Application of Petri Nets to Communication Networks (LNCS 1605), pp. 150–175.
Copyright © 1999 Springer. The original publication is available at www.springerlink.com.
DOI 10.1007/BFb0097776.

Timed Petri Net Models of ATM LANs

M. Reid and W.M. Zuberek

Department of Computer Science, Memorial University
St.John's, NL, Canada A1B 3X5

Abstract. The Asynchronous Transfer Mode (ATM) is a fast packet–switching communication method using small fixed-length cells. A model of an ATM LAN is presented which provides a realistic representation of data transmission by modeling both the ATM network and the applications running over it. Colored Petri nets are used to create a compact model that is capable of representing a variety of different protocols at a high level of detail. The model is designed to allow easy reconfiguration or addition of details at different levels of the system. Simulation is used to evaluate the performance of the modeled system, and some results are compared to actual data gathered from the campus network at Memorial University.

1 Introduction

The basic premise of ATM (Asynchronous Transfer Mode) [8] is that information of all types (i.e., video, audio, data) is divided into small fixed-length data units (cells), which can then be sent across switching networks to be recombined at the receiving end. The challenge for an ATM communication system designer is to ensure that this cell–based communication system can provide correct behavior for a video stream as well as a phone call.

Since many of the performance promises of ATM depend on its ability to transmit cells at high rates with little or no loss, much of the research has focused on switch design [4], with the input data represented by fairly simple models. However, ATM systems are now starting to appear as data network backbones. One particular method of using ATM to transmit data is LAN emulation, which attempts to simulate an Ethernet in such a way that the user is unaware that the ATM network exists. There has not been a great deal of study on how such an ATM backbone will behave under a real network load, or how ATM will affect the applications using it.

This paper presents a model of an ATM LAN that provides a realistic model of data transmission over ATM. It does so by modeling the applications that are running over the ATM LAN, as well as the ATM network itself. Most current research uses simple stochastic state–based models or queueing models to describe an ATM network; such approaches, however, ignore the synchronization aspects of network protocols. This synchronization can easily be represented in Petri net models [21, 14]. Furthermore, Petri net models can also represents the applications directly in terms of different protocols and numbers of active users, and this permits the designer to estimate the impact of these variables on an ATM backbone.

An analysis of network behavior on the campus of Memorial University has shown that while network protocols can exhibit quite complex behavior under specific situations, much of this behavior does not appear on a LAN in relatively non–congested periods of operation. If the net behavior under congested conditions is less important than detecting when congestion might occur, then many elements of protocol behavior can be ignored. The remaining behavior is remarkably similar across a set of protocols carried at Memorial's network.

To capture this similarity of behavior while still permitting the individual protocols to act independently, the model is based on timed Petri nets with deterministic and exponentially distributed firing times [25]. For modeling systems that contain similar components, colored Petri nets are quite convenient because similar components can essentially be superimposed on one another, and distinguished by token attributes (called colors). The colored tokens can operate independently from one another or interact, as required by the modeled system. The structure of a colored net model represents the basic behavior common to all components (in this case, protocols), while the different colors are used to model the differences from the common model, such as temporal characteristics or packet sizes of individual protocols.

The model presented in this paper is modular, with well defined boundaries between modules, and with modules corresponding to easily identifiable entities of the modeled LAN. The modules can be modified independently from each other, and re–configured into different network models to reflect different structures of physical entities and protocol stacks in the modeled networks. The modular structure of the model can also be used to obtain approximate, preliminary results from a simplified model, in which some of the modules are replaced by simple elements with 'typical' or 'average' properties.

The approach to building the resulting composite model, the attention paid to modularity, and the focus on capturing interactions between applications and protocols become visible and important only when the entire network, not just its component entities, is considered. This composite model provides a very useful template for simple modeling of communication networks. Following this template will require only minor modeling effort, while the returns in terms of results can be significant if the model is validated properly and the results interpreted thoughtfully.

Although analytical solutions are possible for many classes of Petri nets, the model uses a number of extensions which prevent such analysis. Therefore, simulation is used to evaluate the performance of the model [26]. Once the model is validated (by comparing its performance to the real data collected by monitoring the campus network), it can be used for different performance studies, including:

- investigation of the effect of an increased load of a particular protocol on the network load by other protocols,
- investigation of the effects of additional network elements (bridges, switches) on the performance of the network,

– identification of network bottlenecks,
– optimization of network structure to maximize its performance, and so on.

Some simple results are included as an illustration of such studies.

The description of the proposed net model is rather informal as the emphasis of this paper is on a structured modeling approach rather than on the derivation of a specific model. Consequently, some more detailed aspects of the model are not addressed here; more details and other simulation results can be found in [20].

The paper is organized in five sections. Section 2 describes the modeled environment which corresponds to the campus network at Memorial University. Section 3 recalls basic concepts of timed Petri nets and then discusses the Petri net model of the network. Some performance results are presented in Section 4, while Section 5 contains a short discussion of the proposed approach and a number of concluding remarks.

2 Modeled Environment

The protocols and data used in this model correspond to the communication network implemented on the Memorial University campus. The protocols can be represented by the protocol stack shown in Fig.2.1.

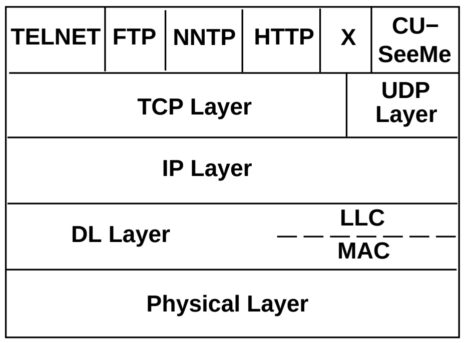

Fig.2.1. Protocol stack.

The protocols (from top) include:

– Application protocols, which usually act as the direct point of contact between the user and the network; these protocols include:
  - TELNET, providing a remote terminal connection from one host to another; TELNET is normally used for text–based interactive computing; RLOGIN, which performs a similar function as TELNET, is combined with TELNET in this study;

  - FTP, probably the most commonly used application for transferring files from one host to another;
  - NNTP, used for transferring USENET articles from host to host;
  - HTTP, used to transmit most of the information on WWW, although some other protocols are used as well;
  - X WINDOWS: X is a client–server based system for the management of remote graphics displays;
  - CU–SeeMee, a video conferencing application designed (at Cornell University) to work over the Internet.

– TCP/IP, a suite of networking protocols that have gained wide acceptance as the basis for the global Internet. The TCP protocol is built on a connectionless datagram service (the Internet Protocol, IP) with primarily two higher level services – UDP, which offers a connectionless service, and TCP, which provides a reliable connection–oriented service. Most higher–layer applications use one of these two services to transmit data between hosts.

– Ethernet, a Data Link (DL) layer Media Access Control (MAC) system designed for Local Area Networks (LANs), based on a bus topology with control of the medium distributed among the stations attached to the bus. Access to the transmission medium is by a method commonly known as CSMA/CD (Carrier Sensing Multiple Access / Collision Detection); each station can detect if another station is transmitting, and if so, it refrains from attempting to send the packet for a random period of time and then tries again. The Logical Link Control (LLC) sublayer is the other part of the divided Data Link Layer [8].

ATM is a method of transferring information (data, voice, video) using small fixed–length cells. ATM is connection–oriented; a data transfer between two entities over an ATM network will follow a path determined (by a signaling protocol) before the transfer begins. Each data connection represents a different virtual path. Although cells are always in sequence on any given virtual channel, the channels are multiplexed together through switching devices and underlying media. The ATM protocol stack is shown in Fig.2.2.

