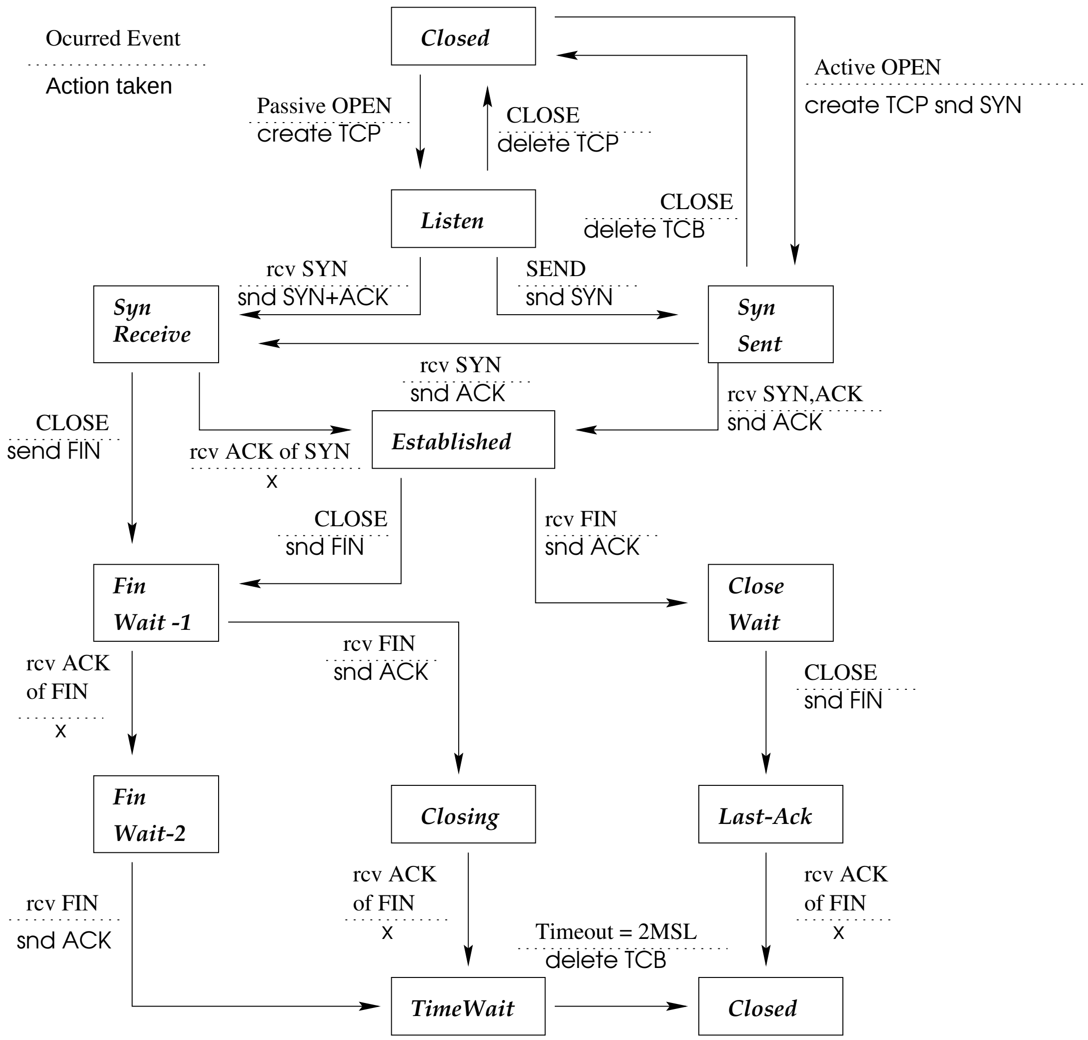


Figure 7: TCP connection state diagram.

5.2 How to Use the Simulation Entity

The entity is a simple Intersim-Java entity and can be connected with other entities has explained in section 4.3.

