MODELING AND PERFORMANCE ANALYSIS OF ATM LANs

MICHAEL E. REID

Modeling and Performance Analysis of ATM LANs

by

Michael E. Reid

A thesis submitted to the School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

April 1997

St. John's Canada

0-612-36172-1

Abstract

Asynchronous Transfer Mode (ATM) is a method of data transmission using small fixed-length cells. This thesis presents a model of an ATM LAN which provides a realistic representation of data transmission over the system by explicitly modeling both the ATM network and the applications running over that network. Coloured timed 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 detail at different levels of the system. Simulation is used to evaluate the performance of the model, and results are compared to actual data gathered from the Memorial University campus network.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the following:

- Dr. Wlodek Zuberek, my advisor, for his guidance and patience during the work on this thesis,
- the thesis examiners, for their detailed comments,
- Memorial University, for financial support,
- the Department of Computing and Communications, for their willingness to grant me both time and disk space,
- Rick Collins, who provided the CUSEEME data,
- my extended family, who provided sympathy, babysitting and meals,

but most of all my wife Susan and daughter Alicia. I could not have completed this work without their patience and support.

Chapter 1

Introduction

The transmission of information is usually performed by systems specialized for different types of information. That is, there are specific systems for data, others for video, a third type for voice, and so on. A single system that can transport all types of information has obvious advantages - for example, it could allow information carriers to offer multiple services using one basic infrastructure. Another potential benefit could be the true integration of services at the user station - i.e. voice, video, and data on one user device.

One system that promises to deliver this integration is Asynchronous Transfer Mode, or ATM. The basic premise of ATM is that all information is divided into small fixed-length data units (cells), which can then be sent across switching networks to be re-combined at the receiving end. The challenge for an ATM designer is to ensure that this cell transmission system can provide the correct behaviour for a video stream as well as a phone call. This challenge has also resulted in a considerable body of research into ATM related topics.

Since many of the performance promises of ATM depend on its ability to transmit cells at extremely high rates with little or no loss of cells, much of the research has focussed on switch design, 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 thesis presents a model of an ATM LAN that provides a more realistic representation of data transmission over ATM. It does this by explicitly modeling the applications that are running over an ATM LAN, as well as the ATM network itself. Most current research uses simple stochastic inputs or queuing models to describe an ATM network; such an approach, however, ignores the synchronized nature of network protocols. This synchronization is easily represented in the Petri net based model presented in this thesis. Furthermore, the model also represents the applications directly in terms of protocols and number of active users, which permits the modeler to estimate the impact of these variables on an ATM backbone.

Petri nets are a type of directed graph which have become quite popular for modeling a wide variety of concurrent systems [41]. Their similarity to finite automata eases the understanding and utilization of Petri nets for most modelers of computer systems, and the wide range of extensions to the basic net formalism allows the modeler to choose a type of net that best matches his requirements.

Network protocols, especially those that have been continually refined like TCP/IP, can exhibit quite complex behaviour. However, an analysis of network behaviour on the campus backbone of Memorial University has shown that much of this behaviour does not appear on a LAN in relatively non-congested periods of operation. If the modeler assumes that behaviour under congested conditions is less important than detecting when congestion might occur, then many elements of protocol behaviour can be ignored. The behaviour that remains is remarkably similar across a set of protocols carried on the Memorial network.

To capture this similarity of behaviour while still permitting the individual protocols to act independently, the model is based on coloured timed Petri nets with exponentially distributed firing times. Coloured Petri nets are useful for representing systems that contain many repeated components; the different components can essentially be superimposed on one another, and distinguished by associating the tokens in the net with attributes called "colours". The coloured tokens can operate independently from one another or interact as required by the designer. The structure of the net model represents the basic behaviour common to all the protocols, while the different colours are used to represent the differences between the protocols, such as temporal characteristics or packet size.

To allow the modeler to easily modify parts of the model, the net is designed in a modular fashion in relation to a protocol stack description of the subject network. This ability to easily change a particular section without affecting the surrounding parts of the net permits separate testing of each module, as well as the power to re-arrange or re-combine elements of the model to reflect different configurations of the subject network.

Although analytical solutions are possible for many classes of Petri nets, the model uses a number of theoretical extensions which prevent such analysis. Therefore, simulation is used to evaluate the performance of the net. This simulation is used to investigate:

- the conformance of the model to real protocol behaviour,
- the cumulative effect of many users and protocols on a network,
- the effect of a particular protocol on an ATM LAN,
- the effect of different configurations of the subject network.

Where possible, the results from simulation are compared to data traces taken from the subject network.

The systems and protocols modeled in this thesis, including a description of the subject network, are discussed in Chapter 2. A brief survey of current research on ATM performance is also included there. Chapter 3 presents the model design in greater detail, and describes some elements of the design process. Chapter 4 describes the parameters of the model and the way they are determined; it also presents simulation results and data trace comparisons. Some possible extensions to the model are discussed in Chapter 5, with Chapter 6 summarizing the work and results.

Chapter 2

Modeling Environment

This chapter discusses some of the protocols and systems that make up the network architecture modeled in this thesis. Some background information is provided, as well as a description of how the systems are implemented in the Memorial University campus environment. The chapter concludes with a brief survey of current research on modeling and analysis of ATM networks.

2.1 Protocols

Protocol behaviour is a key part of any network. The discussion starts with the high level application protocols, and proceeds down through the standard layer model to the ATM cell level.

2.1.1 Application Protocols

The application protocols are those most familiar to network system users – they usually act as the direct point of contact between the user and the network.

TELNET: provides 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 thesis.) TELNET is described in RFC 854 (see Section 2.1.2 for more detail on RFC's).

FTP: the File Transfer Protocol is probably the most commonly used application at Memorial for transferring files from one host to another. It is described in RFC 959.

NNTP: the Network News Transfer Protocol is used for transferring USENET news articles from host to host. It is described in RFC 977.

WWW: the World Wide Web is the popular name for a global information system on the Internet. The protocol used to transmit most WWW information is HTTP (Hypertext Transfer Protocol), although actual WWW usage is often a mixture of HTTP and other protocols. HTTP is specified in RFC 1945.

X Windows: 'X' is a client-server based system for the management of remote graphics displays. X protocols and standards are maintained by the X Consortium, a group of academic and industry users.

CU-SeeMe: Originally developed at Cornell University, CU-SeeMe is a videoconferencing application designed to work over the Internet.

2.1.2 TCP/IP

The acronym **TCP/IP** (Transmission Control Protocol/Internet Protocol) [11] refers to a suite of networking protocols that have gained wide acceptance as the basis for the global Internet. TCP/IP was developed in the mid to late 1970s by DARPA, the Defense Advanced Research Projects Agency. By early 1980 it had become the communication standard for the ARPANET, the predecessor of the current Internet. The TCP/IP suite has developed over time, with numerous extensions and improvements. All TCP/IP standards are publicly accessible over the Internet, and are published in documents called *RFCs*, or Requests For Comments.

The TCP/IP protocol stack (see Figure 2.1) is built on a connectionless datagram service (IP), with primarily two higher level services - UDP, which offers a connectionless service, and TCP, which provides a reliable connection-oriented service. Most Internet applications use one of these two services to transmit data from one host to another. Most of the protocols modeled in this thesis use TCP as their transport service, although UDP is also discussed.

TCP [49] is designed to provide:

- basic data transfer,
- reliability (guaranteed delivery using positive acknowledgements),

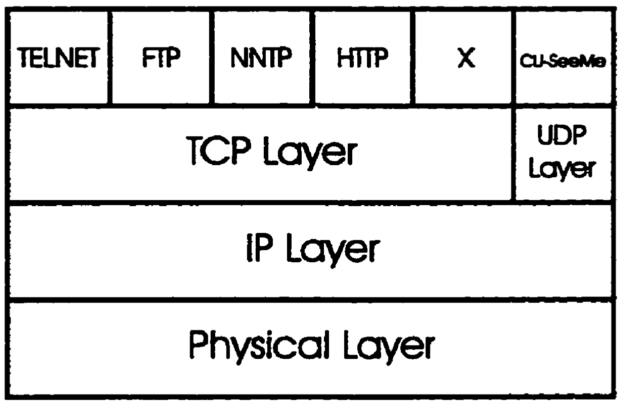

Figure 2.1: TCP/IP protocol stack.

- flow control (sliding windows),
- multiplexing (full duplex transmission),
- connection (handshaking for setting up and ending a connection, as well as status information during the life of a connection).

There have been a number of additions to the TCP protocol. These are documented in Internet RFCs.

2.1.3 Ethernet

Ethernet [37, 11] is a packet transmission system designed for Local Area Networks (LANs). It is based on a shared 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* for *Carrier Sensing Multiple Access/Collision Detection*. Each station can detect if another is transmitting, and, if so, refrains from attempting to send. Once the medium is free, stations with data to transmit will attempt to gain control. Since the start of the transmission takes a finite time to propagate to all stations on the medium, it is possible that more than one station may attempt to send at the same time (a "collision"). When transmitting stations detect a collision, they immediately stop and wait a random period of time before attempting to transmit again.

Ethernet normally operates at 10 Mbps, although there are recent standards for operating Ethernet at 100 Mbps [39].

2.2 ATM

Memorial chose ATM technology to serve as the backbone of its campus network. This section provides a brief description of ATM and associated protocols.

2.2.1 Basic Concepts

Asynchronous Transfer Mode (ATM) [13, 18] is a method of transferring information (i.e. data, voice, or 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 before the transfer begins. Each data connection represents a different virtual channel or 'circuit'. Although cells are always in sequence on any given virtual circuit, the circuits are multiplexed together through switching devices and underlying media.

Although ATM is connection-oriented, it is intended to be used by a variety of

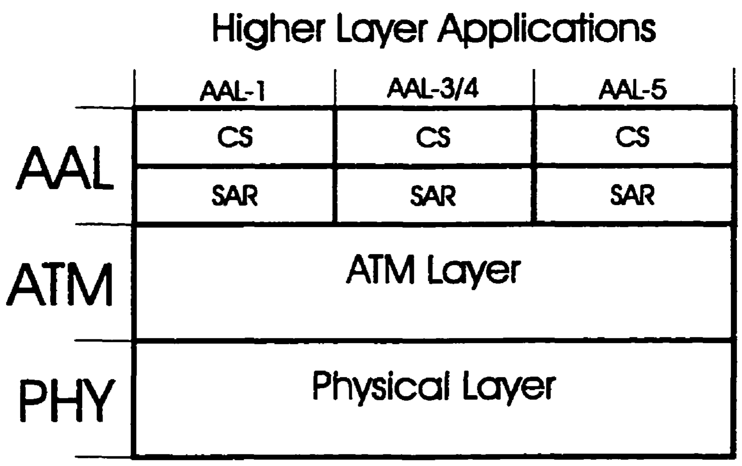

Figure 2.2: ATM protocol stack.

applications, such as LAN traffic or voice transmission. A number of interfaces called *ATM Adaptation Layers* provide different classes of service to upper level applications (Figure 2.2 shows the ATM protocol stack). AAL-1, for example, 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 sizes for LAN traffic. The AAL layer is subdivided into two parts: Convergence Sublayer (CS), which prepares higher level blocks of data for transmission, and the Segmentation and Reassembly (SAR), which actually does the conversion of the data into cell payloads. In all cases, the information is eventually broken into 53 byte (5 byte header, 48 byte payload) cells at the ATM layer. The use of small fixed-length cells permits the design of very fast ATM switching devices, as well as reducing queuing delay and jitter [13].

One of the strengths of ATM is that it is not tied to any particular physical medium or transmission rate. ATM networks currently operate on both copper and fiber optic media, with rates from T1 (1.544 Mbps) to OC-12 (622 Mbps). ATM standards are set by the *ATM Forum*, an organization made up of telephone companies, software and hardware manufacturers, and interested user organizations.

2.2.2 LAN Emulation

Most existing applications requiring a network do not interface directly with ATM. Since they are designed to use more traditional protocols such as IP or IPX [33], there has been a considerable amount of work in designing interfaces that allow these protocols to operate over ATM. The upper level application is unaware that ATM is involved in the data transfer - it merely sees the normal IP or IPX interface. The IP-over-ATM standards of the IETF (Internet Engineering Task Force) described in RFC1483 and RFC1577, and MPOA [1] are examples of work in this area.

Another attempt to interface traditional protocols and systems with ATM is LAN Emulation [31]. This standard simulates a MAC layer (either Ethernet/IEEE 802.3 or Token Ring) over an ATM network. Any application or protocol that would normally operate over an Ethernet or 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 part of an Emulated LAN is called a LAN Emulation Client (LEC). Each Emulated LAN must also have devices that perform certain functions for the Emulated LAN - these functions are:

- **LAN Emulation Configuration Server**, which provides locations of other services,
- **LAN Emulation Server**, which provides locations of other clients in the same Emulated LAN,
- **Broadcast and Unknown Server**, which simulates the broadcast behaviour of a LAN by replicating broadcast type packets to all clients of a particular Emulated LAN.

Most of these services are activated when a client (which can be a single computer or a bridge between an ATM and an older type of network) first joins an Emulated LAN or sends a broadcast packet. When one host wishes to send data to another on the same emulated LAN, it uses the LES for that LAN to find the ATM level address of the other host. It then sets up a direct ATM VC (Virtual Circuit) between the two devices for the actual data transfer.

2.3 Memorial's Network

The data in this thesis are taken from the campus network of Memorial University of Newfoundland. At the time this thesis is being written, the University is transitioning from a campus backbone based on Ethernet over fiber optic cable to an ATM network using LAN Emulation.

Figure 2.3 shows a typical data path over the old backbone at the University. Both the backbone and the end-point LANs are 10 Mbps Ethernet, while the intervening

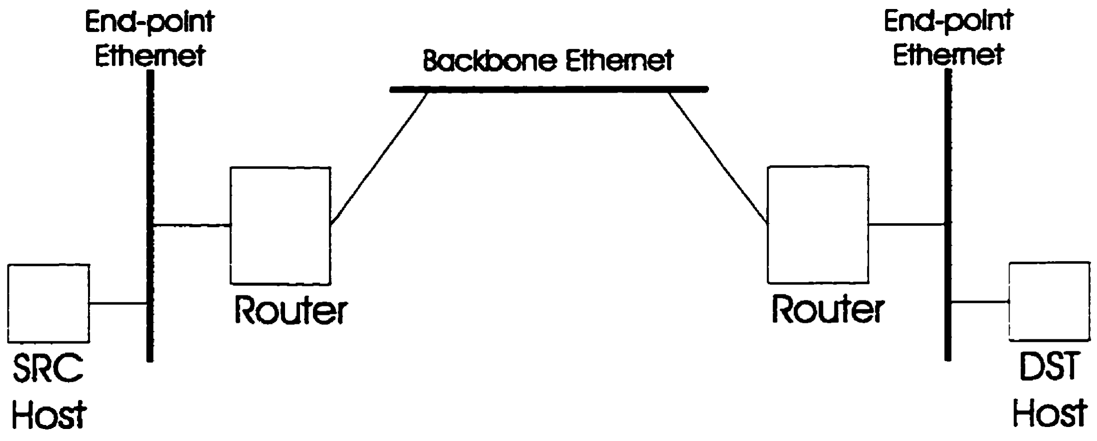

Figure 2.3: Typical data path - old backbone.

devices are routers.

The new ATM backbone replaces the Ethernet and routers in the middle of the data path 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*, then the data path is as shown in Figure 2.4; 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 Figure 2.5; the data path includes a router as an intervening device.

The links between the ATM devices are fiber optic cables using OC-3c SONET framing. (SONET stands for Synchronous Optical NETwork [18].) The normal raw transmission rate of OC-3c is 155.52 Mbps.

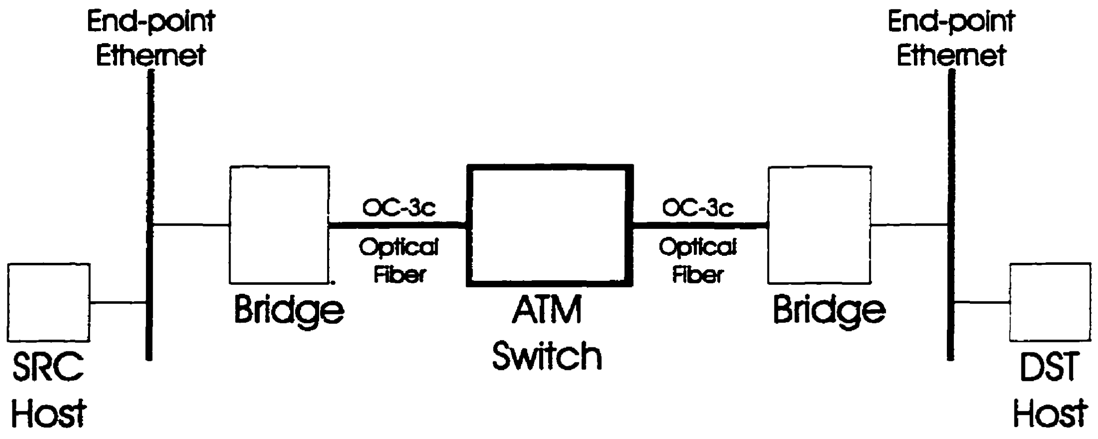

Figure 2.4: Typical data path - same virtual LAN.

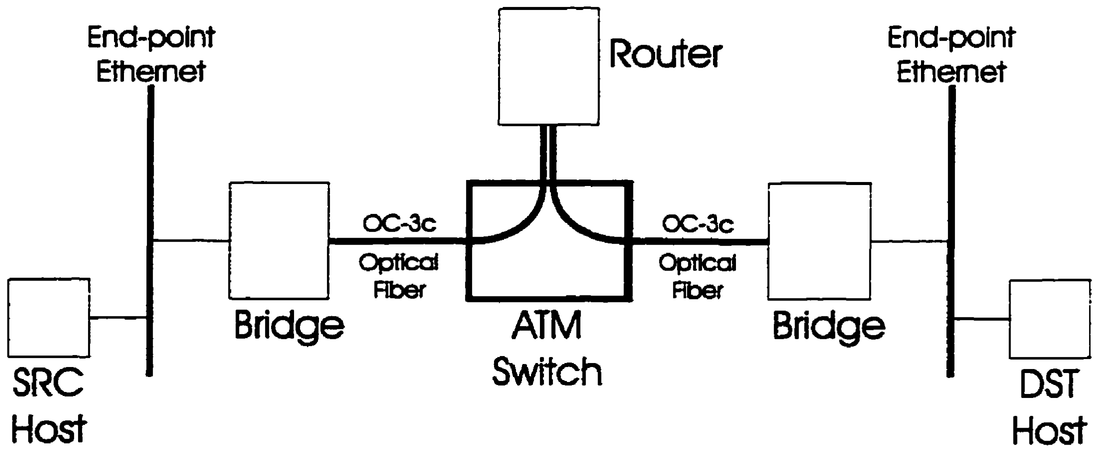

Figure 2.5: Typical data path - different virtual LAN.

2.4 Current Research

ATM is still very much a system under development. The ATM Forum is continually producing new standards, and revising current ones. Many aspects of ATM generate fierce debate between researchers, vendors, and users of ATM equipment.

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 network of switches. A summary of such methods can be found in [55]. Conti [12] provides an exhaustive list of models applied to FDDI [28] and DQDB [9], while an older survey in [51] categorizes modeling methods according to the OSI Reference Model.

A common input model used in ATM analysis is the MMPP (Markov Modulated Poisson Process) [44]. This and related models are used in [58] and [14] to estimate cell loss probabilities in ATM networks. Queueing models are used to study transmission delays in ATM networks [42, 43], and also buffer allocation within an ATM switch [32].

A "fluid flow" model is used in [6] as an alternative to queueing models in studying the "leaky bucket" policing mechanism[1] in ATM. Other such mechanisms are studied in [50] using an MMPP related model, while [16] uses teletraffic and signal processing theory.

Direct simulation has also been used to analyze ATM performance. A simulation comparison of ATM, Frame Relay [24], and DQDB is described in [48]. [22] analyzes a policy mechanism, and [8] describes a distributed simulation of a multi-node ATM

[1]ATM supports Quality of Service (QoS) parameters; policing mechanisms are devices that ensure a source complies with the contracted service levels in the network.

network. A simulator specifically built for ATM-based systems is described in [36].

The issues involved in operating traditional protocols over ATM has generated a number of studies of IP performance over ATM. These studies have concentrated on directly attached workstations - i.e. a host with an ATM interface card. The issue of protocol overhead is examined in [7, 2], while [19, 46, 57] analyze the effects of TCP/IP and system design on IP-ATM performance. [53] evaluates two strategies for effective discard of packets in an ATM environment, while [38] examines in detail a deadlock situation that can occur with TCP over ATM.

The study presented in this thesis uses a simple behavioural model of ATM. A good agreement of simulation results with measurements indicates that even this simple model is quite satisfactory for many performance analyses. The next section describes the conceptual model used in this thesis, some background on the modeling technique employed (timed coloured Petri nets), and the derivation and construction of the model itself.

Chapter 3

Model

The interconnection of LANs with ATM networks is still a fairly new area of interest. Many standards are still under development, and there remain some areas of fierce debate, such as the best deployment of routing traditional protocols over an ATM switch network. Much of the research has concentrated on the behaviour of the ATM switches themselves using abstract models to represent the data passing through the switch. These models have generally represented broad classes of input - "data" versus "video", for example. This creates an extremely abstract environment outside the switch. It was felt that explicitly representing the higher-level protocols that drive the network would provide a more detailed and realistic input model for an ATM network. Furthermore, in a production ATM environment such as the one at Memorial, the emphasis is usually not the expected cell loss rates of the switches (for example), but rather what effect using an ATM network has on the users who pass data across it.

The system modeled in this thesis is described in the reference model (Figure 3.1).

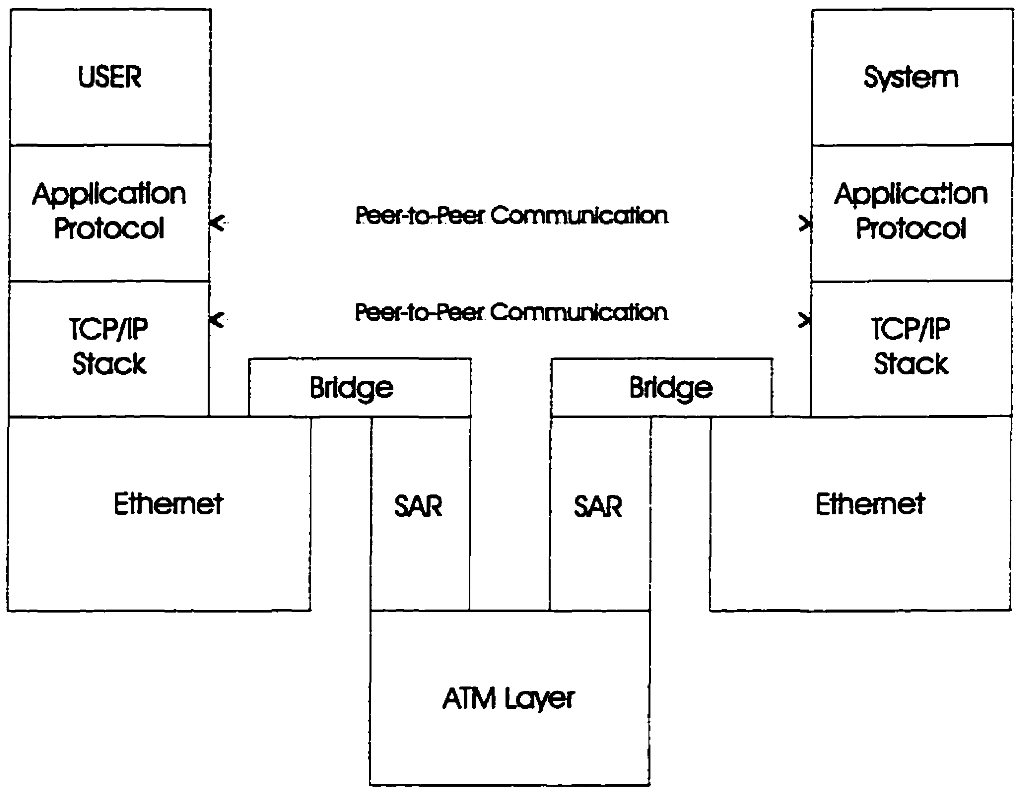

Figure 3.1: Reference Model

This reference model can be thought of as a "cross-section" through the backbone configuration shown in Figure 2.4, using the classic network layer model.

In designing the model, a number of broad goals were identified:

- The model should be *modular*. It should be possible to change the behaviour of sections of the model without requiring major changes to other sections. The modular approach also allows the designer to gradually build the model from smaller pieces.

- The model should be closely tied to the physical implementation and behaviour of the system studied.

- The model should have a high degree of flexibility. It should be possible to easily re-arrange modules to represent different environments.

The modeling method used in this thesis is based on Petri nets [52, 41, 47]. Petri nets have a number of advantages for modeling:

- Concurrency and synchronization is "naturally" represented in Petri net models.
- A sound theoretical foundation has been developed for Petri nets, as well as a number of extensions to the basic concept. The many extensions allow a modeler to choose a construct that best suits the system to be modeled.
- There are a large number of software tools available for analyzing various types of Petri nets.
- Petri nets can be studied by analytical methods as well as simulation.

It was decided that the model will follow the layered structure of the reference model. Each layer will be represented by a module of the net, allowing the modeler to change a particular layer independently of the others. Where peer-to-peer conversations [56] are involved (at the User/Application and TCP/IP layers, for example), the preference was to closely imitate the action of realistic peer-to-peer conversations - the data flow is actually down through the lower layers in the stack and back up to the peer level in the other half of the conversation. It was also decided to have all conversations originate from one side of the network. This is a typical pattern for much of Memorial's network, with many end-point LANs consisting entirely of personal computers that interact with central servers for various functions. In this case, network conversations almost always start from the PC end of the link.

Section 3.1 describes the basic theory of Petri nets, and the extensions used in this thesis. Section 3.3 introduces the model for the upper layers of the reference model (i.e., User/Application), while Section 3.4 models the TCP/IP protocol stack. Sections 3.5 and 3.6 describe how the model represents an Ethernet and the ATM network respectively. Finally, some key model design decisions are discussed in Section 3.8.

3.1 Petri Nets

This section gives a short introduction to Petri nets and the extensions to Petri nets that are used in the model. Some of the general references, such as [41], can be consulted for further information and sources.

