TASK QUARTERLY 8 No 3 (2004), 333–358

A METHOD TO BUILD NETWORK-OF-QUEUES-BASED SIMULATORS FOR COMMUNICATION SYSTEMS

LEONARDO PASINI AND SANDRO FELIZIANI

Department of Mathematics and Computer Science, University of Camerino, via Madonna delle Carceri, 62032 Camerino, Macerata, Italy
{*leonardo.pasini, sandro.feliziani*}*@unicam.it*

(Received 7 May 2004; revised manuscript received 7 June 2004)

Abstract: In this study we build a library of new object types whose architecture is based on models of networks-of-queues to simulate communication networks. We also define a class of procedures to automatically generate a simulator of a generic communication network described by the library's objects.

Keywords: queuing systems, computer simulation, communication systems

1. Introduction

The context of this study is performance evaluation of communication networks. In particular, we develop a technique to create a model of a communication network with the use of devices functioning as a network-of-queues model. Therefore, discrete events simulation is the technique that we have applied to evaluate performance of the communication network model to be examined. Particularly, we started from Fdida and Pujolle's [1] concepts on models of communication protocols concerning connections. We then built a library of objects to represent the functioning of a communication network bearing in mind the first three levels of the ISO (International Standards Organization) model.

In this study we present a class of procedures to automatically generate a simulator of a generic communication network described by the library objects. We also illustrate the way a specific communication network is unequivocally associated with the description file of the network. This file is given as an input file of the procedure that generates the simulator of the communication network to be examined. The functioning of the simulator allows us to verify and optimise the protocol behaviour according to different network types. We conclude the paper with a case study.

The programming context is QNAP2 V9.3 [2].

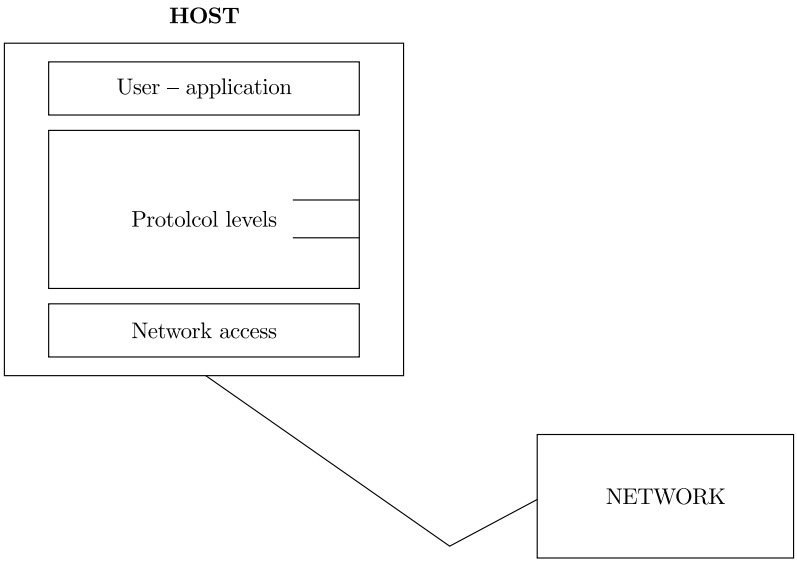

Figure 1. Modular host representation

2. Context

The communication systems we analyse are based on a communication protocol with acknowledgement. Specifically, we use a modular host representation as shown in Figure 1.

This representation has three levels: the user, the protocol and the network access component.

We contemplate a generic communication system that can be simulated with a finite number of hosts, connected by means of routers and half-duplex and full-duplex communication channels. The representation methodology of this system class is based on the following object set, of which the first level description is offered.

Each object will then be translated into a model based on networks of queues and customers in Section 2. In this way a library of QNAP2 object types will be built.

Flow

A flow locates a source host and a destination host within a communication network. It will be characterized by the path that connects the source host to the destination host. The flow path is the sequence of routers and channels crossed by the packet flow. In this context, a data flow is characterized by the data rate of the source host. Each flow of data in the network is associated with a flow of acknowledgments from the destination host to the data source host.

Packets

They represent the system's users. A packet's length, expressed in bytes, depends on the information typology it stands for. In the present context, a packet corresponds to one of the following types: information, data or ack. Data are generated by a source and are routed to the queue that represents the protocol management level within the host. Information packets represent messages with actual information interchanged by hosts. Ack packets are messages acknowledging receipt

of an information packet. Packets are shaped according to a model so as to have two fields specifying the flow they belong to and their length.

Host

This device constitutes a centre of generation and transfer of packets in the network as well as reception and processing of packets incoming from the network. A host contains a source to generate packet flows and, at the same time, can be a recipient of packet flows in the network. It has a specific processing capability, as it contains an internal service device, labelled the protocol, to process packets in two service modalities, *viz.* transmission and reception. Additionally, each host checks the flow of packets to be sent to the network. These devices are connected to the network though a communication channel of bandwidths different for transmission and reception.

Routers

These service devices can receive packets directly from hosts or other routers connected by specific transmission channels. Incoming packets are processed by an internal service device in a period of time that depends on the router's processing frequency. Packets are routed to outgoing transmission channels according to a criterion based on acknowledgment of the destination host.

Connection lines between host and router

These devices constitute service centres that simulate the physical transmission of packets from the host network board to the router interface.

Connection lines between router and router

These devices constitute service centres that simulate the physical transmission of packets from one router to another.

3. The object library

In this section, the above-described objects are introduced into a network-of-queues model to simulate their functioning. When applying such a model, we define a new type of data within a QNAP2 programming context for each object of Section 2, so as to implement a new object library based on queues that can be used to simulate a generic communication network.

3.1. Definition of the object flow

While implementing a communication network model, each host within the system will be assigned to an unequivocal identifier determined by an integer number. This allows considering two integer member variables within each IDSH and IDDH flow. These variables identify the source host and the destination host of the flow. In the communication network, the integer member variables ID and IDA are unequivocal identifiers of the flow and the corresponding acknowledgement flow, respectively. The integer member variable LE indicates the length of packets of the flow. The real variable IA indicates the mean time between arrivals of two data packets of the flow of the source of the generating host.

The QNAP2 code to define the object flow is given below. For each router crossed by the flow, the ROUTABLE array specifies the identifier of either the router or the following host in the path:

```
OBJECT FLOW(ID);
  INTEGER ID, IDA;
  INTEGER IDSH, IDDH;
  INTEGER LE;
  INTEGER ROUTABLE(NROUTER,1);
  REAL IA;
END;
```

3.2. Definition of the object packet

Packets are the system's users. CUSTOMER is a pre-defined variable type in QNAP2. The PACKET variable type will therefore be defined as a CUSTOMER subtype containing some additional member variables. Packets can belong to three different typologies, according to the value of the string member variable TYP:

- data indicates that the packet has been generated by the user application,
- info indicates that the packet has to be processed by the host protocol and afterwards transmitted to the network,
- ack indicates that the packet is a host reply to an info packet received by a sender host.

When a packet originates inside a host, it is assigned to a determinate data flow, which it maintains until it gets to the destination host. Here, once it has been processed, it is assigned to the corresponding acknowledgement flow and sent back into the network. An integer member variable, IDF, identifies the flow the packet belongs to. Besides, there are two integer member variables, IDSH and IDDH, that identify the packet's source and destination hosts, respectively. These variables are employed if the system representation does not use the object flow type.

The QNAP2 code to define the object packet is as follows:

```
CUSTOMER OBJECT PACKET;
  STRING TYP;
  INTEGER IDF;
  INTEGER IDSH, IDDH;
END;
```

In the present programming context, CLASS is a pre-defined variable type used to subdivide users into classes and specify different service modalities for the user classes within the service devices. We will therefore define two user classes with the following instruction:

```
CLASS EMISSION, RECEPT;
```

EMISSION and RECEPT indicate the two transit modalities of packets within the host system.