The TCP entity is expecting to have the applications connected has upper layers and a network protocol has a lower layer. The entity can be connected with more than one application, but only one connection is expected with the lower layer (if more than one is used, the rest will be ignored and the first one will receive all the events). The lower layer though expected to be a network protocol (e.g. IP protocol) can be any type of entity so long as it is able to send the segment to the destination node.

TCP communicates with the lower layer Only to exchangesegments. When sending a segment the event that is passed has the tag with value of 1 (`SEGMENT` constant). It also has five objects:

- The segment to be passed to the destination TCP entity;
- The identity of the destination node;

- The size of the data carried in the segment;
- The sequence number of the segment and
- The acknowledge number carried in the segment.

The first three are compulsory in order to the lower layer completes the transmission. The last two are used to monitoring the segments in the lowers layers and get some simulation results.

When receiving a new segment the TCP expects an event with the `SEGMENT` value in the tag. Two objects are expected: The first one with the segment and the second with the identity of the origin node.

When interfacing with the upper layers the TCP entity has to cope with several calls, here they are listed by tag value (these constants can be found in the `TCP_constants` interface, make the application class implement the interface easier to use):

- OPEN : Opens a connection. Is compulsory to specify the port number (the port number is a `String`) as the first object. If the connection is to be open in the active mode, also the destination node identity and the number of the destination port as to be passed as the second and third objects. If none of this is specified the connection is open in passive mode and goes to `LISTEN` state (waits to be connected).
- SEND : Orders the TCP entity to pass the specified data to the destination connection. Two objects are expected in this call, the first one has the port number (used to inform the TCP of which connection this send refers to), the second one has the size of the data to send. Since this is a simulation, no actual data is sent, just the size of the simulated data. If the connection is in `LISTEN` state the identity of the destination node and the port number must also be set in the fourth and fifth positions.
- RECEIVE : Gets data from the *stream*. The port number has to be specified in the first object and the size of the data expected in the second. The TCP entity will only respond when enough data as arrived to send to the application. The answer is an event with the tag equal to `DATA` where the amount of data is specified in the first object.
- CLOSE : Closes the connection. Makes the TCP entity start the handshake process with the destination to close the connection specified by the port number in the first object position.
- ABORT : Abruptly closes the connection. The connection (specified by the port in the first object position) is closed without any handshake process. If the user uses the `CLOSE` call any outstanding sends will be finished, but with the abort call the connection is closed at that moment.
- STATUS : Asks for the state of the connection. The event has to have the port number in the first position. The user will receive an event with the `STATUS` value in the tag. The first object position has a string with the name of the present state of the connection.

The user may also receive asynchronous events with the tag `ERROR`. The first object is a message with an explanation of the occurred error.

This TCP simulation entity does not cope with urgent data or security and preferences of communications.

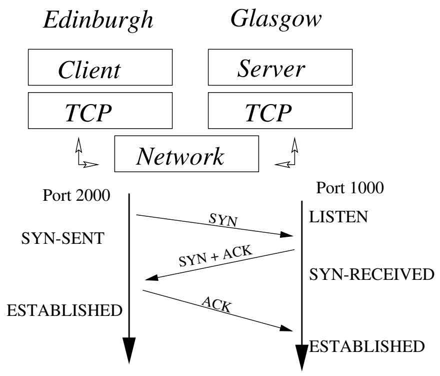
5.3 Introducing Congestion Avoidance Features

The original TCP specification already has a flow control problem, in order to use the available resources better. The intervening nodes of the connection are constantly exchanging the capacity of their receiver buffer, this parameter is called `window`. It indicates the allowed number of bytes that the sender may transmit before receiving further permission. This way no data is lost due to lack of capacity at the receiver.