3.1.1 Basic Properties

The basic form of Petri nets, as defined in [47, 52, 41], is a bipartite graph made up of a set of places P, a set of transition T, and a set of directed arcs A which connect places with transitions and transitions with places. Tokens are assigned to places, with the particular distribution of tokens among the places at any one time being called a

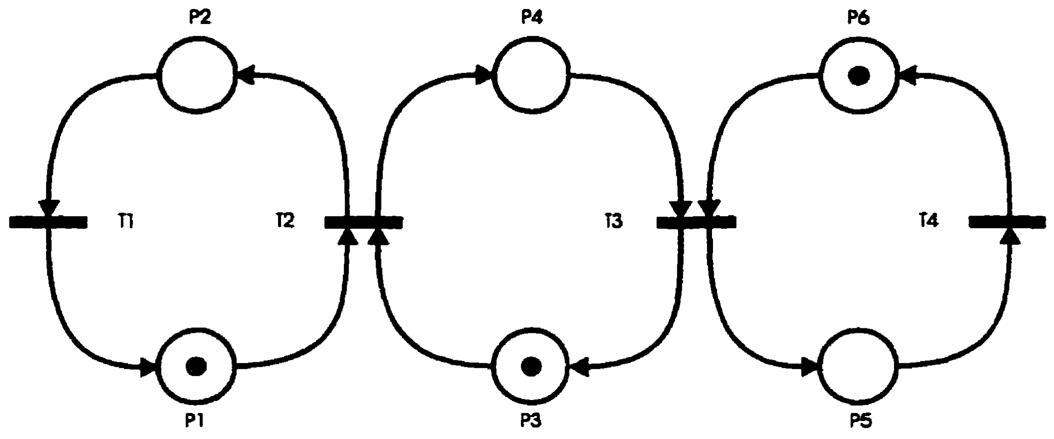

Figure 3.2: Basic Petri net model: producer/consumer bounded-buffer model.

marking. The behaviour of a petri net is controlled by the distribution of tokens and the firing of transitions; a transition is enabled for firing when all the input places to that transition contain tokens. The firing of a transition removes one token from each of the input places and adds a token to each of the output places of that transition.

Figure 3.2 is an example of a simple Petri net. It represents two processes communicating via a buffer – one writes to the buffer and the other reads from the buffer assuming the buffer is non-empty. The "write" process is represented by place P1 and P2 and transitions T1 and T2. When T2 fires, it removes one token from P1 and P3 and deposits single tokens into P2 and P4. This represents a "write" action to the buffer. (The sum of tokens in P3 and P4 determines the buffer capacity.) With a token in P4, T3 is now enabled, and on firing, removes the token from P4 (a "read" action) and deposits tokens into P3 and P5. Both T1 and T4 can fire, returning the producer and consumer back to their original states.

The theoretical bases of Petri nets are well-explored; [41] provides an overview of the subject. Petri nets can be used to determine a wide range of properties about modeled systems. For example, [15] and [3] describe the use of Petri nets in protocol verification – that is, ensuring that protocols operate as intended and are free from deadlock or other design flaws.

There are a number of extensions to basic Petri nets. Inhibitor arcs [61] provide a "test if zero" condition; a transition with inhibitor arcs is enabled if the places attached to those arcs are empty of tokens. Another extension is the use of multiple arcs from the same place and transition, allowing a transition to remove or deposit multiple tokens at one firing.

Some properties of Petri nets are determined by the structure of the net without reference to a particular marking. Part of this work is the study of conflicts, where multiple transitions are enabled by the same marking, and the firing of one transition prevents the other from firing. The simplest type of conflict in a Petri net is known as a "free-choice" type, where multiple transitions share the same input places. Since all such transitions are enabled simultaneously, the resolution of the conflict can be made by assigning probabilities to each transition and making a random choice.

Two extensions to Petri nets that are important to performance modeling are described in the next two sections: associating timing information with a net, and the use of "colours" to distinguish tokens from one another. A third section discusses in more detail which extensions are employed in the model.

3.1.2 Timed Nets

While basic Petri nets are useful in a variety of modeling applications, they do not represent events that take place over time. For example, the simple net in Figure 3.2 cannot model the speed at which the read and write actions occur. There have been a variety of methods proposed to add timing information to Petri nets. *Timed* nets [59] associate a deterministic or exponentially distributed firing time with each transition. Stochastic Petri nets (SPN's) [40] also assign exponentially distributed firing times to transitions, but the two types use different procedures for handling the interaction between transitions and tokens. (This interaction is called the "firing semantics" of the model.) Some other classes of nets with timing information are Generalized Stochastic Petri Nets (GSPN's)[34] which include both exponential and immediate (non-timed) transitions, and Extended Stochastic Petri Nets (ESPN's) [17] which use non-exponential firing time distributions.

The model used in this thesis follows the timed net paradigm of firing semantics. When a transition fires, the token(s) from the input places are removed at the beginning of the firing duration, and then token(s) are deposited into the output places at the end of the firing period. For example, to turn the sample net in Figure 3.2 into a timed net, we would associate exponentially distributed firing times with T2 and T3. When T2 fires, the token would be removed from P1 until the end of the duration of that firing, at which time tokens are placed in P2 and P4. Therefore, the transition T3 is not enabled until the *end* of the T2 firing time.

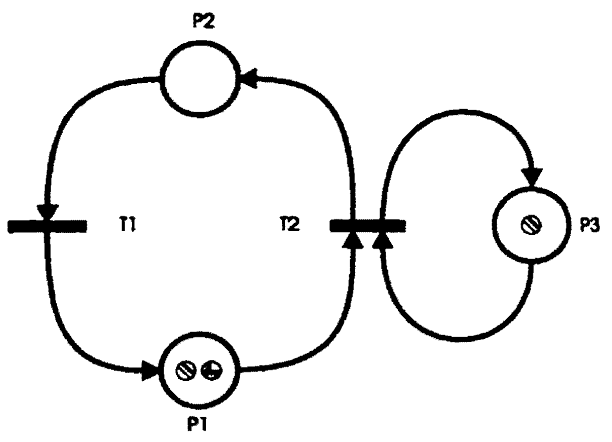

Figure 3.3: Basic Petri net model: coloured version.

Nets with timing information have become quite common in performance modeling. [35] describes a model of a LAN system similar to Ethernet, while [21] uses Stochastic Reward Nets (a modified version of SPN's) to model operating systems.

3.1.3 Coloured Petri Nets

In basic Petri nets, all the tokens are considered identical – only the number of tokens in each place is important. In coloured Petri nets [27], however, the tokens have attributes called colours which make the tokens distinct from one another. Since there are now different classes of tokens, transitions and transition firings become much more complicated. To be enabled, a transition must not only have sufficient numbers of tokens in its input places, but the tokens must be of the required colours. The number and colours of tokens deposited in the output places when the transition fires must also be specified.

The chief benefit of coloured nets is that they simplify the overall structure of the net. Many net models, if designed using non-coloured nets, are made up of repeated subnets combined together. For example, the net model of a LAN in [35] uses an identical subnet for each station attached to the network. By using different colours of tokens, these subnets can be "folded" onto one another, and yet still operate independently. Figure 3.3 shows a coloured version of the net in Figure 3.2. The "read" and "write" processes have now been folded together, with one token of colour r and another of colour w in place P1. Transition T2 now operates in two ways:

1. If a token of colour w is in P1 and in P3, then T2 is enabled and, on firing, removes a token of colour w from each of P1 and P3 and places a token of colour w in P2 and a token of colour r in P3.
2. If a token of colour r is in P1 and in P3, then T2 is enabled. On firing, it removes one token r from P1 and one token r from P3, depositing a token of colour r into P2 and one into P3.

The two events represented by T2 enabling are called "occurrences" of the transition, each corresponding to one of the original transitions in the non-coloured net. It is possible to fold the net even further into one with only a single place and transition. This requires an additional colours and a more complicated transition description. Generally, increased folding (reduced structure) results in more colours and greater transition complexity.

Some simple examples of coloured Petri nets can be found in [27]. [23] discusses hierarchical structures using coloured nets.

3.1.4 Model Semantics and Notational Conventions

This thesis uses timed coloured Petri nets to model the network system described in the previous chapter. Infinite timed firing semantics is employed – tokens are removed from the input places at the beginning of the transition firing time, and several firings of the same transition can overlap.

The model makes frequent use of *free-choice* structures, described by "choice" probabilities. The model also uses "marking-dependent" probabilities where the probability of firing a particular occurrence is determined by the relative number of tokens of a particular colour in an input place.

The following notation is used in the diagrams in this thesis. Places are notated by the usual hollow circles. Immediate transitions are indicated by thin bars and timed transitions by hollow bars. The notation 'kn', where n is a number indicating the packet type from Section 3.4, is used next to arcs to indicate the particular packet type removed or deposited by a transition. (All occurrences in the User/Application and TCP/IP stack levels have the same packet types per arc.) If this notation is missing from an arc in the User/Application or TCP/IP modules, the arc is assumed to be 'k0' (i.e. a control token). If the notation is missing from arcs in the network layers, the arc is assumed to have an occurrence for multiple packet types, as is the case for most of the arcs in the Ethernet and ATM layers.

3.2 Model Overview

The complete Petri net model is shown in Figure 3.4 indicating the sections corresponding to the layers of the reference model (Figure 3.1). The model can be considered as a stack of identical superimposed nets, one for each application-level protocol. The superimposed nets are wholly independent at the source and destination processes at either side, but there are links between the layers at the intervening network sections, representing resources such as an Ethernet which can only transmit one frame at a time.

Inside each layer, each transition can be thought of as a list of occurrences. This list can grow up to the maximum number of sessions per application-level protocol. Again, each session is independent of the others at the User/Application and TCP/IP sections, but interact at the network levels.

The remainder of this Chapter describes each section of the model in more detail in relation to the reference model (Figure 3.1).

3.3 User/Application Level

The basic model at this level is of a user running an upper level application (such as TELNET, FTP, etc.) between two computers connected by a network. The user is assumed to operate in the following way:

1. There is a thinking state, in which no transmissions occur.

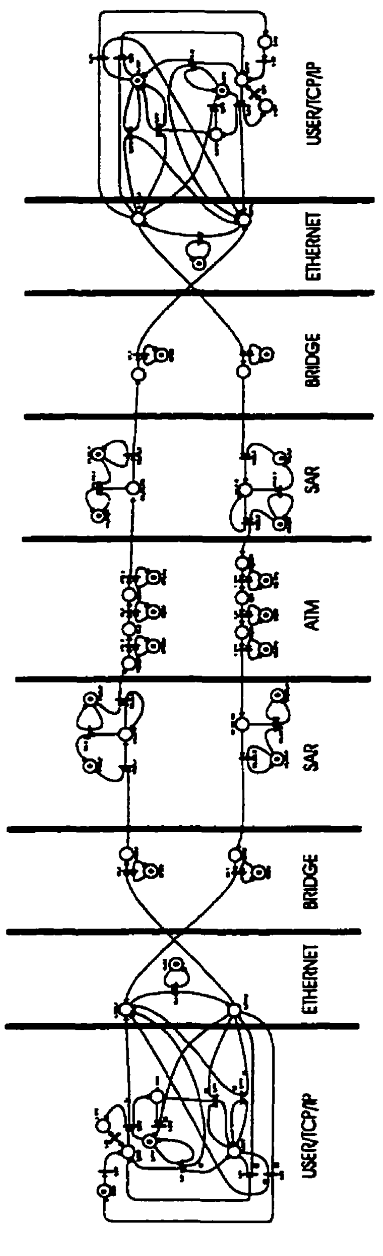

Figure 3.4: High level view of model.

2. A request (e.g., a command) is sent to the remote computer. This causes a data transmission from the originating host (labelled SRC or " Source") to the destination host (DST or "Destination").

3. The SRC host then waits for a reply from the DST host (i.e., a new screen or a data transfer).

4. The SRC host returns to the original thinking state, and repeats the cycle.

Figure 3.5 shows the net model of the User/Application level. The thinking time is represented by the transition **S_TNK**, while the data transmissions are represented by the "Network" transitions and dotted arcs. Control is transferred to the DST process, which has a certain delay to process the request (**D_TNK**) before replying with a data transmission back to the SRC process.

The token indicated in Figure 3.5 represents the initial marking for the model. Each token indicates a single session (user) using a particular application-level protocol. Each protocol is represented by a different token colour. Since the behaviour of the **S_TNK** transition can be different for each colour, the colours separate the behaviour of different protocols. The multiplicity of tokens of each colour in place **S_SEND** represents the number of simultaneous sessions of each protocol type. The infinite firing semantics used in this model ensures that each session operates independently of all other sessions in the model.

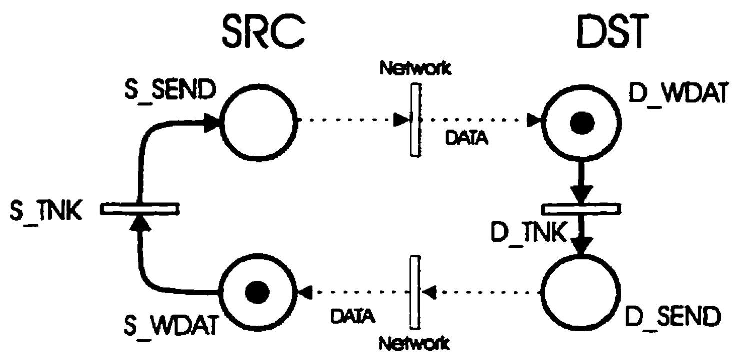

Figure 3.5: User/Application level model.

3.4 TCP/IP Level

When originally designed for the ARPANET, TCP/IP was built to work over slow point-to-point connections. Sliding windows, positive acknowledgements, re-transmissions, and piggybacking helped improve reliability and performance of the protocol on congested links. In a LAN environment, however, some of these techniques are rarely seen. Re-transmissions, for example, are relatively rare in Memorial's internal traffic (intra-LAN or inter-LAN) as opposed to LAN to Internet traffic. This is a function of both the small delay times found in the LAN environment as well as the much smaller probability of packet loss. Since increased functionality results in increased model complexity, a trade-off point must be reached where enough functionality is included to provide a reasonable approximation of system behaviour.

3.4.1 TCP

Figure 3.6 shows an outline of a net model of the User/Application level with a TCP/IP stack. (A more detailed diagram of the TCP/IP stack is shown in Figure 3.9.) The stack provides a TCP (connection-oriented) path between the SRC and DST processes at the User/Application level. It provides the following functionality:

- Transmission and reception of data packets.
- Positive Acknowledgements. The SRC or DST processes send back a packet to acknowledge the successful receipt of a data packet.
- Piggybacking. Rather than send a specific packet to acknowledge a previously received data packet, the process may embed the acknowledgement in the first data packet that it returns to the other process.
- Sliding Windows. A process does not wait for an acknowledgement before sending the next packet. Instead, it has a "window" of unacknowledged packets at any one time.

The TCP/IP model shown in Figure 3.6 assumes that packets cannot be lost, and that a SRC or DST process will always respond fast enough to prevent re-transmissions. Table 3.1 shows the number of packets involved in a particular TCP function for various packet traces from Memorial's network backbone. Note that the functions are not mutually exclusive – a packet can be both an acknowledgement and a re-transmission. (The relatively large number of re-transmissions for the X protocol are probably the

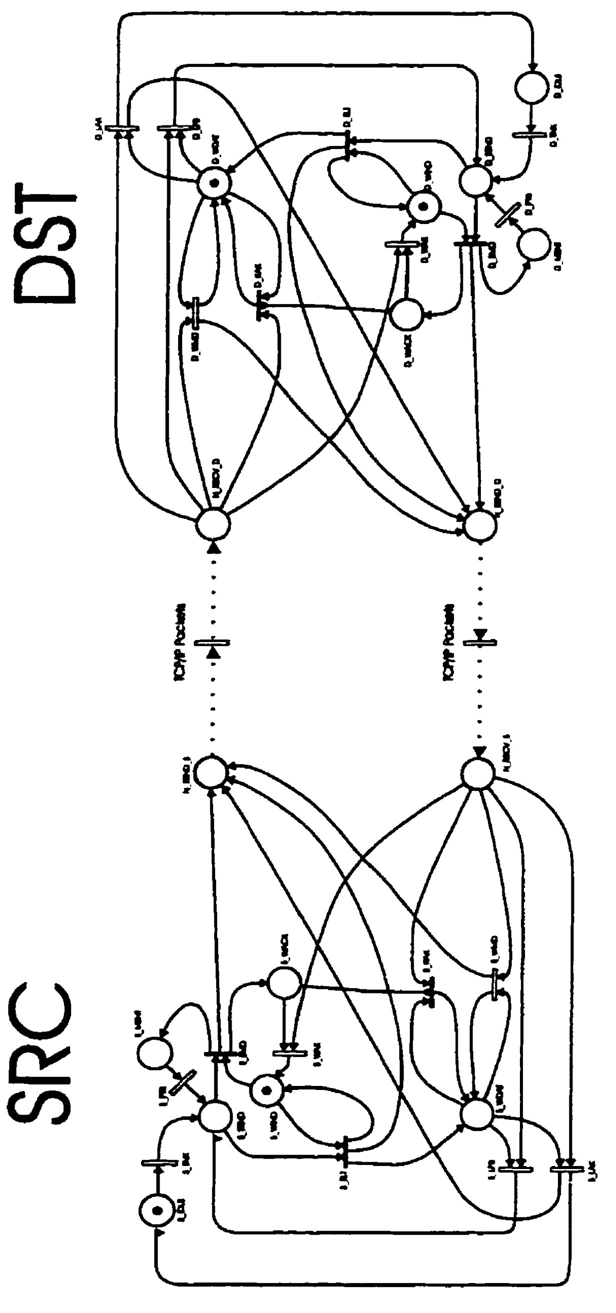

Figure 3.6: TCP/IP stack level model.

Function	Frequency in Trace Data			
	TELNET	FTP	NNTP	X
Acknowledgements	33%	35%	33%	24%
Piggybacking	31%	18%	21%	74%
Re-transmissions	1.5%	2.9%	8%	27%
Sliding Windows	1.9%	31%	31%	26%

Table 3.1: Impact of TCP/IP functions on protocol behaviour.

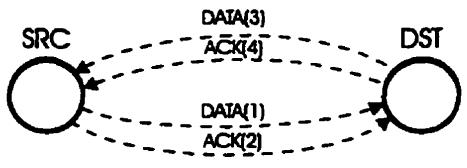

Figure 3.7: Packet flows in model (single data type).

result of a long network path from source to destination across the campus network for the connections included in the raw data.)

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. We can broadly characterize the types of packets flowing between SRC and DST into four types (see Figure 3.7):

1. *DATA(1)*: SRC to DST: data packet,
2. *ACK(2)*: SRC to DST: acknowledgement packet,
3. *DATA(3)*: DST to SRC: data packet,
4. *ACK(4)*: DST to SRC: acknowledgement packet.

(The notation *DATA(1)* describes the type of packet. The numeral in parentheses distinguishes the different packets, while the remainder of the name is a reminder of the packet's function. For example, a DATA(3) packet is type 3 - a data packet from the DST process to the SRC process. The 'DATA' part of the name indicates that the packet carries actual application protocol data.)

Piggybacking is assumed to be a part of all data packets. That is, if a process expects an ACK for a previous transmission, it will accept a DATA packet as both a new data transmission and an implicit acknowledgement of the previous transmission.

However, most network applications send a *group* of data packets, wait for a group of packets in reply, send another group, and so on. Table 3.2 shows typical data group sizes (in packets) for various application-level protocols. For real network processes, the end of a group can be detected through the data contained in the packets. A TELNET session, for example, echos keystrokes until it reads a carriage return character, at which point it processes the command. The model does not have any information about the contents of the packets, so a different mechanism had to be found to signal the end of a data group. This was done by creating two more packet types (LST(2) and LST(5)) to represent the last packet in a data group (one for each direction). This brings the total number of packet types (per application protocol) to six. The revised packet flow diagram is shown in Figure 3.8.

Each process sends a certain number (possibly zero) of *MID* packets, followed by a single *LST* data packet. (It is assumed that the receiving process is sending back

	TELNET	FTP	NNTP	X
SRC to DST	1.32	11.6	1.02	1.70
DST to SRC	1.46	9.8	5.75	1.72

Table 3.2: Mean size of data groups (in packets).

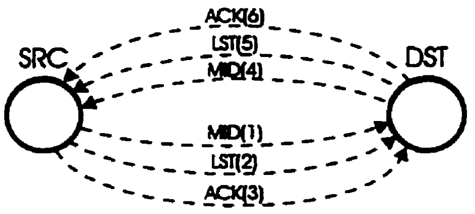

Figure 3.8: Packet flows in model (multiple data types).

acknowledgment packets for each data packet sent.) When the receiving process detects a 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 colour is used for the User/Application level, and a further six colours are used for the six packet types representing the lower level behaviour of an application protocol. For example, a model of FTP applications would use one colour for the user session state, and six more for packet types transmitted between processes. A model of FTP *and* TELNET would require 14 colours, and so on.

This distinguishing of packet types also allows the model to represent behaviour based on packet size. For example, the transmission delay of a packet through a network, which is often dependent on the size of the packet, can be based on the mean packet size for each type, rather than the overall mean packet size.

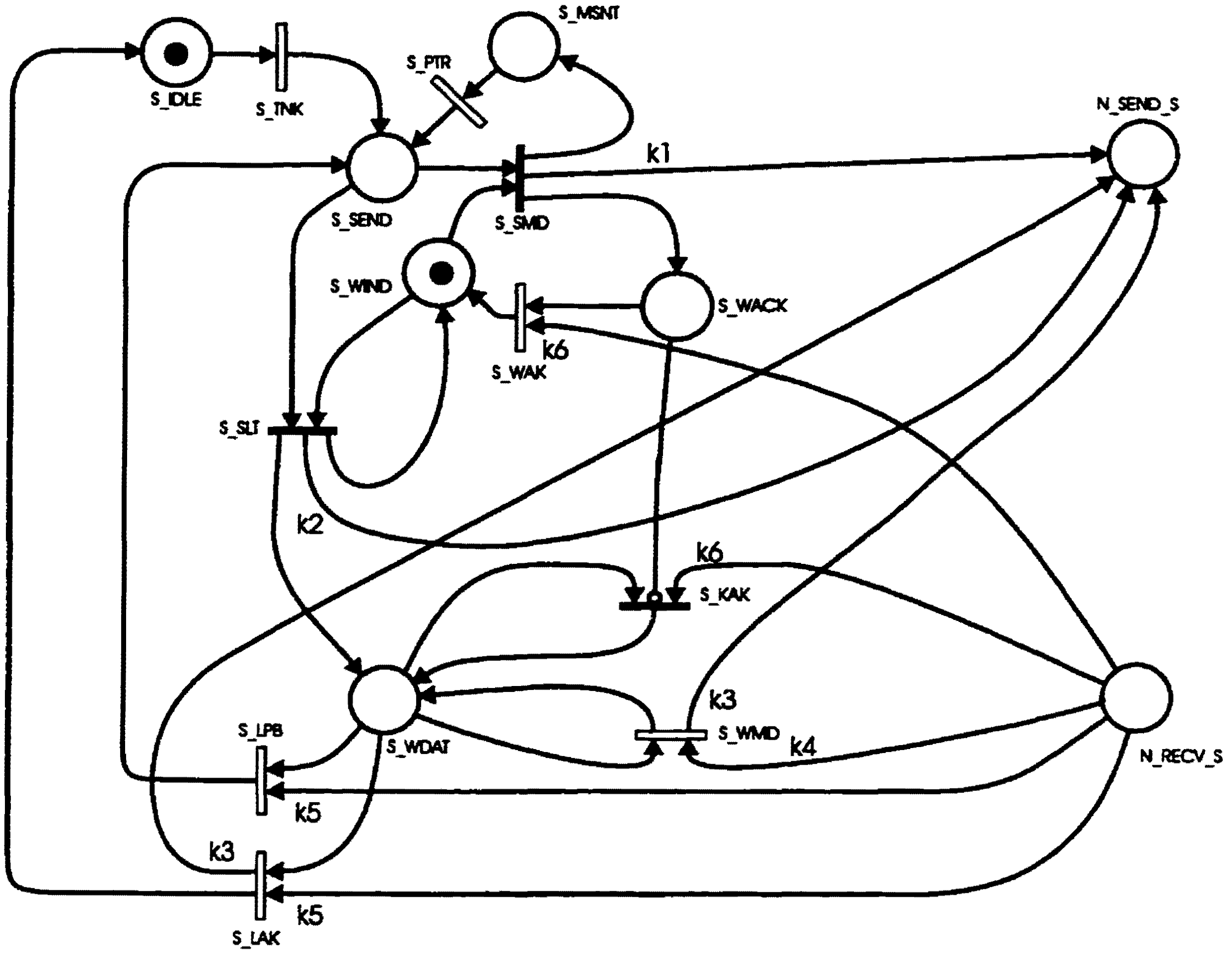

Figure 3.9: TCP Source (SRC) process.

Figure 3.9 shows the model of a SRC process combining the User/Application level and the TCP/IP level. Control tokens cycling through places **S_IDLE**, **S_SEND**, and **S_WDAT** represent the User/Application level shown in Figure 3.5. When the control token is in place **S_SEND**, both transitions **S_SMD** ('Send MID') and **S_SLT** ('Send LST') are enabled. This forms a free-choice structure, which is controlled by choice probabilities assigned to the two enabled transitions. For example, by assigning choice probabilities we can cause **S_SMD** to fire in 20% of cases, and **S_SLT** in 80% of cases.

When **S_SMD** fires, it places a token representing packet type MID(1) into place **N_SEND_S**, modeling the host attempting to transmit a data packet through the next layer in the reference model. A control token is also removed from place **S_WIND**. If no tokens are present in **S_WIND**, it indicates that the number of unacknowledged packets in the sliding window is at maximum, and no more data packets can be transmitted until some acknowledgments arrive. Control tokens are also deposited in places **S_WACK** and **S_MSNT**. At this point, the token in **S_WACK** waits until an acknowledgment packet (ACK) arrives from the DST process, at which time transition **S_WAK** fires and deposits a control token back in place **S_WIND** (i.e. the sliding window moves forward by one packet). **S_PTR** will fire as well, returning a control token to place **S_SEND**. Since we have returned to the free-choice structure described earlier, this cycle will continue for each time **S_SMD** is fired (sending a MID(1) packet) until **S_SLT** fires (sending a LST(2) packet indicating the end of a data group).

The size of the data group is determined by the probability assigned to **S_SLT**. If p is the probability of choosing **S_SLT**, then the size of the data group is a geometric random variable [54] with probability density:

$$P\{X = n\} = (1-p)^{n-1}p, \quad n = 1, 2, ... \tag{3.1}$$

Since p can be different for each application protocol, it is possible to model the mean data group size on a per-protocol basis.

When **S_SLT** fires, transmitting a LST(2) packet and ending the data group transmitted by the SRC process, it deposits a control token 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 at some point. The model needs to handle the arrival of MID(4), LST(5), and ACK(6) packets in place **N_RECV_S**, which represents the TCP/IP stack accepting a packet from the lower layer in the reference model.

An ACK(6) packet is handled by transition **S_KAK**, which removes the token from **N_RECV_S** and simply returns the control token back to **S_WDAT**. The inhibitor arc from place **S_WACK** to **S_KAK** enforces priority for received ACK(6) tokens. (See section 3.8.2.)

A MID(4) packet arriving in place **N_RECV_S** is handled by the transition **S_WMD**, which deposits an ACK(3) packet in place **N_SEND_S** to acknowledge the MID(4) packet. (MID type packets are always acknowledged in the model.)

