Simulation Experiments with Protocol Interactions in ATM Networks

Stephen Cusack and Rob Pooley
Department of Computer Science
The University of Edinburgh
email: *sdc@dcs.ed.ac.uk* & *rjp@dcs.ed.ac.uk*

June 14, 1995

Abstract

This paper outlines ongoing work in performance evaluation of ATM networks through the use of simulation. This work is particularly aimed at estimating the effects at the application level of higher level protocol interactions. Current modelling of ATM networks has been concentrated at the switch level. An objective of this work is to broaden the context by examining the network performance and characteristics at a level typically seen by network management. An object-oriented methodology and the techniques of discrete event simulation are being investigated as a means of providing a framework in which to conduct these experiments.

1 Introduction

Most modelling of ATM networks has concentrated on switch level modelling, often of individual switches (see [3] for an example). This is understandable in the early stages of the design of hardware based on this new technology, but it is not enough to understand the likely overall performance of high bandwidth communications systems. Work on driving these switch level models has concentrated on defining the characteristics of expected sources of work, most importantly real time video, and the switching between these sources, for instance by defining a Markov modulated Poisson process. The aim of the work described is to broaden the context within which ATM performance is considered.

Among the issues under consideration are:

- How realistic are the workload models proposed when ATM networks are largely driven by bridges from LANs operating *Ethernet* and other conventional protocols?
- What are the effects of protocol interactions? In particular, what impact does the possible loss of cells in an asynchronous network have on higher level protocols operating across such networks? Since no re-transmission is forced within the ATM network, how does the buffering of messages either side of the network get affected?
- How can pathological conditions in an ATM network (which may bring the network down) be recognised and possibly avoided? Which network workload states may lead to critical conditions? Can the network recover from such states?

In order to answer this type of question, the level of analysis of ATM networks has to be abstracted above the switch level to a model which incorporates the entire network. This level of abstraction is that typically seen by a network manager who monitors the performance of their network. To model at this level, a discrete event simulation framework for ATM and related network models is being created. This is being written using an object oriented approach made possible by the DEMOS simulation package [1]. Presently, the model constituents are in the process of continued testing and refinement which is leading to the design and implementation of experiments to examine aspects of total network and combined network behaviour.

This paper is structured as follows:

- Section 2 details some of the motivation for the work in progress.
- Section 3 briefly describes the features of the DEMOS simulation environment and details the current state of the simulation model.
- Section 4 gives an indication of some of the problems and solutions already encountered.
- Section 5 describes anticipated future work.
- Section 6 gives some early conclusions.

2 Motivation for the work

2.1 Overview

ATM technology and the interest in it has reached a point where governments and public network operators are considering ATM as a vehicle for providing B-ISDN [4, 5] services. Effective roll-out of ATM technology to the business and private sectors will require the understanding of how to design ATM networks to carry both real-time and data traffic as efficiently as possible.

Modelling ATM networks effectively will require the careful design and implementation of realistic workload models (to simulate expected traffic on the network) and efficient simulators. The use of inaccurate models, however, could well lead to sub-optimal ATM network designs being adopted in the future due to 'worst case' design rules having been used. The goal of this work is to fuse suitable workload models with efficient simulation models to provide insights into the design and performance of ATM networks.

2.2 Areas of interest

As mentioned in section 1, one area of interest is the validity of existing workload models in the context of ATM networks being driven by bridges from private LAN or MAN networks. This is of particular relevance to public network operators who will almost certainly offer ATM interconnection to their customers' own private networks. It is planned that appropriate models to simulate the expected traffic from this variety of source will be developed as 'plug in' modules for the simulator under development. This is in addition to appropriate 'plug in' modules designed to simulate existing single applications (such as MPEG video transmission).

The effects of protocol interaction are also an important area of study. As ATM networks have no standard method of cell re-transmission (cells from higher level protocol data packets may be dropped anywhere en route) the simulator is being designed such that even at the high level of abstraction the effect of cell loss on the higher level protocols controlling transmission may be observed. It will be important to see if the advantages of a high speed cell transmission system are compromised by the delays caused by the retransmission of data packets which arrive incomplete at their destination. Performing this analysis with simulated entire networks, rather than two applications sharing a common switch, is particularly relevant.