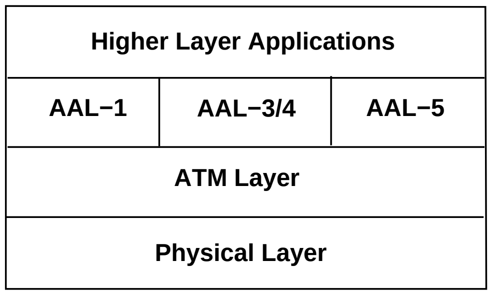

Fig.2.2. ATM protocol stack.

A number of higher–level protocols, called the ATM Adaptation Layer (AAL), provide different classes of service to upper level applications [8]. For example, AAL–1 provides a constant bit rate time division multiplexor service suitable for voice transmission, while AAL–5 provides variable rate service for data blocks of varying size for LAN traffic. In all cases, the information is eventually broken into 53–byte cells (5–byte header and 48–byte payload) at the ATM layer. The use of small fixed–length cells permits the design of very fast ATM switching devices.

The ATM Adaptation Layer is divided into two sublayers, the Segmentation And Reassemly (SAR) sublayer and the Convergence sublayer (CS). The SAR sublayer brakes down the original frame or other data unit into cells to be sent by the ATM Layer, and also reconstructs the original data unit from a sequence of cell at the other side of the connection. The CS provides the mechanism for mixing the different requirements of voice, video and data by defining a number of classes of service, each with appropriate parameters for the service. These are used to provide the proper quality of service (QoS) parameters on the connection.

Most existing applications do not interface directly with the ATM. Since they are designed to use more traditional protocols such as IP or IPX, there has been a considerable amount of work in designing interfaces that allow these protocols to operate over ATM. The IP–over–ATM standards of the IETF and MPOA [2] are examples of work in this area.

Another attempt to interface traditional protocols and communication systems with ATM is LAN emulation [8], which simulates a MAC layer (either Ethernet or Token Ring) over an ATM network. Any application or protocol that would normally operate over an Ethernet or a Token Ring network can work without modification on a LAN emulation network; the presence of ATM is hidden from the upper level applications.

Each host that is a part of an emulated LAN performs certain services for the emulated LAN. Most of these services are activated when a client (which can be a single computer or a bridge between an ATM and another type of network) first joins an emulated LAN or sends a broadcast packet. A data transfer between two hosts is usually carried over a single ATM VC (virtual circuit) without the involvement of the LAN emulation services.

At Memorial University campus, the new backbone replaces the Ethernet and routers with an emulated LAN built on ATM. The end–point LANs attach to ATM/Ethernet bridges which act as LAN emulation clients. If the two end–point LANs are in the same virtual LAN (i.e., several separate LANs that ‘look’ like one big LAN), then the data path is as shown in Fig.2.3; the data flows between the two ATM/Ethernet bridges via a direct ATM VC. However, if traffic is between two end–point LANs that are not in the same virtual LAN, then the data path is as shown in Fig.2.4; the path includes a router as an intervening device. An increasing number of (new) hosts are connected directly to the ATM medium.

A considerable variety of modeling and analysis methods have been applied to ATM, the main focus being the representation of input traffic to a switch or

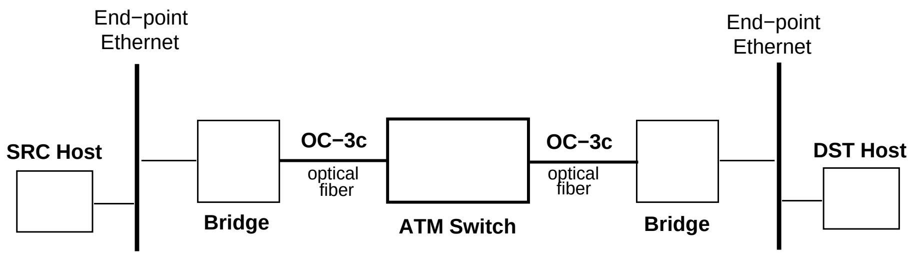

Fig.2.3. Typical data path – same virtual LAN.

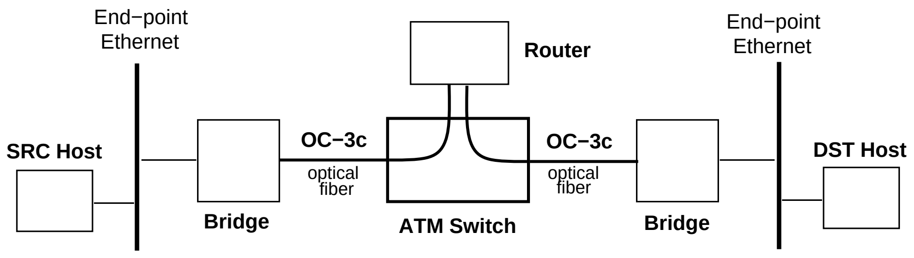

Fig.2.4. Typical data path – different virtual LANs.

network of switches. A summary of such methods can for instance be found in [23].

A popular input model used in ATM analysis is the MMPP (Markov Modulated Poisson Process) [17]. This and related models are used in [24] and [7] to estimate cell loss probabilities in ATM networks. Queueing models are used to study transmission delays in ATM networks [15, 16], and also buffer allocation within an ATM switch [12].

Discrete–event simulation has also been used to analyze ATM performance. A simulation comparison of ATM, Frame Relay, and DQDB is given in [19]. A simulator, specifically developed for ATM–based systems, is described in [1].

The issues involved in operating traditional protocols over ATM have generated a number of studies of IP performance over ATM. The issue of protocol overhead has been examined in [3, 17], while [18] analyzes the effect of TCP/IP and system design on IP–ATM performance. [22] evaluates two strategies for effective discard of packets in an ATM environment, while [13] examines in detail a deadlock situation that can occur with TCP over ATM.

The approach presented in this paper uses a simple behavioral model of ATM. A good conformance of simulation results and real measurements indicates that even this simple model is quite satisfactory for many performance studies.

3 Petri Net Model

This section first briefly recalls basic concepts of timed Petri nets, and then describes the timed net model of an ATM network.

3.1 Basic concepts of timed Petri nets

The inhibitor (place/transition) Petri net is usually defined as a system composed of a finite, nonempty set of places P, a finite, nonempty set of transitions T, a set of directed arcs A, connecting places with transitions and transitions with places, $A \subset P \times T \cup T \times P$, a set of inhibitor arcs B, connecting places with transitions, $B \subset P \times T$, and an initial marking function m_0 which assigns nonnegative numbers of so called tokens to places of the net, $m_0 : P \rightarrow \{0, 1, ...\}$. Usually the set of places connected by (directed) arcs to a transition is called the input set of a transition, and the set of placed connected by (directed) arcs outgoing from a transition, its output set. Similarly, the set of places connected by inhibitor arcs to a transition is called its inhibitor set.

A place is shared if it belongs to the input set of more than one transition. A net is conflict–free if it does not contain shared places. A shared place is (generalized) free–choice if all transitions sharing it have the same input sets and inhibitor sets. Each free–choice place determines a class of free–choice transitions sharing it. It is assumed that selection of a transition for firing in a free–choice class of transitions is a random process which can be described by (free–choice) probabilities assigned to transitions in each free–choice class. Moreover, it is usually assumed that the random choices in different free–choice classes are independent one from another.

A shared place is guarded if for any two transitions sharing it there exists another place which belongs to the input set of one of these two transitions, and the inhibitor set of the other transition. If a place is guarded, at most one of the transitions sharing it can be enabled by any marking function.