3.3. Definition of the object host

The network-of-queues model simulating the functioning of the host is shown in Figure 2.

This architecture shows the following internal queue stations:

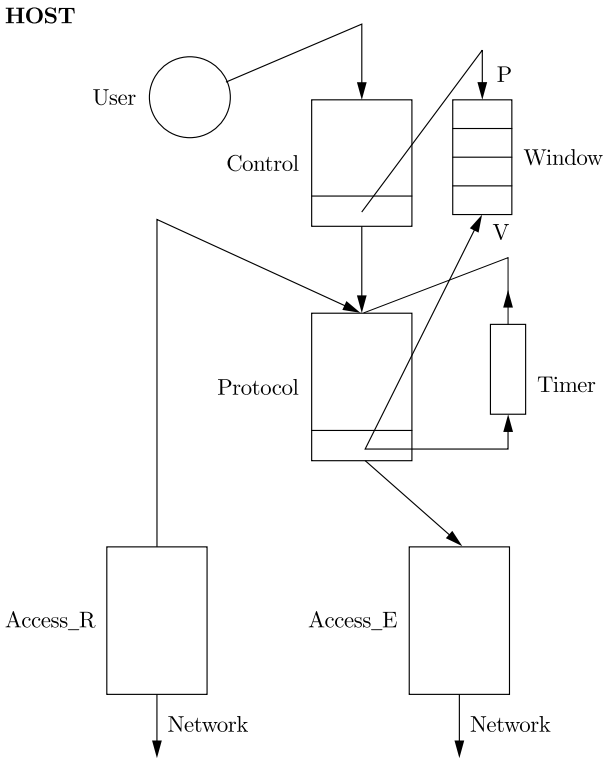

Figure 2. Host queuing model

- **User**: this queue is a source that generates data type packets ascribing to each packets its destination in the network. With reference to Figure 1, it generates the flow from the user-application level of the host to the communication network.
- **Control**: this queue is the service device within which packets send their access requests to the protocol queue, a device where access is limited to a finite number of users. If the protocol device is busy, packets are kept in the control queue.
- **Window**: this queue manages access requests to the protocol device originating from packets in the control queue. It is a semaphore and can assign a WIN_SIZE maximum number of access rights to packets that simultaneously request access to the protocol device. When access rights to the protocol have been exhausted, the following request is kept in the window queue and the packet that sent it is kept in the control queue. Packets that leave the protocol device free their access rights, which are then re-assigned by window.
- **Protocol**: this queue implements a service device that serves packets according to their class. The service algorithm of EMISSION classed packets, which are generated by the user-application level of the host, is described by the PROTOCOL_EMI procedure. The service algorithm of RECEPT classed packets, arriving at the host from the network, is described by the PROTOCOL_REC procedure.

- **Access_E**: this queue simulates the connection device to the output communication channel of the host towards the network. The service algorithm of this queue's packets is described by the ACC_EMI procedure. The LINE_CAP parameter simulates the bandwidth supported by the emission device connected to the network.
- **Access_R**: this queue simulates the connection device to the input communication channel of the host from the network. The service algorithm of this queue's packets is described by the ACC_REC procedure. The REC_CAP parameter simulates the bandwidth supported by the reception device connected to the network.

The following QNAP2 code defines of a new object type host that implements the architecture shown in Figure 2.

```
OBJECT HOST(ID,HFN);
  QUEUE USER,CONTROL,PROTOCOL;
  QUEUE ACCESS_E,ACCESS_R,WINDOW,TIMER;
  REF QUEUE NETWORK;
  INTEGER HFN;
  INTEGER HF(HFN);
  REAL HFP(HFN);
  INTEGER ID;
  INTEGER ID_R;
  INTEGER WIN_SIZE;
  INTEGER LI,LA;
  REAL T_ARR;
  REAL T_EMI_I,T_EMI_A;
  REAL T_REC_I,T_REC_A;
  REAL TIME_OUT;
  REF PACKET RP;
END;
```

Figure 2 shows the queues that are member variables of each host. We will afterwards describe the general functioning of service devices built on these variables by using template stations, which are procedures typical for this programming context.

In a network model to be simulated, the integer variables ID and ID_R are unequivocal identifiers of a host object and the router object it is connected to.

We have supposed that each host in our model structure is connected to a network access router.

The real T_ARR variable indicates the time between arrivals of two packets in the flow generated by the user queue. The integer variables LI and LA are the length of packets of the info and ack types, expressed in bytes.

WIN_SIZE is the dimension of the window that controls access to the protocol queue, while TIME_OUT is the time interval after which a packet for which the corresponding ack has not been received is re-transmitted.

T_EMI_I and T_REC_I are, respectively, the internal emission and reception process times for packets of the info type, while T_EMI_A and T_REC_A are the corresponding values for packets of the ack type.

NETWORK is a queue pointer that represents the connection between host and router.

Let's now define, by means of template stations, the functioning of the host's internal devices that generate the packets flow and route it to the protocol queue.

```
/STATION/
  NAME = *HOST.USER;
  TYPE = SOURCE;
  SERVICE = BEGIN
    RP:=NEW(PACKET);
    RP.IDF:=DISCRETE(HF,HFP);
    RP.TYP:="data";
    EXP(TIME_ARR);
    TRANSIT(RP,CONTROL,EMISSION);
  END;
  TRANSIT = OUT;
/STATION/
  NAME = *HOST.CONTROL;
  SERVICE = P(WINDOW);
  TRANSIT = PROTOCOL;
/STATION/
  NAME = *HOST.WINDOW;
  TYPE = SEMAPHORE,MULTIPLE(WIN_SIZE);
```

The DISCRETE function used in the service algorithm of the source returns a flow index among those contained in the HF array according to the HFP probabilities. Thus, each packet generated by the source is assigned to one of the flows generated inside the host. The TIME_ARR service time must be calculated as the time between arrivals of two packets in the process resulting from the superposition of the single flows generated by the host.

The P procedure in the control queue forces the present customer to request an access right to the window semaphore queue.

The service devices built on the protocol and timer queues of a generic host are defined by the following procedures.

```
/STATION/
  NAME = *HOST.PROTOCOL;
  SCHED = PRIOR;
  SERVICE(EMISSION) = PROTOCOL_EMI(T_EMI_I,T_EMI_A);
  SERVICE(RECEPT) = PROTOCOL_REC(T_REC_I,T_REC_A);
  TRANSIT = OUT;
/STATION/
  NAME = *HOST.TIMER;
  SERVICE = BEGIN
  CST(TIME_OUT);
  TYP:="data";
  TRANSIT(PROTOCOL,EMISSION,1);
END;
TRANSIT=OUT;
```

The internal service procedures for the customer classes EMISSION and RECEPT are PROTOCOL_EMI and PROTOCOL_REC, respectively.

```
PROCEDURE PROTOCOL_EMI(TEI,TEA);
  REAL TEI,TEA;
  BEGIN
    WITH CUSTOMER::PACKET DO
      IF TYP="data" THEN BEGIN
```

```
        TYP:="info";
        EXP(TEI);
        COPY_P(NEW(PACKET),CUSTOMER::PACKET.IDF);
        TRANSIT(HOST.ACCESS_E);
      END ELSE BEGIN
        EXP(TEA);
        TRANSIT(HOST.ACCESS_E);
      END;
  END;
```

This service procedure calls the following COPY_P procedure to send a copy of the packet outgoing from the Timer queue to simulate the management of transmission errors.

```
PROCEDURE COPY_P(RP,I);
REF PACKET RP;
INTEGER I;
BEGIN
  RP.IDF:=I;
  TRANSIT(RP,HOST.TIMER);
END;
```