This feature proved in 1988 not to be sufficient. The network suffered a series of congestion collapses and was noted that the TCP responsible for some of these. Since then many improvements have been made to the protocol. In this project was also simulated a version of TCP known `Tahoe`. In this version several congestion avoidance and control features were introduced: Slow-start; new round-trip timing; Congestion avoidance and Fast Retransmit all the algorithms are described elsewhere [8, 17].

6 Experimental Results

Many experiments can be done with the TCP entity. During this project several experiments were carried out.



at 0.0 Glasgow: rcv OPEN, Port 1000 new state LISTEN

at 0.1 Edinburgh: rcv OPEN, Port 2000 new state SYN-SENT

at 0.1 Edinburgh: send SYN to Glasgow, port 1000

at 0.6 Glasgow: rcv SEGMENT

at 0.6 Glasgow: Port 1000 new state SYN-RECEIVED

at 0.6 Glasgow: send send SYN+ACK to Edinburgh, port 2000

at 1.1 Edinburgh: rcv SEGMENT

at 1.1 Edinburgh: send ACK to Glasgow, port 1000

at 1.1 Edinburgh: Port 200 new state ESTABLISHED

at 1.6 Glasgow: rcv SEGMENT

at 1.6 Glasgow: Port 1000 new state ESTABLISHED

Figure 8: TCP simulation example.

Figure 8 illustrates a simulation of the TCP "three-way hand shake" process that the TCP connections pass through to start communicating. A client is running on a machine in *Edinburgh* and opens a connection with the server in *Glasgow*.

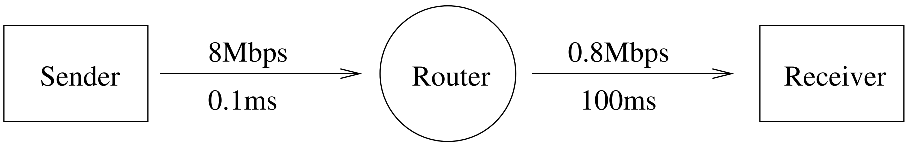


Figure 9: Network simulation topology.

For the next experiments a different network was built (Figure 9). The circle indicates a finite-buffer drop-tail gateway, and the squares are representing network nodes that communicate through the TCP protocol.

The Figures 10-13 present graphs of the arrival and departure of packets in the router. The x-axis shows the arrival and departure time of a packet (filled and empty squares, respectively) and acknowledgements (circles). If the packet passing through the router does not have to experience any appreciable queueing delay tehn they generate two marks so close together on the graph as to appear as a single mark. The graphs are representing the communication during `Established` state.

With this simulation topology both the TCP versions created were compared. As it can be seen in Figures 10 and 11 the initial specification of TCP is faster than the Tahoe. However this is with no other traffic passing through the network and the receiver entity having enough buffer space to receive.

In the later experiments (Figures 12 and 13) background traffic was introduced in the first two seconds. It forces the gateway to drop the 14th packet (represented by a cross). The Tahoe version uses the Slow Start and Congestion Avoidance algorithms to avoid a congestion state. Also the Fast Retransmit procedure enables the lost packet to be discover before the timeout. By comparison it can be seen that the Tahoe version is 4-5 times faster than the initial version, which instead of backing off and trying to avoid congestion keeps sending the data to the network. In this case the background traffic stops very rapidly, but if it stayed that way the network would experience congestion collapse.