A shared place which is not free–choice and is not guarded, is a conflict place. The class of enabled transitions sharing a conflict place depends upon the marking function, so the probabilities of firing conflicting transitions must be determined in a dynamic (i.e., marking–dependent) way. A simple but usually satisfactory approach is to use relative frequencies of transition firings assigned to conflicting transitions [9]; the probability of firing an enabled transition is then determined by the ratio of transition's (relative) frequency to the sum of (relative) frequencies of all enabled transitions in a conflict class. Another generalization is to make such relative frequencies (and probabilities of firings) dynamic, depending upon the marking function, for example, by using the number of tokens in a place rather than a fixed, constant number as the relative frequency.

In ordinary nets the tokens are indistinguishable, so their distribution can be described by a simple marking function $m : P \rightarrow \{0, 1, ...\}$. In colored Petri nets [11], tokens have attributes called colors. Token colors can be quite complex, for example, they can describe the values of (simple or structured) variables or the contents of message packets. Token colors can be modified by (firing) transitions and also a transition can have several different occurrences (or variants) of its firings, for different combinations of colored tokens.

The basic idea of colored nets is to 'fold' an ordinary Petri net. The original set of places is partitioned into a set of disjoint classes, and each class is replaced by a single place with token colors indicating which of the original places the tokens belong to. Similarly, the original set of transitions is partitioned into a set of disjoint classes, and each class is replaced by a single transition with occurrences indicating which of the original transitions the firing corresponds to.

In order to study performance aspects of Petri net models, the duration of activities must also be taken into account and included into model specifications. Several types of Petri nets 'with time' have been proposed by assigning 'firing times' to the transitions or places of a net. In timed nets, firing times are associated with transitions (or occurrences), and transition firings are 'real–time' events, i.e., tokens are removed from input places at the beginning of the firing period, and they are deposited to the output places at the end of this period (sometimes this is called a 'three–phase' firing mechanism as opposed to a 'one–phase', instantaneous firings of nets without time or stochastic nets).

In timed nets, all firings of enabled transitions are initiated in the same instants of time in which the transitions become enabled (although some enabled transition cannot initiate their firings). If, during the firing period of a transition, the transition becomes enabled again, a new, independent firing can be initiated, which will 'overlap' with the other firing(s). There is no limit on the number of simultaneous firings of the same transition (sometimes this is called 'infinite firing semantics'). Similarly, if a transition is enabled 'several times' (i.e., it remains enabled after initiating a firing), it may start several independent firings in the same time instant.

The firing times of transitions can be constant (or deterministic), or can be random variables described by a probability distribution function. Exponentially distributed firing times (sometimes also called stochastic or Markovian firing times) are particularly popular because of the memoryless property of models with such firing times.

The firing times of some transitions may be equal to zero, which means that the firings are instantaneous; all such transitions are called immediate (while the other are called timed). Since the immediate transitions have no tangible effect on the (timed) behavior of the model, it is convenient to first fire the (enabled) immediate transitions, and then (still in the same time instant), when no more immediate transitions are enabled, to start the firings of (enabled) timed transitions. It should be noted that such a convention introduces the priority of immediate transitions over the timed ones, so the conflicts of immediate and timed transitions should be avoided. Similarly, the free–choice classes of transitions must be 'uniform', i.e., all transitions in each free–choice class must be either immediate or timed.

There are three basic approaches to analysis of timed Petri net models. For some classes of nets, structural methods, and in particular invariant analysis, can provide performance characteristics. In other cases, when the model is bounded and not excessively complex, the (exhaustive) reachability analysis can be used to find the stationary probabilities of states, and to derive performance characteristics from these stationary probabilities. Finally, the most general (but also the least flexible) approach to analysis of net models is to use discrete–event simulation. The last approach is used to obtain some performance characteristics of net models developed in this paper.

3.2 Complete Model

The system modeled in this paper is described in the reference model shown in Fig.3.1; this model can be thought of as a “cross–section” through the backbone configuration (Fig.2.3).

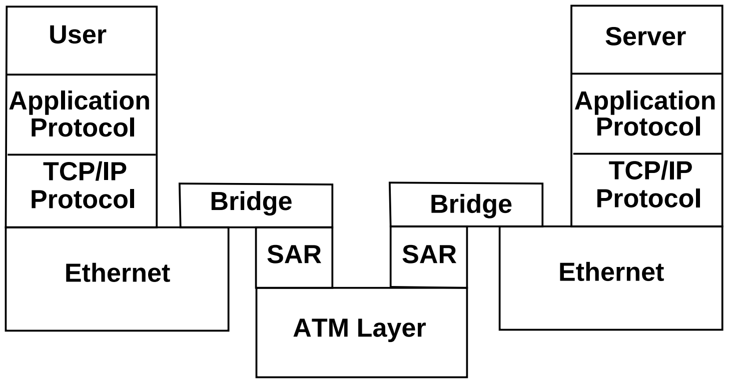

Fig.3.1. Reference model.

The model follows the layered structure of the reference model. Each layer is represented by a module of the net, which allows to change a particular layer independently of the other layers. Moreover, it was decided to originate all transfers from one side of the network, which is a typical pattern for much of Memorial’s traffic (personal computers and workstations requesting various services from central servers).

The complete model is shown in Fig.3.2, with sections corresponding to the layers of the reference model. The user and application protocols from Fig.3.1 are represented by single transitions with corresponding delays.

The complete colored model can be considered as a stack of identical nets superimposed one upon another, with different nets representing different application protocols. The superimposed nets are independent at the SRC (source) and DST (destination) processes at each end, but dependencies exist between the layers at the intervening network sections, representing resources such as an Ethernet which can only transmit one frame at a time. In each layer, each transition should be regarded of as a collection of occurrences corresponding to different application protocols. Again, each transmission is independent of the others at the User/Application and TCP/IP layers, but the transmissions must be serialized at the network level.

3.3 User/Application Level

The basic model at this level is a user running an upper level application (such as TELNET or FTP). The user is assumed to spend some time thinking, after

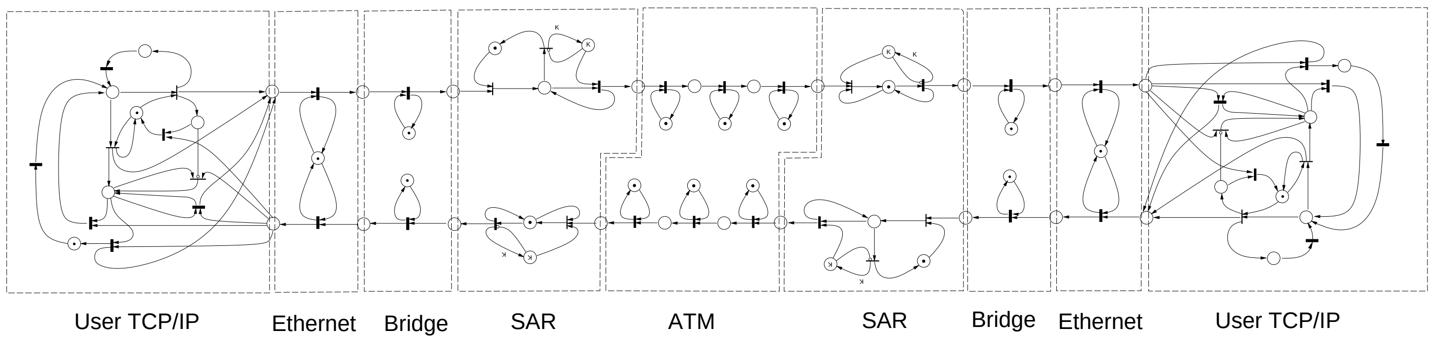

Fig.3.2. Complete model.

which a request is sent to the remote computer. This causes a data transmission from the originating host (SRC) to the destination host (DST). The SRC host then waits for a reply from the DST host, and when it is received, the SRC host returns to the thinking state.