An arriving LST(5) packet in place **N_RECV_S** indicates the end of the data group from the DST process. At this point, the SRC process can either return an acknowledgement for the LST(5) packet, or start the transmission of the next data group immediately, which implies that the acknowledgement is piggybacked onto the first data packet. This is modeled using a free-choice structure. When a control token is in place **S_WDAT** and a LST(5) token arrives in place **N_RECV_S**, either transition **S_LPB** will fire with probability α, or **S_LAK** with probability $1 - \alpha$. Piggybacking is dependent on host and application factors; if an application is ready to send data when a LST(5) packet arrives, or the host/application can generate a response within a short period of time, the TCP/IP stack will send the data with the piggybacked acknowledgement. Otherwise, it will send a specific ACK(3) packet to inform the DST process of the successful arrival of the LST(5) packet. The parameter α can vary for each application-level protocol.

Once an arriving MID(4), LST(5), or ACK(6) packet is handled by the SRC process, the process reverts to the start state, either through the transition **S_IDLE**, indicating that the User/Application level requires processing time (and an acknowledgement has been sent), or directly to place **S_SEND** to start the next data group from the SRC. This represents the piggybacked case, where the User/Application is ready to send data.

The DST process is an exact mirror image of the SRC process from a structural point of view, and follows the same cycle. However, it exists in the opposite state as the SRC process (i.e. it waits while the SRC process is sending data, and vice versa), and it typically has different timing parameters, since the responses of the DST usually represent software replies to a command or query.

3.4.2 UDP

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

A UDP process is modeled as a subnet of the TCP process. Figure 3.10 shows a UDP process subnet, with the unused TCP portions shown by dotted lines. The UDP model basically trades data groups, with the SRC process sending a series of MID(1) packets followed by a single LST(2) packet to the DST process. (UDP does not use acknowledgements, so the timing between packets is dependent on the application.) As in the TCP process, the size of the data group is modeled as a geometric random variable.

Once the DST process receives the LST(2) packet, it begins transmitting its own data group, finishing with a LST(5) packet. The SRC process then begins again.

3.5 LAN (Ethernet) Level

The Ethernet level of the reference model is described by a simple net structure that performs two basic functions: a) it adds a transmission delay for each packet, and b) it forces the different application protocols to interact. (Except for the windowing mechanism at the TCP/IP level, each session of each protocol has, until now, been able

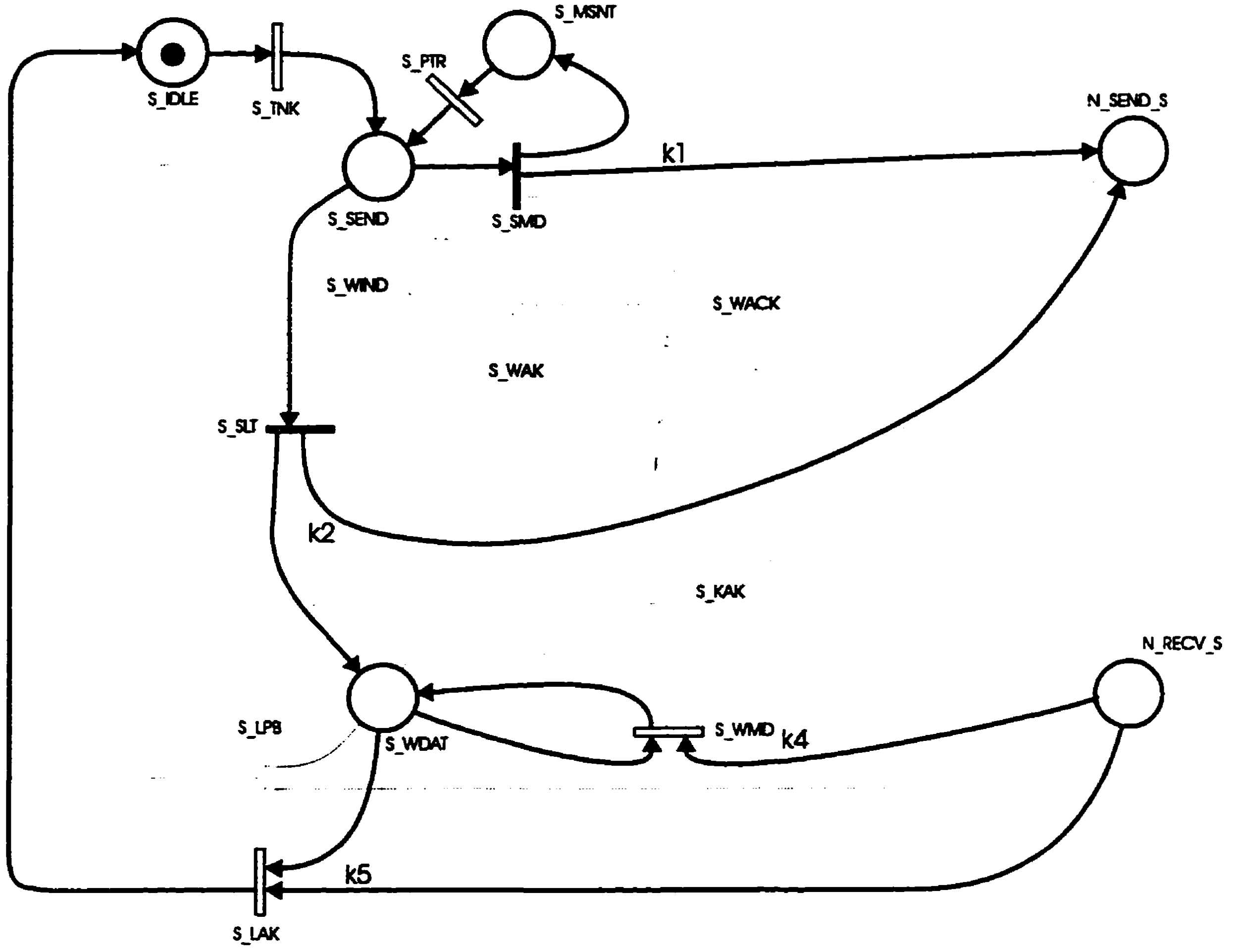

Figure 3.10: UDP Source (SRC) process.

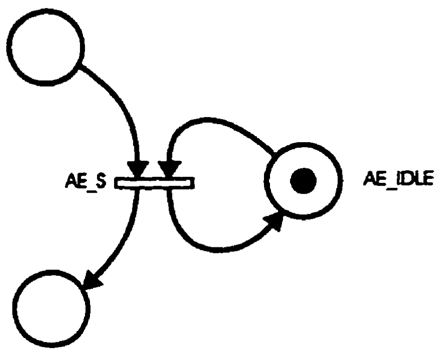

Figure 3.11: Ethernet model.

to operate independently of the other sessions.)

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

Figure 3.11 shows the model of the Ethernet layer. The controlling place **AE_IDLE** contains one token, which ensures that only one occurrence of transition **AE_S** can fire at the same time. The actual transmission times are modeled by deterministic transitions, which are unique to each packet type of each application protocol. The occurrence probabilities of **AE_S** are marking-sensitive, which results in a colour being chosen based on the relative frequency of that colour versus the other colours in the input place of the Ethernet.

3.6 ATM Level

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

3.6.1 AAL (SAR) Function

The AAL layer is modeled in two sections. The first (Figure 3.12) 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 (Figure 3.13) represents the reverse process. It inputs a number of cells and re-creates the original packet for transmission to the next layer.

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.

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 (i.e. only one "packet" token passes through to **SAR_IN_P2**). Place **SAR_IN_P3** controls the number of

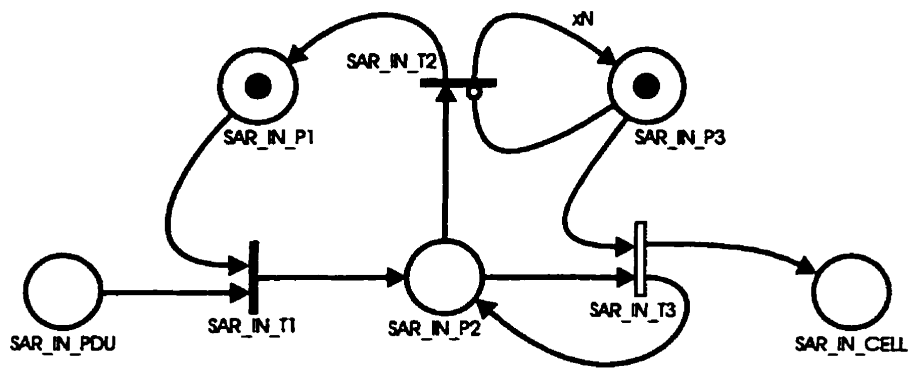

Figure 3.12: AAL (SAR) level - segmentation.

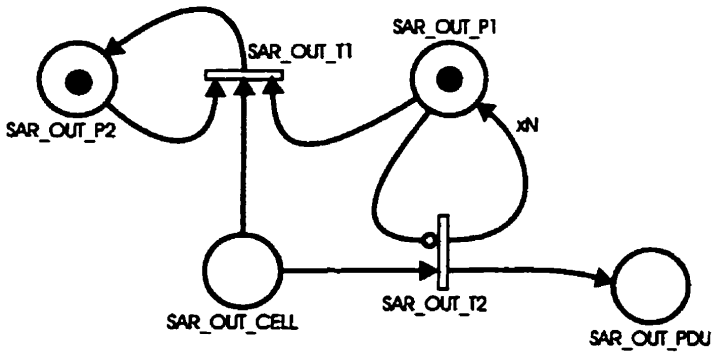

Figure 3.13: AAL (SAR) level - reassembly.

cells that are generated by 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 colour representing the TELNET MID(1) packet are placed in place **SAR_IN_P3**. When a token is deposited in place **SAR_IN_P2**, transition **SAR_IN_T3** will fire a set number of times determined by the colour of the token in place **SAR_IN_P2**, thus generating a specific number of cells for each packet type. When the last cell has been generated, all tokens of the current colour have been removed from place **SAR_IN_P3**. This enables transition **SAR_IN_T2**, which removes the token from place **SAR_IN_P2** (indicating that all cells have been generated), replaces the required number of tokens in **SAR_IN_P3** (to segment the next packet of this type), and returns the control token to place **SAR_IN_P1**, to start the segmentation of the next packet.

The reassembly section (Figure 3.13) operates in a similar manner to represent the re-creation of packets from a stream of cells. Tokens are deposited in place **SAR_OUT_CELL** representing the arrival of cells at the SAR layer. Place **SAR_OUT_P1** contains a number of coloured tokens for each packet type equal to one less than the number of cells the packet is divided into. As cells arrive, they are removed by transition **SAR_OUT_T1**. Place **SAR_OUT_P2** contains one control token to ensure that only one cell is "re-assembled" at a time. When all the cells for a packet have arrived (indicated by the removal of *all* the tokens for that colour from place **SAR_OUT_P1**), transition **SAR_OUT_T2** fires, depositing a single token of the re-assembled packet's colour in the output place **SAR_OUT_PDU**. **SAR_OUT_T2** also returns the correct number of tokens to place **SAR_OUT_P1** in preparation for

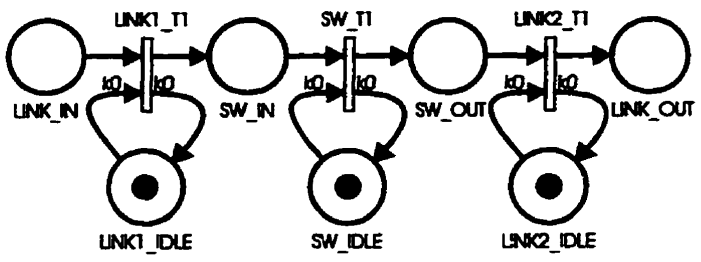

Figure 3.14: ATM switch fabric.

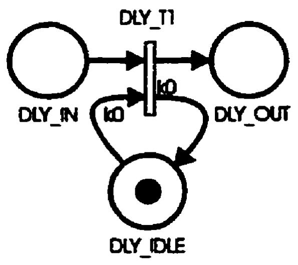

Figure 3.15: Generic Delay.

the next cell stream for that colour.

3.6.2 ATM Switch

The ATM switching fabric is modeled by a series of delay modules as shown in Figure 3.14. The delays represent the latencies of the two OC-3c SONET links and the ATM switch (Figure 2.4). The use of three delays rather than one longer delay allows a stream of cells to move through the layer with several in the system at any one time.

3.7 Generic Delay

Figure 3.15 shows a generic delay mechanism for adding timing delays in the model where appropriate. Transition **DLY_T1** has an occurrence for each packet type, so the delay can be set uniquely for each type. Place **DLY_IDLE** contains a single control token to ensure that only one occurrence fires at a time. Delays such as this are used to model bridge latency, for example.

If a delay is required where uniqueness is not needed, a simple transition is sufficient.

3.8 Discussion

In any model design process, the modeler must make design decisions that can have a major effect on the usefulness of the model. This section describes two such decisions in greater detail.

3.8.1 Single versus Multi Session Models

A key element of the design of the model is that it should be able to represent multiple simultaneous conversations of a set of application-level protocols. Some information is lost, however, in the multi-session version. There are multiple control tokens in operation, and there is no way to exactly match a particular control token with the other tokens it generates. For example, **S_SLT** fires, sending a LST(2) packet to the DST process and placing a control token in **S_WDAT**. There is no difference between an arriving LST(5) caused by our particular control token, and by an earlier or later firing of **S_SLT**.

To verify that the multi-session model is still valid, a model with a single protocol and n sessions was compared to a model with n unique single-session protocols, each with parameters identical to the multi-session model. The multi-protocol model more closely resembles 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 requires many more colours and is not feasible for large numbers of sessions, however.) It was found that the multi-session model gave slightly higher results for packets and bytes per second, but the differences between the models were not statistically significant ($p = 0.552$) when compared using standard hypothesis testing [25].

3.8.2 Priority of Received Acknowledgements

An ACK(6) packet deposited in place **N_RECV_S** (Figure 3.6) is either acknowledging a previously transmitted MID(1) or a LST(2) packet. TCP normally uses sequence numbers (see [49]) to differentiate between the two cases. However, since the model does not reflect that level of detail, a simpler mechanism is required. In a single-session case, where there is only one user per protocol, the location of the control packet determines which data packet is being acknowledged. That is, if a control token is in place **S_WACK**, representing a transmitted MID(1) packet, then the acknowledgement is for that packet. Otherwise, if a control token is in place **S_WDAT**, then the model correctly accepts the acknowledgement for a transmitted LST(2) packet.

In the multi-session version of the model, however, there is no way to match a particular ACK(6) packet with the MID(1) or LST(2) packet that generated it. Furthermore, since the place **S_KAK** will accept ACK(6) packets and return control to place **S_WDAT**, it will tend to leave session control tokens in **S_WACK**, eventually leading to a deadlock in the net.

Two solutions were developed for this problem. The first was to put an inhibitor arc between place **S_WACK** and transition **S_KAK**. This effectively gives MID(1) packets priority for any arriving ACK(6) packet, and removes the potential for deadlock. The other option was to add an additional packet type and have an ACK(6) for MID(1) acknowledgements and an ACK(7) for LST(2) acknowledgements. The second solution, while somewhat more realistic, also adds two colours per protocol and two occurrences per transition for the intervening layers in the reference model. As well, no additional benefit regarding packet size distribution is realized, since both types would be the same size. (The split between MID and LST packet types, in contrast, actually improves the size distribution because the two types typically have different mean sizes.)

In simulation tests, the first option showed slightly higher values for throughput and average burst rate. When compared using hypothesis testing [25] the results were not found to be statistically significant ($p = 0.514$). It was decided to use the first option.

This chapter described the derivation and construction of a timed, coloured Petri net model of an ATM-based network. The next chapter discusses how the implementation details of the model (i.e. actual timing and probability values) are determined. It also covers the data collection process that resulted in those values, and some results from simulation of the model.

Chapter 4

Model Performance

This chapter describes how data was gathered from Memorial's campus network, how that data was analyzed, and how it was used to derive the model parameter values. The chapter concludes with some validation results (i.e. tests that ensure that model behaviour is a reasonable estimate of actual system behaviour), and some simulation results from the model.

4.1 Data Collection

The collection and analysis of actual network data formed a key part in both the development of the structure of the model and the derivation of model parameters for simulation. This section presents some the details of the data collection process.

4.1.1 Network Monitoring Software: TCPDUMP

TCPDUMP [26] is a public domain TCP/IP monitoring program that has become a commonly used tool for network management. It normally produces a line of ASCII output describing each packet that appears on an attached network. The user can filter the output to select certain events - specific TCP port numbers. or a particular IP subnet, for example. Although TCPDUMP is limited by the host computer hardware in both event granularity, (i.e. the smallest time interval it can recognize), and the maximum monitoring speed, it is an excellent tool for network monitoring over long time periods.

4.1.2 Data Sources

The collection of data for the model started with an examination of protocol frequency on the Memorial network. The Ethernet backbone of the campus network was monitored for a week. and packet and byte counts per protocol (as determined by TCP/UDP port numbers) were recorded. Table 4.1 shows the relative frequencies for the most common protocols.

Traces of some of the most common protocols were then taken over the space of a week. Each trace was made up of at least four separate sub-traces randomly spaced over the week to reduce bias from system load, both on the network and the monitoring system(s) themselves. Since the author expects the usage of the network to include more sound and video applications in the future, an additional trace was made of a

Protocol	Packets	Bytes
SHELL	13.80%	33.69%
TELNET	40.18%	14.45%
NNTP	5.99%	12.11%
NFS	7.75%	9.71%
SMTP	3.80%	9.37%
X	11.10%	7.21%
LOGIN	8.25%	3.97%
FTP (DATA)	1.46%	3.76%
WWW	1.43%	1.59%
OTHER	6.24%	4.14%

Table 4.1: Relative Frequency of Network Protocols.

CU-SeeMe session for comparison purposes.

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. (Figure 4.1 shows the data collection locations in relation to the system model.) Locating the monitoring stations as close to the end-systems as possible reduced measurement errors caused by network delay.

Since we are interested in the network load of a single user, a further complication was determining what actually constituted a "user session". A TCP connection has distinct start and end points that can be detected by the TCP flags reported in the data traces. For a TELNET session, a TCP connection and a "user session" are equal - the connection is set up when the user logs in to the remote host, and remains until the user is finished his terminal session. For other protocols, however, a single "user session" (such as someone accessing a WWW site with a browser) may create multiple

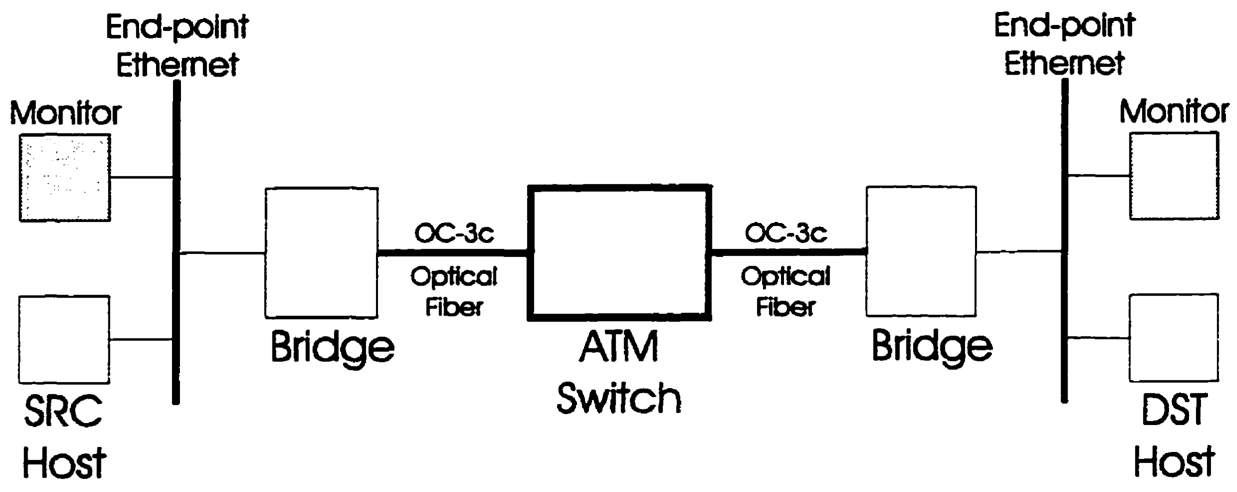

Figure 4.1: Data collection configuration.

TCP connections. For these protocols, two models were made from each trace. The first equates a "user session" with each TCP connection. The second model was generated by erasing either the source port numbers or the destination IP addresses (depending on the protocol) in the data trace. This had the effect of concatenating consecutive TCP connections into one longer connection representing the total activity of the user. (Since most of the sessions were from individual personal computers, the unique IP addresses of the source hosts served to separate the different user sessions.) It is felt that the actual load of a user session lies somewhere between the two extremes.

4.2 Simulation Software

Two software packages were used in the design and simulation of the model in this thesis. The first package, Visual SimNet [20], is a PC based Petri Net simulator. It was used extensively in the early design stage of building the model, primarily because the graphical interface was a significant aid in laying out and testing various parts of the model. However, the program did not support "infinite firing semantics" for transitions (i.e. multiple transition firings with overlapping times). An attempt to approximate this behaviour by creating multiple transitions for each single transition in the model was not successful due to internal limitations in the software.

Following a survey of Petri net software, a second software package, TPNsim [60], was chosen as the simulation software for this thesis. In addition to supporting "infinite firing semantics", the software had some additional features that made it attractive for this work: the author of the software was available to add features as required, output routines specific to this model could be added easily, and the program could be integrated into a suite of programs and scripts to perform various types of simulations and data comparisons.

TPNsim reads a net represented in a Net Description Language. This language is briefly described in Appendix B, while Appendix C contains a specification of a version of the model discussed in this thesis. The simulation portion of the software implements the event-driven approach [30]. The package also supports structural analysis (i.e. analysis of net properties based solely on the topology of the net [41]), as well as reachability analysis of Petri nets.

4.3 Model Tuning

The model described in Chapter 3 captures the basic behaviour of a multi-protocol environment over ATM. A key premise of the model, however, is that while the basic behaviour of the various protocols is similar (represented by the structure of the coloured Petri net model), the individual protocols can behave differently. The use of individual colours allows the modeler to separate the protocols by assigning different timing and probability parameters to different colours. This section discusses how those parameters are determined.

4.3.1 Performance Parameters

Since the model used in this thesis was developed in sections representing various layers of a communication system, the preferred method of gathering actual performance data would be to analyze each layer independently. This did not prove to be feasible in all cases, however. Analyzing the User/Application and TCP/IP stack layers in specific detail requires modifying code and gaining access to systems and resources that are beyond the scope of this thesis. For example, existing studies of TCP/IP stack performance involved either detailed code analysis or modification [10] or the use of a hardware logic analyzer [29]. Neither of these methods were possible on the scale required for this work.

It was therefore decided to estimate these parameters based on what could be observed at the network level.

		Second Packet					
		MID(1)	LST(2)	ACK(3)	MID(4)	LST(5)	ACK(6)
First Packet	MID(1)	12	4	1	0	0	97
	LST(2)	0	0	9	236	1579	409
	ACK(3)	36	1351	27	461	430	3
	MID(4)	0	0	879	67	76	0
	LST(5)	13	819	1401	0	0	1
	ACK(6)	52	49	1	258	150	0

Table 4.2: "Next Packet Frequency" matrix for TELNET protocol.

The first step in defining the timing and probability parameters is evaluating the actual behaviour of a protocol in reference to the basic model shown in Figure 3.7. Since the model has six distinct packet types for each protocol, one obvious metric is the number of occurences of each type. However, this provides little useful information on the nature of a particular protocol.

The SRC and DST processes are synchronized with packets representing the communication between the two. Since each packet signals an event or state change for one or both of the processes, the relationship between adjacent packets should provide far more information on the behaviour of the protocol than simple packet counts. An example of such a "next packet" matrix is shown in Table 4.2.

The matrix shows the number of occurrences of each packet type given that the previous packet was of type n. For example, the top right location in the matrix shows that a MID(1) type packet was followed by an ACK(6) packet 97 times in the data trace. It is important to note that the "next packet" data is on a *per-connection* basis. That is, we are interested in how many times an ACK(6) follows a MID(1) for the same

		Second Packet					
		MID(1)	LST(2)	ACK(3)	MID(4)	LST(5)	ACK(6)
First Packet	MID(1)	10.5	3.5	0.9	0.0	0.0	85.1
	LST(2)	0.0	0.0	0.4	10.6	70.7	18.3
	ACK(3)	1.6	58.5	1.2	20.0	18.6	0.1
	MID(4)	0.0	0.0	86.0	6.6	7.4	0.0
	LST(5)	0.6	36.7	62.7	0.0	0.0	0.0
	ACK(6)	10.2	9.6	0.2	50.6	29.4	0.0

Table 4.3: "Next Packet Probability" matrix for TELNET protocol.

user session.

Some useful versions of this matrix are generated by scaling. The "probability matrix" (Table 4.3) is formed by treating each row as a probability density. This "probability matrix" is similar to a transition matrix of a Markov process [54].

A similar matrix is generated for timing information. Table 4.4 shows the "timing" matrix for the same TELNET data trace. The matrix shows the average observed delay (in milliseconds) between packets of different types. Measurements are from the start of each packet. Zero values indicate either that no packet combinations of that type were observed, or that the difference was smaller than the clock granularity of the measuring device (approximately 1 μsec.).

Other important parameters are shown in Tables 4.5, 4.6, and 4.7. **SWIN** and **RWIN** (Table 4.5) refer to the average data group size (in numbers of packets) from the SRC and DST process, respectively. The **SOUT** and **ROUT** parameters show the average unacknowledged packet window size for the two processes. The **THINK** parameter (Table 4.6) shows the average delay between the last data packet from the

		Second Packet					
		MID(1)	LST(2)	ACK(3)	MID(4)	LST(5)	ACK(6)
First Packet	MID(1)	0.159167	0.040000	0.150000	0.000000	0.000000	0.041959
	LST(2)	0.000000	0.000000	0.052222	0.018347	0.009417	0.047262
	ACK(3)	1.961111	3.701066	36.984074	0.187874	0.096419	0.026667
	MID(4)	0.000000	0.000000	0.134903	0.107761	0.202895	0.000000
	LST(5)	0.053077	0.150708	0.184104	0.000000	0.000000	0.010000
	ACK(6)	0.309615	0.982245	0.060000	0.199574	0.088000	0.000000

Table 4.4: "Timing" matrix for TELNET protocol (msec).

DST process and the first data packet from the SRC process; For a TELNET session this could approximate the time a user takes to respond to a screen of data. The **DREPLY** parameter is the opposite - it shows the average delay between the arrival of the last data packet from the SRC and the first data packet from the DST process; **DREPLY** approximates the processing time of the host. The last parameter in Table 4.6 is the average inter-arrival times between packets. This parameter is on a per-connection basis – the inter-arrival times are for the same user session. All measurements are taken from the beginning of each packet.