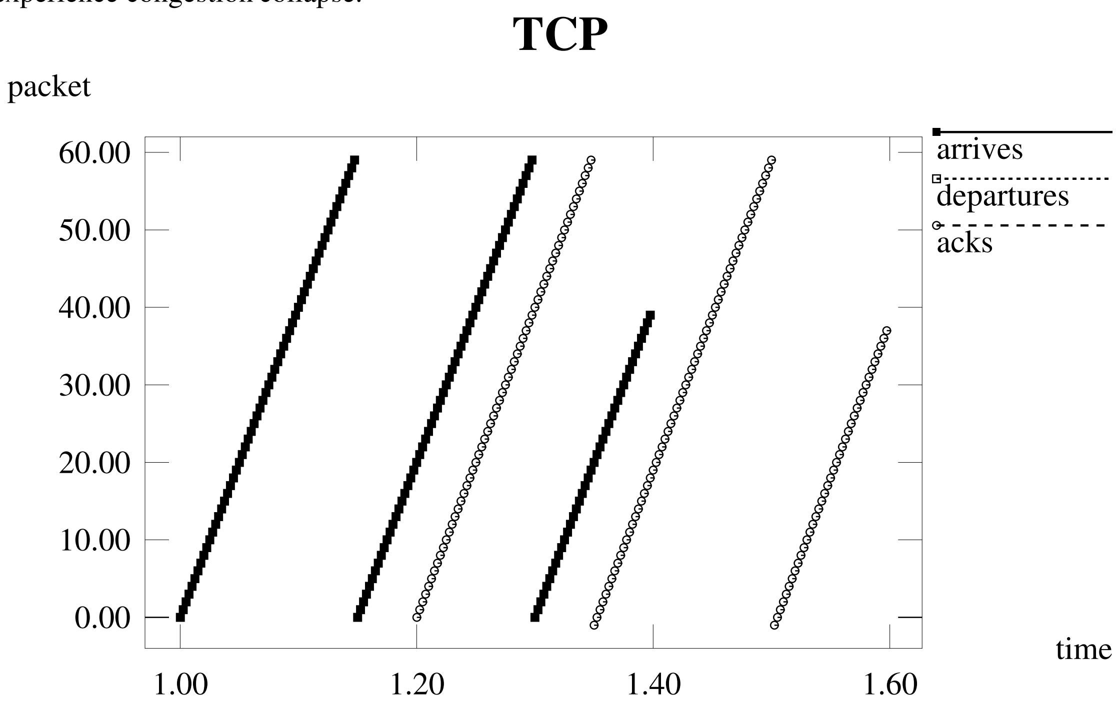


Figure 10: TCP simulation without packet lost.