Fig.3.3 sketches the net model at the User/Application level (as usual, timed transitions are represented by 'thick' bars, and immediate transitions by bars; moreover, inhibitor arcs have small circles instead of arrowheads). The think time is represented by the (firing time of) transition S_TNK, while the "Network" transitions (with the dashed arcs) model data transmissions. The DST process (transition D_TNK) introduces certain delay (to process the request) before replying with a data transmission back to the SRC process.

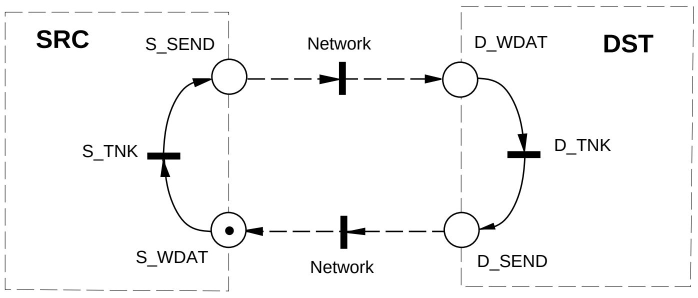

Fig.3.3. User/Application level model.

The token in S_WDAT represents the initial marking of the model, with each protocol represented by a token of different color.

3.4 TCP/IP Level

Fig.3.4 shows a net model of the User/Application level with a TCP/IP model. The model assumes that packets cannot be lost, and that SRC and DST processes will always respond fast enough to prevent re–transmissions. It is quite straightforward to add timeout mechanisms [25] which would retransmit the lost or distorted packets. On the other hand, very low probability of such events results in quite insignificant influence of these events on the performance of the network, so they are not represented in Fig.3.4.

While the User/Application level is only concerned with the data flowing between the SRC and DST processes, the TCP/IP level deals with data and acknowledgements, since TCP provides guaranteed delivery. The packets transmitted between SRC and DST can be broadly characterized into four types:

- SRC to DST: data packets;
- SRC to DST: acknowledgement packets;
- DST to SRC: data packets;
- DST to SRC: acknowledgement packets.

Most network applications send a group of data packets, wait for a group of packets in reply, send another group, and so on. Tab.3.1 shows typical data group sizes for various protocols.

Tab.3.1. Mean size of data groups (in packets).

direction	TELNET	FTP	NNTP	X
SRC to DST	1.06	11.6	1.02	1.70
DST to SRC	1.46	9.8	5.75	1.72

For real network processes, the end of a group can be detected through the data contained in the packets. A TELNET session, for example, echoes keystrokes until it recognizes the end of the line, at which point it processes the command. The model does not have any information about the content of the packets, so a different mechanism is provided to signal the end of a data group; this is done by creating special packet types, LST, to represent the last packet in a data group (one in each direction). The 'regular' data packets are denoted as MID packets.

So, each process sends a certain number (possibly zero) of MID packets, followed by a single LST packet (it is assumed that the receiving process is sending back an acknowledgement packet, ACK, for each data packet received). When the receiving process detects an LST packet, it knows that the sender has finished the data group and is now ready to receive data packets in reply.

In the model, one color is used for the User/Application level, and further six colors are used for the six packet types (i.e., MID (type '1'), LST (type '2') and ACK (type '3') packets in the direction from SRC to DST, and another three types, MID (type '4'), LST (type '5') and ACK (type '6'), for packets in the opposite direction). This distinguishing of packet types also allows the model to represent behavior based on packet type. For example, the transmission delay of a packet through the network, which is often dependent on the size of the packet, can be based on the packet size for each type rather than the overall average packet size.

In Fig.3.4, the control tokens cycling through places `S_IDLE`, `S_SEND` and `S_WDAT` represent the User/Application level sketched in Fig.3.3. When the control token is in place `S_SEND`, both transitions `S_SMD` and `S_SLT` are enabled. This (generalized) free–choice structure is described by choice probabilities assigned to the two transitions; for example, by assigning choice probability 0.8 to `S_SMD` (which represents sending MID(1) packets) and 0.2 to `S_SLT` (which represents sending LST(2) packets), each firing of `S_SLT` will correspond (on average) to 4 firings of `S_SMD`. Data in Tab.3.1 are used to determine the values of these choice probabilities for different protocols.

When `S_SMD` fires, a token (representing a MID(1) packet) enters `N_SEND_S`, which models the host that transmits a data packet through the next layer of the reference model. A control token is also removed from place `S_WIND`. If no tokens are present in `S_WIND`, the number of unacknowledged packets in the sliding window is at maximum, and no more data packets can be transmitted until some acknowledgements arrive. Tokens are also deposited in places `S_WACK`

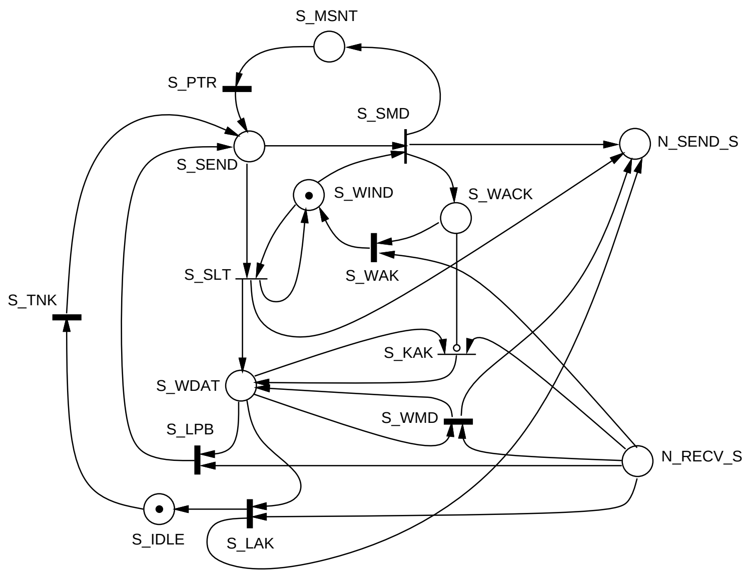

Fig.3.4. TCP/IP source (SRC) model.

and S_MSNT. The token in S_WACK waits until an acknowledgement packet (ACK) arrives from the DST process, and then transition S_WAK fires and deposits a control token back in place S_WIND (i.e., the sliding window moves forward by one packet). The firing of transition S_PTR returs a token to place S_SEND (after a delay representing the time needed for assembling a new packet of data). This cycle continues sending MID(1) packets to the DST process until S_SLT fires and sends an LST(2) packet (which indicates the end of data group).

The size of the data group is determined by the probability assigned to S_SLT (this probability can be different for each protocol); if p is the choice probability of S_SLT, then the size of the data group is a (discrete) random variable X with geometric probability density function, so:

$$\text{Prob}\{X = k\} \;=\; (1-p)^{k-1}p, \quad k = 1, 2, ...$$

When S_SLT fires (sending an LST(2) packet), a token is deposited into S_WDAT. This represents the User/Application level in “wait” mode; a command or request has been sent to the DST process and a reply should arrive (after some delay) in place N_RECV_S as a sequence of MID(4) packets followed by an LST(5) packet.

ACK(6) packets are accepted by transition S_KAK. The inhibitor arc from S_WACK to S_KAK enforces the priority for ACK(6) tokens.