Table 4.7 shows basic packet counts, relative frequencies, and average packet sizes for the six packet types (packet sizes are Ethernet frame sizes). The total number of packets in the trace and the global average size is also shown.

Parameter	Count	Average (Packets)	Variance	Maximum
SWIN	2236	1.050984	0.273015	-
RWIN	2236	1.457066	1.166389	-
SOUT	2236	1.173443	0.317880	5
ROUT	2236	1.116657	0.199538	5

Table 4.5: Miscellaneous parameters for TELNET protocol (1).

Parameter	Count	Average (sec.)	Variance
THINK	2221	2.456006	330.749099
DREPLY	2233	0.046740	0.018706
Packet Inter-Arrival Time	8421	0.818788	107.531884

Table 4.6: Miscellaneous parameters for TELNET protocol (2).

Parameter	Count	Freq.	Average (Bytes)	Variance
Packet Size: MID(1)	114	0.01%	60.736842	5.505356
Packet Size: LST(2)	2236	0.26%	60.036225	0.365130
Packet Size: ACK(3)	2321	0.28%	60.000000	0.000000
Packet Size: MID(4)	1022	0.12%	401.822896	102815.225215
Packet Size: LST(5)	2236	0.26%	167.937835	45783.729020
Packet Size: ACK(6)	511	0.06%	60.000000	0.000000
TOTAL	8440	-	130.006754	36900.417895

Table 4.7: Miscellaneous parameters for TELNET protocol (3).

4.3.2 Model Parameters

Given the performance parameters determined in Section 4.3.1, the next step is to find a set of model parameters for which the model behaviour matches the actual system behaviour as closely as possible.

Tables 4.8 and 4.9 show the parameters of the model. Each parameter is classified as one of the following types:

Operational : unique text strings that serve to separate different protocols and sections of the model from each other.

Structural : a parameter which alters the basic structure of the model. UDP models, for example, do not use all the transitions and places of TCP models.

Marking : a parameter which determines the initial marking of a place.

Time : a parameter which determines the firing time of a transition.

Probability : a parameter which determines the choice probability in a free-choice configuration.

The actual values for the various parameters are derived from a number of sources. Parameters in group A (Table 4.8) are determined by the particular protocol. Parameters in group B (Table 4.8) control the operation of the User/Application and TCP/IP stack levels in the model for the SRC process. (The actual net transitions or places of

Group	Parameter Name	Type	Transition/Place
A	name	Operational	-
	abbr	Operational	-
	Protocol	Structural	Various
	ns	Marking	**S_IDLE,D_WDAT**
B	Think	Time	**S_TNK**
	SendMidProb	Probability	**S_SMD**
	SendLstProb	Probability	**S_SLT**
	PTRTime	Time	**S_PTR**
	WinSize	Marking	**S_WIND**
	SendDelay1	Time	**S_WAK**
	MidAckDelay	Time	**S_WMD**
	LstPBDelay	Time	**S_LPB**
	LstAckDelay	Time	**S_LAK**
	RecvPBProb	Probability	**S_LPB**
	RecvThProb	Probability	**S_LAK**
C	DThink	Time	**D_TNK**
	DSendMidProb	Probability	**D_SMD**
	DSendLstProb	Probability	**D_SLT**
	DPTRTime	Time	**D_PTR**
	DWinSize	Marking	**D_WIND**
	DSendDelay1	Time	**D_WAK**
	DMidAckDelay	Time	**D_WMD**
	DLstPBDelay	Time	**D_LPB**
	DLstAckDelay	Time	**D_LAK**
	DRecvPBProb	Probability	**D_LPB**
	DRecvThProb	Probability	**D_LAK**

Table 4.8: List of model parameters (part 1).

Group	Parameter Name	Type	Transition/Place
D	E_1	Time	**E_S**
	E_2	Time	**E_S**
	E_3	Time	**E_S**
	E_4	Time	**E_S**
	E_5	Time	**E_S**
	E_6	Time	**E_S**
E	P_1	Marking	-
	P_2	Marking	-
	P_3	Marking	-
	P_4	Marking	-
	P_5	Marking	-
	P_6	Marking	-
F	SAR_1	Marking	**SAR_IN_P3**
	SAR_2	Marking	**SAR_IN_P3**
	SAR_3	Marking	**SAR_IN_P3**
	SAR_4	Marking	**SAR_IN_P3**
	SAR_5	Marking	**SAR_IN_P3**
	SAR_6	Marking	**SAR_IN_P3**
G	SAR_MAX	Time	**SAR_IN_T3**
	BR_LAT	Time	Bridge Delay
	OC3_LAT	Time	**LINK1_T1**
	ATM_LAT	Time	**SW_T1**

Table 4.9: List of model parameters (part 2).

the model that are affected by each parameter are also listed in Tables 4.8 and 4.9.) Group C parameters are the DST process analogues for the group B parameters.

Ethernet timing parameters are listed in group D, while group E parameters govern the actual size (at the Ethernet frame level) for each packet type. The ATM SAR level functions are partially controlled by parameters in group F. Group G contains parameters used in the bridge and ATM layers of the model.

The assignment of values to the various parameters is discussed in more detail in the remainder of this section.

Group B Parameters

This group contains timing and probability parameters. The *SendMidProb* and *SendLstProb* parameters control the average number of packets in a data group transmitted by the process. Since we know that the size of the data group (in the model) is a geometric random variable (see Equation 3.1, page 38), it follows that:

$$p = \frac{1}{k} \tag{4.1}$$

where p (i.e. parameter *SendLstProb*) is the probability of sending a LST(3) packet, and k is the **SWIN** value measured from the trace data (see Section 4.3.1). *SendMidProb* is simply $1 - p$.

The marking parameter *WinSize* is determined by the maximum outstanding packet window parameter **SOUT** from the trace data (Section 4.3.1).

Parameter Name	Time (1)	Time (2)	Ethernet Delay
Think	*(3,1)*	*(3,2)*	ϵ_3
PTRTime	*(1,1)*	*(1,2)*	-
SendDelay1	*(6,1)*	*(6,2)*	ϵ_6
MidAckDelay	*(4,3)*		ϵ_4
LstPBDelay	*(5,1)*	*(5,2)*	ϵ_5
LstAckDelay	*(5,3)*		ϵ_5
DThink	*(6,4)*	*(6,5)*	ϵ_6
DPTRTime	*(4,4)*	*(4,5)*	-
DSendDelay1	*(3,4)*	*(3,5)*	ϵ_3
DMidAckDelay	*(1,6)*		ϵ_1
DLstPBDelay	*(2,4)*	*(2,5)*	ϵ_2
DLstAckDelay	*(2,6)*		ϵ_2

Table 4.10: Derivation of Group B and C timing parameters.

The timing parameters in group B are derived from the timing and packet count matrices from the trace data. Each element of the "next packet" timing matrix is evaluated using a simple structural or path trace through the net. The value of the element can then be used to estimate the timings of the transitions involved. For example, the *(4,3)* element of the timing matrix (i.e., the average time between a MID(4) packet and an ACK(3) packet), upon inspection, will be equal to the average time of transition **S_WMD** (parameter *MidAckDelay*) plus a delay ϵ representing the average transmission time of a MID(4) packet. (The timing matrices are based on trace data which only logs the beginning of each packet. The packet hasn't really "arrived" until the end of the packet, however, so this delay must be included.) The parameter *MidAckDelay* can thus be estimated as $t - \epsilon$, where t is the *(4,3)* element of the timing matrix and ϵ is the packet transmission delay.

Table 4.10 shows the derivation of the group B and C timing parameters. Parameters with a single time are derived as already discussed. Other parameters, however, affect two elements in the timing matrix because of the various free-choice situations in the model. In these cases, the time parameter is estimated by using a linear combination of the two times scaled by the relative packet counts of each element (Table 4.2). That is, given the timing matrix T and the packet count matrix P, if parameter k is involved in times $T(l, j_1)$ and $T(l, j_2)$, then k is estimated by:

$$k = \frac{T(l,j_1)P(l,j_1) + T(l,j_2)P(l,j_2)}{P(l,j_1) + P(l,j_2)} - \epsilon_l \tag{4.2}$$

The remaining parameters in group B, *RecvPBProb* and *RecvThProb*, determine whether the SRC process returns a unique ACK(3) packet or sends another data packet, implying a piggybacked acknowledgement. These parameters are estimated from the packet count matrix (Table 4.2). The frequency of piggybacking is simply the frequency of an LST(5) packet being followed by either a MID(1) or LST(2) packet ($P(5,1) + P(5,2)$), while the sending of an explicit acknowledgement is indicated by an LST(5) packet followed by an ACK(3) packet ($P(5,3)$). *RecvPBProb* is thus:

$$\frac{P(5,1) + P(5,2)}{P(5,1) + P(5,2) + P(5,3)} \tag{4.3}$$

with *RecvThProb* = 1 - *RecvPBProb*.

Group C Parameters

These parameters are exact analogs of the group B parameters, and are derived in the same manner.

Group D Parameters

Group D parameters are the transmission times for Ethernet frames, and are used both in the derivation of group B and C parameters (as the ϵ_l in Table 4.10), and the timing of the Ethernet section of the model. They are derived by taking the average frame size for each packet type from the data trace.

Group E Parameters

These parameters are simply the average frame size (in bytes) of each packet type, as determined from the data trace.

Group F Parameters

Group F parameters control the division of Ethernet frames into cells at the SAR layer. Each parameter SAR_n represents the number of cells required to transmit a packet of type n from a higher layer.

Group G Parameters

These parameters control various parts of the ATM and bridge layers.

SAR_MAX : The maximum rate at which cells are generated at the SAR layer. It is reported in [4] that the ATM bridge device (Cisco Catalyst 5000)[1] can generate cells at the maximum rate of an OC-3c SONET connection. Since the theoretical maximum for cell transmission over OC-3 is 149.76 Mbps [7], or 353,207.5 cells/sec, a deterministic delay of 2.831 μsec per cell will generate cells at that rate.

BR_LAT : This parameter represents the average delay between the last bit of a packet/frame to arrive at a device and the first bit to appear on another port (LIFO latency) [5]. It is reported in [4] that the Cisco Catalyst 5000 has an Ethernet-to-Ethernet latency of 8 μsec. It is assumed that the per-frame latency of the device for Ethernet-to-ATM is at least as great, and that value was used in the simulations in this thesis. Further discussion on this value can be found in section 4.4.1.

OC3_LAT : The average transmission delay across an OC-3c SONET link. Ignoring propagation delay, this value is set to provide the maximum per-cell transmission rate: 2.831 μsec.

ATM_LAT : The average delay (latency) across the ATM switch. The switch (Cisco Lightstream A100) backplane is rated at 2.4 Gbps, and is capable of switching multiple OC-3c SONET cell streams simultaneously. The switch latency is set at 2.831 μsec per cell as well, since

[1] "Cisco" and "Catalyst 5000" are trademarks of Cisco Systems Inc.

it can easily switch one OC-3c SONET cell stream at the maximum transmission rate.

4.3.3 Effect of Windowing in Trace Data

The derivation of the Group B and C timing parameters was found to produce reasonable results for protocols with small amounts of windowing (e.g., TELNET). However, as windowing increased the model results began to vary from the actual data. Windowing results in a looser relationship between a packet and its explicit acknowledgment – to the point where there is little guarantee that the ACK packet that follows a DATA packet in a trace is acknowledging that particular packet. To solve this problem, the method described in the previous section was applied to the data traces with the TCP sequence numbers used to correctly match the DATA and ACK packets. This greatly improved the accuracy and stability of the model.

4.4 Model Results

This section presents some results from simulation of the model, starting with validation data, which compares the model to actual network behaviour. The section concludes with a series of results showing the effect of various application protocols on parts of the network system.

4.4.1 Model Validation

An important part of the modeling process is *validation* [30], which is comparing the model results with actual measurements to determine if the model is a reasonable approximation of the modeled system. Since many of the timing parameters are determined from measurements on an Ethernet, we can compare the behaviour of the model with actual data from an Ethernet. However, direct comparisons at the user, system stack, or ATM level require measurement devices that were unavailable for this work. Despite that, some measurements were obtained from a number of sources that could be used in model validation.

A version of the model with a single Ethernet connecting a SRC and DST process was compared to actual user sessions on an Ethernet. Figure 4.2 shows a scattergram of packet rates for a number of observed sessions and the model results for each trace. The dashed line is the ideal; it represents an exact match between the model and the actual data.

At the ATM level, it was possible to compare cell rates at a fairly coarse level of granularity (i.e., every minute) by gathering cell transmission totals from the ATM switch. A trace was recorded at the Ethernet level of data coming from the ATM link of the Catalyst 5000. At the same time, cell totals were recorded from the ATM switch for that data VC. The two traces could be accurately synchronized because the packet transmission containing the data from the ATM switch was embedded in the Ethernet trace. A theoretical cell value could be computed from the Ethernet trace and compared

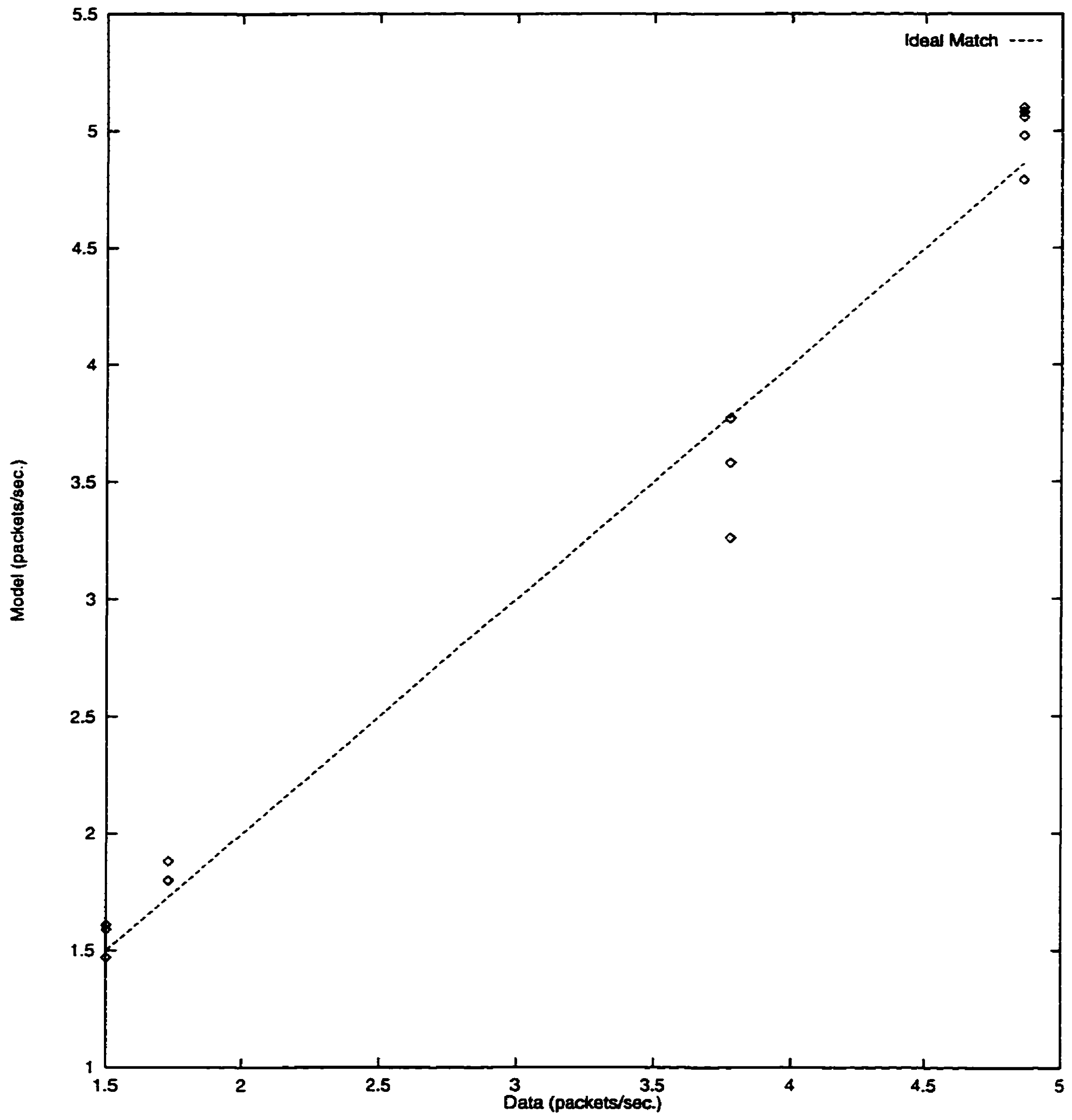

Figure 4.2: Theoretical vs. actual Ethernet packet transmission rates.

with the measured cell totals. Figure 4.3 shows a scattergram of the theoretical and measured cell totals. The linearity of the graphs indicates a good match between the model and actual data.

The performance statistics on the Catalyst 5000 reported in [4] used an ATM test configuration almost identical to the system modeled in this thesis. Of particular interest are the Ethernet-to-Ethernet (across ATM) latency measurements. Figure 4.4 compares the latency values reported in [4] with the values produced by the model. Both show similar slopes, indicating the model is a good predictor of overall behaviour. The gap between the two lines indicates that there is some latency not accounted for in the model. This suggests that the 8 μsec latency value for the Catalyst 5000 is probably too low for Ethernet-to-ATM transmission.

The current version of the model uses either deterministic or exponentially distributed firing times for transitions. Since some transitions (such as the **S_TNK** or "thinking time" transition) can have a major impact on the performance of the whole system, the distribution of those times could have an impact as well. Figure 4.5 compares the actual "thinking time" distribution of a TELNET session with an exponential distribution with the same mean value, as would be used in the equivalent model. The two are roughly similar, although the actual distribution has less data in the tail of the graph.

Since the model uses six different packet types, the mix of packet sizes can be closer to reality than a single average value. Figure 4.6 compares the actual packet size distribution of an FTP protocol data trace and a single average packet size. Figure 4.7

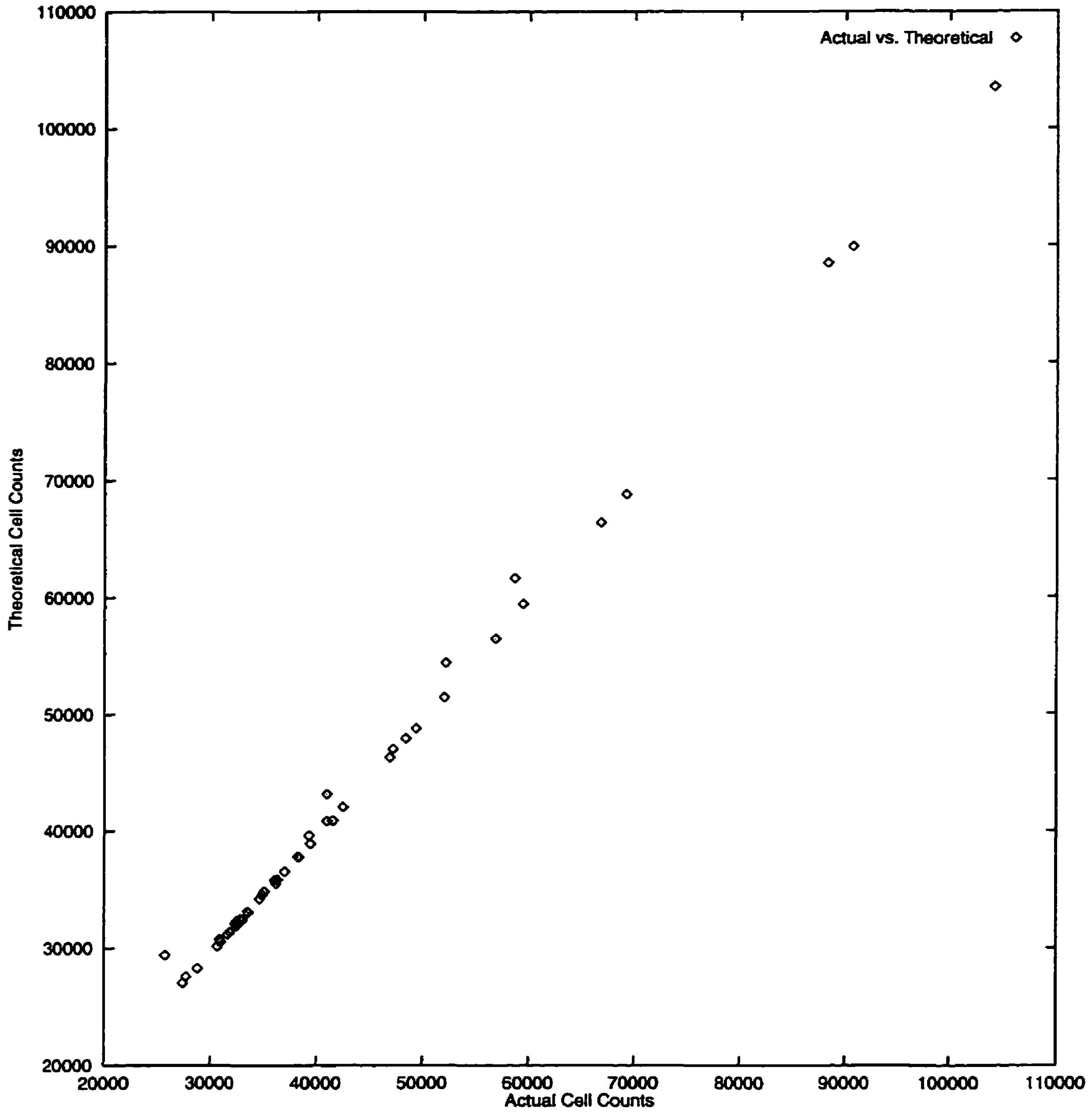

Figure 4.3: Theoretical vs. actual cell transmission rates.

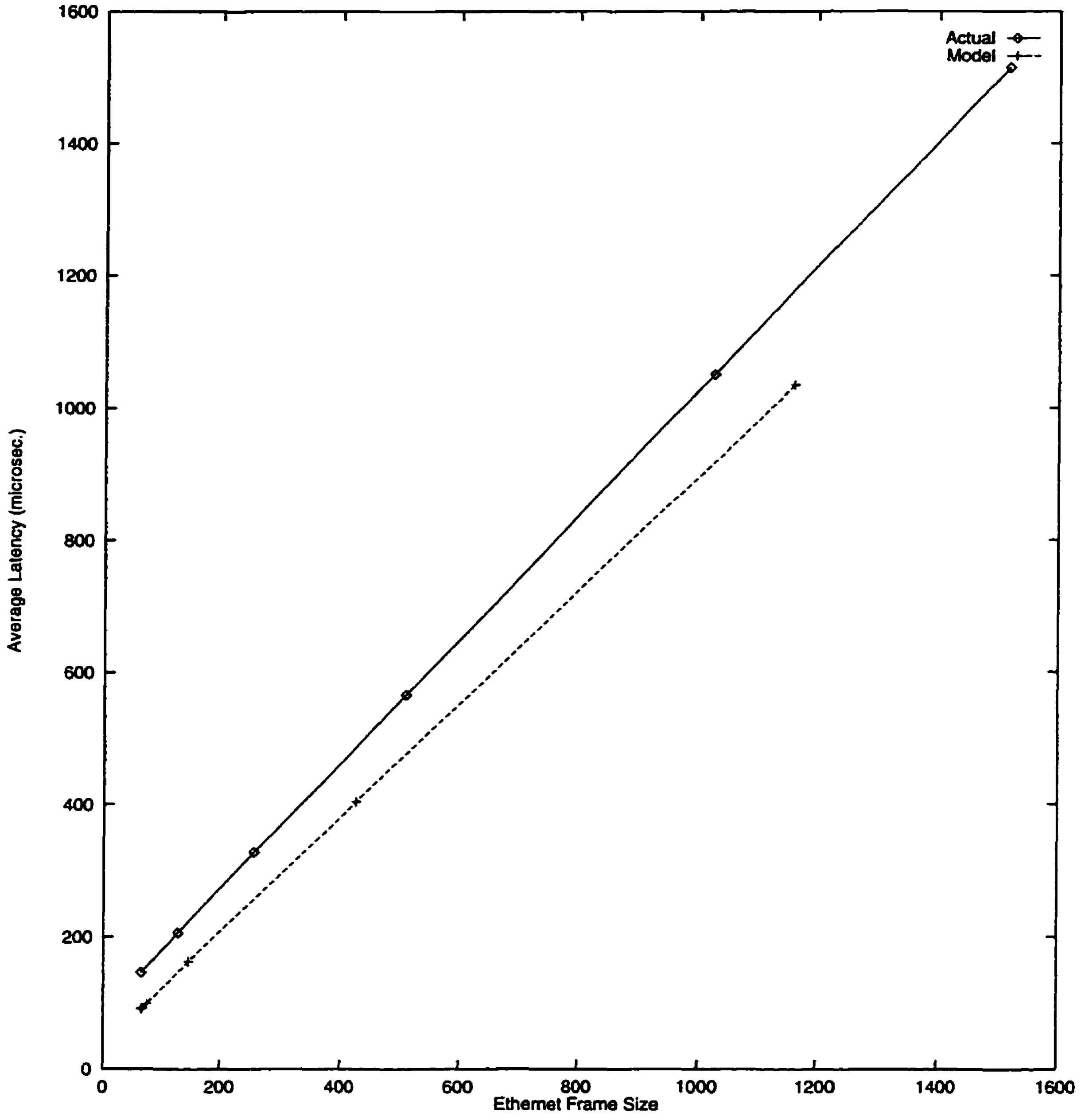

Figure 4.4: Model vs. actual ATM latency.