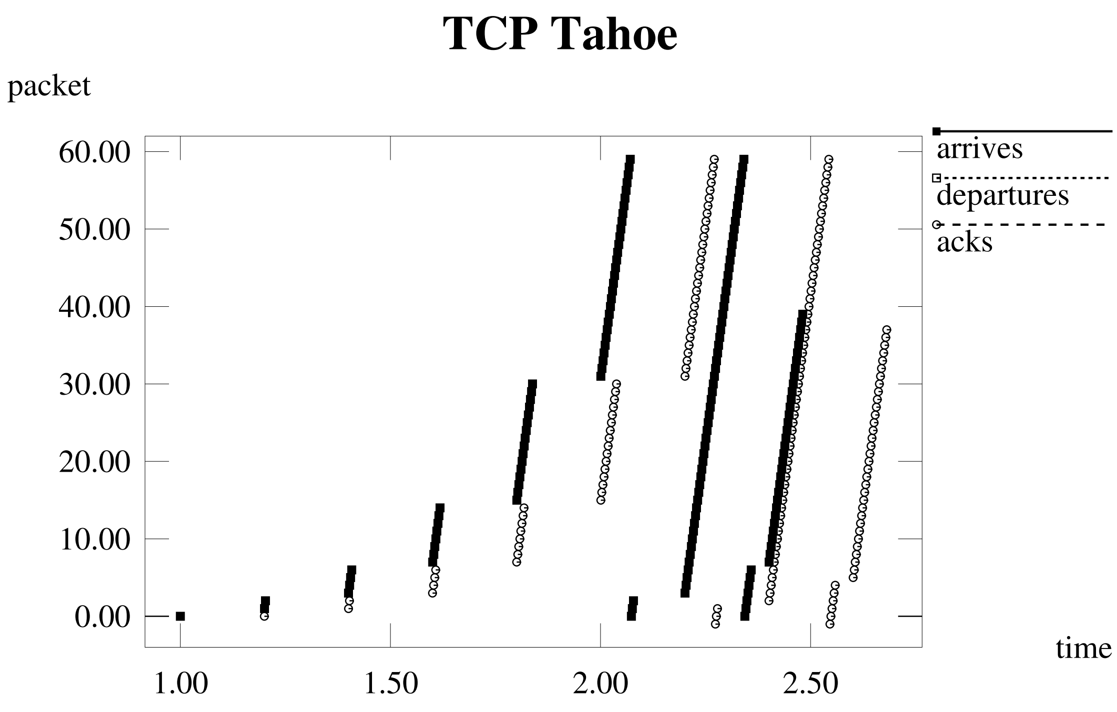


Figure 11: TCP Tahoe simulation without packet lost.

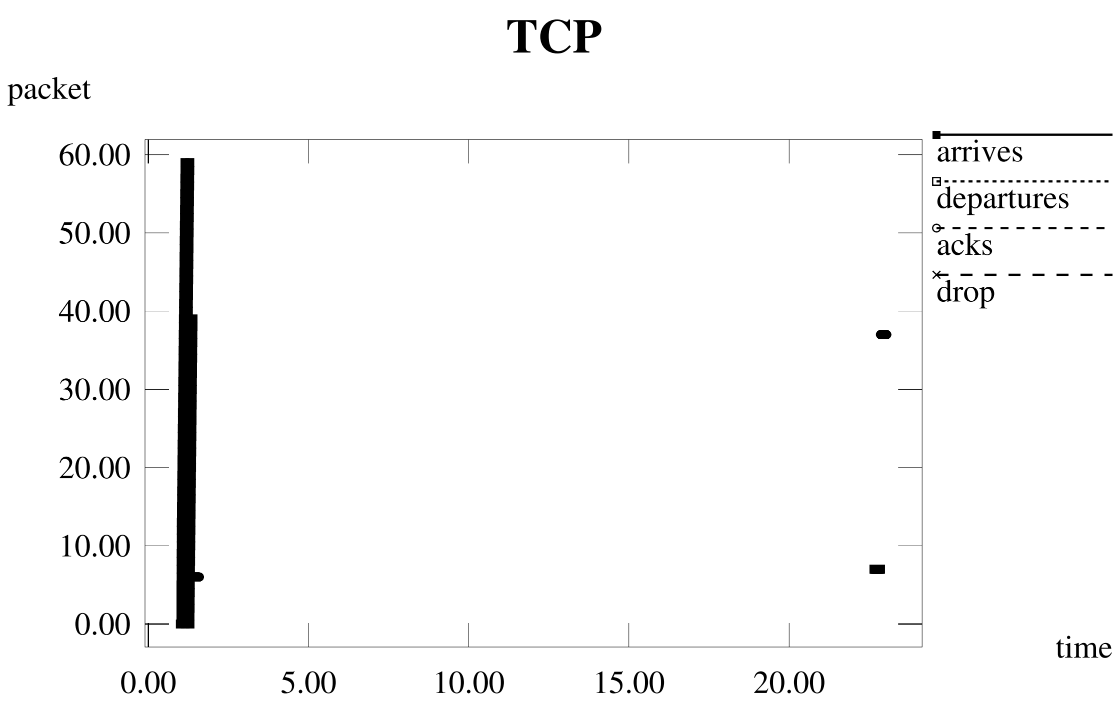


Figure 12: TCP simulation with one packet lost.