The other area of interest highlighted so far from the high level of abstraction concerns network management. Simulating an entire network will allow sensitivity analysis to be performed on the network such that the network can be forced into abnormal states and the associated behaviour monitored. Combinations of conditions could be forced onto the network to check the cumulative effect. An exciting possibility of this work is that it could lead to simplified models and operational guidelines for network managers as to what conditions may cause their network to either fail or enter an unstable state.

3 The Object Framework

3.1 Overview

This section details the structure of the first attempt to provide a suitable framework for the experiments envisioned. As the work is still in progress, the choice of tool may alter as problems and changing needs arise. A choice had to be made as to an appropriate simulation tool for the work and the following properties were deemed essential:

- A process based language utilising discrete events.
- An object oriented design capability for easy scalability and re-use of the objects created.
- For the system to be efficient.
- For the system to be simple enough for rapid prototyping and modification of the model.

For the 'first cut' of the simulator the **DEMOS** (Discrete Event Modelling On SIMULA) system was chosen. SIMULA [2] itself contains sufficient simulation primitives to build simulation models, but leaves many of the implementation issues up to the user. DEMOS provides a standardised approach by allowing the creation of models composed of **entities**. Entities may compete for specified resources, give and take items from buffers, interrupt and restart each other, wait for specified events to occur and wait for specified amounts of simulation time. Simulation models can be built relatively quickly from the primitives supplied in DEMOS. Particularly useful features include a selection of random number distributions and the automatic reporting facilities available. An example of the use of DEMOS to model WAN protocols may be seen in [6].

3.2 Basic DEMOS constructs used

The following classes of DEMOS object were used:

BIN representing an unbounded buffer.

WaitQ representing a master/slave relationship, where one process gains temporary control of another.

CondQ where a process can `WaitUntil` a condition is true. An entity which can alter the condition issues a `Signal` command to prompt waiting processes to reevaluate the condition.

INTERRUPT where one process can awaken another from a `Hold` and set a flag for it to check.

3.3 Increasing the efficiency of the simulations

As regards efficiency, it is clear that trying to simulate an entire network by individually simulating each point-to-point cell transmission is impractical even on a powerful computer. In this work the lowest form of transmission is as a group (or 'unit' as it is called in the software) of cells. One way of imagining a unit is as a burst of cells. The thinking behind this approach is that we are mainly interested in 'critical' events such as cell loss due to buffer overflow. During non-critical activity the successful transmission of a group of cells is represented by a time lag (a DEMOS `Hold` command) equal to the time taken for the transmission thus avoiding the need for a great deal of computation. When a critical event occurs the entities logically connected either side of the buffer involved are designed to interrupt each other appropriately. On each interrupt the state of the buffer is altered to accurately mirror the state of the buffer at that simulation time (i.e. the buffer contains the correct number of cells). In the worst case scenario, the model effectively degenerates to simulating cell by cell transmission into and out of the buffer to ensure that cells are lost only when they really would be.

It is hoped that for really large scale simulations this approach will ensure that large numbers of cell transmissions can be simulated in reasonable amounts of compute time.

3.4 Abstracting from real networks

A typical ATM network may comprise of workstations connected either directly to the ATM network or via a multiplexing router or bridge from a LAN or MAN. The ATM network itself will contain switches for routeing cells to different destinations. Of course, a workstation or LAN/MAN may well be directly connected to a switch. Clearly the basic building blocks of a simulation model hoping to simulate such networks must at least include the following building blocks:

- A generic 'source' entity used to simulate the various types of traffic representing the communication from computer applications.
- A generic 'sink' to receive communicated cells from a source entity.
- A 'multiplexor' entity to simulate a shared transmission line.
- A 'de-multiplexor' entity to complement the multiplexor.
- A 'switch' entity to route cells between different virtual channels and paths.

It was realized that a completely separate switch entity was unnecessary as the function of a switch could be performed by connecting a multiplexor and demultiplexor appropriately. With this in mind the entities described in the following sections were designed with consistent interfaces to accommodate the modelling of arbitrary ATM networks.