A MID(4) packet arriving in place N_RECV_S is accepted by transition S_WMD. Its firing deposits a token (an ACK(3) packet) in place N_SEND_S to acknowledge the arrival of a MID(4) packet. An arriving LST(5) packet indicates the end of the data group. At this point, the SRC process can either return an acknowledgement or start the transmission of the next data group, which implies that the acknowledgement is piggybacked onto the first data packet. This is modeled by a free–choice structure of transitions S_LPB and S_LAK. Piggybacking is described by the choice probability of S_LPB; this probability can be different for each protocol.

Once an arriving MID(4), LST(5) or ACK(6) packet is handled by the SRC process, SRC returns to the start state, either through the transition S_TNK (which indicates that the User/Application level requires some 'thinking' time), or directly to place S_SEND, starting transmitting the next data group.

The DST process is a mirror image of the SRC process. However, it typically has different timing properties (the responses of the DST process represent software replies to a command or request).

3.5 UDP model

UDP provides connectionless service between two hosts. Much of the functionality required in a TCP model is not needed for UDP; UDP leaves reliability of service to the higher layers of the protocol stack. As a result, a UDP model does not need the windowing, acknowledgements or piggybacking described in the previous section.

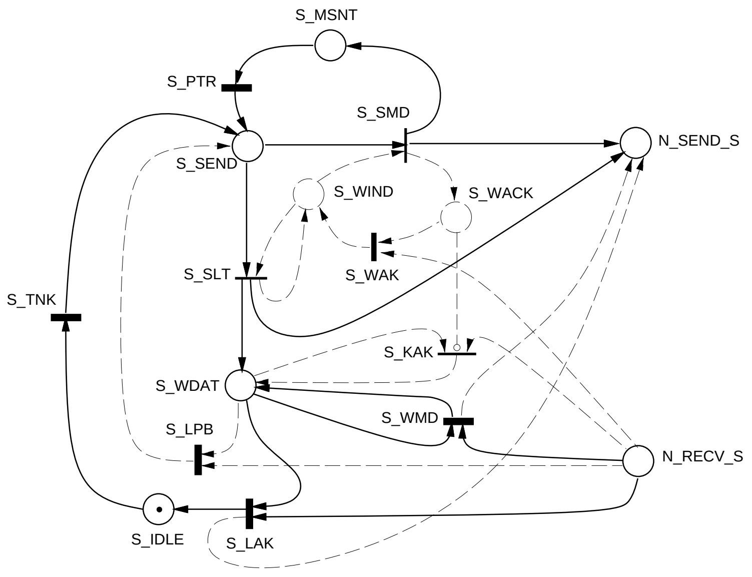

Fig.3.5. UDP source (SRC) process.

A UDP process is modeled as a subnet of the TCP process. Fig.3.5 shows a UDP model, with the unused TCP portions showed by dashed lines. The UDP model basically exchanges data groups, with the SRC process sending a series of MID(1) packets followed by a single LST(2) packet (UDP does not use acknowledgements, so the timing between packets depends on the application). As for the TCP protocol, the size of the data group is modeled by a geometrically distributed random variable.

For UDP, once the DST process receives an LST(2) packet, it begins transmitting its own data groups, finishing with an LST(5) packet. The SRC process then begins again.

3.6 LAN level

The Ethernet level of the reference model is described by a simple net (Fig.3.6) that performs two basic functions: (i) it adds a transmission delay to each packet, and (ii) it introduces 'serialization' (and blocking) as only one packet of one protocol can be transmitted at the same time.

It should be noticed that with the exception of the windowing mechanism at the TCP/IP level, each session of each protocol has, until now, been able to operate independently of the other sessions.

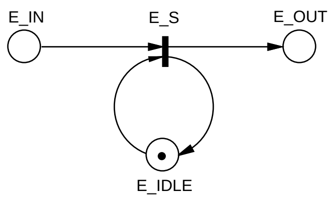

Fig.3.6. Ethernet model.

This Ethernet model assumes that all sessions are on unique hosts. That is, interactions between sessions on the same host are not represented. This is reasonably accurate for the SRC processes if these processes originate on PCs or workstations. It is less accurate for the DST processes, since these typically represent a smaller number of servers. However, delays caused by buffering on the hosts are included by default in the timing parameters used in the model. The collision mechanism of Ethernet is not modeled.

The controlling place `E_IDLE` in Fig.3.6 contains one token which assures that only one occurrence of transition `E_S` can fire at the same time. The actual transmission times are modeled by deterministic transitions, with different firing times for each packet type of each protocol. The occurrence probabilities of `E_S` are marking–dependent [26], so the relative frequencies of chosing a color are determined by the numbers of tokens of that color in place `E_IN` of the Ethernet model.

3.7 ATM level

The ATM level of reference model has two subsections: the AAL layer which provides the segmentation and reassembly (SAR) functionality (i.e., dividing packets into cells and combining cells into packets), and the cell switching functionality of the ATM layer itself.

The AAL layer is modeled in two sections. The first (Fig.3.7) shows the segmentation part of the layer – it takes a token (in the input place) representing the arrival of a packet, and generates a series of tokens representing a number of cells for transmission over an ATM switching network. The second section (Fig.3.8) represents the reverse process; it inputs a number of cells and re–creates the original packet for transmission to the next layer.

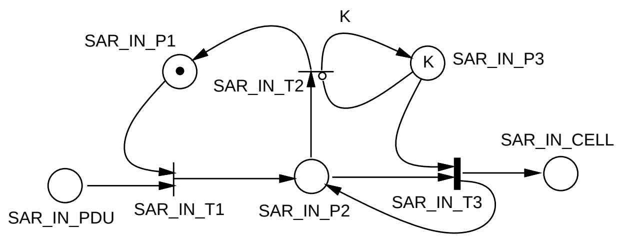

Fig.3.7. AAL (SAR) level – segmentation.

These SAR layer models take advantage of the fact that ATM is connection–oriented – cells for a particular packet must arrive in order, and there cannot be any interleaving of cells. Because of very low probability of cell losses and misinsertions, and insignificant influence of such events on the performance of the network, these very infrequent events are not represented here.

In the segmentation section, a token deposited in place `SAR_IN_PDU` represents the arrival of a packet at the AAL level. Place `SAR_IN_P1`, containing one control token, ensures that only one packet is segmented at a time. Place `SAR_IN_P3` controls the number of cells that are generated for each individual packet type. For example, if packet MID(1) of the TELNET protocol is, on average, divided into 5 cells, then 5 tokens of the color representing the TELNET MID(1) packets are placed in `SAR_IN_P3`. When a token is deposited in `SAR_IN_P2` (place `SAR_IN_P2` is guarded), transition `SAR_IN_T3` will fire a number of times corresponding to the color of the token in place `SAR_IN_P2`, generating a specific number of cells for each packet type. When the last cell has been generated, all tokens of this color have been removed from `SAR_IN_P3`, so transition `SAR_IN_T2` can fire, which removes the token from `SAR_IN_P2`, replaces the required number of tokens in `SAR_IN_P3` (to allow segmentation of another packet of this type), and returns the control token to place `SAR_IN_P1`, to start the segmentation of the next packet (of any type). It should be noticed that the values of K are associated with colors, so the numbers of cells can be different for different application protocols.

The reassembly section (Fig.3.8) re–creates a packet form a stream of cells. Tokens, which represents the arrival of cells at the SAR level, are received in place `SAR_OUT_CELL`. Place `SAR_OUT_P1` contains a number of colored tokens (for each packet type) equal to the number of cells the packet is divided into.

As cells arrive, they are accepted by transition SAR_OUT_T1. Place SAR_OUT_P2 (also a guarded place) contains one control token to ensure that only one cell is processed at a time. When all cells of a packet have arrived (which is indicated by removal of all tokens of the corresponding color from place SAR_OUT_P1), transition SAR_OUT_T2 fires, depositing a single token of the reassembled packet's color in place SAR_OUT_PDU. SAR_OUT_T2 also deposits the required number of tokens into place SAR_OUT_P1 in preparation for the next stream of cells of that color.