The service procedure PROTOCOL_EMI identifies packets of the data type generated by the host's internal source, submits them to a process mean time T_EMI_I and changes their type into info. It then uses the COPY_P procedure to send a copy of the packet to the host's timer queue and moves the packet to the host's ACCESS_E queue for transmission to the network. After a process mean time T_EMI_A the ack packets are sent to the ACCESS_E queue.

```
PROCEDURE PROTOCOL_REC(TRI,TRA);
REAL TRI,TRA;
BEGIN
  WITH CUSTOMER::PACKET DO
    IF TYP="info" THEN BEGIN
      TYP:="ack";
      IDF:=FLOW#(CUSTOMER::PACKET.IDF).IDA;
      EXP(TRI);
      TRANSIT(HOST.PROTOCOL,EMISSION);
    END ELSE BEGIN
      EXP(TRA);
      IF HOST.TIMER.FIRST<> NIL THEN BEGIN
        TRANSIT(HOST.TIMER.FIRST,OUT);
        V(HOST.WINDOW);
      END;
    TRANSIT(OUT);
    END;
END;
```

The service procedure PROTOCOL_REC identifies the packets of the info type incoming from the network. In this case, the packet type is transformed into ack and its flow identifier is assigned the corresponding ack flow identifier. After a process mean time T_REC_I the packet is re-sent to the protocol queue with the EMISSION class and the ack type. If the in-service packet's type is not info, the PROTOCOL_REC procedure identifies an incoming packet of the ack type. In such a case, after a process mean time T_REC_A the procedure controls the timer queue status. If this queue is

not empty, the first customer is removed. Then the V procedure returns an access right to the window semaphore queue.

Let us now define, with reference to Figure 2, the functioning of the two host internal devices, ACCESS_E and ACCESS_R.

The ACCESS_E queue represents the physical connection of the host to the network in the emission modality towards the router it is connected to. Using the definition of the template station of this queue, for a generic host, the following QNAP2 code is obtained.

```
/STATION/
  NAME = *HOST.ACCESS_E;
  SERVICE = ACC_EMI(LI,LA);
  TRANSIT=NETWORK;
```

The service procedure for packets inside this queue is described by the following ACC_EMI(LI,LA) procedure:

```
PROCEDURE ACC_EMI(LENGTH_I,LENGTH_A);
INTEGER LENGTH_I,LENGTH_A,LENGHT;
BEGIN
  WITH CUSTOMER::PACKET DO
  IF TYP = "info" THEN LENGTH:= LENGTH_I
  ELSE LENGTH:= LENGTH_A;
  EXP((LENGTH*8)/LINE_CAP);
END;
```

The above service procedure controls the typology of each in-service packet, assigning an appropriate value in bytes to its length. Then, the procedure simulates the transmission time of the packet assigning a service mean time that corresponds to (LENGTH*8)/LINE_CAP. LINE_CAP indicates the available transmission band expressed in bpms. At the end of its service a packet is routed in the NETWORK towards the input queue of the router the host is connected to.

At the same time, the ACCESS_R queue represents the physical connection of the host to the network in the reception modality from the router it is connected to. Using the definition of the template station of this new queue, for a generic host, the following QNAP2 code is obtained:

```
/STATION/
  NAME = *HOST.ACCESS_R;
  SERVICE = ACC_REC(LI,LA);
  TRANSIT=PROTOCOL;
```

The service procedure for packets inside this queue is described by the following ACC_REC(LI,LA) procedure:

```
PROCEDURE ACC_REC(RLENGTHI,RLENGTHA);
INTEGER RLENGTHI,RLENGTHA,LENGHT;
BEGIN
  WITH CUSTOMER::PACKET DO
  IF TYP = "info" THEN LENGTH:= RLENGTHI
  ELSE LENGTH:= RLENGTHA;
  EXP((LENGTH*8)/REC_CAP);
END;
```

This service procedure also controls the packet's typology and assigns a proper value to its length. Then the procedure simulates the incoming reception time

of the packet in the host assigning a service mean time that corresponds to (LENGTH*8)/REC_CAP. REC_CAP represents the host's incoming bandwidth, available from the network, expressed in bpms. At the end of its service, the host's incoming packet is routed towards the host's protocol queue with the RECEPT incoming class.

3.4. Definition of the object router

The network-of-queues model simulating the functioning of the router is shown in Figure 3.

Its architecture includes a single server internal queue CPU, defined by a RATE variable expressed in bpms, that can process packets in transit.

In the service device of the CPU a not-null probability of packets getting lost is assumed, represented by a P_ERR internal variable.

An unequivocally determined identification number, ID, is associated with a generic router of the network. The router's architecture includes NCR channels communicating with the other routers of the network it is connected to and NCH channels communicating with the hosts it is connected to at the same time.

Each router receives packets coming from a connection channel to another router or a host. Each incoming packet belongs to a specific flow and is put in the queue of the CPU internal device to be processed. Outgoing packets from the router CPU are routed following the instruction of the routing table of the flow they belong to.

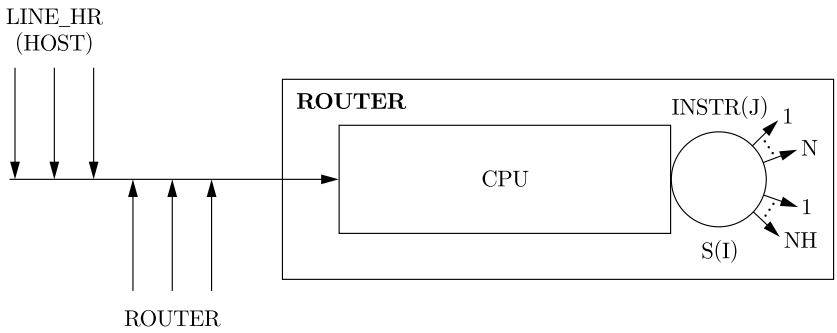

Figure 3. Router queuing model

The following QNAP2 code defines of the new router object type implemented by the architecture shown in Figure 3.

```
OBJECT ROUTER(ID,NCR,NCH);
  INTEGER ID, NCR, NCH;
  INTEGER CH(NCH);
  QUEUE CPU;
  REF QUEUE S(NCH);
  REF QUEUE INSTR(NCR);
  REAL PROB(NCR);
  REAL RATE;
  REAL P_ERR;
END;
```

The integer variable NCH indicates the number of hosts directly connected to the router in the network. CH is an array of integers containing NCH identifiers related

to the hosts which are directly connected to the router. The internal variable S(NCH) is an array of NCH queue pointers. When a router is created, each pointer will be assigned to the queue of the channel that connects the router to a host.

Similarly, the integer variable NCR indicates the number of routers connected to the router instance in the network. The internal variable INSTR(NCR) is an array of NCR queue pointers. When a router is created, each pointer will be assigned to the queue of the channel that connects the router to another router. PROB(NCR) is an array for probabilities of routing to connected routers, used only when the system implements a probabilistic routing in the network.