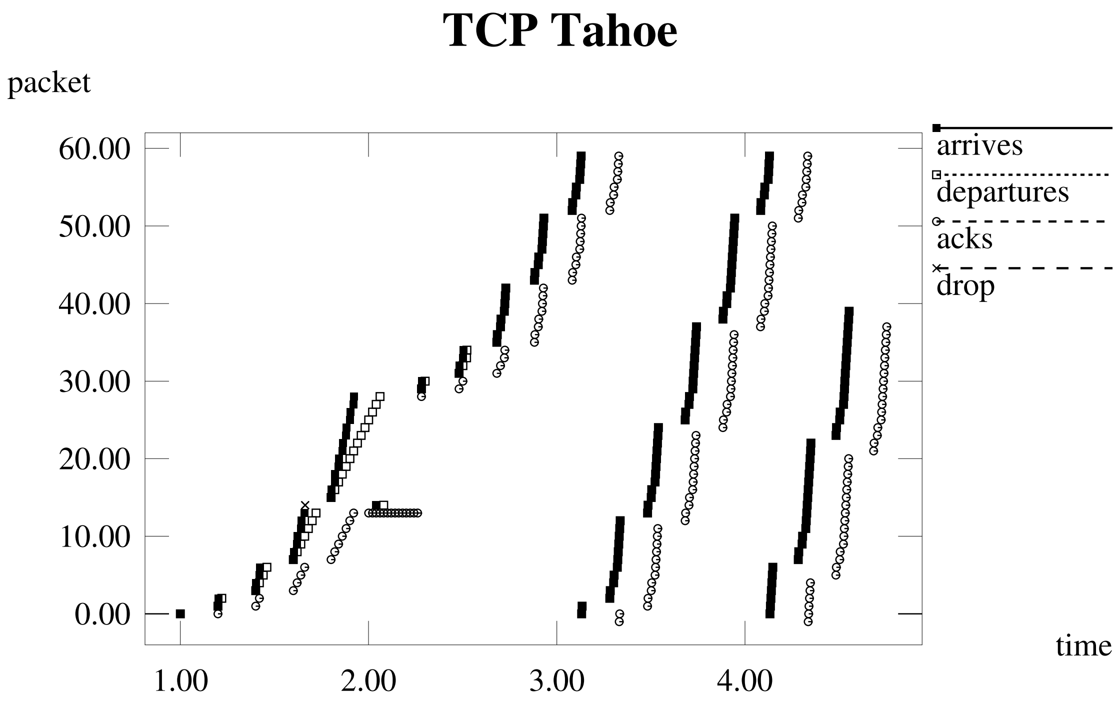


Figure 13: TCP Tahoe simulation with one packet lost.

7 Related Work

Simulating networks is not a new problem and several other packages exists. Most of them are very rigid in the imposed structure or simply are used to test a specific protocol in a controlled situation (e.g. [19]).

Some other like REAL [18] and VINT [20], give some freedom to the user to build their own network structure. Although they might be easier to use, the user is restricted to the available protocols in the software package. Also REAL and VINT they do not simulate the underlying network, they are concentrated in the transport protocols (especially in TCP). No simulator was found with the same degree of modularity as Intersim-Java was found during this project.

Also since Intersim-Java is implemented with Java it is platform independent, which means that the simulation made by the user will run in anywhere.

8 Future Work

The Intersim-Java package is still in a very immature state and can be improved in several directions. More methods can/should be added to the simulation package giving to the user more control over the simulation, and to enable other features.

A decisive feature to implement in future releases is visualisation. A plain text output is not very appealing and can be hard to interpret by the user. The Simjava package already has some built-in features to make visualisation, which can be easily introduced in the Intersim-Java package.

A GUI could also be created to make the creation of simulations more user friendly.

Also to use the Java capabilities, Intersim-Java could have an applet version which could be used in Web pages to run small simulations. These can be useful to illustrate some network features, as well as for teaching netowrk concepts and TCP/IP.

An other interesting improvement would be to create an client-server that could communicate with a system executing a long-running simulation and be able to visualise the results of the simulation on-line. The client could be applet for an easier access. The server might be running on a HPC machine.