3.4.1 The Unit entity

The 'unit' in this work is the lowest form of cell transmission. It is best to think of a unit as a typical 'burst' of cells. The unit entities passed around in the simulation carry information such as the original number of cells in the unit and a counter detailing how many cells get lost during the transmission lifetime. A simple addressing scheme is also implemented for use in switches (multiplexor/demultiplexor pairs).

3.4.2 The Basic Source and Sink entities

The basic source entity generates units (representing bursts of cells) governed by a user defined method (e.g. in an example source a Poisson distribution is used to determine the number of cells in a unit and a negative exponential distribution is used to determine the length of time between units). VBR and CBR services can be represented by appropriately setting the number of cells per unit and the inter-unit generation times. The source is connected to a DEMOS `BIN` resource (effectively a integer counter representing the number of items in a buffer as opposed to actually storing the items) acting as a buffer. A DEMOS `WaitQ` is also connected to store unit entities passing from the source to its sink. The source is passed a *reference* (the SIMULA name for a pointer) to the entity acting as its sink as well as a parameter which sets the source's transmission speed.

The basic sink entity also has connections for a `BIN` resource and a `WaitQ`. Several parameters are passed to the sink including a reference to its source entity and a number to set the cell receipt speed. The source and sink entities are also passed the user selected size of the buffer interconnecting them.

Connecting a basic source and sink together with an appropriate buffer (`BIN` resource) and `WaitQ` connected between them simulates the classic producer-buffer-consumer problem. The action of the source is to generate unit entities according to the chosen method and then to simulate the transmission of this number of cells into the buffer. The first step is to place the newly created unit entity into the `WaitQ` to be serviced by the sink entity. The next step is to *interrupt* the sink entity if it is currently receiving cells. This forces the sink to update the `BIN` by removing the number of cells it would have received by that simulation time. This ensures that the true amount of free space is available in the buffer. The time taken for the transmission is the product of the source transmission speed and the number of cells in the unit. If there is enough space in the buffer for all of the cells in the unit the source updates the `BIN` to represent the placing of the cells in the buffer and then holds for the transmission time before wakening to produce the next unit as appropriate. If there is not enough space for all of the cells in the unit the source updates the `BIN` with the number of cells it can send and holds for a time to mark the transmission of those cells. It then wakes and interrupts the sink to update the `BIN` status again. This repeats until the source reaches the end of the time allocated for the transmission of that unit. If any cells of the unit remain unsent then they have been lost and the appropriate counter in the unit is set accordingly.

The sink entity *coopts* unit entities produced by the source and placed in the `WaitQ` for servicing. The sink simulates the emptying of the buffer by holding for a time to represent the time it would take to remove the cells from the buffer (simply the product of the receipt speed and the number of cells to receive). When the required time is up, the sink wakes and updates the `BIN` accordingly unless it is woken beforehand by an interrupt from the source. If an interrupt occurs, the sink calculates how many cells it would have removed from the buffer by that simulation time and updates the `BIN` accordingly. More complex interactions occur when the source and sink are dealing with the same unit at the same time. The added complication here is that the sink does not yet know whether the source will lose any cells. If the source loses cells it produces a special interrupt which is only processed if the source and sink are dealing with the same unit. Upon this type of interrupt the sink has to change several parameters such that it will only try to receive the number of cells successfully transmitted in that unit. Once the sink has waited for a time representing the number of cells successfully transmitted it may either hold (for some specified time) or immediately wait until a new unit arrives from the source in the `WaitQ`.

Slight variants of the source and sink entities have been created for use with the multiplexor and demultiplexor entities as described in section 3.4.3.

3.4.3 The Multiplexor and Demultiplexor entities

The multiplexor entity is designed to communicate with slightly modified sink entities. The difference in the sink entity is that it interrupts the multiplexor to determine whether it is allowed to start removing cells from its input buffer. Multiple instantiations of these sinks may be connected to the multiplexor. The multiplexor contains some method of call admission (currently a very simple bespoke bandwidth sharing algorithm) provided by the user. When a sink is allowed to proceed it interacts with its source and `BIN` as before. The multiplexor passes details of the unit being accepted by the sink to the demultiplexor. A DEMOS `CondQ` is used to allow the sinks connected to a multiplexor to determine whether they are allowed to receive. A sink simply issues a `WaitUntil` command depending on a logical condition linked to the `CondQ`. When the multiplexor changes a condition it issues a `Signal` command to the `CondQ` which wakes each sink holding in a `WaitUntil` and makes them reevaluate the condition. The multiplexor holds references to each sink and conversely each sink holds the reference to the multiplexor. This allows each entity to examine and set parameters in the other to avoid the need for messy global parameters.