The following QNAP2 code defines the procedure describing the functioning of the CPU internal queue:

```
/STATION/
  NAME = *ROUTER.CPU;
  SCHED = FIFO,PRIOR;
  SERVICE = BEGIN
  WITH CUSTOMER::PACKET DO BEGIN
    EXP((FLOW#(IDF).LE*8)/RATE);
    IF DRAW(1-P_ERR) THEN BEGIN
      IF ISTPR THEN
        TRANSIT(S(FLOW#(IDF).ROUTABLE(ID,1)),RECEPT))
      ELSE TRANSIT(INSTR(FLOW#(IDF).ROUTABLE(ID,1));
    END;
  END;
END;
TRANSIT = OUT;
```

The service algorithm of the CPU code recognises the flow an in-service packet belongs to and its length in bytes, and assigns the packet a service mean time inversely proportional to the server speed expressed in bpms. The algorithm allows for an error probability, P_ERR, so that the packet is eliminated through the TRANSIT parameter of the station. If the packet is not eliminated, the ISTPR function is called to determine the type of outgoing routing from the router's CPU. The ISTPR function checks if the destination host of the flow the packet belongs to is directly connected to the router. When that is the case, the packet is routed to the host's ACCESS_R queue. The routing is obtained through the transit procedure using indexes memorised in the flow's routing table. If the destination host of the flow is not directly connected to the router, routing takes place in the queue of the connection channel of the next router on the path of the relevant in-service packet's flow.

The code that defines the ISTPR function is as follows:

```
BOOLEAN FUNCTION ISTPR;
  INTEGER I;
  REF ROUTER RR;
  REF FLOW RF;
  BEGIN
    RR:=INCLUDIN(QUEUE)::ROUTER;
    RF:=FLOW#(CUSTOMER::PACKET.IDF);
    WHILE ((I<RR.NCH) AND (RESULT=FALSE)) DO BEGIN
      I:=I+1;
      IF (RF.IDDH=RR.CH(I)) THEN RESULT:=TRUE;
    END;
  END;
```

As an alternative, if the communication system implements probabilistic routing, the service algorithm of the CPU queue is modified, as – in such case – packets which are not aimed at a host directly connected to the router are routed to the NCR outgoing RR lines according to probabilities PROB(NCR).

The definition of the CPU's functioning in the case of probabilistic routing of packets outgoing from routers is as follows:

```
/STATION/
  NAME = *ROUTER.CPU;
  SCHED = FIFO,PRIOR;
  SERVICE = BEGIN
    WITH CUSTOMER::PACKET DO
     BEGIN
       IF TYP = "info" THEN EXP((HOSTS#(IDSH).LI * 8)/RATE)
       ELSE EXP((HOSTS#(IDSH).LA * 8)/RATE);
       IF DRAW(P_ERR) THEN TRANSIT(OUT) ELSE
       IF (HOSTS#(IDDH).ID_R = ID) THEN  TRANSIT(S(IDDH),RECEPT);
     END;
  END;
  TRANSIT=INSTR(1 STEP 1 UNTIL NCR),PROB,OUT;
```

In the present case, where the flow object is not used, IDSH and IDDH are the member variables of the packet object and contain the actual values concerning the present in-service customer.

3.5. Definition of the object LINE_HR

In the present context, the LINE_HR object represents a connection channel between a host and a router. Its architecture is shown in Figure 4 and contains a single server internal queue, R. Packets coming from the ACCESS_E queue of the connected router are routed to R. The time of packets' propagation through this channel is exponentially distributed, with a T mean value expressed in ms. After crossing the HR line, packets are sent to the CPU queue of their destination, determined by the member variable ROUT.

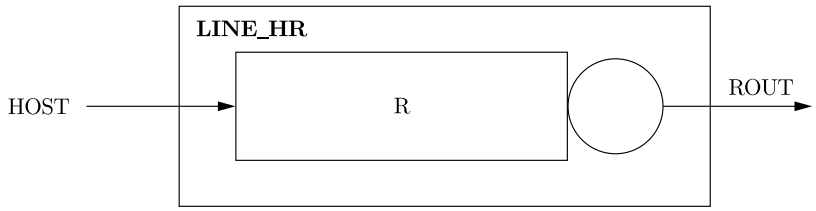

Figure 4. LINE_HR queuing model

The following code defines the new object type:

```
OBJECT LINE_HR(T);
  QUEUE R;
  REF QUEUE ROUT;
  REAL T;
END;
```

This object represents the physical connection between the network card of a host and the router. The real variable T indicates the packets' propagation time within the line.

When building the model, the queue pointer ROUT is assigned to the CPU queue of the router to which transiting packets are sent. The following QNAP2 procedure specifies the code to define the functioning of the internal queue R:

```
/STATION/
  NAME = *LINE_HR.R;
  TYPE = INFINITE;
  SERVICE = EXP(T);
  TRANSIT = ROUT;
```

3.6. Definition of the object LINE_RR

The LINE_RR object represents a one-way connection between two routers, of architecture shown in Figure 5.

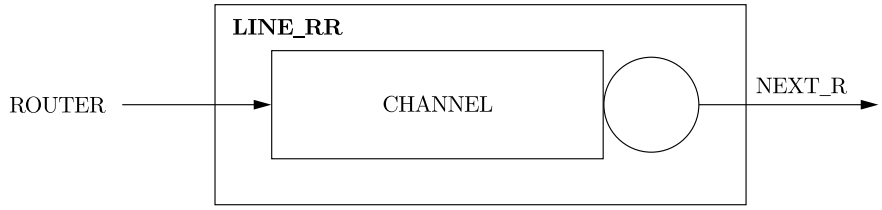

Figure 5. LINE_RR queuing model

If we need two-way communication between two routers, we must create two corresponding lines.

Each instance of a LINE_RR object in a specific model is endowed, as any other component in the present context, with an integer, unequivocally determined identifier. The router-to-router connection is implemented by a single server internal queue, CHANNEL, where packets are transmitted in service mean time TEM expressed in ms.

The pointer to the outgoing router is the member variable NEXT_R.

The following QNAP2 code defines this object type:

```
OBJECT LINE_RR(ID,TEM);
  QUEUE CHANNEL;
  REF QUEUE NEXT_R;
  REAL TEM;
  INTEGER ID;
END;
```

3.7. Building the library

A file is compiled containing the definitions of the objects as above as well as the procedures describing the functioning of their internal queues, originating a library file. In the QNAP2 programming context, it requires the following steps.

First of all, we have to declare a variable of the FILE type, which is a type pre-defined in the programming context. Then, in a code section opened by an /EXEC/ command, the following pre-defined procedures are to be followed: FILASSIGN – to assign a physical file to the file object that has been created, OPEN – to open the file in the writing mode, SAVE – to save the context in the file with a specific name.

The following QNAP2 code illustrates the programme section creating the object library and must be placed in the final part of the file, before the /END/ command that closes the QNAP2 source file.

```
/DECLARE/ FILE F;
/EXEC/ BEGIN
    FILASSIGN(F,"LIB1.lis");
    OPEN(F,2);
    SAVE(F,"OBJECTS");
  END;
/END/
```

4. A programme to generate the simulator

In this section we will illustrate a class of BuildMod procedures to generate simulators of communication networks based on the object library described above.

A generic communication network will be represented by a network made up of objects of the library defined in Section 3.

The network description will therefore be contained in a text file called Model.dat. The format of data in Model.dat files depends on the structure of the BuildMod procedure used, as the procedure executes reading operations from the Model.dat files according to a pre-defined scheme. These operations are carried out using GET and GETLN functions to manage inputs in a QNAP2 context.

The BuildMod procedure, when executed, reads data from a Model.dat file and generates the components of the analysed network, assigning actual values read from the Model.dat file to the internal variables.

The result of the procedure's execution is a simulator of the communication system described by the corresponding Model.dat file.

4.1. Inclusion of the object library

The programme code defining the BuildMod procedure is included in a library file. The file must initially contain a code module to carry out the inclusion of the first library file, where the new object types are defined.

The following QNAP2 code is an example of a possible structure of the module to include the first library.