Finally an almost infinite work can be done implementing protocol simulation entities to add to the class. The modularity of the class is more useful if the protocols already are implemented and the user only has to join them together. So having a good set of protocols simulated is an important issue for the simulator.

9 Conclusions

Network simulators, though they are very useful for understand how networks work, are very hard to write. Network configurations can change a great deal from problem to problem, which makes it very difficult to create a flexible network simulator.

Modular simulation of networks still has space for a large amount of work. Had the scholarship programme been longer the simulation package could have evolved further and more features could have been implemented.

In order to create modularity the simulations of the protocols have to be more detailed, requiring more computational resources run a simulation. Big network simulations require the use of HPC might, depending on the degree of detail in the simulation.

This project proved that is possible to construct a network simulator with large freedom amount of for the user create the necessary structure of the simulation. A modular simulator also enables the user to create their own entities or use those others have already implemented. This makes it easier and faster to simulate networks.

References

[1] ANSI: American National Standards Institution, `http://www.ansi.org/`

[2] IEEE: Institute of Electrical and Electronics Engineers, `http://www.ieee.org/`

[3] J.Postel, “Internet Protocol - DARPA Internet Program Protocol Specification”, RFC 791, USC/Information Sciences Institute, September 1981

[4] J.Postel, “Transmission Control Protocol - DARPA Internet Program Protocol Specification”, RFC 793, USC/Information Sciences Institute, September 1981

[5] ISO: International Organisation for Standardisation, `http://www.iso.ch/welcome.html`

[6] JANET: The UK Academic & Research Network, `http://www.ja.net/`

[7] J.Postel, "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, September 1981

[8] V.Jacobson, "Congestion Avoidance and Control", Proceedings of the SIGCOMM'88 Symposium,pag. 314-332, August 1988

[9] V.Jacobson, R.Braden, D.Borman, "TCP Extensions for Long-Delay Paths", RFC 1072, October 1988

[10] V.Jacobson, R.Braden, L.Zhang, "TCP Extension for High-Speed Paths", RFC 1185, October 1990

[11] V.Jacobson, R.Braden, D.Borman, "TCP Extensions for High Performance", RFC1323, May 1992

[12] L.S.Brakmo, S.E.O'Malley, L.L.Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", Proceedings of the SIGCOMM'94 Symposium, pag. 24-35, August 1994

[13] JavaSoft Home page, `http://www.javasoft.com/`

[14] R.McNab, F.W.Howell, "Using Java for Discrete Event Simulation", Proceedings of the Twelfth UK Computer and Telecommunications Performance Engineering Workshop (UKPEW), Univ. of Edinburgh, pag. 219-228, September 1996

[15] F.W.Howell, "A discrete event simulation library for C++", Available from `http://www.dcs.ed.ac.uk/home/hase/projects/hase++.html`, February 1996

[16] Jade Simulation International Corporation, "SIM++ User Manual", Calgary, Canada, 1992

[17] W.Richard Stevens, "TCP/IP Illustrated, Volume 1: The Protocols", Addison Wesley, 1994

[18] S.Keshav, "REAL: A Network Simulator", UCB CS Tech Report 88/472, California, USA, December 1988

[19] J.Evans, "TCP and ATM in Wide Area Networks", `http://www.magic.net/tcp`

[20] VINT Project, `http://catarina.esc.edu/vint/`

Hernani Pedroso will graduate in late September 1997 with BSC in Computer Science from the University of Coimbra (Portugal). His main interests are Parallel Computing, Distributed Systems, Distributed Simulation and Fault Tolerance. Motor-cycling is his hobby passion though basketball and tennis are considered fun.

Martin Westhead was the proposer and supervisor of this project. He graduated from Edinburgh University in Artificial Intelligence and Computer Science, in the summer of 1993. He is being working in the EPCC (Edinburgh Parallel Computing Centre) in Training & Research as an Applications Consultant.