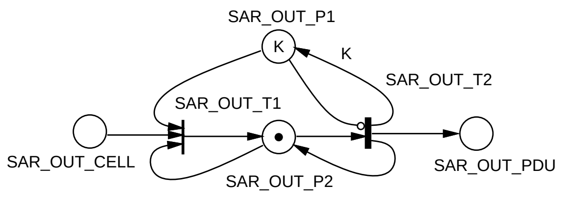

Fig.3.8. AAL (SAR) level – reassembly.

The color of the token arriving to SAR_OUT_PDU indicates the type of the application protocol the cell belongs to, and it selects the matching color of tokens in SAR_OUT_P1, so, cells of different application protocols can be re–assembled in different ways (consistent with the segmentation process in Fig.3.7).

3.8 ATM switch

The ATM switching fabric is modeled by a series of delays as shown in Fig.3.9. The delays represent the latencies of the two OC-3c SONET links and the ATM switch (Fig.2.3). The use of these three delays rather than one longer delay represents the concurrency of the stream of incoming cells (transition LINK1_T1), outgoing cells (transition LINK2_T1) and the switching process (transition SW_T1).

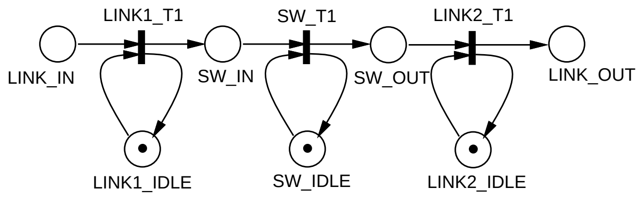

Fig.3.9. ATM switch fabric.

3.9 Other considerations

An important aspect of the model is that it should be able to represent multiple simultaneous interactions of a set of protocols. The model does not represent the contents of the packets being transmitted, so it is not obvious that the multi–session versions, obtained by simply using multiple control tokens in the corresponding modules, provide accurate representations of the real behavior. With multiple control tokens, it may be difficult to match a particular control token with the other tokens it generates. For example, when `S_SLT` fires (Fig.3.4) sending an LST(2) packet to the DST process and placing a control token in `S_WDAT`, there is no difference between an arriving LST(5) packet caused by this particular control token, and caused by an earlier or later firing of `S_SLT`.

In order to verify that such a multi–session model is valid, a different model was developed, in which unintensional interference of different sessions was eliminated by assigning a different set of colors to each session. The single protocol with n concurrent sessions (multi–session model) was compared to a multi–protocol model, i.e., a model with n unique single–session protocols, each with parameters identical to the multi-session model. The multi–protocol model more closely resembles the reality, since the unique protocols completely separate the n sessions in the same way that connection ID's and sequence numbers separate packets on a real network (the multi–protocol model uses many colors and is not feasible for large number of session). It was found that the multi–session model gave slightly higher results for the numbers of packets and bytes per second, but the results were not statistically significant when compared using standard hypothesis testing [10].

Another consideration is the priority of acknowledgements. In Fig.3.4, an ACK(6) packet deposited in place `N_RECV_S` is acknowledging either a previously transmitted MID(1) packet, or an LST(2) packet. TCP normally uses sequence numbers to differentiate between these two cases. However, since the model does not reflect that level of detail, a simpler mechanism is needed. In a single–session case, where there is only one user per protocol, so the location of the control packet determines which data packet is being acknowledged. That is, if the control token is in place `S_WACK`, representing a transmitted MID(1) packet, then the acknowledgement is for that packet. Otherwise, if the control token is in place `S_WDAT`, then the model accepts the acknowledgement for a transmitted LST(2) packet.

In the multi–session version of the model, however, there is no simple way to match a particular ACK(6) packet with the MID(1) or LST(2) packet that generated it. Two solutions were developed to address this. The first was to introduce an inhibitor arc from `S_WACK` to `S_KAK`, which effectively gives MID(1) packets a priority over any arriving ACK(6) packet. The other option was to add an additional packet type, ACK(7), and use an ACK(6) as acknowledgements for MID(1) packets, and ACK(7) as acknowledgements for LST(2) packets. The second solution, while somewhat more realistic, also adds two colors per protocol and two occurrences per transition for the intervening layers in the reference model.

The simulation results for the first option showed slightly higher values for throughput and average burst rate. When compared using hypothesis testing

[10], the difference was not found to be significant. Therefore it was decided to use the first option.

4 Model Performance

Many parameters of the model (firing times of transitions, probabilities of firings in free-choice classes, relative frequencies of firings for conflict classes of transitions) can be established on the basis of real, physical data characterizing the Memorial's network. The Ethernet backbone of the campus network was monitored for a week (the data were collected using the TCPDUMP public domain TCP/IP monitoring program), and packet and byte counts per protocol (as determined by TCP/IP port numbers) were recorded. Tab.4.1 shows the relative frequencies for the most common protocols.

Tab.4.1. Packet and byte counts for the most common protocols.

Protocol	packets	bytes
SHELL	13.8 %	33.7 %
TELNET	40.2 %	11.5 %
NNTP	5.99 %	12.1 %
NFS	7.75 %	9.72 %
SMTP	3.80 %	9.37 %
X	11.1 %	7.21 %
LOGIN	8.25 %	3.97 %
FTP (data)	1.46 %	3.76 %
HTTP	1.43 %	1.59 %
OTHER	6.24 %	4.14 %

Since many of the model parameters are estimated from timing data taken at the network level, it was necessary to take two trace sets for each protocol; one for the source processes and one for the destination processes (the "Monitor" boxes in Fig.4.1 show the data collection points in the analyzed system).

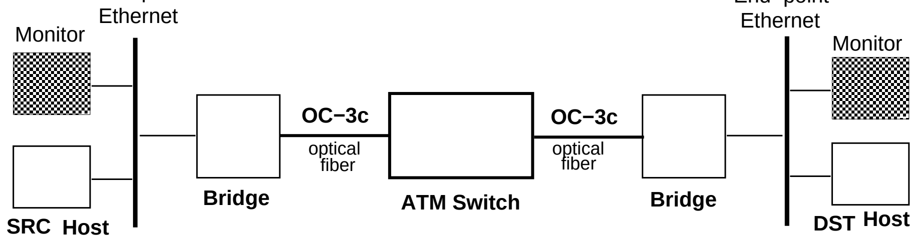

Fig.4.1. Data collection configuration.

The parameters for the SRC and DST processes were estimated by analyzing the pairs of adjacent packets in the recorded traces. An example of a matrix describing the "next packet" probability density for the TELNET protocol is shown in Tab.4.2 (for example, entry [2,4] is the probability that a packet of type "2" is followed by a packet of type "4", in the same user session).

Tab.4.2. "Next packet probability" matrix for TELNET protocol.

TELNET	1	2	3	4	5	6
1	0.105	0.035	0.009	0.000	0.000	0.851
2	0.000	0.000	0.004	0.106	0.707	0.183
3	0.016	0.585	0.012	0.200	0.186	0.001
4	0.000	0.000	0.860	0.066	0.074	0.000
5	0.006	0.367	0.627	0.000	0.000	0.000
6	0.102	0.096	0.002	0.506	0.294	0.000

A similar matrix was generated for timing information. Other parameters, also extracted from the recorded traces for the TELNET protocol, are shown in Tab.4.3.