The module creates a new object of the FILE type, F, and executes the following operations: assigns the previously-compiled library file to F and opens the file in the reading mode. It restores the OGGETTI model from file F with the RESTORE procedure, as follows:

```
/DECLARE/ FILE F;
/EXEC/ BEGIN
  FILASSIGN(F,"LIB1.lis");
  OPEN(F,1);
  RESTORE(F,"OGGETTI");
END;
```

4.2. The NewLine procedure

This procedure is used as part of the BuildMod procedure. NewLine allows one to improve the data reading mode from the Model.dat text file containing data

from the communication network. The data are memorised in records, each of them terminated with a ";" character.

The NewLine procedure, when executed, detects the character at the end of a record and moves the following reading operation to the next record. If the NewLine execution does not detect a record's end character, the procedure interrupts BuildMod's execution and writes an error message in the output file of the simulator's generation.

The NewLine procedure is defined by the following code, placed in the same section as the BuildMod procedure and is:

```
/DECLARE/
PROCEDURE NewLine;
  STRING S;
  BEGIN
    S:=GETLN(STRING,2);
    IF INDEX(S,";")=0 THEN BEGIN
    WRITELN(" (!); not found:[",S,"]");
    ABORT;
  END;
END;
```

4.3. The BuildMod procedure

The BuildMod procedure is structured in code sections specific for the following actions:

- generation of instances constituting the model to be simulated, for each object type of the library;
- generation of the necessary connections among the components described above.

The procedure uses some global variables, such as N_Hosts, N_Router *etc.*, employed to memorise the number of components of the model to be simulated read from the corresponding Model.dat file. The BuildMod procedure generates different simulators starting from different Model.dat data files.

The code of a BuildMod procedure's prototype is given below, in which the flow object is not used. The generated simulator implements probabilistic routing of packets coming from routers.

The BuildMod procedure is characterised by a subdivision of code sections carrying out the following actions in order to build the simulator:

- definition of global variables,
- generation of the system's routers,
- generation of RR lines and connections to outgoing routers,
- generation of outgoing routing procedures from the routers,
- generation of hosts, HR lines and router-to-host connections,
- generation of host-to-router connections.

The data reading operations are carried out with the GET and GETLN functions on a Model.dat data file describing the system to be simulated.

```
PROCEDURE BuildMod;
  INTEGER N_Hosts,N_Router,N_LineRR,idh,idr,idv,I,J,K;
  BEGIN
```

Definition of global variables:

```
N_Hosts := GET(INTEGER);
N_Router:= GET(INTEGER);
N_LineRR:=GET(INTEGER);
NHTEST:= N_Hosts;
NRTEST:=N_Router;
NVTEST:=N_Vie;
LINE_CAP:=GET(REAL);
REC_CAP:=GET(REAL);
NewLine;
T_MAX:=GET(REAL);
PERIODO:=GET(REAL);
NewLine;
```

Generation of the system's routers:

```
IF (N_Router > 0) THEN
  FOR I:=1 STEP 1 UNTIL N_Router DO BEGIN
    idr :=GET(INTEGER);
    ROUTER#(idr):=NEW(ROUTER,idr,GET(INTEGER));
    WITH ROUTER#(idr) DO BEGIN
    ROUTER#(idr).RATE := GET(REAL);
    ROUTER#(idr).P_ERR := GET(REAL);
  END;
  NewLine;
END;
```

Generation of RR lines and connections to outgoing routers:

```
IF (N_LineRR > 0) THEN
  FOR I:=1 STEP 1 UNTIL N_LineRR DO BEGIN
    idv :=GET(INTEGER);
    LINERR#(idv):=NEW(LINE_RR,idv,GET(REAL));
    WITH LINERR#(idv) DO BEGIN
      idr:=GET(INTEGER);
      NEXT_R:=ROUTER#(idr).CPU;
    END;
    NewLine;
  END;
```

Generation of outgoing routing procedures from the routers:

```
FOR I:=1 STEP 1 UNTIL N_Router DO BEGIN
  idr:=GET(INTEGER);
  WITH ROUTER#(idr) DO BEGIN
    FOR J:=1 STEP 1 UNTIL N DO BEGIN
      K:=GET(INTEGER);
      INSTR(J):=LINERR#(K).CANALE;
      PROB(J):=GET(REAL);
    END;
  END;
  NewLine;
END;
```

Generation of hosts, HR lines and router-to-host connections:

```
IF (N_Hosts > 0) THEN
  FOR I:=1 STEP 1 UNTIL N_Hosts DO BEGIN
    idh:=GET(INTEGER);
    HOSTS#(idh):=NEW(HOST,idh);
```

```
    WITH HOSTS#(idh) DO BEGIN
      HOSTS#(idh).T_ARR := GET(REAL);
      HOSTS#(idh).T_EMI_I := GET(REAL);
      HOSTS#(idh).T_EMI_A := GET(REAL);
      HOSTS#(idh).T_REC_I := GET(REAL);
      HOSTS#(idh).T_REC_A := GET(REAL);
      HOSTS#(idh).TIME_OUT := GET(REAL);
      HOSTS#(idh).WIN_SIZE := GET(INTEGER);
      HOSTS#(idh).LI := GET(INTEGER);
      HOSTS#(idh).LA := GET(INTEGER);
    END;
    LINE_HR#(idh):=NEW(LINE_HR,GET(REAL));
    idr:=GET(INTEGER);
    HOSTS#(idh).ID_R := idr;
    ROUTER#(idr).S(idh):=HOSTS#(idh).ACCESS_R;
    NewLine;
  END;
```

Generation of host-to-router connections:

```
IF (N_Hosts > 0) THEN
  FOR I:=1 STEP 1 UNTIL N_Hosts DO BEGIN
    HOSTS#(I).NETWORK:=LINE_HR#(I).R;
    LINE_HR#(I).ROUT:=ROUTER#(HOSTS#(I).ID_R).CPU;
  END;
END;
```

4.4. Building the BuildMod library

The file containing the BuildMod procedure definition code closes with the following section code to create a second library file, Lib2.lis.

Procedures to be used are the same as those in Section 3.7:

```
/DECLARE/ FILE I;
/EXEC/
  BEGIN
    FILASSIGN(I,"LIB2.LIS");
    OPEN(I,2);
    SAVE(I,"BuildMod");
  END;
/END/
```

5. Study on a communication system

The application presented in this section builds a simulator for the communication network described in Figure 6.

The system operates according to the following hypotheses:

- each host is connected to a router of the network;
- inside a host, each generated packet is randomly assigned to a destination host;
- in each router, packets whose destination is a host unconnected through a direct HR line are probabilistically routed to RR lines.

The flow object type, introduced in Section 3.1, is not used to represent this system; instead, the flow of packets is controlled through internal variables of the object packet, defined in Section 3.2, and of the object router, defined in Section 3.4.

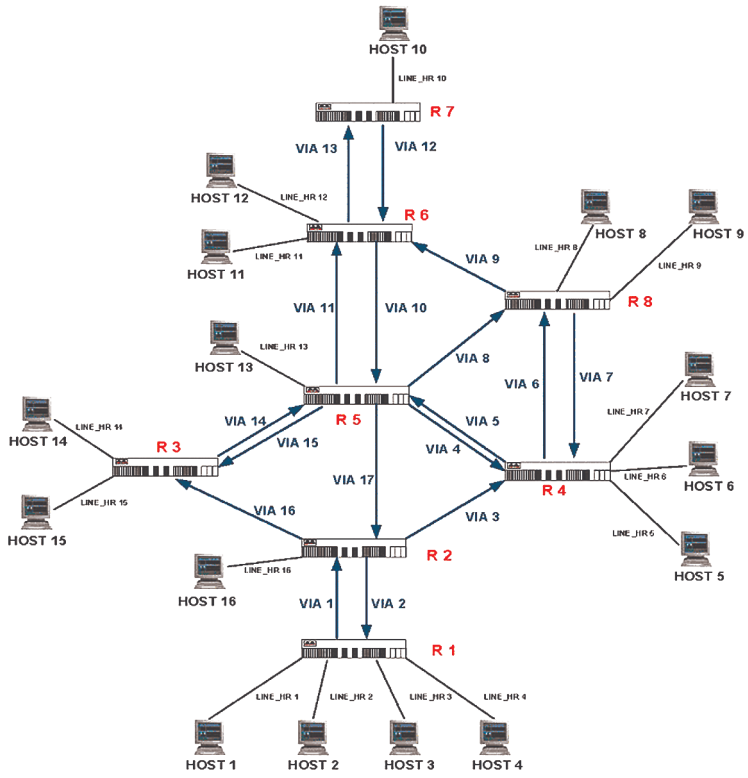

Figure 6. Communication network to be simulated

The BuildMod procedure prototype illustrated in Section 4.3 is consistent with this representation.