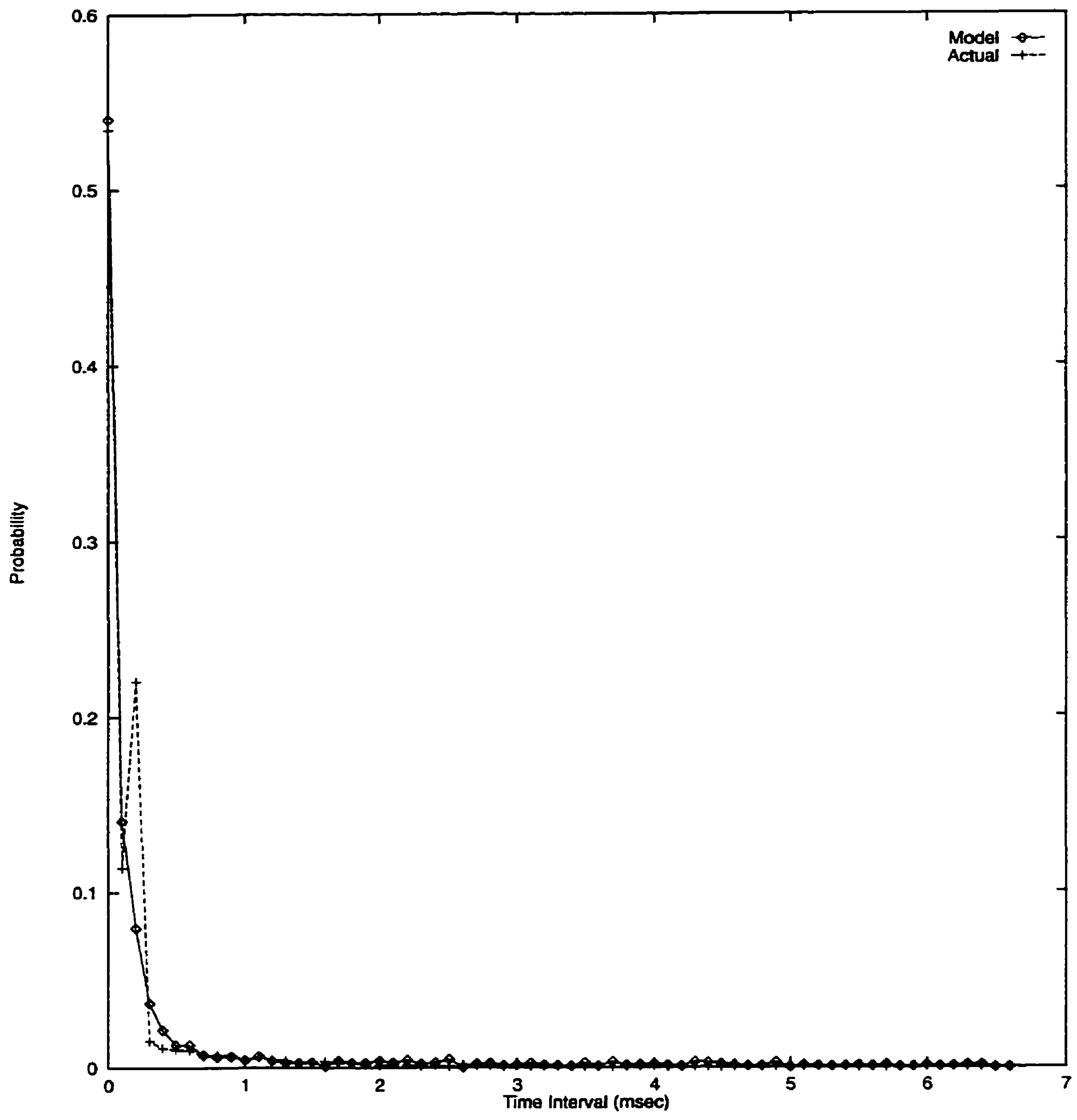

Figure 4.5: TELNET "think time": actual and exponential model.

compares the same FTP packet size distribution and the model using six packet types. The model distribution reflects the bi-modal nature of the real FTP distribution – most of the packets are either at the minimum or the maximum size for an Ethernet. (Table 5.1 shows the results of the χ^2 goodness-of-fit test between the distributions in Figures 4.6 and 4.7).

A method of improving the accuracy of the model packet size distribution is described in Chapter 5.

4.4.2 Protocol Results

This section presents some simulation results from the model. Figures 4.8 and 4.9 show the average load placed on the Ethernet in Figure 2.4, on a per-protocol basis, by number of concurrent user sessions. The protocols ending in 'U' (i.e. WWWU, etc.) are the versions with concatenated TCP connections as discussed in Section 4.1.2.

Figures 4.10 and 4.11 show the ATM send and receive data rates for the same data traces. Figures 4.12 and 4.13 show some of the various types of information that can be obtained from the model. Figure 4.12 compares the maximum ATM burst rate for two protocols, using a 10 msec window. Figure 4.13 shows part of the ATM cell inter-arrival distribution for 64 concurrent TELNET users. Figure 4.14 shows the one-way delay per packet caused by network load (i.e. delays caused by other packets in the system, but not including the average latency caused by hardware). Figure 4.15 compares the network delay effect of FTP and TELNET on a single CU-SeeME session.

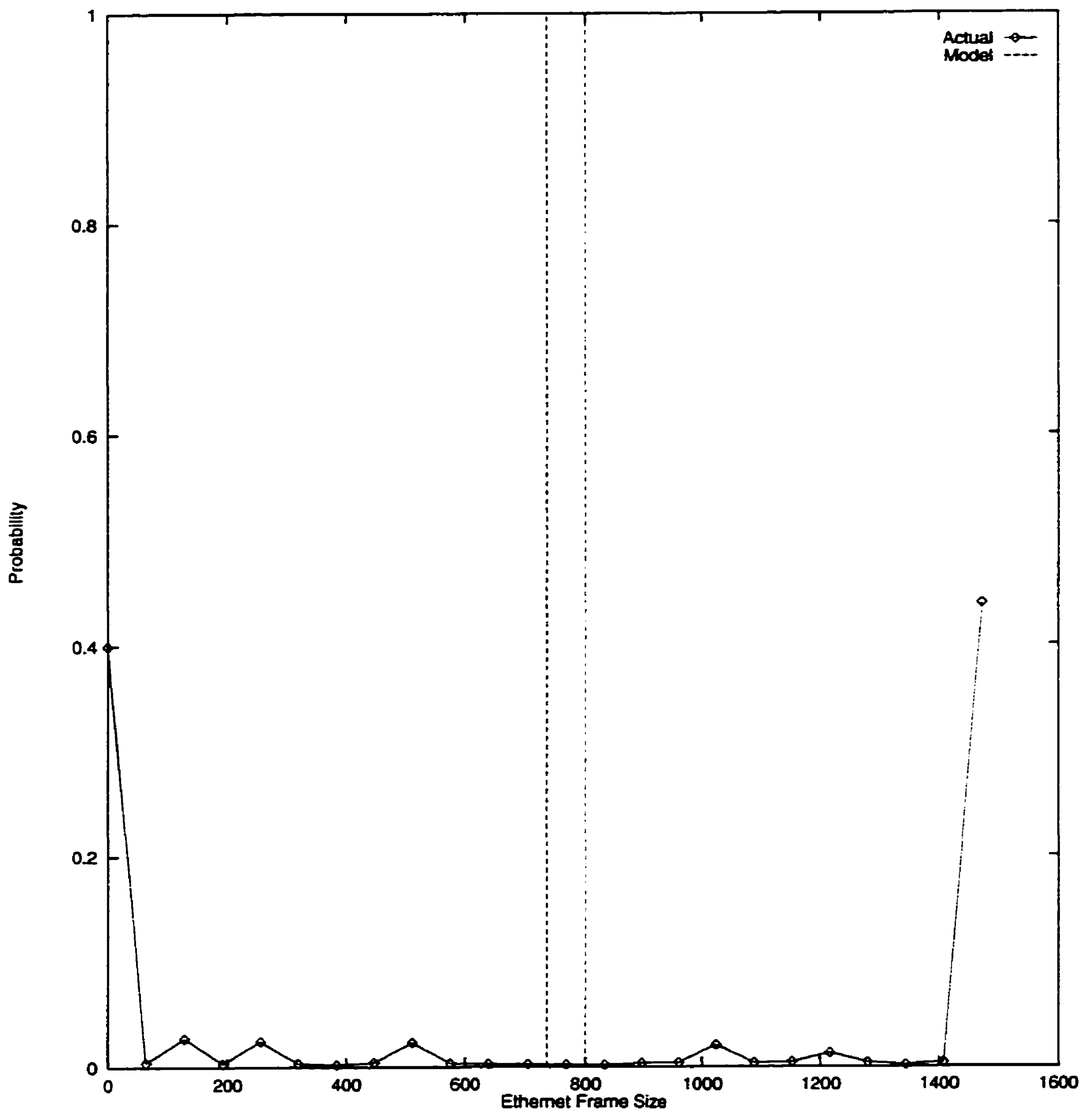

Figure 4.6: Packet size distribution for FTP protocol (single value).

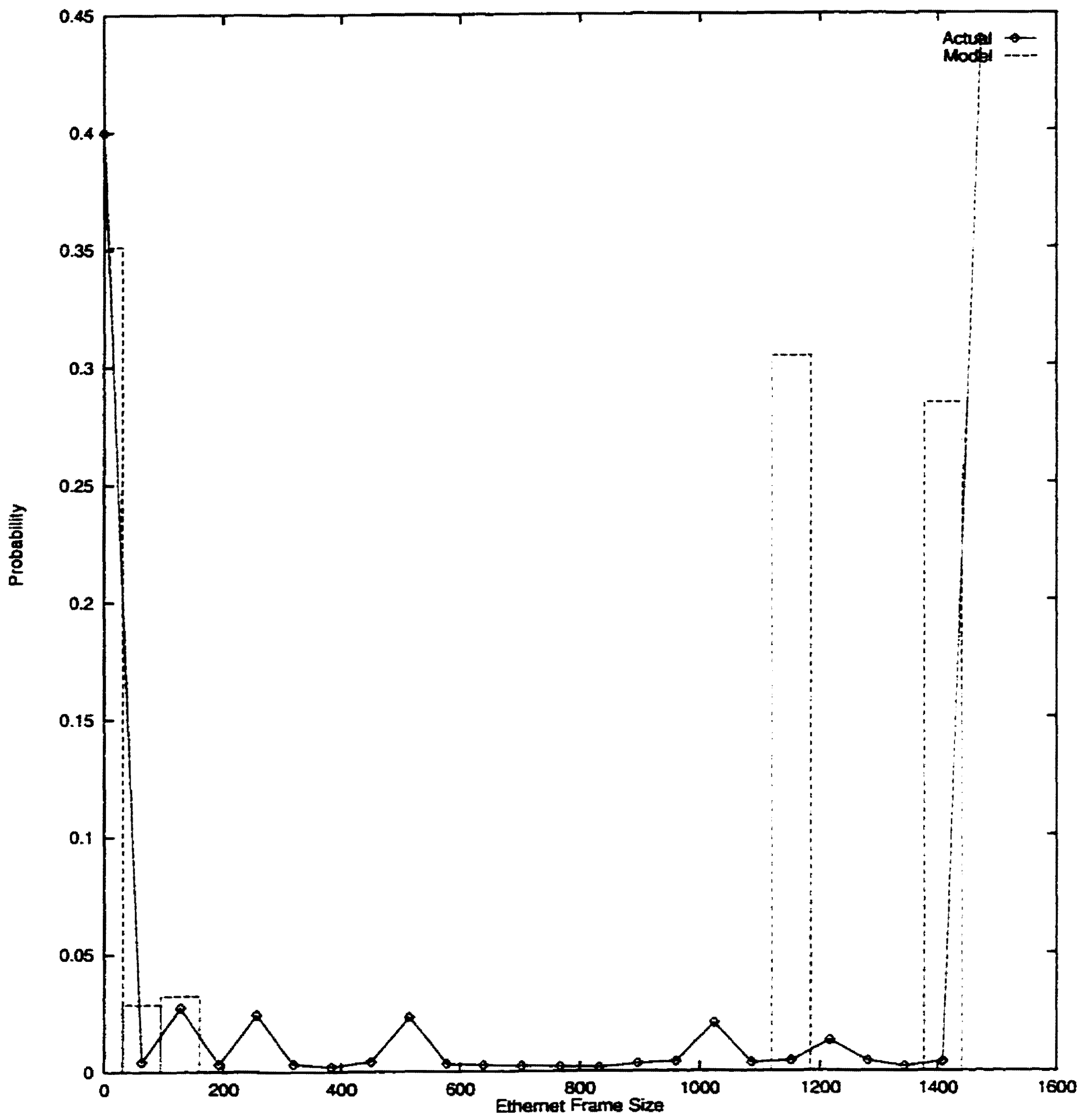

Figure 4.7: Packet size distribution for FTP protocol (multiple packet types).

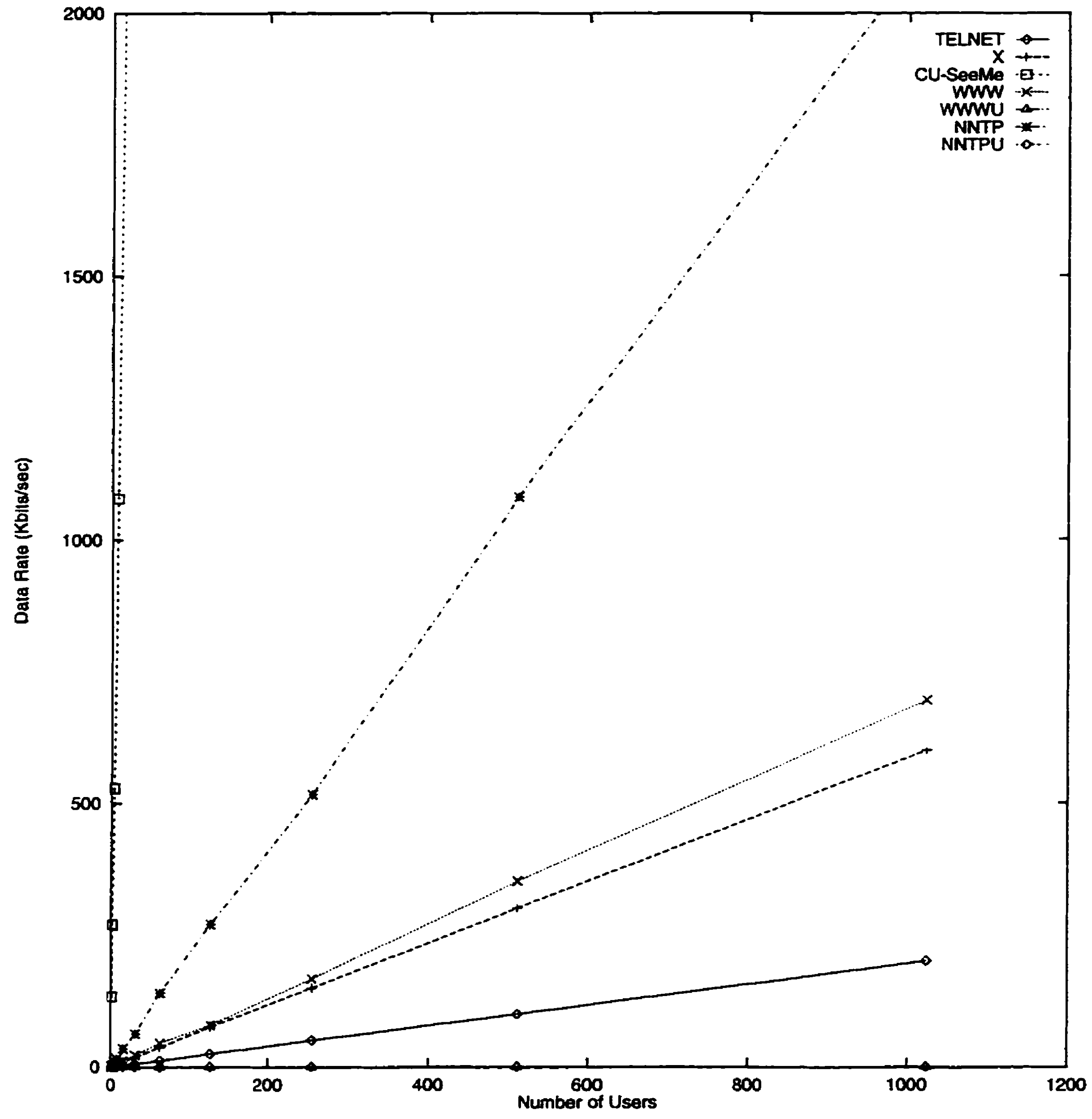

Figure 4.8: Network load by protocol - data rate.

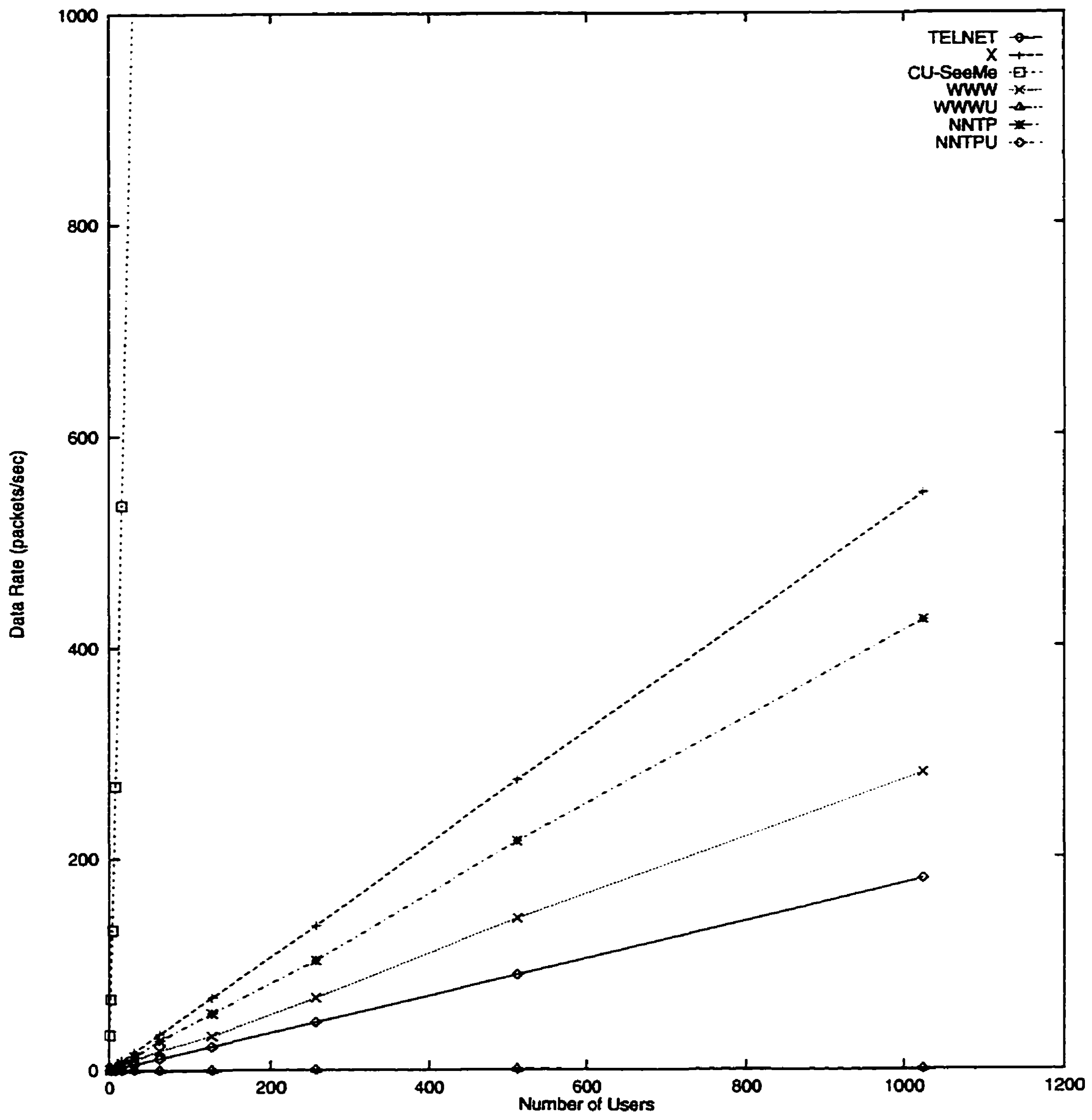

Figure 4.9: Network load by protocol - frame rate.

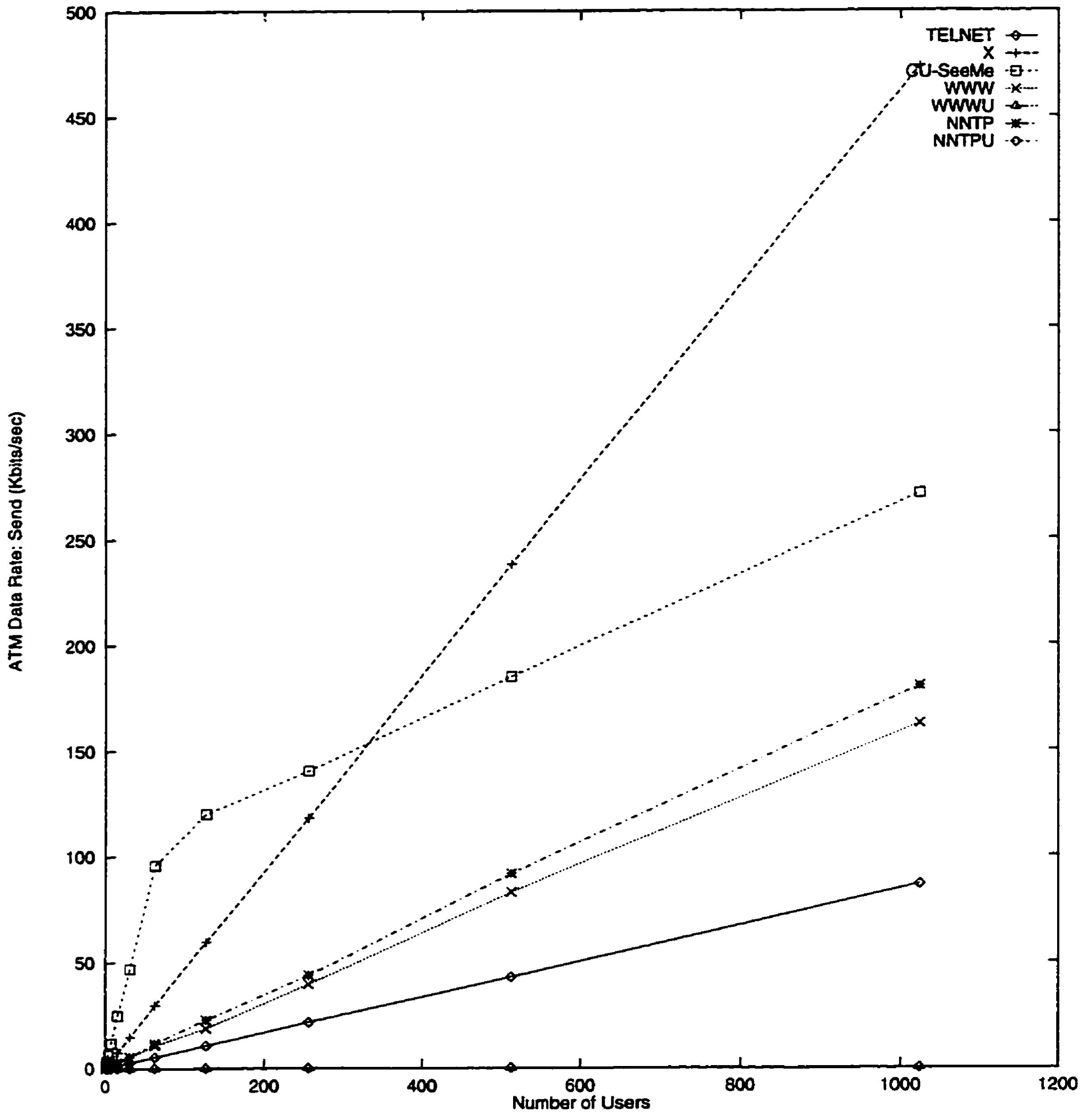

Figure 4.10: Network load by protocol - ATM data rate (send).

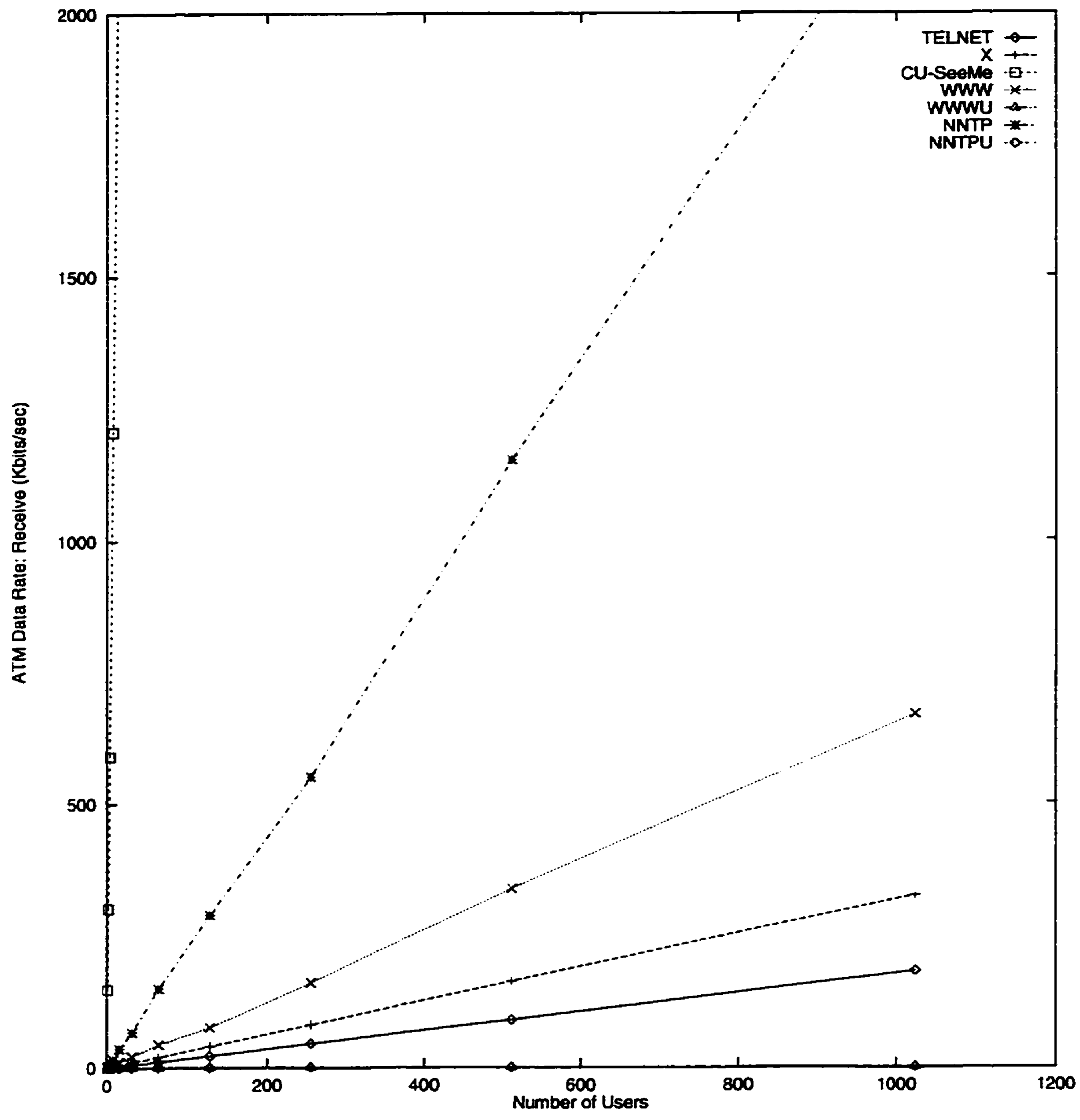

Figure 4.11: Network load by protocol - ATM data rate (receive).