Tab.4.3. Values of model parameters for TELNET protocol.

parameter	*average value*	*units*
SRC group size	1.051	packets
DST group size	1.456	packets
SRC window size	1.173	packets
DST window size	1.117	packets
packet size (1)	60.74	bytes
packet size (2)	60.04	bytes
packet size (3)	60.00	bytes
packet size (4)	401.8	bytes
packet size (5)	167.9	bytes
packet size (6)	60.00	bytes
thinking time	2.456	sec
reply delay	0.047	sec
packet interarrival time	0.8188	sec

The results obtained by simulation of the developed model [26] correspond quite well to the recorded data. For example, Fig.4.2 compares the latency values reported in [5] (the solid line) with the simulated results (the dashed line); both plots exhibit linear relationship between the frame sizes and the average latencies, and both plots are quite close one to another.

Fig.4.3 compares the distribution of packet sizes for FTP protocol; the solid line shows the distribution of the data collected from the network, with two 'peaks', one for the very short frames, and another one for long frames.

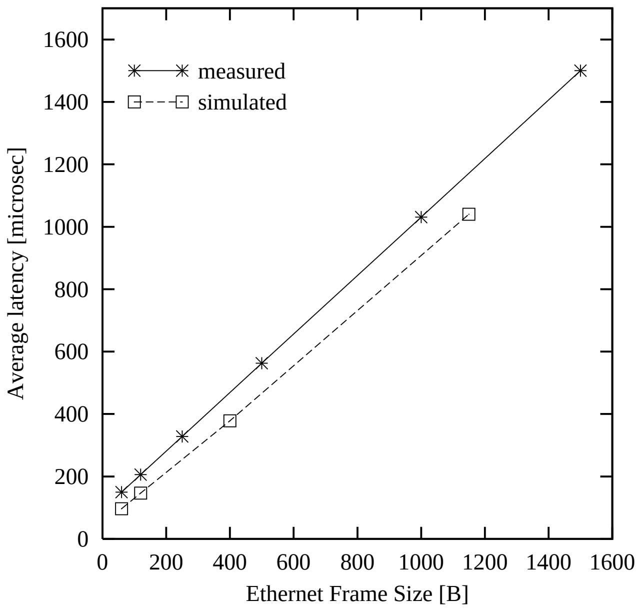

Fig.4.2. Simulated vs real ATM latency.

The simulated results also have a characteristic peak for very short frames, but there is a double peak for long frames; this double peak should be converted into a single peak, similar to the measurement data, by tuning model parameters.

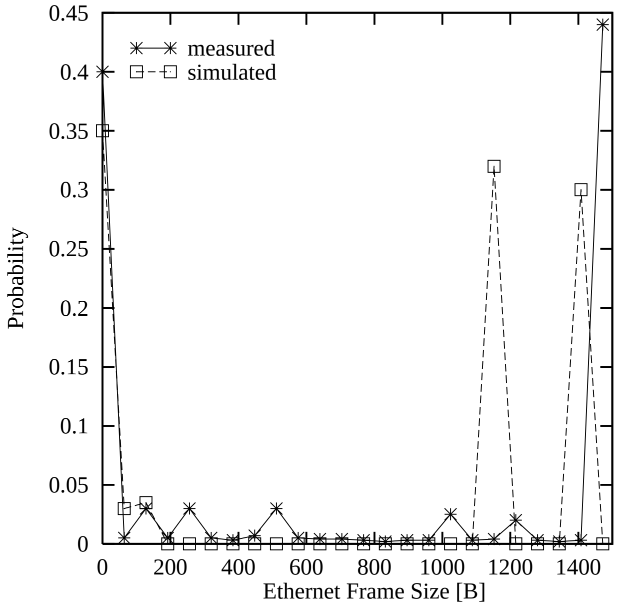

Fig.4.3. Packet size distribution for FTP protocol.

Fig.4.4 shows an example of information that can be obtained from the model [20]; it compares the maximum ATM burst rate for two protocols, using a 10 msec window. The figure indicates that even a low impact protocol like TELNET (the solid line) can create short bursts of cells at high speed, far above the average behavior. It also shows that the load of protocols can be limited by activities of other protocols – in this case, increasing (with the number of users) load generated by TELNET restricts the effective load of FTP (represented by the dashed line).

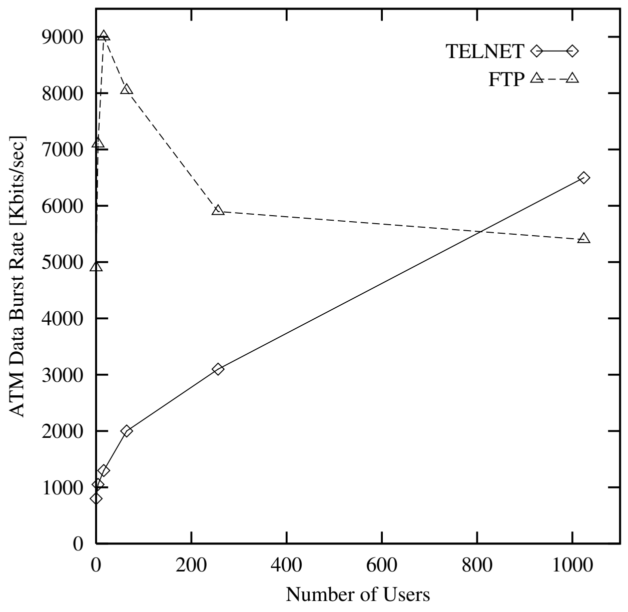

Fig.4.4. Network load by protocol – max ATM data rate.

Fig.4.5 shows the effect of TELNET and FTP protocols on CU–SeeMe; as the number of users in the system increases (it is assumed that all users have the same network traffic characteristics), the average delay of CU–SeeMe packets initially increases very slowly, but after certain number of users, this average delay grows rather quickly. The 'critical' number of users is equal to 64 (the "knee" point of the delay curve), so the limit on the number of users in order to avoid excessive delays should be 64 (for the assumed traffic characteristics).

The model can be extended in a number of ways by very simple modifications. For example, multiple Ethernets (at one or both sides of the ATM link) can be represented by increasing the number of (initial) tokens in `E_IDLE` (Fig.3.6); this allows multiple simultaneous firings of transition `E_S`, up to the number of tokens in `E_IDLE`. Fig.4.6 shows that as the number of Ethernets increases, the average load per network decreases. The ATM load initially increases but then stabilizes, which indicates a bottleneck in the system (a bridge could be this bottleneck).

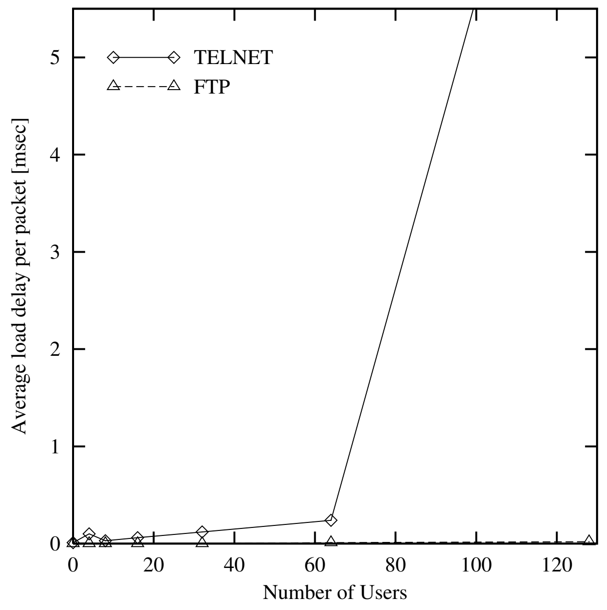

Fig.4.5. Load delay per packet – CU–SeeMe and telnet/FTP.