The following sections show the phases of generating the simulator and some of the results of the simulation experiments.

5.1. Network description

The description of the communication network to be simulated is accomplished with a Ms-Excel file. This file defines the actual values, in the model, of the network system's global variables and all the connections among hosts, communication lines and routers. The specifications relevant to each component of the system have to be place in the description file. The file is organized in 5 worksheets with specific functions described as follows.

5.1.1. Sheet 1: GLOBALS

The above data define the dimension of the network under analysis. They specify the number of hosts, routers and RR lines in the communication system, as well as the download and upload speed of packets. The model concerns an ADSL network with an emission speed of 128Kb/s and a reception speed of 256Kb/s.

The actual values of the simulation's control parameters are also defined in the above sheet. TMAX indicates the maximum time of the simulation and PERIOD – the execution period of the system's testing procedures during a simulation. The fixed simulation time is 500 seconds, while the time unit for simulation is milliseconds.

The testing procedure will be executed in the system every 1000 ms from the instant a simulation begins.

5.1.2. Sheet 2: ROUTERS

Sheet 2 of the ModelDescription.xls file is made up of four columns. The first column contains routers' identifiers in the model. For each router identifier, the other fields of the same record include, in sequence, the following values:

- number of output RR lines from the router,
- process capability of the router in b/ms,
- error probability.

5.1.3. Sheet 3: LINE_RR

Sheet 3 of the ModelDescription.xls is made up of 3 columns. The first column contains RR line identifiers in the model. For each router identifier, the other fields of the same record include, in sequence, the following values:

- mean transmission time on the line in ms,
- the identifier of the output router in the line.

5.1.4. Sheet 4: INSTR_ROUTER

In sheet 4 the first column contains routers' identifiers in the model. For each router identifier, the number of fields in the same record varies and depends on the number of RR lines outgoing from the router identified in sheet 2. Each record includes sequences of two fields that respectively contain:

- the identifier of an output RR line of the router,
- packets' routing probability on that line.

5.1.5. Sheet 5: Host

Sheet 5 of the ModelDescription.xls file is made up of 12 columns. The first one contains host identifiers in the model. For each host identifier, the other fields of the same record include, in sequence, the following values:

- Tarr: the time between arrival of two packets,
- TEI: emission time of a packet whose type is information,
- TEA: emission time of a packet whose type is ack,
- TRI: reception time of a packet whose type is information,
- TRA: reception time of a packet whose type is ack,
- TIME_OUT: waiting time in the timer queue before a copy is sent,
- WIN_SIZE: dimension of the access control window,
- LI: dimension of a packet whose type is information,
- LA: dimension of a packet whose type is ack,
- LINE_HR: service time within the host's line HR,
- Routers: identifier of the router the host is connected to.

5.2. Generation of the Model.dat file

The generation process of the Model.dat file starts from the system's description file, ModelDescription.xls. We have placed an MS-Visual Basic code module in this file and built the following data export macro:

```
Sub esportation()
N = Worksheets.Count
For I = 1 To N
Worksheets.Item(I).SaveAs("c:\home\pasini\qnap\model\"+
 Worksheets.Item(I).Name),
xlTextPrinter
Next
ThisWorkbook.SaveAs "c:\home\pasini\qnap\model\excelmodel",
    xlWorkbookNormal
End Sub
```

The macro generates, for each internal sheet of the ModelDescription.xls file, a *.prn extension file that contains data of the relevant Excel sheet.

We have created a batch file in the same working directory, Catmod.bat, whose code is:

```
if exist Model.dat del Model.dat
copy *.prn Model.dat
del *.prn
```

The execution of the Catmod.bat file produces an updated version of the Model.dat file in the working directory, deleting previous versions, if any, as well as the intermediate files to export data.

If we apply this process to generate the Model.dat file to the communication network shown in Figure 6 and described by the ModelDesription.xls file described in Section 5.1, we obtain a Model.dat file of the following text:

Sheet 1

```
& GLOBAL VARIABLES
&Nb Hosts      Nb Router  Nb Line RR  Cap. Emis. (b/ms)  Cap. Ric.(b/ms)
      16           8           17        128.0                256.0;
& SIMULATION
& TMAX           PERIOD
  500000.0         1000.0;
```

Sheet 2

```
& ROUTERS
& ID   Exit Rate(b/ms) P_Err
  1      1    100.0    0.001;
  2      3    123.0    0.001;
  3      1    111.0    0.001;
  4      2    122.0    0.001;
  5      5    100.0    0.001;
  6      2    123.0    0.001;
  7      1    111.0    0.001;
  8      2     50.0    0.001;
```

Sheet 3

```
& Line RR
&ID          Propag(ms)     Router...
   1            32.0           1;
   2            32.0           2;
   3            54.0           4;
   4            34.0           5;
   5            11.7           4;
   6            22.0           8;
   7            22.0           4;
   8            12.4           8;
   9            21.0           6;
  10            10.5           6;
  11            10.5           5;
```

```
  12            20.0          7;
  13            20.0          6;
  14            11.0          3;
  15            11.0          5;
  16            32.0          3;
  17            54.0          2;
```

Sheet 4

```
& Routing ROUTER
&Router_ID   L_RR  Prob  L_RR  Prob   L_RR  Prob   L_RR  Prob  L_RR Prob
    1          2   1.0;
    2          1   0.4    3     0.3   16    0.3;
    3         15   1.0;
    4          4   0.6    6     0.4;
    5          5   0.2    8     0.1   10    0.2   14    0.2    17   0.3;
    6         11   0.6    12    0.4;
    7         13   1.0;
    8          7   0.5    9     0.5;
```

Sheet 5

```
& HOSTS
&ID   Tarr  TEI   TEA   TRI TRA  TIME_OUT WIN_SIZE LI  LA LineHR Router
1     120.0 12.1   1.0   4.3 1.1   800.0      10    55   1   12.0     1;
2     123.0 12.0   2.0   4.6 1.2   804.0      11    55   1   13.0     1;
3     124.0 13.0   1.5   6.4 1.3   780.0      12    55   2   12.6     1;
4     135.0 15.0   1.0   5.0 1.8   800.0      20    55   2   10.0     1;
5     120.0 12.0   1.0   4.3 1.1   803.0      12    55   1   11.0     4;
6     123.0 12.3   2.0   5.2 1.2   803.0      12    55   1   12.0     4;
7     201.0 13.0   1.5   5.0 1.3   803.2      14    55   2   13.0     4;
8     204.0 14.0   1.8   4.1 1.5   802.4      12    55   1   15.0     8;
9     112.0 13.0   0.6   5.0 1.1   802.0      13    55   1   12.0     8;
10    180.0 13.7   0.8   6.0 1.2   803.0      14    55   2   10.3     7;
11    110.0 12.5   0.9   3.0 1.6   780.0      15    55   1    9.6     6;
12    112.0 11.3   1.1   2.5 1.9   801.0      13    55   2    8.5     6;
13    202.0 10.7   1.0   2.9 2.0   789.0      12    55   1   13.0     5;
14    145.0 11.9   2.0   2.5 1.0   803.0      11    55   2   11.3     3;
15    143.0 12.5   0.8   2.4 1.3   804.2      10    55   1   12.2     3;
16    180.0 13.0   0.9   3.0 1.7   805.1      15    55   2   14.7     2;
```

5.3. Generation of the simulator

In order to generate the simulator pertinent to the communication system described by the Model.dat file, the following code has to be executed:

```
/DECLARE/ FILE H;
/EXEC/ BEGIN
    FILASSIGN(H,"Model.dat");
    OPEN(H,1);
  END;
/CONTROL/ UNIT = GET(H);
/EXEC/ BEGIN
    BuildMod;
    CLOSE(H);
  END;
```

In this case, the FILASSIGN procedure assigns a file H object to the Model.dat in the working directory. This file is opened in the reading modality. Using the UNIT parameter of the /CONTROL/ command, we define FILE H as the logic unit where the GET and GETLN functions read data.