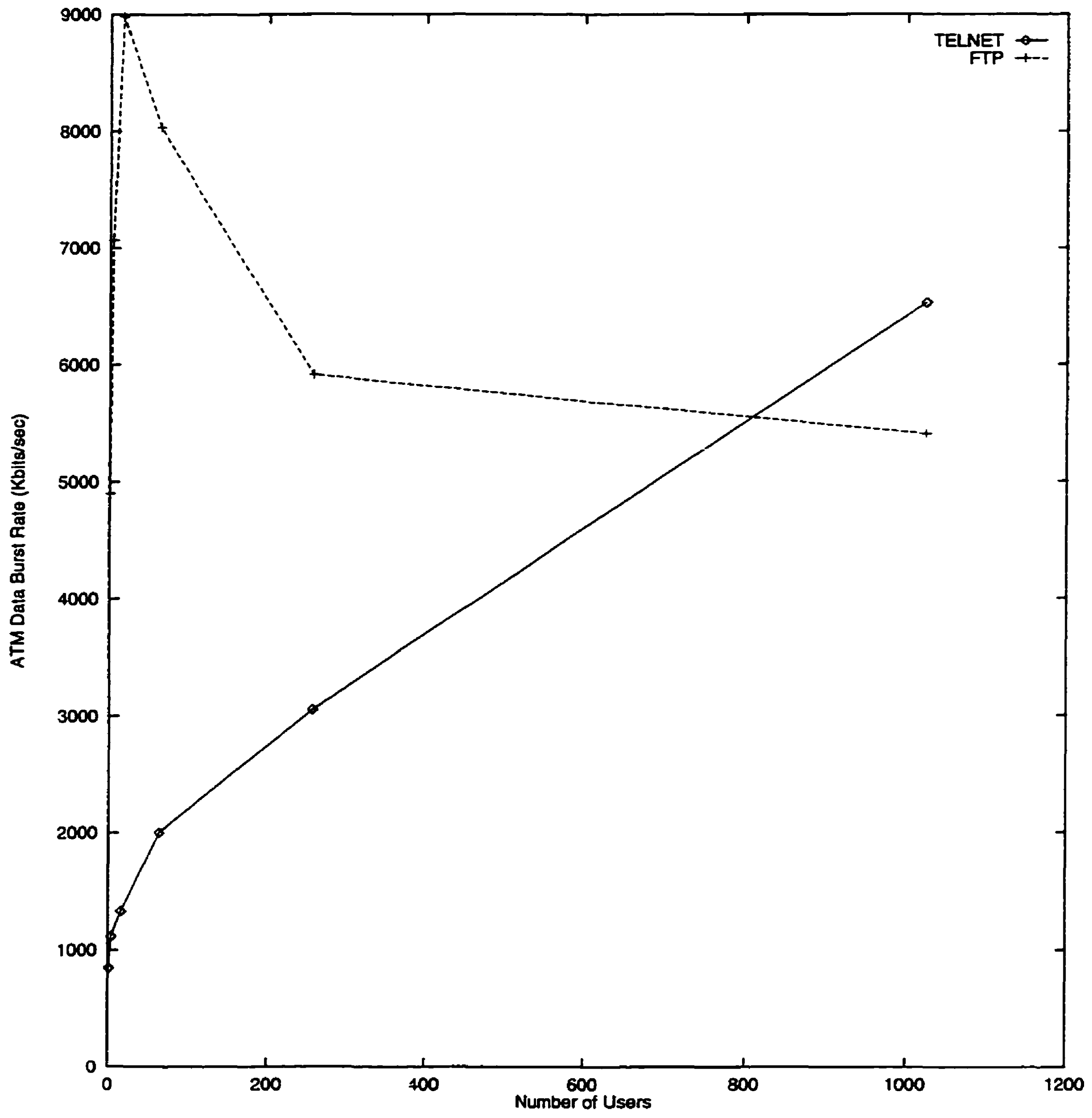

Figure 4.12: Network load by protocol - maximum ATM data rate per user (0.01 ms burst).

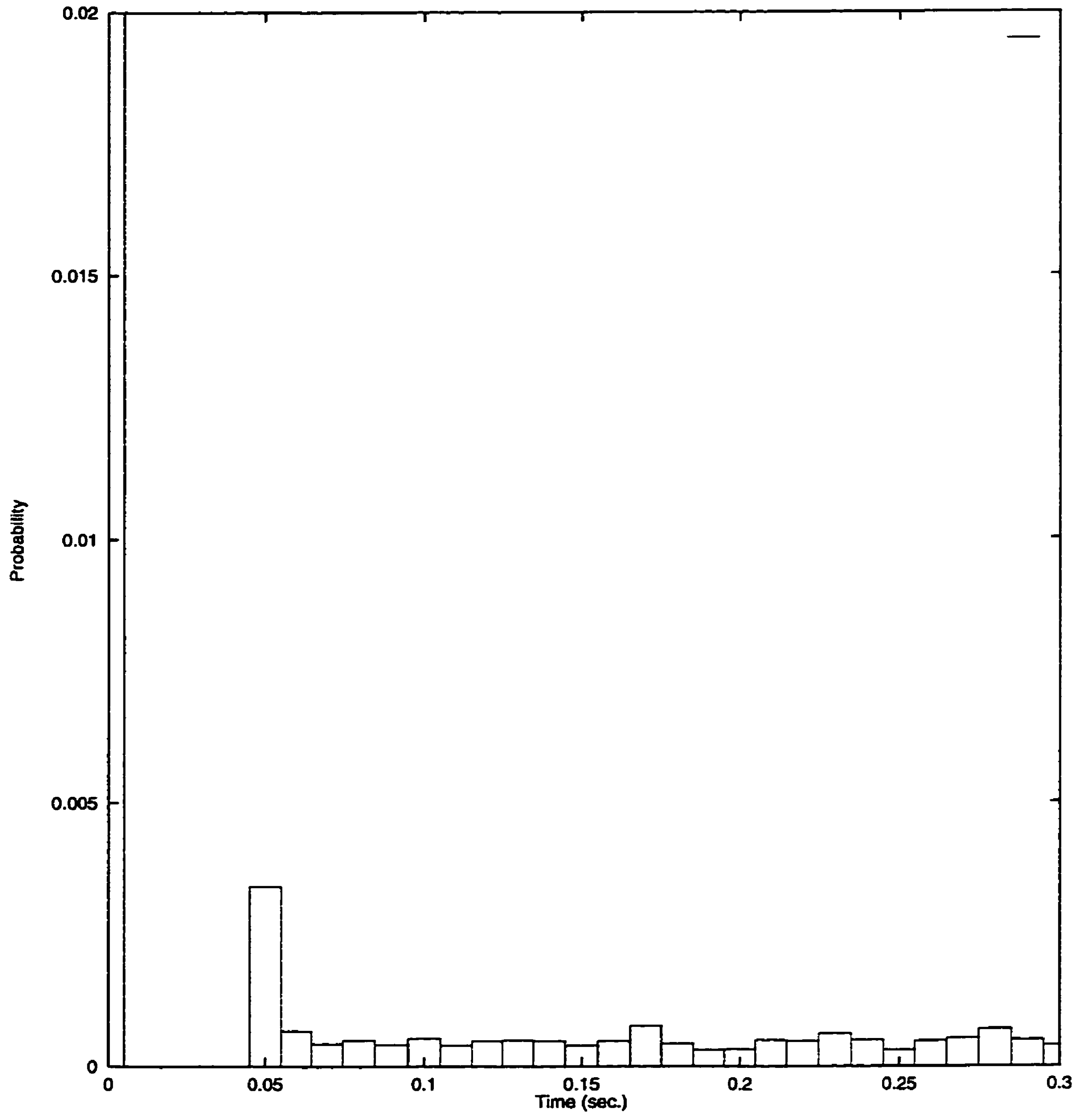

Figure 4.13: ATM cell inter-arrival time distribution.

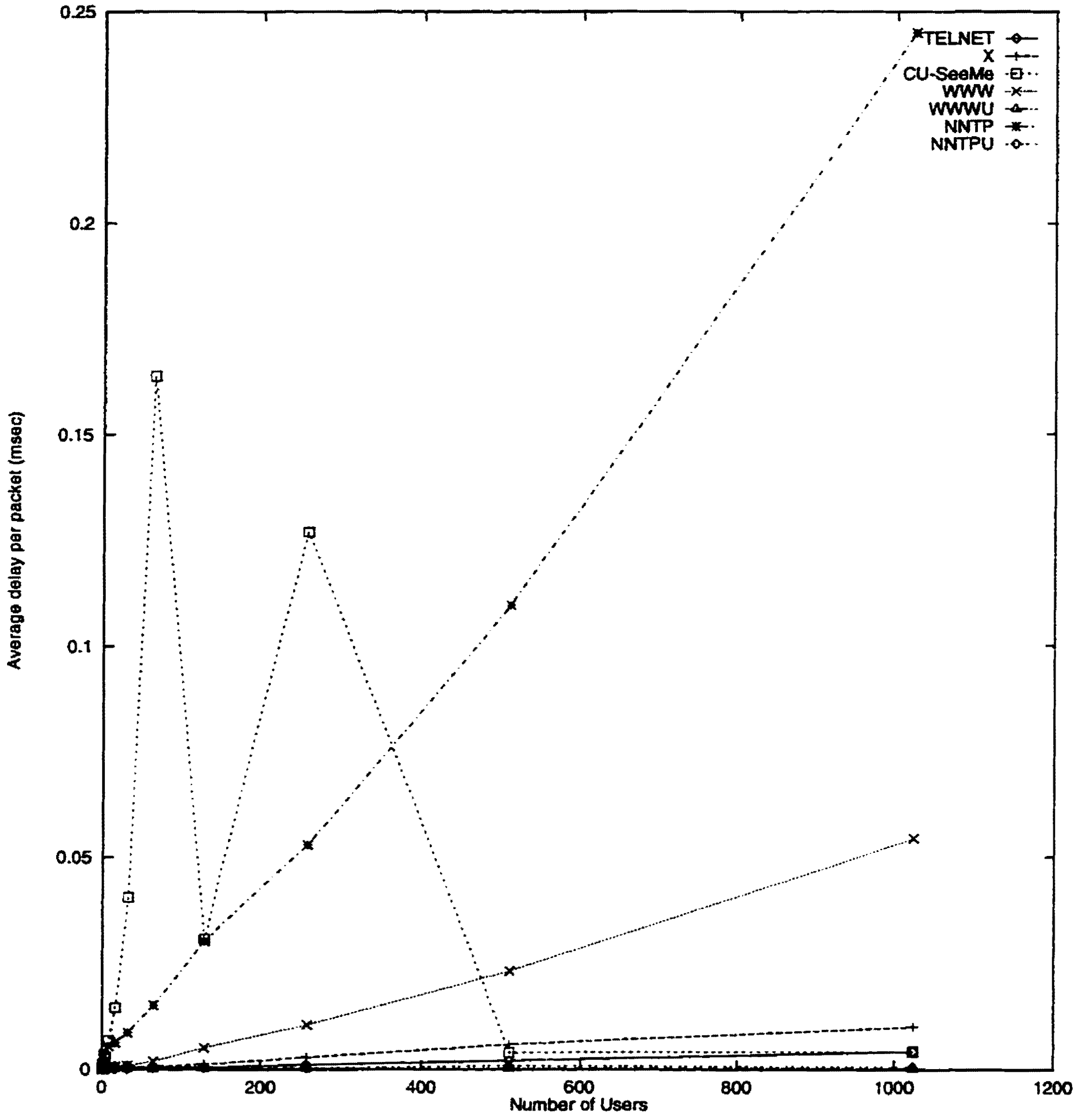

Figure 4.14: Load-induced network delay per packet.

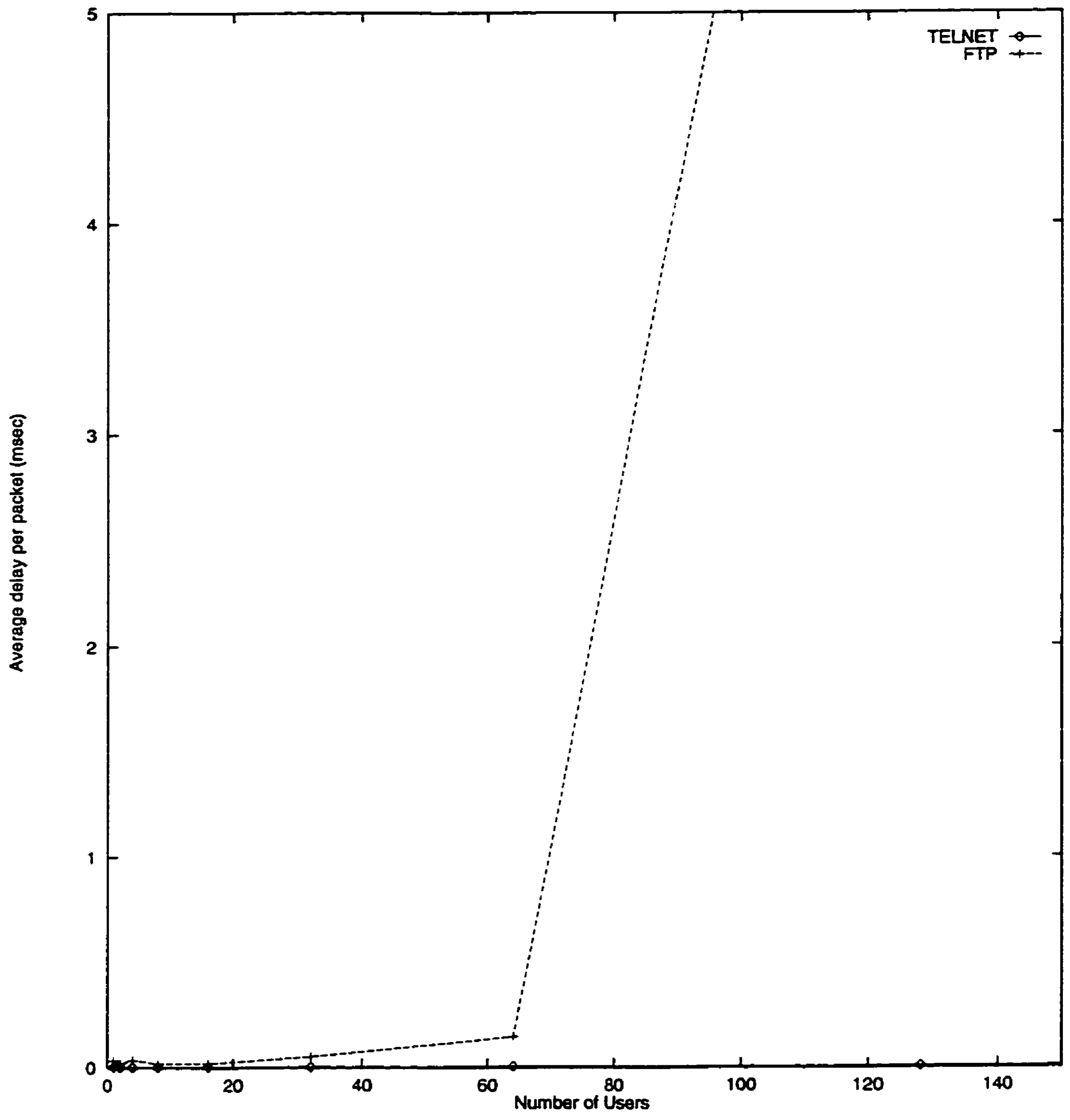

Figure 4.15: Load delay per packet - CU-SeeMe with TELNET/FTP.

4.5 Discussion

The network load graphs (Figures 4.8 and 4.9) show significant differences between the various protocols. CU-SeeMe appears to have a major impact on network performance. The versions of the protocols representing multiple TCP sessions appear to have little impact. The lower data rates are caused by the user's thinking time between actions.

Generally, the two sides of the ATM link show quite different data rates (Figures 4.10 and 4.11). The curved behaviour of the CU-SeeMe graph is caused by the protocol saturating the Ethernet. The ATM burst rate (Figure 4.12) shows that even a low-impact protocol like TELNET can create short bursts of cells at high speeds far above the average behaviour. The ATM cell inter-arrival distribution (Figure 4.13) shows a large spike (representing over 50% of the distribution), a gap, and then a long even tail. The initial spike is caused by the fast cell generation rate at the SAR layer, while the tail represents the longer gaps between packets at the Ethernet layer.

The load delay graph (Figure 4.14) shows odd behaviour of CU-SeeMe caused by a saturated Ethernet, while the similar graph showing the effect of two protocols on CU-SeeME (Figure 4.15) demonstrates that load delay could be used to find the network congestion point. For example, Figure 4.15 suggests that the number of users in the system should be less than 64 (the point at which the delay graph climbs steeply). However, the linearity of most of the graphs suggests the a more detailed model of Ethernet collision behaviour may be required before more accurate load decisions can be made.

The next chapter describes some extensions to the model - relatively simple modifications that expand the range of network configurations the model can simulate.

Chapter 5

Extensions and Discussion

One of the goals in the design of the model presented in this thesis was to ensure that new network configurations could be modeled with relatively simple modifications to the basic structure. This chapter describes two such modifications.

5.1 Extensions to the Model

This section describes two extensions to the basic model to simulate alternate network design situations. The first is a small change to the actual model to describe the effect of multiple Ethernets instead of the single Ethernet used in the previous chapter. The second extension doesn't actually change the model, but rather exploits the model's ability to handle multiple application-level protocols to provide a more accurate representation of the packet size distribution of a single application-level protocol.

5.1.1 Multiple Ethernets

The basic configuration of the model (Figure 2.4) has one source Ethernet and one destination Ethernet. In many cases, however, a single emulated LAN will involve multiple Ethernets at one or both sides of the ATM link (Figure 5.1). As a first step in modeling this configuration, we can simply change the number of control tokens at the Ethernet level model in place **AE_IDLE**. This allows multiple simultaneous firings of the Ethernet transition, up to the number of tokens in **AE_IDLE**, thus simulating multiple Ethernets.

The effect of spreading 16 FTP sessions over multiple Ethernets is shown in Figure 5.2. As the number of Ethernets increases, the average load per network drops. The ATM load starts at an initial rate where the Ethernet is the bottleneck and rises to a new equilibrium value. The fact that the ATM data rate doesn't rise further indicates that there is a new limiting factor in the system – either the protocol itself or the ATM-to-Ethernet bridge.

5.1.2 Improving Packet Size Distribution

The packet size distributions in Figures 4.6 and 4.7 show that the use of six packet types gives a much better approximation of the actual FTP packet size distribution. It is possible to improve this approximation even more, however, by using multiple FTP sessions. Essentially, we create two versions of the FTP protocol with the same timing parameters, but with different mean packet sizes. Figure 5.3 shows the revised packet

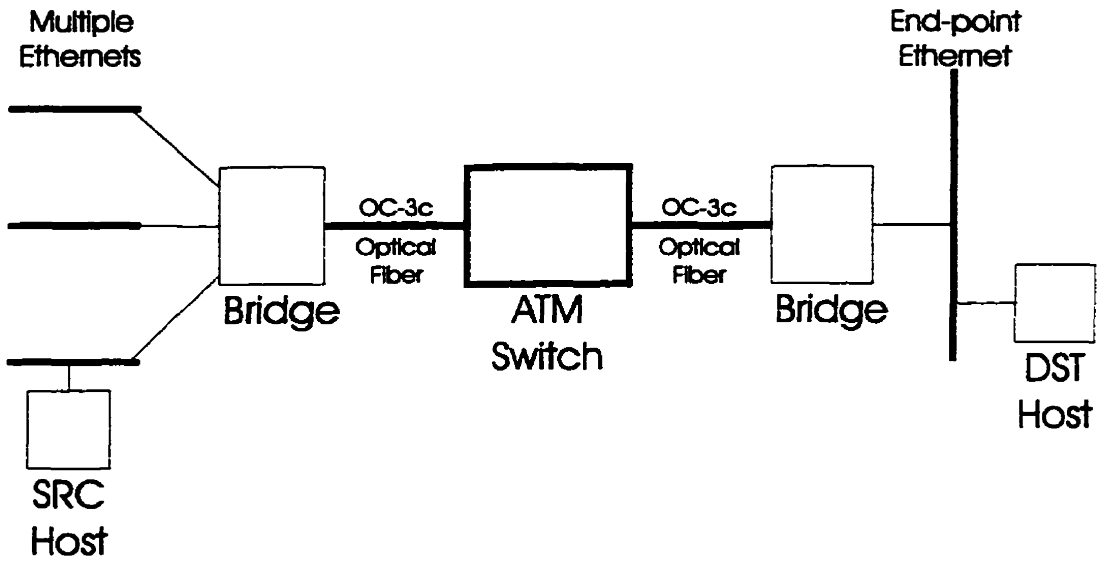

Figure 5.1: LAN Emulation configuration with multiple Ethernets.

size distribution created by taking one of the packet types, splitting it in two at the median and using two "FTP" protocols, each set at the mean of one half of the data for that type.

Table 5.1 shows the results of the χ^2 goodness-of-fit test for a single average (Figure 4.6), the 'six packet type' distribution shown in Figure 4.7, and the revised distribution in Figure 5.3. As long as the number of sessions for each "protocol" are equal, the average packet size distribution will be as shown in Figure 5.3. This procedure increases model complexity by doubling the number of occurrences, however, so there is a balance point at which the gain of an improved distribution is offset by the increased model size.

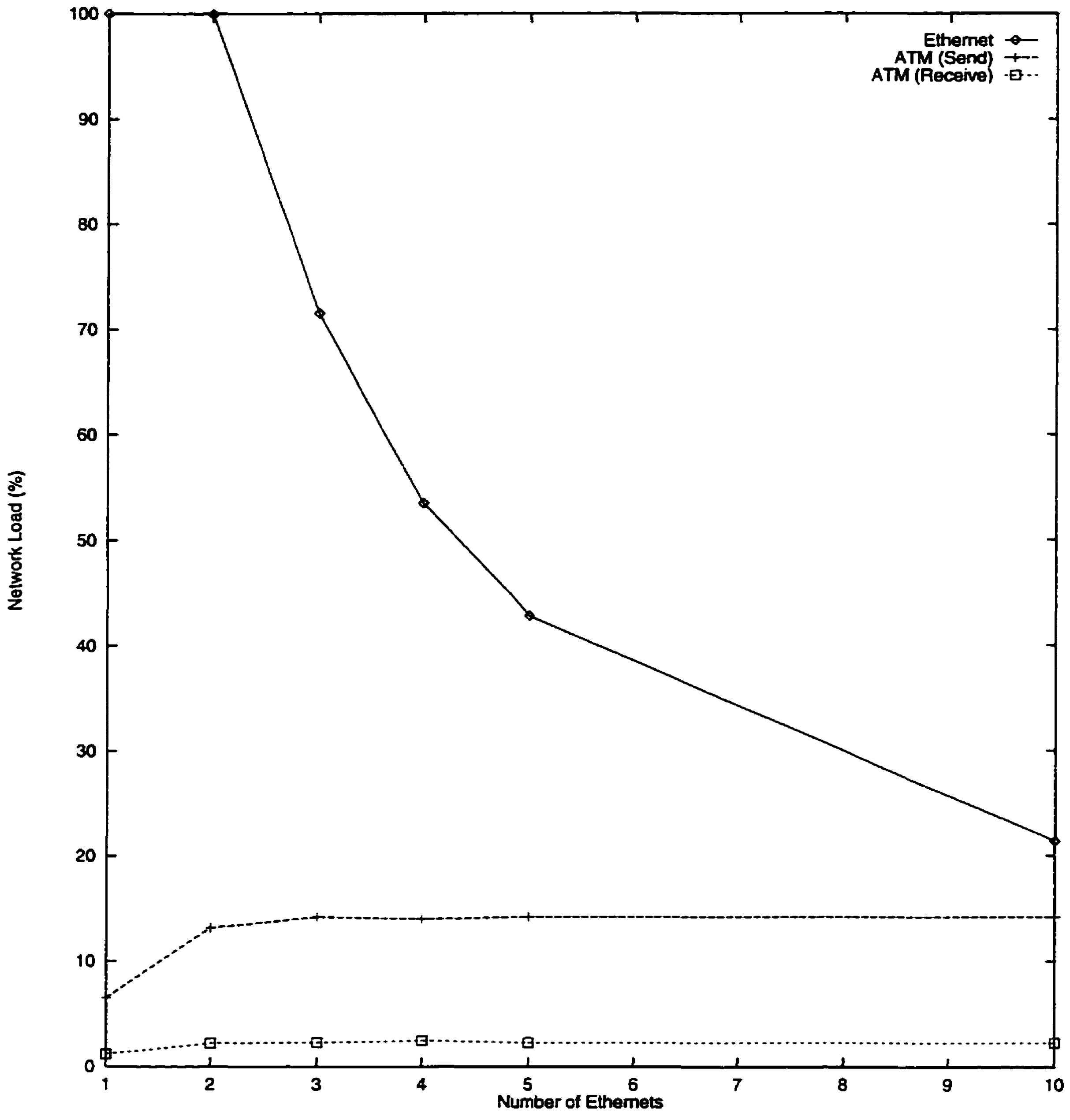

Figure 5.2: Effect on Ethernet and ATM load with multiple Ethernets.

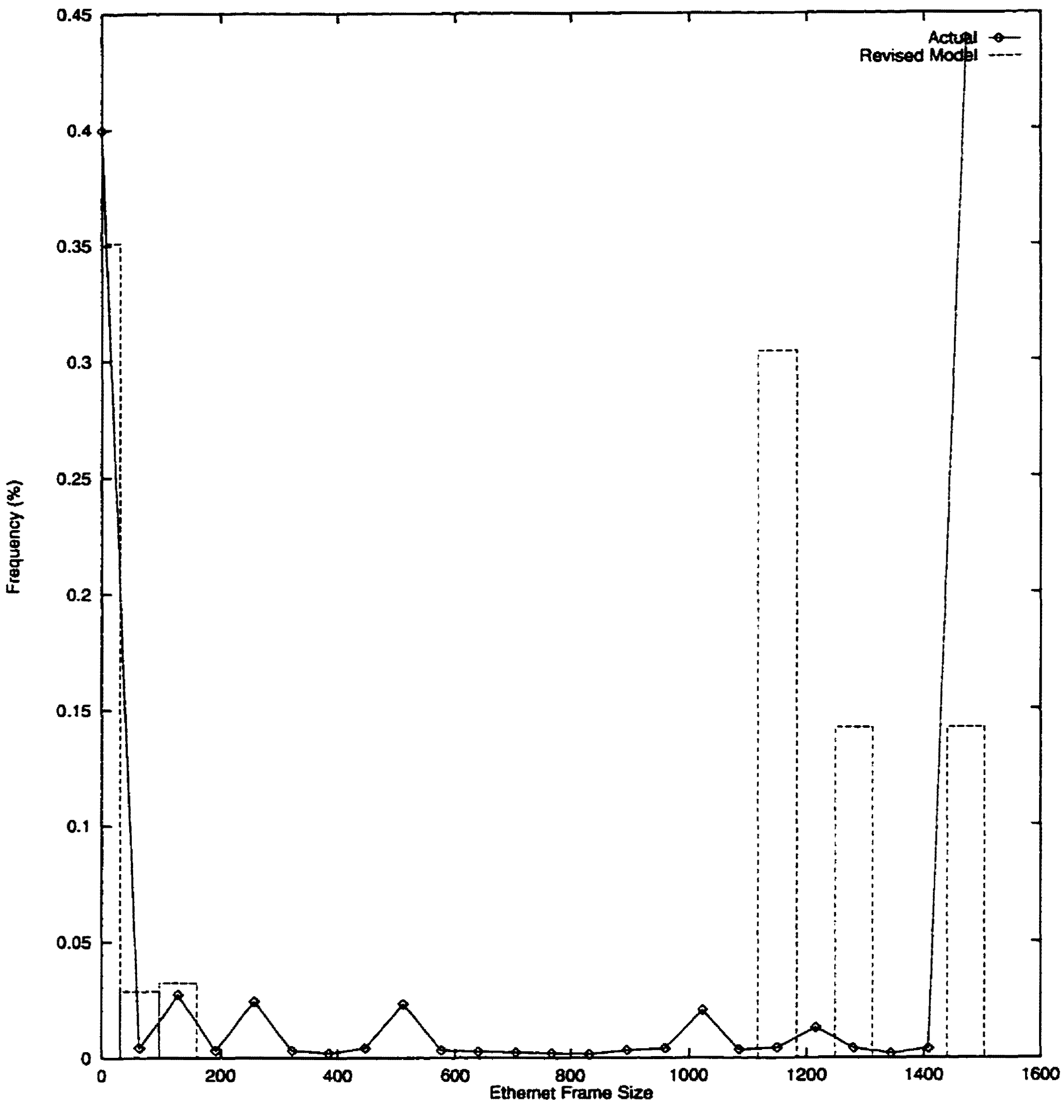

Figure 5.3: Revised packet size distribution for FTP protocol.