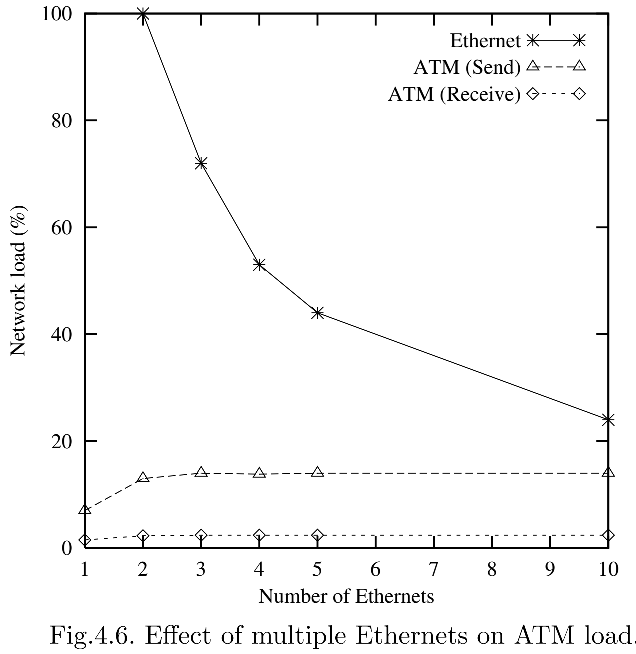

Fig.4.6. Effect of multiple Ethernets on ATM load.

5 Concluding Remarks

A timed Petri net model of an ATM LAN provides a reasonably accurate representation of the behavior of the original system, so the model can be used for detailed investigations of various parts of the system. The model can easily be extended by introducing additional elements, for example, bridges or routers between bridges. The ability to represent multiple distinct protocols can be extended past the basic concept to study the interactions between protocols, timing delays and where those delays occur.

The proposed model provides good conformance to protocol behavior under normal load, it directly represents network traffic in terms of users and protocols, its structure can easily be modified, and it allows to study properties of different network configurations (without actually implementing them).

Although only a very simple model of the workload has been used for obtaining the results presented in this paper, the simulation results conform to the real, measured quantities quite well. Moreover, more elaborate models of the workload can easily be introduced by adding a few additional net elements to the model [6].

The model has some limitations. It becomes less accurate when it passes the point of congestion because it does not represent many effects which affect the behavior of the network if congestion is taken into account (a more complex Ethernet model would be needed and retransmission of lost packets would be required).

In extending the proposed model, some additional constructs would be of benefit. The ATM layer is complicated by the requirement that cells remain in strict order, yet can be buffered at various points in the system. As well, current work on hierarchical net structures would be beneficial to this model, simplifying the introduction of mechanisms that cannot be layered easily (e.g., lost packet mechanism).

The complete model is quite complicated but it has a very modular structure. It would be interesting to check if structural methods could be successfully applied to analysis of this type of net models.

Acknowledgments

A number of interesting comments and constructive remarks of two anonymous referees are gratefully acknowledged.

The Natural Sciences and Engineering Research Council of Canada partially supported this work through Research Grant A8222.

References

1. Ajmone Marsan, M., Cigno, R.L., Munafo, M., Tonietti, A., “Simulation of ATM computer networks with CLASS”; in: “Computer Performance Evaluation: Modelling Techniques and Tools”, pp.159–179, Springer Verlag 1994.

2. Alles, A., "ATM internetworking"; Technical Report, CISCO Systems Inc. 1995.
3. Armitage, G.J., Adams, K.M., "How inefficient is IP over ATM anyway?"; IEEE Network, vol.9, no.1, pp.18–26, 1995.
4. Awdeh, R.Y., Mouftah, H.T., "Survey of ATM switch architectures"; Computer Networks and ISDN Systems, vol.27, no.12, pp.1567–1613, 1995.
5. Bradner, S., " Bradner Reports – Catalyst 5000 switch"; Technical Report, Cisco Systems Inc, Sept. 1995.
6. Chen, P-Z., Bruell, S.C., Balbo, G., "Alternative methods for incorporating non–exponential distributions into stochastic timed Petri nets"; Proc. 3-rd Int. Workshop on Petri Nets and Performance Models (PNPM'89), Kyoto, Japan, pp.187–196, 1989.
7. Descloux, A., "Stochastic models for ATM switching networks"; IEEE Journal on Selected Areas in Communications, vol.9, no.3, pp.450–457, 1991.
8. Goralski, W.J., "Introduction to ATM networking"; McGraw Hill 1995.
9. Holliday, M.A., Vernon, M.K., "Exact performance estimates for multiprocessor memory and bus interference"; IEEE Trans. on Computers, vol.36, no.1, pp.76-85, 1987.
10. Huntsberger, D.V., Billingsley, P., "Elements of statistical inference" (5-th ed.), Allyn and Bacon 1981.
11. Jensen, K., "Coloured Petri nets"; in: "Advanced Course on Petri Nets 1986" (Lecture Notes in Computer Science 254), Rozenberg, G. (ed.), pp.248-299, Springer Verlag 1987.
12. Lin, A.Y.M., Silvester, J.A., "Queueing analysis of an ATM switch with multichannel transmission"; Performance Evaluation Review, vol.18, no.1, pp.96–105, 1990.
13. Moldeklev, K., Gunningberg, P., "How a large ATM MTU causes deadlocks in TCP data transfers"; IEEE-ACM Trans. on Networking, vol.3, no.4, pp.409–422, 1995.
14. Murata, T., "Petri nets: properties, analysis and applications"; Proceedings of IEEE, vol.77, no.4, pp.541–580, 1989.
15. Ohba, Y., Murata, M., Miyihara, H., "Analysis of interdeparture processes for bursty traffic in ATM networks"; IEEE Journal on Selected Areas in Communications, vol.9, no.3, pp.468–476, 1991.
16. Onvural, R.O., "On performance characteristics of ATM networks"; Proc. SuperComm/ICC '92, pp.1004–1008, 1992.
17. Onvural, R.O., "Asynchronous Transfer Mode Networks: Performance Issues"; Artech House 1994.
18. Perloff, M., Reiss, K., "Improvements to TCP performance in high–speed ATM networks"; Communications of the ACM, vol.38, no.2, pp.91–109, 1995.
19. Petr, D.W., Frost, V.S., Neir, L.A., Demirtjis, S., Braun, C., "Simulation comparison of broadband networking technologies"; SIMULATION 64, pp.42–50, 1995.
20. Reid, M., "Modeling and performance analysis of ATM LANs"; M.Sc. Thesis, Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada A1B 3X5, 1997.
21. Reisig, W., "Petri nets – an introduction"; Springer Verlag 1985.
22. Romanow, A., Floyd, S., "Dynamics of the TCP traffic over ATM networks"; IEEE Journal on Selected Areas in Communications, vol.15, no.4, pp.633–641, 1995.
23. Stamoulis, G.D., Anagnostou, M.E., Georgantas, A.D., "Traffic source models for ATM networks: a survey"; Computer Communications, vol.17, no.6, pp.428–438, 1994.

24. Yamada, H., Sumita, S., “A traffic measurement method and its application for cell loss probability on ATM networks”; IEEE Journal on Selected Areas in Communications, vol.9, no.3, pp.305–314, 1991.
25. Zuberek, W.M., “Timed Petri nets – definitions, properties and applications”; Microelectronics and Reliability (Special Issue on Petri Nets and Related Graph Models), vol.31, no.4, pp.627–644, 1991.
26. Zuberek, W.M., “Modeling using timed Petri nets – event–driven simulation”; Technical Report #9602, Department of Computer Science, Memorial University of Newfoundland, St. John’s, Canada A1B 3X5, 1996.