Thus, the execution of the BuildMod procedure yields the simulator of the system described in Model.dat.

6. Simulation and results

The simulation of the system generated by the BuilMod procedure is obtained through the call of the QNAP2 SIMUL procedure.

This procedure calls discrete events simulation. The simulator is based on a scheduler and a generator of random numbers. The length of the simulation is to be defined in advance with the TMAX parameter.

The following code assigns the maximum length of the simulation, read from the Model.dat file, to TMAX. The time unit to express this length is in accordance with the one used in the model. In our context, the simulation time unit is expressed in milliseconds, ms. The simulation's length has been fixed at 500 000 ms.

```
/CONTROL/
  CLASS=ALL QUEUE;
  TMAX=T_MAX;
/EXEC/ BEGIN
    SIMUL;
  END;
/END/
```

The CLASS parameter of the /CONTROL/ command allows us to request results, for a list of queues, according to user classes. Results are given according to the user classes for all the queues.

The simulation results are arranged in a standard chart that describes the following elements:

- NAME = name of the examined queue,
- SERVICE = mean length of a service, calculated as a result of the measures simulated,
- BUSY PCT = busy percentage of the queue, calculated as a result of the measures simulated,
- CUST NB = mean number of customers in the queue, calculated as a result of the measures simulated,
- RESPONSE = mean response time of the queue, that is the sum of the mean service time and the mean wait time,
- SERV NB = number of users served in the queue.

```
                      *** SIMULATION ***
                    ... TIME =       500000.00
*************************************************************************
*   NAME   *  SERVICE * BUSY PCT *  CUST NB * RESPONSE *  SERV NB *
*************************************************************************
*          *          *          *          *          *          *
*ROUTER 1. *          *          *          *          *          *
*CPU     . * 2.324    *0.2845    *0.5330    * 4.353    *     61215*
*(EMISSION)* 2.324    *0.2845    *0.5330    * 4.353    *     61215*
*          *          *          *          *          *          *
*ROUTER 2. *          *          *          *          *          *
*CPU     . * 1.883    *0.4265    * 1.131    * 4.993    *    113269*
*(EMISSION)* 1.883    *0.4265    * 1.131    * 4.993    *    113269*
```

```
*          *          *          *          *          *          *
*ROUTER 3. *          *          *          *          *          *
*CPU     . * 2.090    *0.3533    *0.7611    * 4.503    *     84509*
*(EMISSION)* 2.090    *0.3533    *0.7611    * 4.503    *     84509*
*          *          *          *          *          *          *
*ROUTER 4. *          *          *          *          *          *
*CPU     . * 1.878    *0.4576    * 1.277    * 5.243    *    121818*
*(EMISSION)* 1.878    *0.4576    * 1.277    * 5.243    *    121818*
*          *          *          *          *          *          *
*ROUTER 5. *          *          *          *          *          *
*CPU     . * 2.277    *0.9994    * 530.8    * 1204.    *    219473*
*(EMISSION)* 2.277    *0.9994    * 530.8    * 1204.    *    219473*
*          *          *          *          *          *          *
*ROUTER 6. *          *          *          *          *          *
*CPU     . * 1.819    *0.4883    * 1.476    * 5.499    *    134224*
*(EMISSION)* 1.819    *0.4883    * 1.476    * 5.499    *    134224*
*          *          *          *          *          *          *
*ROUTER 7. *          *          *          *          *          *
*CPU     . * 2.020    *0.2173    *0.3762    * 3.496    *     53799*
*(EMISSION)* 2.020    *0.2173    *0.3762    * 3.496    *     53799*
*          *          *          *          *          *          *
*ROUTER 8. *          *          *          *          *          *
*CPU     . * 4.549    *0.6664    * 3.216    * 21.95    *     73258*
*(EMISSION)* 4.549    *0.6664    * 3.216    * 21.95    *     73258*
*          *          *          *          *          *          *
*VIA    1. *          *          *          *          *          *
*CANALE  . * 32.00    *0.0000E+00* 2.786    * 32.00    *     43524*
*(EMISSION)* 32.00    *0.0000E+00* 2.786    * 32.00    *     43524*
*          *          *          *          *          *          *
*VIA    2. *          *          *          *          *          *
*CANALE  . * 32.00    *0.0000E+00* 2.817    * 32.00    *     44021*
*(EMISSION)* 32.00    *0.0000E+00* 2.817    * 32.00    *     44021*
*          *          *          *          *          *          *
*VIA    3. *          *          *          *          *          *
*CANALE  . * 54.00    *0.0000E+00* 3.516    * 54.00    *     32552*
*(EMISSION)* 54.00    *0.0000E+00* 3.516    * 54.00    *     32552*
*          *          *          *          *          *          *
*VIA    4. *          *          *          *          *          *
*CANALE  . * 34.00    *0.0000E+00* 4.416    * 34.00    *     64945*
*(EMISSION)* 34.00    *0.0000E+00* 4.416    * 34.00    *     64945*
*          *          *          *          *          *          *
*VIA    5. *          *          *          *          *          *
*CANALE  . * 11.70    *0.0000E+00* 1.008    * 11.70    *     43071*
*(EMISSION)* 11.70    *0.0000E+00* 1.008    * 11.70    *     43071*
*          *          *          *          *          *          *
*VIA    6. *          *          *          *          *          *
*CANALE  . * 22.00    *0.0000E+00* 1.895    * 22.00    *     43062*
*(EMISSION)* 22.00    *0.0000E+00* 1.895    * 22.00    *     43062*
*          *          *          *          *          *          *
*VIA    7. *          *          *          *          *          *
*CANALE  . * 22.00    *0.0000E+00* 1.417    * 22.00    *     32202*
*(EMISSION)* 22.00    *0.0000E+00* 1.417    * 22.00    *     32202*
*          *          *          *          *          *          *
*VIA    8. *          *          *          *          *          *
*CANALE  . * 12.40    *0.0000E+00*0.5320    * 12.40    *     21451*
*(EMISSION)* 12.40    *0.0000E+00*0.5320    * 12.40    *     21451*
```

```
*          *          *          *          *          *          *
*VIA    9. *          *          *          *          *          *
*CANALE  . * 21.00    *0.0000E+00* 1.364    * 21.00    *     32465*
*(EMISSION)* 21.00    *0.0000E+00* 1.364    * 21.00    *     32465*
*          *          *          *          *          *          *
*VIA   10. *          *          *          *          *          *
*CANALE  . * 10.50    *0.0000E+00*0.8974    * 10.50    *     42731*
*(EMISSION)* 10.50    *0.0000E+00*0.8974    * 10.50    *     42731*
*          *          *          *          *          *          *
*VIA   11. *          *          *          *          *          *
*CANALE  . * 10.50    *0.0000E+00* 1.583    * 10.50    *     75361*
*(EMISSION)* 10.50    *0.0000E+00* 1.583    * 10.50    *     75361*
*          *          *          *          *          *          *
*VIA   12. *          *          *          *          *          *
*CANALE  . * 20.00    *0.0000E+00* 1.995    * 20.00    *     49875*