	Single Average Value	Single Protocol	Double Protocol
χ^2	493.3	41.2	25.7

Table 5.1: χ^2 values for FTP packet size distribution.

5.2 Discussion

The examples in this chapter are illustrations of the flexibility in the model. It is possible to re-combine various modules to represent other types of configurations. For example, we could add multiple bridges feeding the same switch, with token colours separating the different ATM virtual circuits. Another possibility would be to examine the configuration shown in Figure 2.5 which adds a router between the two bridges. This would require four more SAR sections (since cells would have to be combined back into packets for level 3 routing to occur) and a new module representing the behaviour of the router. Software support for this type of modular model design exists [45].

The FTP example shows the possiblities of using multiple protocols in the model. In addition to packet size distribution, it would be possible to create a protocol with more complex timing behaviour by combining multiple versions with slightly different parameters to obtain the desired results.

Chapter 6

Conclusions

The goal of the research presented in this thesis was to create a performance model of an ATM LAN that would provide a reasonable representation of system behaviour. As well, the model should provide a flexible framework for more detailed investigation of various sections of the model without requiring drastic changes throughout the entire structure. The preference was to design a model with clearly defined layers, and have the interface between the layers closely tied to the physical model. This would allow the re-design of a module to stay localized within that section. Most of the current models of ATM have tended to focus on one section of the system – the switch, for example, or policing mechanisms. Here the goal was to create a more generic model that could absorb the results of some of these detailed studies.

Since the model had to represent both hardware and software behaviour, a modeling paradigm was required that could easily represent both. A variety of modelling methods have been used for similar work (see Section 2.4), such as queueing models, or analytical work based on Markovian models. It was felt that these methods did not capture the synchronization inherent in network protocols, and were not capable of providing the variety of functionality required. Therefore, it was decided to base the model on Petri nets, which have been used to model a wide variety of concurrent systems. This proven flexibility was felt to be a significant advantage in a model intended to allow easy re-design. Petri nets naturally represent concurrency and synchronization, there are a variety of extensions to basic Petri nets which allow the modeler to choose the functionality required, and a number of software packages are available to analyze the model.

To provide the modularity required of the model, the ATM LAN system was divided into layers. At the interfaces between the layers, the tokens in the Petri net model represent actual Ethernet frames or ATM cells. To allow the representation of multiple high level network protocols, an extension to basic Petri nets called *coloured timed Petri nets* was used. The use of coloured tokens allows the different protocols to operate independently, yet use the same abstracted net model. The results in Chapter 4 demonstrate that the model can accurately represent the packet size distribution and data rates of a variety of quite distinct protocols. Furthermore, since the tokens in the model represent actual packets or cells, the model is capable of producing the equivalent of a network trace, which can be analyzed in the same way as real network data to gain other statistics of interest.

The flexibility of the model is demonstrated by the extensions discussed in Chapter 5. The modeller can easily create alternate configurations by removing or altering specific modules, or making multiple copies. For example, the model can be modified to study the direct interaction of TCP/IP with ATM, or the introduction of routing functionality in the bridge devices. The ability to represent multiple distinct protocols can be extended past the basic concept to provide greater detail or study the interactions between protocols, such as how other protocols introduce specific timing delays, and where those delays occur.

It can therefore be concluded that the proposed model satisfies the four major objectives indicated in the introduction:

- the model provides good conformance to protocol behaviour under normal load,
- the model directly represents network traffic in terms of users and protocols,
- the model can produce a variety of information about the modeled system,
- the model structure is easily modified to study different network configurations.

The model as presented in Chapter 3 has some limitations. The model becomes less accurate as it passes the point of congestion, although some indication of where this point lies is discernible from the model results. To proceed past this point will require a more complex Ethernet model that includes more of the collision behaviour of the medium, as well as a protocol structure that models re-transmissions or lost packets, although this may require breaking down the strict interfaces between layers. However, the model is accurate under current performance levels of Memorial's campus backbone.

As further detail is added, 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. A single place as defined in this model cannot provide that, so a special "queue" entity would simplify the addition of detail in this area. As well, current work on hierarchical net structures would be applicable to this model. Such structures would help the introduction of a lost packet mechanism, since such a mechanism may require a module that all others would have to be able to access. Arc weights that can vary depending on other factors may help provide a more flexible windowing system in the TCP layer.

Bibliography

[1] ALLES, A. ATM Internetworking. Tech. rep., Cisco Systems Inc., 1995.

[2] ARMITAGE, G. J., AND ADAMS, K. M. How inefficient is IP over ATM anyway? *IEEE Network 9*, 1 (Jan. 1995), 18–26.

[3] BERTHELOT, G., AND TERRAT, R. Petri nets theory for the correctness of protocols. In *Protocol Specification, Testing, and Verification: Proceedings of the IFIP WG 6.1 Second International Workshop* (1982), C. Sunshine, Ed., North-Holland, pp. 325–342.

[4] BRADNER, S. Bradner Report - Catalyst 5000 switch. Tech. rep., Cisco Systems Inc., Sept. 1995.

[5] BRADNER, S., AND MCQUAID, J. Benchmarking methodology for network interconnect devices. Tech. rep., Internet Engineering Task Force - Benchmarking Methodology Working Group, 1994.

[6] BUTTO, M., CAVALLERO, E., AND TONIETTI, A. Effectiveness of the "leaky bucket" policing mechanism in ATM networks. *IEEE Journal on Selected Areas in Communications 9*, 3 (Apr. 1991), 335–342.

[7] CAVANAUGH, J. D. Protocol overhead in IP/ATM networks. Tech. rep., Minnesota Supercomputer Center, Inc., Aug. 1994.

[8] CHAI, A., AND GHOSH, S. Modeling and distributed simulation of a broadband-ISDN network. *COMPUTER 26*, 9 (Sept. 1993), 37–51.

[9] CHEN, C. Y. R., MAKHOUL, G. A., AND MELIKSETIAN, D. S. A queueing analysis of the performance of DQDB. *IEEE-ACM Transactions on Networking 3*, 6 (Dec. 1995), 872–881.

[10] CLARK, D. D., JACOBSEN, V., ROMKEY, J., AND SALWEN, H. An analysis of TCP processing overhead. *IEEE Communications Magazine 27*, 6 (June 1989), 23–29.

[11] COMER, D. E. *Internetworking with TCP/IP*, 2nd ed. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1991.

[12] CONTI, M., GREGORI, E., AND LENZINI, L. Metropolitan area networks (MAN's): Protocols, modeling and peerformance evaluation. In *Performance Evaluation of Computer and Communication Systems*, L. Donatiello and R. Nelson, Eds., vol. 729 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993, pp. 81–120.

[13] DE PRYCKER, M. *Asynchronous Transfer Mode - Solution for Broadband ISDN*, 2nd ed. Ellis Horwood, 1993.

[14] DESCLOUX, A. Stochastic models for ATM switching networks. *IEEE Journal on Selected Areas in Communications 9*, 3 (Apr. 1991), 450–457.

[15] DIAZ, M. Modelling and analysis of communication and cooperation protocols using Petri net models. In *Protocol Specification, Testing, and Verification: Proceedings of the IFIP WG 6.1 Second International Workshop* (1982), C. Sunshine, Ed., North-Holland, pp. 465–510.

[16] DITTMAN, L., JACOBSEN, S. B., AND MOTH, K. Flow enforcement algorithms for ATM networks. *IEEE Journal on Selected Areas in Communications 9*, 3 (Apr. 1991), 343–350.

[17] DUGAN, J. B., TRIVEDI, K. S., GEIST, R. M., AND NICOLA, V. F. Extended stochastic Petri nets - applications and analysis. In *Performance '84*, E. Gelenbe, Ed. North-Holland, 1984.

[18] FLANAGAN, W. A. *ATM User's Guide*. Flatiron Publishing, Inc., 1994.

[19] FOUQUET, Y. A., SCHNEEMAN, R. D., CYPHER, D. E., AND MINK, A. ATM performance measurement: Throughput bottlenecks and technology barriers. Tech. rep., National Institute of Standards and Technology, Gaithersburg, M.D., 1994.

[20] GARBE, W. *Visual SimNet Manual*, 1.31 ed., 1995.

[21] GREINER, S., BOLCH, G., PULIAFITO, A., AND TRIVEDI, K. S. Performance evaluation of dynamic priority operating systems. In *Petri Nets and Performance Models* (1995), Center for Advanced Computing and Communications, IEEE Computer Society Press, pp. 241–250.

[22] HSU, S., AND ILYAS, M. A simulation study of bursty data traffic with hybrid source smoothing in an ATM node. *Computers and Industrial Engineering 25*, 1-4 (1993), 151–154.

[23] HUBER, P., JENSEN, K., AND SHAPIRO, R. M. Hierarchies in coloured Petri nets. In *Advances in Petri Nets 1990: Lecture Notes in Computer Science vol 483*, G. Rozenberg, Ed. Springer-Verlag, 1991, pp. 313–341.

[24] HUNT, R. Frame relay - protocols, architecture, operation and performance. *Computer Communications 19*, 9-10 (Aug. 1996), 830–847.

[25] HUNTSBERGER, D. V., AND BILLINGSLEY, P. *Elements of Statistical Inference*, 5th ed. Allyn and Bacon, Inc., Boston, 1981.

[26] JACOBSON, V., LERES, C., AND MCCANNE, S. *TCPDUMP man page (version 2.0)*. Lawrence Berkely Laboratory, 1991.

[27] JENSEN, K. *Coloured Petri Nets*, vol. 1. Springer-Verlag, 1992.

[28] JOSHI, S. P. High-performance networks: A focus on the fiber distributed data interface (FDDI) standard. *IEEE Micro 6*, 3 (June 1986), 8–14.

[29] KAY, J., AND PASQUALE, J. The importance of non-data touching processing overheads in TCP/IP. *SIGCOMM 93 23*, 9 (1993), 259–268.

[30] KOBAYASHI, H. *Modeling and Analysis: An Introduction to System Performance Methodology*. Addison-Wesley Publishing Company, Inc., 1978.

[31] LAN EMULATION SUB-WORKING GROUP, A. F. LAN emulation over ATM - version 1.0. Tech. Rep. af-lane-0021.000, ATM Forum, 1995.

[32] LIN, A. Y. M., AND SILVESTER, J. A. Queuing analysis of an ATM switch with multichannel transmission. *Performance Evaluation Review 18*, 1 (May 1990), 96–105.

[33] MADRON, T. W. *Local Area Network - The Next Generation*, 2nd ed. John Wiley and Sons, Inc., New York, 1990.

[34] MARSAN, A. M., CONTE, G., AND BALBO, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems 2*, 2 (1984), 93–122.

[35] MARSAN, M. A., CHIOLA, G., AND FUMAGALLI, A. An accurate performance model of a CSMA/CD BUS LAN. *Advances in Petri Nets 1987 LNCS 266* (1987), 146–161.

[36] MARSAN, M. A., CIGNO, R. L., MUNAFÒ, M., AND TONIETTI, A. Simulation of ATM computer networks with CLASS. In *Computer Performance Evaluation: Modelling Techniques and Tools. 7th International Conference*, G. Haring and G. Kotsis, Eds. Springer-Verlag, Vienna, Austria, May 3-6, 1994, 1994, pp. 159–179.

[37] METCALFE, R. M., AND BOGGS, D. R. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM 19*, 7 (July 1976), 395–404.

[38] MOLDEKLEV, K., AND GUNNINGBERG, P. How a large ATM MTU causes deadlocks in TCP data transfers. *IEEE-ACM Transactions on Networking 3*, 4 (Aug. 1995), 409–422.

[39] MOLLE, M., AND WATSON, G. 100Base-T/IEEE 802.12/Packet Switching. *IEEE Communications Magazine 34*, 8 (Aug. 1996), 64–73.

[40] MOLLOY, M. K. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers 31*, 9 (Sept. 1982), 913–917.

[41] MURATA, T. Petri nets: Properties, analysis, and applications. *Proceedings of the IEEE 77*, 4 (Apr. 1989), 541–580.

[42] OHBA, Y., MURATA, M., AND MIYIHARA, H. Analysis of interdeparture processes for bursty traffic in ATM networks. *IEEE Journal on Selected Areas in Communications 9*, 3 (Apr. 1991), 468–476.

[43] ONVURAL, R. O. On performance characteristics of ATM networks. In *SuperComm/ICC '92* (1992), IEEE Press, pp. 1004–1008.

[44] ONVURAL, R. O. *Asynchronous Transfer Mode Networks: Performance Issues.* Artech House, Boston, 1994.

[45] OSWALD, H., ESSER, R., AND MATTMAN, R. An environment for specifying and executing hierarchical Petri nets. In *Proceedings of the 12th International Conference on Software Engineering* (1990), IEEE Press, p. 164.

[46] PERLOFF, M., AND REISS, K. Improvements to TCP performance in high-speed ATM networks. *Communications of the ACM 38*, 2 (Feb. 1995), 91–109.

[47] PETERSON, J. L. *Petri net theory and the modelling of systems.* Prentice-Hall, 1981.

[48] PETR, D. W., FROST, V. S., NEIR, L. A., DEMIRTJIS, A., AND BRAUN, C. Simulation comparison of broadband networking technologies. *SIMULATION 64*, 1 (1995), 42–50.

[49] POSTEL, J. Transmission Control Protocol - DARPA Internet program protocol specification. Tech. Rep. RFC 793, USC/Information Sciences Institute, Sept. 1981.

[50] RATHGEB, E. P. Modeling and performance comparison of policing mechanisms for ATM networks. *IEEE Journal on Selected Areas in Communications 9*, 3 (Apr. 1991), 325–334.

[51] REISER, M. Communication-system models embedded in the OSI reference model, a survey. In *Computer Networking and Performance Evaluation. Proceedings of the IFIP WG 7.3 International Seminar on Computer Networking and Performance Evaluation* (1985), T. Hasegawa, H. Takagi, and Y. Takahashi, Eds., IFIP WG 7.3, North-Holland, pp. 85–111.

[52] REISIG, W. *Petri nets – an introduction.* Springer Verlag, 1985.

[53] ROMANOW, A., AND FLOYD, S. Dynamics of TCP traffic over ATM networks. *IEEE Journal on Selected Areas in Communications 15*, 4 (May 1995), 633–641.

[54] ROSS, S. M. *Introduction to Probability Models*. 3rd ed. Academic Press, Inc., 1985.

[55] STAMOULIS, G. D., ANAGNOSTOU, M. E., AND GEORGANTAS, A. D. Traffic source models for ATM networks: a survey. *Computer Communications 17*, 6 (1994), 428–438.

[56] TANENBAUM, A. S. *Computer Networks.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

[57] WOLMAN, A., VOELKER, G., AND CHANDRAMOHAN, A. T. Latency analysis of TCP in an ATM network. Tech. Rep. 93-03-03, Department of Computer Science and Engineering, Unviversity of Washington, 1993.

[58] YAMADA, H., AND SUMITA, S. A traffic measurement method and its application for cell loss probability in ATM networks. *IEEE Journal on Selected Areas in Communications 9*, 3 (Apr. 1991), 305–314.

[59] ZUBEREK, W. M. Timed Petri nets – definitions, properties and applications. *Microelectronics and Reliability (Special Issue on Petri Nets and Related Graph Models) 31*, 4 (1991), 627–644.

[60] ZUBEREK, W. M. Modeling using timed Petri nets – event-driven simulation. Tech. Rep. #9602, Department of Computer Science, Memorial University of Newfoundland, St. John's, NF, Canada A1B 3X5, 1996.

[61] ZUBEREK, W. M. Modeling using timed Petri nets – model description and representation. Tech. Rep. #9601, Department of Computer Science, Memorial University of Newfoundland, St. John's, NF, Canada A1B 3X5, 1996.

Appendix A

List of Acronyms

AAL:	ATM Adaptation Layer
ATM:	Asynchronous Transfer Mode
ASCII:	American Standard Code for Information Interchange
BUS:	LAN Emulation Broadcast-and-Unknown Server
CS:	Convergence Sub-layer
CSMA/CD:	Carrier Sensing Multiple Access/Collision Detection
DQDB:	Distributed Queue Double Bus
FDDI:	Fiber Distribution Data Interface
FTP:	File Transfer Protocol
HTTP:	HyperText Transfer Protocol
IETF:	Internet Engineering Task Force
IP:	Internet Protocol
IPX:	Internet Packet Exchange
LAN:	Local Area Network
LEC:	LAN Emulation Client

LECS: LAN Emulation Configuration Server

MAC: Medium Access Control

NNTP: Network News Transfer Protocol

OSI: Open Systems Interconnection

QoS: Quality of Service

RFC: Request For Comments

SAR: Segmentation And Reassembly

SONET: Synchronous Optical NETwork

TCP: Transport Control Protocol

UDP: User Datagram Protocol

VC: ATM Virtual Circuit

WWW: World Wide Web

Appendix B

Net Description Language

The following description of the net description language is taken from [61].

Net description is 'transition oriented', i.e., nets are specified as collections of transitions, and each transition contains all parameters associated with it.

The syntax of model description, in the BNF notation, is as follows:

```
<model-descr> ::= <color-list> <net-class> <net-descr> <imarking>
<color-list> ::= <colors> | <empty>
<net-class> ::= <class> | <empty>
<net-descr> ::= <net-header> ( <transitions> )
<net-header> ::= Mnet | Dnet | net
<transitions> ::= <transition> | <transitions> ; <transition>
<transition> ::= <t-header> = <input-output-list>
             | <t-header> <occurrence-list>
<occurrence-list> ::= <occurrence> | <occurrence-list> , <occurrence>
<occurrence> ::= { <o-name> <type> <time> <prob> = <input-output-list> }
<t-header> ::= <t-indent> <type> <time> <prob>
<t-ident> ::= # <integer> | # <name>
<o-name> ::= <name> | <empty>
<type> ::= :D | :M | :X | <empty>
<time> ::= * <rational> | <empty>
```

```
<prob> ::= , <rational> | , <integer> / <integer> | <ref> | <empty>
<rational> ::= <integer> | <integer> . <integer>
<ref> ::= [ <place_id> ] | [ <place_id> : <color> ]
<input-output-list> ::= <input-list>
                | <input-list> / <output-list>
<input-list> ::= <arc> | <input-list> , <arc>
<output-list> ::= <arc> | <output-list> , <arc>
<arc> ::= <place-id> | <place-id> - | <place-id> : <weight> <color>
<place-id> ::= <integer> | <name>
<weight> ::= <integer>
<color> ::= <name> | <empty>
<name> ::= <letter> | <name> <letter> | <name> <digit> | <name> _
```

The type of the net can be indicated in the net header or in the class directive:

```
<class> ::= class = D ; | class = M ;
```

The type of a transition or an occurrence (M–timed, D–timed) can also be indicated by the `type` elements; such a specification overrides the net type. The specification `X` indicates the type opposite to the one indicated for the net.

For occurrences without `type`, `time`, or `prob` elements, the values of `type`, `time` and `prob` specified for the transition are used. Transitions and occurrences with empty `time` elements denote immediate transitions and occurrences, and are equivalent to `time` equal to 0.

Probability element `prob` specifies the free-choice probabilities of occurrences or relative frequencies of conflicting occurrences. Empty element `prob` is equivalent to probability equal to 1.

Marking–dependent relative frequencies are indicated by place/color references `ref` of the `prob` element. During conflict resolution, the number of (colored) tokens in the place indicated by `ref` is used as the relative frequency of transition/occurrence firings. Usually `ref` is one of the transition's input places.

Arcs without weight are equivalent to arcs with weight equal to 1. Inhibitor arcs are specified as arcs with weight equal to 0.

All colors used in net descriptions must be declared in the list of colors. This list must precede the net description:

```
<colors> ::= color ( <color-list> ) ;
<color-list> ::= <color> | <color-list> , <color>
<color> ::= <name>
```

The initial marking function is specified as a list of marked places:

```
<imarking> ::= mark ( <marking-list> ) ;
<marking-list> ::= <marked-place> | <marking-list> , <marked-place>
<marked-place> ::= <place> | <place> : <count> <color>
<count> ::= <integer>
<color> ::= <name> | <empty>
```

Appendix C

Model - Net Description

This version of the model has two protocols: CU-SeeMe (1 user) and FTP (1024 users).