The demultiplexor entity communicates with slightly modified source entities. The difference to the source entity is that the user supplied mechanism for generating units has been replaced by the demultiplexor passing the unit received from the multiplexor directly to the source. The source uses this information to interact with its attached sink and `BIN` as before. Once again entity references are used to avoid messy global variables.

As previously stated the need for a separate switch entity has been avoided by intending that the multiplexor and demultiplexor entities (along with the modified source and sink entities) can be used as a switch. To enable this, the multiplexor and demultiplexor have been provided with abilities to introduce extra delay on unit transmission and to decode the simple addressing scheme carried by each unit. Suitable parameter passing adjusts the behaviour as desired.

3.5 The current state of the Object Framework

So far working versions of each of the entities described in section 3.4 have been produced and are in the process of continual refinement. Work is continuing on the verification of the behaviour of the entities when interconnected to form basic building blocks such as a switch. Figure 1 shows an example of how you would build a simple network consisting of several workstations connected to a switch from the current selection of framework entities.

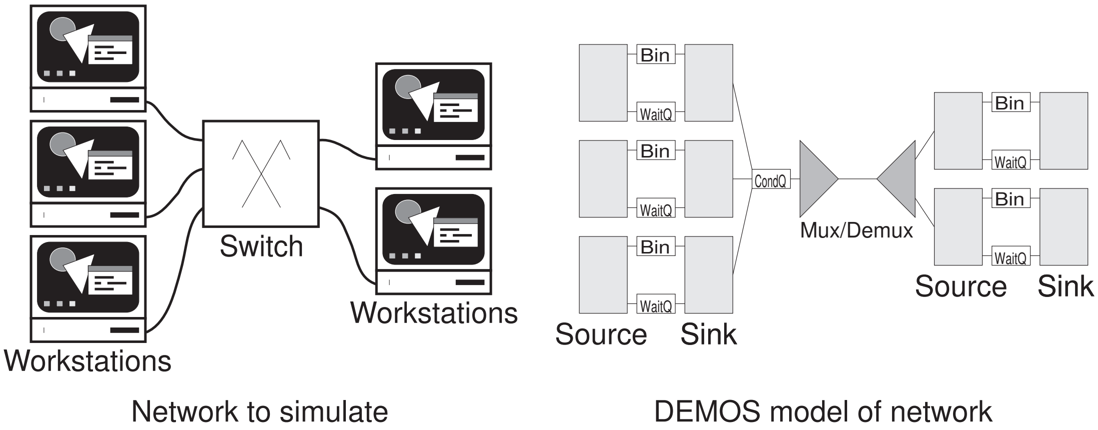

Figure 1: Simulating a simple network with the present object framework.

4 Early Experiences

At this early stage in the work it is possible to comment on some of the strengths and weaknesses of the approach and the tools chosen.

DEMOS has proved to be a reliable and flexible tool with which to build the object framework. The use of objects has led to some fairly rapid entity implementations. As long as the choice of interface remains stable the design of the entity may be modified easily. This approach will allow the design of workload 'plug in' modules to build in realistic traffic profiles. The main advantage of an object oriented approach is in the ease of creating simulation experiments. Producing an ad hoc network model requires the appropriate entities to be instantiated and 'wired up' to each other and their appropriate resources and queues. In this way successful validation of small scale experiments will allow larger experiments to be constructed quickly. There is little experience of how the DEMOS system will cope with very large simulations but this is of little relevance when developing the techniques for this approach as we only require small simulations for validation.

A difficult problem to solve is how to abstract the model from real networks. The 'unit' entity approach is the first attempt at trying to provide an efficient yet accurate approach to handling large numbers of simulated cell transmissions. It is possible, therefore, that changes may have to be made to this approach if and when problems arise.

The DEMOS system provides a selection of text based automatic reporting for each simulation. However, the most useful output for experiment validation is the event trace file which details each and every action for each and every entity. At the best of times the trace is fairly indigestible, but the timing diagrams derived by hand can be essential in ironing out interaction problems. A better approach would be to consider the implementation of a graphical reporting system to produce suitable timing information automatically. The authors are currently investigating a number of approaches.