*(EMISSION)* 20.00    *0.0000E+00* 1.995    * 20.00    *     49875*
*          *          *          *          *          *          *
*VIA   13. *          *          *          *          *          *
*CANALE  . * 20.00    *0.0000E+00* 1.999    * 20.00    *     49976*
*(EMISSION)* 20.00    *0.0000E+00* 1.999    * 20.00    *     49976*
*          *          *          *          *          *          *
*VIA   14. *          *          *          *          *          *
*CANALE  . * 11.00    *0.0000E+00*0.9478    * 11.00    *     43084*
*(EMISSION)* 11.00    *0.0000E+00*0.9478    * 11.00    *     43084*
*          *          *          *          *          *          *
*VIA   15. *          *          *          *          *          *
*CANALE  . * 11.00    *0.0000E+00* 1.668    * 11.00    *     75825*
*(EMISSION)* 11.00    *0.0000E+00* 1.668    * 11.00    *     75825*
*          *          *          *          *          *          *
*VIA   16. *          *          *          *          *          *
*CANALE  . * 32.00    *0.0000E+00* 2.087    * 32.00    *     32606*
*(EMISSION)* 32.00    *0.0000E+00* 2.087    * 32.00    *     32606*
*          *          *          *          *          *          *
*VIA   17. *          *          *          *          *          *
*CANALE  . * 54.00    *0.0000E+00* 6.987    * 54.00    *     64696*
*(EMISSION)* 54.00    *0.0000E+00* 6.987    * 54.00    *     64696*
*          *          *          *          *          *          *
*HOST   1. *          *          *          *          *          *
*USER    . * 119.1    * 1.000    * 1.000    * 119.1    *      4198*
*          *          *          *          *          *          *
*HOST   1. *          *          *          *          *          *
*CONTROL . * 233.1    *0.9968    * 1009.    *0.1180E+06*      2138*
*(EMISSION)* 233.1    *0.9968    * 1009.    *0.1180E+06*      2138*
*          *          *          *          *          *          *
*HOST   1. *          *          *          *          *          *
*PROTOCOL. * 4.805    *0.8358E-01*0.9303E-01* 5.348    *      8698*
*(EMISSION)* 6.871    *0.6072E-01*0.6356E-01* 7.192    *      4419*
*(RECEPT  )* 2.671    *0.2286E-01*0.2946E-01* 3.443    *      4279*
*          *          *          *          *          *          *
*HOST   1. *          *          *          *          *          *
*ACCESS_E. * 1.808    *0.1598E-01*0.1650E-01* 1.867    *      4418*
*(EMISSION)* 1.808    *0.1598E-01*0.1650E-01* 1.867    *      4418*
*          *          *          *          *          *          *
*HOST   1. *          *          *          *          *          *
*ACCESS_R. *0.8635    *0.7389E-02*0.7678E-02*0.8972    *      4279*
*(RECEPT  )*0.8635    *0.7389E-02*0.7678E-02*0.8972    *      4279*
```

```
*          *          *          *          *          *          *
*HOST   1. *          *          *          *          *          *
*WINDOW  . *0.0000E+00*0.0000E+00*0.9968    * 234.5    *      2128*
*(EMISSION)*0.0000E+00*0.0000E+00*0.9968    * 234.5    *      2128*
*          *          *          *          *          *          *
*HOST   1. *          *          *          *          *          *
*TIMER   . * 221.4    *0.9998    * 9.928    * 2195.    *      2258*
*(EMISSION)* 221.4    *0.9998    * 9.928    * 2195.    *      2258*
*          *          *          *          *          *          *
*LINE_HR1. *          *          *          *          *          *
*R       . * 12.00    *0.0000E+00*0.1060    * 12.00    *      4418*
*(EMISSION)* 12.00    *0.0000E+00*0.1060    * 12.00    *      4418*

                                . . . . .

*HOST  16. *          *          *          *          *          *
*USER    . * 179.7    * 1.000    * 1.000    * 179.7    *      2781*
*          *          *          *          *          *          *
*HOST  16. *          *          *          *          *          *
*CONTROL . * 212.3    *0.9828    * 226.7    *0.3991E+05*      2314*
*(EMISSION)* 212.3    *0.9828    * 226.7    *0.3991E+05*      2314*
*          *          *          *          *          *          *
*HOST  16. *          *          *          *          *          *
*PROTOCOL. * 4.834    *0.8708E-01*0.9718E-01* 5.395    *      9007*
*(EMISSION)* 7.309    *0.6654E-01*0.6962E-01* 7.647    *      4552*
*(RECEPT  )* 2.305    *0.2054E-01*0.2756E-01* 3.093    *      4455*
*          *          *          *          *          *          *
*HOST  16. *          *          *          *          *          *
*ACCESS_E. * 1.906    *0.1735E-01*0.1808E-01* 1.986    *      4552*
*(EMISSION)* 1.906    *0.1735E-01*0.1808E-01* 1.986    *      4552*
*          *          *          *          *          *          *
*HOST  16. *          *          *          *          *          *
*ACCESS_R. *0.8963    *0.7986E-02*0.8313E-02*0.9330    *      4455*
*(RECEPT  )*0.8963    *0.7986E-02*0.8313E-02*0.9330    *      4455*
*          *          *          *          *          *          *
*HOST  16. *          *          *          *          *          *
*WINDOW  . *0.0000E+00*0.0000E+00*0.9828    * 216.6    *      2299*
*(EMISSION)*0.0000E+00*0.0000E+00*0.9828    * 216.6    *      2299*
*          *          *          *          *          *          *
*HOST  16. *          *          *          *          *          *
*TIMER   . * 209.8    *0.9992    * 14.83    * 3103.    *      2381*
*(EMISSION)* 209.8    *0.9992    * 14.83    * 3103.    *      2381*
*          *          *          *          *          *          *
*LINE_H16. *          *          *          *          *          *
*R       . * 14.70    *0.0000E+00*0.1338    * 14.70    *      4552*
*(EMISSION)* 14.70    *0.0000E+00*0.1338    * 14.70    *      4552*
*          *          *          *          *          *          *
**********************************************************************
                    ... END OF SIMULATION ...
              MEMORY USED:    1763370 WORDS OF 4 BYTES
                     ( 11.76  % OF TOTAL MEMORY)
```

The simulation was carried out by using a Personal Computer with a 2.8 GHz Intel Pentium IV CPU, 256 MB of RAM and MS Windows 2000 operative system, and it took a few CPU seconds.

If we consider that the mean transfer time of information generated within a host corresponds to the time that separates the transmission of an information packet from the reception of the corresponding ack, we can take the mean response times of the internal window semaphore queues of the hosts as estimates of these measures. It is therefore possible to say that, for example, the mean time for the information generated by host 1 to cross the network was about 235 ms.

7. Conclusions

We have built a library of objects based on queue systems to represent and simulate communication systems. We have also proposed a scheme to build procedures for the automatic generation of simulators based on this object library.

This has allowed us to verify and optimise the communication protocol conduct according to different queue types. It will thus be possible to measure different parameters of the protocol to optimise its performance criterions according to the service quality.

References

[1] Fdida S and Pujolle G 1990 *Modèles de Systèmes et de Réseaux*, Tome 1: *Performance*, Ed. Eyrolles, Paris

[2] *Simulog, QNAP2 Reference Manual*, ver.9.3