5 Future Work

As this is clearly a work in progress the basic entities described in this paper will have to evolve in order to provide the object framework required for the experiments envisioned. The most obvious defect at this time is the 'one way' nature of data flow, i.e. sources just being able to send to sinks. One possible refinement may well be to have composite entities which contain both a source and a sink and the data reporting facilities consistent with the higher level protocol being investigated. Another obvious refinement is to merge the actions of the multiplexor and demultiplexor into one general 'muxdemux' entity to allow seamless bidirectional flow. The muxdemux entity would be able to connect to either sources or sinks and handle them appropriately. Experience gained from entity development thus far will all be very useful in the construction of such entities.

Other work will concentrate on the handling of data units across the simulation models. This is the first attempt at providing an efficient level of abstraction. As the simulation models grow this approach may even be too detailed and other approaches such as replacing certain key entities with statistical distributions may be necessary. For example, a compound switch entity can be produced at present which contains internally a multiplexor and a demultiplexor. The entity initialisation of the components of such a switch entity is performed through appropriate parameter passing. As the level of abstraction increases we may wish to replace this switch component with a simpler entity which utilises a statistical distribution to simulate the switch function. As long as the interfaces to the new switch remain the same, a trivial code replacement is all that is required (see figure 2).

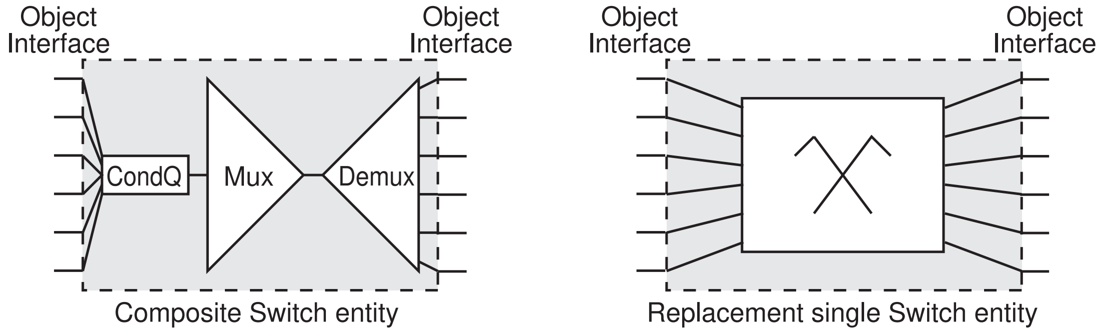

Figure 2: Replacing a compound switch entity in the object framework

In the short term, some work will be concentrated on providing a simpler and quicker form of experiment validation. The current method of following trace files by hand is far too cumbersome. Graphical techniques appear to have the most promise in this area.

In the longer term, experiments on a large scale will have to be planned and implemented in order to try and answer in detail some of the questions posed in section 2.

6 Conclusions

The early work completed thus far has formed a basic framework for simulation of ATM networks. The object oriented design approach and the choice of DEMOS as the implementation system look promising for future development. Much work needs to be done to evolve the current simulator into a system powerful enough to perform the experiments listed. However, as results look promising from this early work the authors are confident of being able to produce such a system.

References

[1] G.M. Birtwistle "Discrete Event Modelling on Simula", Macmillan Press, 1979.

[2] G.M. Birtwistle *et al.* "SIMULA BEGIN", Studentletteratur, Lund, Sweden, 1973.

[3] Joan García-Hara *et al.* "ATMSWSIM An Efficient, Portable and Expandable ATM SWitch SIMulator Tool", Lecture Notes in Computer Science 794, Austria, May 1994

[4] Rainer Händel *et al.* "ATM Networks - Concepts, Protocols, Applications", 2nd Edition, Addison-Wesley, 1994.

[5] Craig Partridge "Gigabit Networking", Addison-Wesley, 1994.

[6] R.J. Pooley and G.M. Birtwistle "Process Based Modelling of Communications Protocols" in S. Schoemaker Ed *Computer Networks and Simulation III*, North Holland, 1986, pp 81-101