It was used to generate the delay data in Figure 4.15.

```
class=M;
color(C9,F0
,F1
,F2
,F3
,F4
,F5
,F6
,CU0
,CU1
,CU2
,CU3
,CU4
,CU5
,CU6
);
net(
    #AS_TNK:M
```

```
            {*341.750900=AS_IDLE:1F0/AS_SEND:1F0},
            {*23.636933=AS_IDLE:1CU0/AS_SEND:1CU0};
#AS_SMD:D
            {,0.990894=AS_SEND:1F0,AS_WIND:1F0/AS_WACK:1F0,
                      AS_MSNT:1F0,N_SEND_S:1F1},
            {,0.004303=AS_SEND:1CU0/AS_MSNT:1CU0,N_SEND_S:1CU1};
#AS_PTR:M
            {*1.190548=AS_MSNT:1F0/AS_SEND:1F0},
            {*88.400000=AS_MSNT:1CU0/AS_SEND:1CU0};
#AS_WAK:M
            {*2.086557=AS_WACK:1F0,N_RECV_S:1F6/AS_WIND:1F0};
#AS_SLT:D
            {,0.009106=AS_SEND:1F0,AS_WIND:1F0/AS_WIND:1F0,
                      AS_WDAT:1F0,N_SEND_S:1F2},
            {,0.995697=AS_SEND:1CU0/AS_WDAT:1CU0,N_SEND_S:1CU2};
#AS_WMD:M
            {*9.244022=AS_WDAT:1F0,N_RECV_S:1F4/AS_WDAT:1F0,
                      N_SEND_S:1F3},
            {=AS_WDAT:1CU0,N_RECV_S:1CU4/AS_WDAT:1CU0};
#AS_KAK:D
            {=AS_WDAT:1F0,AS_WACK:0F0,N_RECV_S:1F6/AS_WDAT:1F0};
#AS_LPB:M
            {*0.627875,0.508772=AS_WDAT:1F0,
                      N_RECV_S:1F5/AS_SEND:1F0};
#AS_LAK:M
            {*6.290358,0.491228=AS_WDAT:1F0,N_RECV_S:1F5/AS_IDLE:1F0,
                      N_SEND_S:1F3},
            {,1.0=AS_WDAT:1CU0,N_RECV_S:1CU5/AS_IDLE:1CU0};
#BR_SSRC_S_T1:D
            {*0.00817=N_RECV_S:1F1,BR_SSRC_S_P1:1C9/ATM_SNDS:1F1,
                      BR_SSRC_S_P1:1C9},
            {*0.00817=N_RECV_S:1F2,BR_SSRC_S_P1:1C9/ATM_SNDS:1F2,
                      BR_SSRC_S_P1:1C9},
            {*0.00817=N_RECV_S:1F3,BR_SSRC_S_P1:1C9/ATM_SNDS:1F3,
                      BR_SSRC_S_P1:1C9},
            {*0.00817=N_RECV_S:1CU1,BR_SSRC_S_P1:1C9/ATM_SNDS:1CU1,
                      BR_SSRC_S_P1:1C9},
            {*0.00817=N_RECV_S:1CU2,BR_SSRC_S_P1:1C9/ATM_SNDS:1CU2,
                      BR_SSRC_S_P1:1C9},
            {*0.00817=N_RECV_S:1CU3,BR_SSRC_S_P1:1C9/ATM_SNDS:1CU3,
                      BR_SSRC_S_P1:1C9};
#ASAR_IN_S_T1:D
            {=ATM_SNDS:1F1,ASAR_IN_S_P1:1C9/ASAR_IN_S_P2:1F1},
            {=ATM_SNDS:1F2,ASAR_IN_S_P1:1C9/ASAR_IN_S_P2:1F2},
            {=ATM_SNDS:1F3,ASAR_IN_S_P1:1C9/ASAR_IN_S_P2:1F3},
            {=ATM_SNDS:1CU1,ASAR_IN_S_P1:1C9/ASAR_IN_S_P2:1CU1},
          {=ATM_SNDS:1CU2,ASAR_IN_S_P1:1C9/ASAR_IN_S_P2:1CU2},
          {=ATM_SNDS:1CU3,ASAR_IN_S_P1:1C9/ASAR_IN_S_P2:1CU3};
#ASAR_IN_S_T2:D
          {=ASAR_IN_S_P2:1F1,ASAR_IN_S_P3:0F1/ASAR_IN_S_P1:1C9,
                ASAR_IN_S_P3:25F1},
          {=ASAR_IN_S_P2:1F2,ASAR_IN_S_P3:0F2/ASAR_IN_S_P1:1C9,
                ASAR_IN_S_P3:3F2},
          {=ASAR_IN_S_P2:1F3,ASAR_IN_S_P3:0F3/ASAR_IN_S_P1:1C9,
                ASAR_IN_S_P3:2F3},
          {=ASAR_IN_S_P2:1CU1,ASAR_IN_S_P3:0CU1/ASAR_IN_S_P1:1C9,
                ASAR_IN_S_P3:3CU1},
          {=ASAR_IN_S_P2:1CU2,ASAR_IN_S_P3:0CU2/ASAR_IN_S_P1:1C9,
                ASAR_IN_S_P3:4CU2},
          {=ASAR_IN_S_P2:1CU3,ASAR_IN_S_P3:0CU3/ASAR_IN_S_P1:1C9,
                ASAR_IN_S_P3:2CU3};
#ASAR_IN_S_T3:D
          {*0.0028312=ASAR_IN_S_P2:1F1,ASAR_IN_S_P3:1F1/ATM_S1:1F1,
                ASAR_IN_S_P2:1F1},
          {*0.0028312=ASAR_IN_S_P2:1F2,ASAR_IN_S_P3:1F2/ATM_S1:1F2,
                ASAR_IN_S_P2:1F2},
          {*0.0028312=ASAR_IN_S_P2:1F3,ASAR_IN_S_P3:1F3/ATM_S1:1F3,
                ASAR_IN_S_P2:1F3},
          {*0.0028312=ASAR_IN_S_P2:1CU1,
                ASAR_IN_S_P3:1CU1/ATM_S1:1CU1,ASAR_IN_S_P2:1CU1},
          {*0.0028312=ASAR_IN_S_P2:1CU2,
                ASAR_IN_S_P3:1CU2/ATM_S1:1CU2,ASAR_IN_S_P2:1CU2},
          {*0.0028312=ASAR_IN_S_P2:1CU3,
                ASAR_IN_S_P3:1CU3/ATM_S1:1CU3,ASAR_IN_S_P2:1CU3};
#OC3_SND1_S_T1:D
          {*0.0028312=ATM_S1:1F1,OC3_SND1_S_P1:1C9/ATM_S2:1F1,
                OC3_SND1_S_P1:1C9},
          {*0.0028312=ATM_S1:1F2,OC3_SND1_S_P1:1C9/ATM_S2:1F2,
                OC3_SND1_S_P1:1C9},
          {*0.0028312=ATM_S1:1F3,OC3_SND1_S_P1:1C9/ATM_S2:1F3,
                OC3_SND1_S_P1:1C9},
          {*0.0028312=ATM_S1:1CU1,OC3_SND1_S_P1:1C9/ATM_S2:1CU1,
                OC3_SND1_S_P1:1C9},
          {*0.0028312=ATM_S1:1CU2,OC3_SND1_S_P1:1C9/ATM_S2:1CU2,
                OC3_SND1_S_P1:1C9},
          {*0.0028312=ATM_S1:1CU3,OC3_SND1_S_P1:1C9/ATM_S2:1CU3,
                OC3_SND1_S_P1:1C9};
#ATMSW_SND_S_T1:D
          {*0.0028312=ATM_S2:1F1,ATMSW_SND_S_P1:1C9/ATM_S3:1F1,
                ATMSW_SND_S_P1:1C9},
          {*0.0028312=ATM_S2:1F2,ATMSW_SND_S_P1:1C9/ATM_S3:1F2,
                ATMSW_SND_S_P1:1C9},
        {*0.0028312=ATM_S2:1F3,ATMSW_SND_S_P1:1C9/ATM_S3:1F3,
                ATMSW_SND_S_P1:1C9},
        {*0.0028312=ATM_S2:1CU1,ATMSW_SND_S_P1:1C9/ATM_S3:1CU1,
                ATMSW_SND_S_P1:1C9},
        {*0.0028312=ATM_S2:1CU2,ATMSW_SND_S_P1:1C9/ATM_S3:1CU2,
                ATMSW_SND_S_P1:1C9},
        {*0.0028312=ATM_S2:1CU3,ATMSW_SND_S_P1:1C9/ATM_S3:1CU3,
                ATMSW_SND_S_P1:1C9};
#OC3_SND2_S_T1:D
        {*0.0028312=ATM_S3:1F1,OC3_SND2_S_P1:1C9/ATM_S4:1F1,
                OC3_SND2_S_P1:1C9},
        {*0.0028312=ATM_S3:1F2,OC3_SND2_S_P1:1C9/ATM_S4:1F2,
                OC3_SND2_S_P1:1C9},
        {*0.0028312=ATM_S3:1F3,OC3_SND2_S_P1:1C9/ATM_S4:1F3,
                OC3_SND2_S_P1:1C9},
        {*0.0028312=ATM_S3:1CU1,OC3_SND2_S_P1:1C9/ATM_S4:1CU1,
                OC3_SND2_S_P1:1C9},
        {*0.0028312=ATM_S3:1CU2,OC3_SND2_S_P1:1C9/ATM_S4:1CU2,
                OC3_SND2_S_P1:1C9},
        {*0.0028312=ATM_S3:1CU3,OC3_SND2_S_P1:1C9/ATM_S4:1CU3,
                OC3_SND2_S_P1:1C9};
#ASAR_OUT_S_T1:D
        {*0.0028312=ATM_S4:1F1,ASAR_OUT_S_P1:1F1,
                ASAR_OUT_S_P2:1C9/ASAR_OUT_S_P2:1C9},
        {*0.0028312=ATM_S4:1F2,ASAR_OUT_S_P1:1F2,
                ASAR_OUT_S_P2:1C9/ASAR_OUT_S_P2:1C9},
        {*0.0028312=ATM_S4:1F3,ASAR_OUT_S_P1:1F3,
                ASAR_OUT_S_P2:1C9/ASAR_OUT_S_P2:1C9},
        {*0.0028312=ATM_S4:1CU1,ASAR_OUT_S_P1:1CU1,
                ASAR_OUT_S_P2:1C9/ASAR_OUT_S_P2:1C9},
        {*0.0028312=ATM_S4:1CU2,ASAR_OUT_S_P1:1CU2,
                ASAR_OUT_S_P2:1C9/ASAR_OUT_S_P2:1C9},
        {*0.0028312=ATM_S4:1CU3,ASAR_OUT_S_P1:1CU3,
                ASAR_OUT_S_P2:1C9/ASAR_OUT_S_P2:1C9};
#ASAR_OUT_S_T2:D
        {*0.0028312=ATM_S4:1F1,ASAR_OUT_S_P1:0F1/ATM_SNDD:1F1,
                ASAR_OUT_S_P1:24F1},
        {*0.0028312=ATM_S4:1F2,ASAR_OUT_S_P1:0F2/ATM_SNDD:1F2,
                ASAR_OUT_S_P1:2F2},
        {*0.0028312=ATM_S4:1F3,ASAR_OUT_S_P1:0F3/ATM_SNDD:1F3,
                ASAR_OUT_S_P1:1F3},
        {*0.0028312=ATM_S4:1CU1,ASAR_OUT_S_P1:0CU1/ATM_SNDD:1CU1,
                ASAR_OUT_S_P1:2CU1},
        {*0.0028312=ATM_S4:1CU2,ASAR_OUT_S_P1:0CU2/ATM_SNDD:1CU2,
                ASAR_OUT_S_P1:3CU2},
        {*0.0028312=ATM_S4:1CU3,ASAR_OUT_S_P1:0CU3/ATM_SNDD:1CU3,
                    ASAR_OUT_S_P1:1CU3};
#BR_SDST_S_T1:D
          {*0.00817=ATM_SNDD:1F1,BR_SDST_S_P1:1C9/N_SEND_D:1F1,
                    BR_SDST_S_P1:1C9},
          {*0.00817=ATM_SNDD:1F2,BR_SDST_S_P1:1C9/N_SEND_D:1F2,
                    BR_SDST_S_P1:1C9},
          {*0.00817=ATM_SNDD:1F3,BR_SDST_S_P1:1C9/N_SEND_D:1F3,
                    BR_SDST_S_P1:1C9},
          {*0.00817=ATM_SNDD:1CU1,BR_SDST_S_P1:1C9/N_SEND_D:1CU1,
                    BR_SDST_S_P1:1C9},
          {*0.00817=ATM_SNDD:1CU2,BR_SDST_S_P1:1C9/N_SEND_D:1CU2,
                    BR_SDST_S_P1:1C9},
          {*0.00817=ATM_SNDD:1CU3,BR_SDST_S_P1:1C9/N_SEND_D:1CU3,
                    BR_SDST_S_P1:1C9};
#BD_TNK:M
          {*58.319897=BD_IDLE:1F0/BD_SEND:1F0},
          {*25.740089=BD_IDLE:1CU0/BD_SEND:1CU0};
#BD_SMD:D
          {,0.897980=BD_SEND:1F0,BD_WIND:1F0/BD_WACK:1F0,
                    BD_MSNT:1F0,N_SEND_D:1F4},
          {,0.975946=BD_SEND:1CU0/BD_MSNT:1CU0,N_SEND_D:1CU4};
#BD_PTR:M
          {*9.884468=BD_MSNT:1F0/BD_SEND:1F0},
          {*30.077134=BD_MSNT:1CU0/BD_SEND:1CU0};
#BD_WAK:M
          {*28.673214=BD_WACK:1F0,N_RECV_D:1F3/BD_WIND:1F0};
#BD_SLT:D
          {,0.102020=BD_SEND:1F0,BD_WIND:1F0/BD_WIND:1F0,
                    BD_WDAT:1F0,N_SEND_D:1F5},
          {,0.024054=BD_SEND:1CU0/BD_WDAT:1CU0,N_SEND_D:1CU5};
#BD_WMD:M
          {*32.810188=BD_WDAT:1F0,N_RECV_D:1F1/BD_WDAT:1F0,
                    N_SEND_D:1F6},
          {=BD_WDAT:1CU0,N_RECV_D:1CU1/BD_WDAT:1CU0};
#BD_KAK:D
          {=BD_WDAT:1F0,BD_WACK:0F0,N_RECV_D:1F3/BD_WDAT:1F0};
#BD_LPB:M
          {*2888.667632,0.003228=BD_WDAT:1F0,
                    N_RECV_D:1F2/BD_SEND:1F0};
#BD_LAK:M
          {*0.301882,0.996772=BD_WDAT:1F0,N_RECV_D:1F2/BD_IDLE:1F0,
                    N_SEND_D:1F6},
          {,1.0=BD_WDAT:1CU0,N_RECV_D:1CU2/BD_IDLE:1CU0};
#BR_RDST_R_T1:D
          {*0.00817=N_RECV_D:1F4,BR_RDST_R_P1:1C9/ATM_RCVD:1F4,
                    BR_RDST_R_P1:1C9},
          {*0.00817=N_RECV_D:1F5,BR_RDST_R_P1:1C9/ATM_RCVD:1F5,
                    BR_RDST_R_P1:1C9},
          {*0.00817=N_RECV_D:1F6,BR_RDST_R_P1:1C9/ATM_RCVD:1F6,
                    BR_RDST_R_P1:1C9},
          {*0.00817=N_RECV_D:1CU4,BR_RDST_R_P1:1C9/ATM_RCVD:1CU4,
                    BR_RDST_R_P1:1C9},
          {*0.00817=N_RECV_D:1CU5,BR_RDST_R_P1:1C9/ATM_RCVD:1CU5,
                    BR_RDST_R_P1:1C9},
          {*0.00817=N_RECV_D:1CU6,BR_RDST_R_P1:1C9/ATM_RCVD:1CU6,
                    BR_RDST_R_P1:1C9};
#BSAR_IN_R_T1:D
          {=ATM_RCVD:1F4,BSAR_IN_R_P1:1C9/BSAR_IN_R_P2:1F4},
          {=ATM_RCVD:1F5,BSAR_IN_R_P1:1C9/BSAR_IN_R_P2:1F5},
          {=ATM_RCVD:1F6,BSAR_IN_R_P1:1C9/BSAR_IN_R_P2:1F6},
          {=ATM_RCVD:1CU4,BSAR_IN_R_P1:1C9/BSAR_IN_R_P2:1CU4},
          {=ATM_RCVD:1CU5,BSAR_IN_R_P1:1C9/BSAR_IN_R_P2:1CU5},
          {=ATM_RCVD:1CU6,BSAR_IN_R_P1:1C9/BSAR_IN_R_P2:1CU6};
#BSAR_IN_R_T2:D
          {=BSAR_IN_R_P2:1F4,BSAR_IN_R_P3:0F4/BSAR_IN_R_P1:1C9,
                    BSAR_IN_R_P3:30F4},
          {=BSAR_IN_R_P2:1F5,BSAR_IN_R_P3:0F5/BSAR_IN_R_P1:1C9,
                    BSAR_IN_R_P3:4F5},
          {=BSAR_IN_R_P2:1F6,BSAR_IN_R_P3:0F6/BSAR_IN_R_P1:1C9,
                    BSAR_IN_R_P3:2F6},
          {=BSAR_IN_R_P2:1CU4,BSAR_IN_R_P3:0CU4/BSAR_IN_R_P1:1C9,
                    BSAR_IN_R_P3:11CU4},
          {=BSAR_IN_R_P2:1CU5,BSAR_IN_R_P3:0CU5/BSAR_IN_R_P1:1C9,
                    BSAR_IN_R_P3:6CU5},
          {=BSAR_IN_R_P2:1CU6,BSAR_IN_R_P3:0CU6/BSAR_IN_R_P1:1C9,
                    BSAR_IN_R_P3:2CU6};
#BSAR_IN_R_T3:D
          {*0.0028312=BSAR_IN_R_P2:1F4,BSAR_IN_R_P3:1F4/ATM_R1:1F4,
                    BSAR_IN_R_P2:1F4},
          {*0.0028312=BSAR_IN_R_P2:1F5,BSAR_IN_R_P3:1F5/ATM_R1:1F5,
                    BSAR_IN_R_P2:1F5},
          {*0.0028312=BSAR_IN_R_P2:1F6,BSAR_IN_R_P3:1F6/ATM_R1:1F6,
                    BSAR_IN_R_P2:1F6},
          {*0.0028312=BSAR_IN_R_P2:1CU4,
                    BSAR_IN_R_P3:1CU4/ATM_R1:1CU4,BSAR_IN_R_P2:1CU4},
          {*0.0028312=BSAR_IN_R_P2:1CU5,
                    BSAR_IN_R_P3:1CU5/ATM_R1:1CU5,BSAR_IN_R_P2:1CU5},
          {*0.0028312=BSAR_IN_R_P2:1CU6,
                    BSAR_IN_R_P3:1CU6/ATM_R1:1CU6,BSAR_IN_R_P2:1CU6};
#OC3_RCV1_R_T1:D
          {*0.0028312=ATM_R1:1F4,OC3_RCV1_R_P1:1C9/ATM_R2:1F4,
                    OC3_RCV1_R_P1:1C9},
          {*0.0028312=ATM_R1:1F5,OC3_RCV1_R_P1:1C9/ATM_R2:1F5,
                    OC3_RCV1_R_P1:1C9},
          {*0.0028312=ATM_R1:1F6,OC3_RCV1_R_P1:1C9/ATM_R2:1F6,
                    OC3_RCV1_R_P1:1C9},
          {*0.0028312=ATM_R1:1CU4,OC3_RCV1_R_P1:1C9/ATM_R2:1CU4,
                    OC3_RCV1_R_P1:1C9},
          {*0.0028312=ATM_R1:1CU5,OC3_RCV1_R_P1:1C9/ATM_R2:1CU5,
                    OC3_RCV1_R_P1:1C9},
          {*0.0028312=ATM_R1:1CU6,OC3_RCV1_R_P1:1C9/ATM_R2:1CU6,
                    OC3_RCV1_R_P1:1C9};
#ATMSW_RCV_R_T1:D
          {*0.0028312=ATM_R2:1F4,ATMSW_RCV_R_P1:1C9/ATM_R3:1F4,
                    ATMSW_RCV_R_P1:1C9},
          {*0.0028312=ATM_R2:1F5,ATMSW_RCV_R_P1:1C9/ATM_R3:1F5,
                    ATMSW_RCV_R_P1:1C9},
          {*0.0028312=ATM_R2:1F6,ATMSW_RCV_R_P1:1C9/ATM_R3:1F6,
                    ATMSW_RCV_R_P1:1C9},
          {*0.0028312=ATM_R2:1CU4,ATMSW_RCV_R_P1:1C9/ATM_R3:1CU4,
                    ATMSW_RCV_R_P1:1C9},
          {*0.0028312=ATM_R2:1CU5,ATMSW_RCV_R_P1:1C9/ATM_R3:1CU5,
                    ATMSW_RCV_R_P1:1C9},
          {*0.0028312=ATM_R2:1CU6,ATMSW_RCV_R_P1:1C9/ATM_R3:1CU6,
                    ATMSW_RCV_R_P1:1C9};
#OC3_RCV2_R_T1:D
          {*0.0028312=ATM_R3:1F4,OC3_RCV2_R_P1:1C9/ATM_R4:1F4,
                    OC3_RCV2_R_P1:1C9},
          {*0.0028312=ATM_R3:1F5,OC3_RCV2_R_P1:1C9/ATM_R4:1F5,
                    OC3_RCV2_R_P1:1C9},
          {*0.0028312=ATM_R3:1F6,OC3_RCV2_R_P1:1C9/ATM_R4:1F6,
                    OC3_RCV2_R_P1:1C9},
          {*0.0028312=ATM_R3:1CU4,OC3_RCV2_R_P1:1C9/ATM_R4:1CU4,
                    OC3_RCV2_R_P1:1C9},
          {*0.0028312=ATM_R3:1CU5,OC3_RCV2_R_P1:1C9/ATM_R4:1CU5,
                    OC3_RCV2_R_P1:1C9},
          {*0.0028312=ATM_R3:1CU6,OC3_RCV2_R_P1:1C9/ATM_R4:1CU6,
                    OC3_RCV2_R_P1:1C9};
#BSAR_OUT_R_T1:D
          {*0.0028312=ATM_R4:1F4,BSAR_OUT_R_P1:1F4,
                    BSAR_OUT_R_P2:1C9/BSAR_OUT_R_P2:1C9},
          {*0.0028312=ATM_R4:1F5,BSAR_OUT_R_P1:1F5,
                    BSAR_OUT_R_P2:1C9/BSAR_OUT_R_P2:1C9},
          {*0.0028312=ATM_R4:1F6,BSAR_OUT_R_P1:1F6,
                    BSAR_OUT_R_P2:1C9/BSAR_OUT_R_P2:1C9},
          {*0.0028312=ATM_R4:1CU4,BSAR_OUT_R_P1:1CU4,
                    BSAR_OUT_R_P2:1C9/BSAR_OUT_R_P2:1C9},
          {*0.0028312=ATM_R4:1CU5,BSAR_OUT_R_P1:1CU5,
                    BSAR_OUT_R_P2:1C9/BSAR_OUT_R_P2:1C9},
          {*0.0028312=ATM_R4:1CU6,BSAR_OUT_R_P1:1CU6,
                    BSAR_OUT_R_P2:1C9/BSAR_OUT_R_P2:1C9};
#BSAR_OUT_R_T2:D
          {*0.0028312=ATM_R4:1F4,BSAR_OUT_R_P1:0F4/ATM_RCVS:1F4,
                    BSAR_OUT_R_P1:29F4},
          {*0.0028312=ATM_R4:1F5,BSAR_OUT_R_P1:0F5/ATM_RCVS:1F5,
                    BSAR_OUT_R_P1:3F5},
          {*0.0028312=ATM_R4:1F6,BSAR_OUT_R_P1:0F6/ATM_RCVS:1F6,
                    BSAR_OUT_R_P1:1F6},
          {*0.0028312=ATM_R4:1CU4,BSAR_OUT_R_P1:0CU4/ATM_RCVS:1CU4,
                    BSAR_OUT_R_P1:10CU4},
          {*0.0028312=ATM_R4:1CU5,BSAR_OUT_R_P1:0CU5/ATM_RCVS:1CU5,
                    BSAR_OUT_R_P1:5CU5},
          {*0.0028312=ATM_R4:1CU6,BSAR_OUT_R_P1:0CU6/ATM_RCVS:1CU6,
                    BSAR_OUT_R_P1:1CU6};
#BR_RSRC_R_T1:D
          {*0.00817=ATM_RCVS:1F4,BR_RSRC_R_P1:1C9/N_SEND_S:1F4,
                    BR_RSRC_R_P1:1C9},
          {*0.00817=ATM_RCVS:1F5,BR_RSRC_R_P1:1C9/N_SEND_S:1F5,
                    BR_RSRC_R_P1:1C9},
          {*0.00817=ATM_RCVS:1F6,BR_RSRC_R_P1:1C9/N_SEND_S:1F6,
                    BR_RSRC_R_P1:1C9},
          {*0.00817=ATM_RCVS:1CU4,BR_RSRC_R_P1:1C9/N_SEND_S:1CU4,
                    BR_RSRC_R_P1:1C9},
          {*0.00817=ATM_RCVS:1CU5,BR_RSRC_R_P1:1C9/N_SEND_S:1CU5,
                    BR_RSRC_R_P1:1C9},
          {*0.00817=ATM_RCVS:1CU6,BR_RSRC_R_P1:1C9/N_SEND_S:1CU6,
                    BR_RSRC_R_P1:1C9};
#AE_S:D
          {*0.935812,[N_SEND_S:F1]=N_SEND_S:1F1,
                    AE_IDLE:1C9/N_RECV_S:1F1,AE_IDLE:1C9},
          {*0.082118,[N_SEND_S:F2]=N_SEND_S:1F2,
                    AE_IDLE:1C9/N_RECV_S:1F2,AE_IDLE:1C9},
          {*0.057600,[N_SEND_S:F3]=N_SEND_S:1F3,
                    AE_IDLE:1C9/N_RECV_S:1F3,AE_IDLE:1C9},
          {*1.136963,[N_SEND_S:F4]=N_SEND_S:1F4,
                    AE_IDLE:1C9/N_RECV_S:1F4,AE_IDLE:1C9},
          {*0.147639,[N_SEND_S:F5]=N_SEND_S:1F5,
                    AE_IDLE:1C9/N_RECV_S:1F5,AE_IDLE:1C9},
          {*0.057600,[N_SEND_S:F6]=N_SEND_S:1F6,
                    AE_IDLE:1C9/N_RECV_S:1F6,AE_IDLE:1C9},
          {*0.081600,[N_SEND_S:CU1]=N_SEND_S:1CU1,
                    AE_IDLE:1C9/N_RECV_S:1CU1,AE_IDLE:1C9},
          {*0.149749,[N_SEND_S:CU2]=N_SEND_S:1CU2,
                    AE_IDLE:1C9/N_RECV_S:1CU2,AE_IDLE:1C9},
                {*0.009600,[N_SEND_S:CU3]=N_SEND_S:1CU3,
                           AE_IDLE:1C9/N_RECV_S:1CU3,AE_IDLE:1C9},
                {*0.413363,[N_SEND_S:CU4]=N_SEND_S:1CU4,
                           AE_IDLE:1C9/N_RECV_S:1CU4,AE_IDLE:1C9},
                {*0.207520,[N_SEND_S:CU5]=N_SEND_S:1CU5,
                           AE_IDLE:1C9/N_RECV_S:1CU5,AE_IDLE:1C9},
                {*0.009600,[N_SEND_S:CU6]=N_SEND_S:1CU6,
                           AE_IDLE:1C9/N_RECV_S:1CU6,AE_IDLE:1C9};
     #BE_S:D
                {*0.935812,[N_SEND_D:F1]=N_SEND_D:1F1,
                           BE_IDLE:1C9/N_RECV_D:1F1,BE_IDLE:1C9},
                {*0.082118,[N_SEND_D:F2]=N_SEND_D:1F2,
                           BE_IDLE:1C9/N_RECV_D:1F2,BE_IDLE:1C9},
                {*0.057600,[N_SEND_D:F3]=N_SEND_D:1F3,
                           BE_IDLE:1C9/N_RECV_D:1F3,BE_IDLE:1C9},
                {*1.136963,[N_SEND_D:F4]=N_SEND_D:1F4,
                           BE_IDLE:1C9/N_RECV_D:1F4,BE_IDLE:1C9},
                {*0.147639,[N_SEND_D:F5]=N_SEND_D:1F5,
                           BE_IDLE:1C9/N_RECV_D:1F5,BE_IDLE:1C9},
                {*0.057600,[N_SEND_D:F6]=N_SEND_D:1F6,
                           BE_IDLE:1C9/N_RECV_D:1F6,BE_IDLE:1C9},
                {*0.081600,[N_SEND_D:CU1]=N_SEND_D:1CU1,
                           BE_IDLE:1C9/N_RECV_D:1CU1,BE_IDLE:1C9},
                {*0.149749,[N_SEND_D:CU2]=N_SEND_D:1CU2,
                           BE_IDLE:1C9/N_RECV_D:1CU2,BE_IDLE:1C9},
                {*0.009600,[N_SEND_D:CU3]=N_SEND_D:1CU3,
                           BE_IDLE:1C9/N_RECV_D:1CU3,BE_IDLE:1C9},
                {*0.413363,[N_SEND_D:CU4]=N_SEND_D:1CU4,
                           BE_IDLE:1C9/N_RECV_D:1CU4,BE_IDLE:1C9},
                {*0.207520,[N_SEND_D:CU5]=N_SEND_D:1CU5,
                           BE_IDLE:1C9/N_RECV_D:1CU5,BE_IDLE:1C9},
                {*0.009600,[N_SEND_D:CU6]=N_SEND_D:1CU6,
                           BE_IDLE:1C9/N_RECV_D:1CU6,BE_IDLE:1C9})
mark(AS_IDLE:1024F0
,AS_IDLE:1CU0
,AS_WIND:16384F0
,BR_SSRC_S_P1:1C9
,ASAR_IN_S_P1:1C9
,ASAR_IN_S_P3:25F1
,ASAR_IN_S_P3:3F2
,ASAR_IN_S_P3:2F3
,ASAR_IN_S_P3:3CU1
,ASAR_IN_S_P3:4CU2
,ASAR_IN_S_P3:2CU3
,OC3_SND1_S_P1:1C9
,ATMSW_SND_S_P1:1C9
```

```
,OC3_SND2_S_P1:1C9
,ASAR_OUT_S_P2:1C9
,ASAR_OUT_S_P1:24F1
,ASAR_OUT_S_P1:2F2
,ASAR_OUT_S_P1:1F3
,ASAR_OUT_S_P1:2CU1
,ASAR_OUT_S_P1:3CU2
,ASAR_OUT_S_P1:1CU3
,BR_SDST_S_P1:1C9
,BD_WDAT:1024F0
,BD_WDAT:1CU0
,BD_WIND:22528F0
,BR_RDST_R_P1:1C9
,BSAR_IN_R_P1:1C9
,BSAR_IN_R_P3:30F4
,BSAR_IN_R_P3:4F5
,BSAR_IN_R_P3:2F6
,BSAR_IN_R_P3:11CU4
,BSAR_IN_R_P3:6CU5
,BSAR_IN_R_P3:2CU6
,OC3_RCV1_R_P1:1C9
,ATMSW_RCV_R_P1:1C9
,OC3_RCV2_R_P1:1C9
,BSAR_OUT_R_P2:1C9
,BSAR_OUT_R_P1:29F4
,BSAR_OUT_R_P1:3F5
,BSAR_OUT_R_P1:1F6
,BSAR_OUT_R_P1:10CU4
,BSAR_OUT_R_P1:5CU5
,BSAR_OUT_R_P1:1CU6
,BR_RSRC_R_P1:1C9
,AE_IDLE:1C9
,BE_IDLE:1C9
);
simulate(2050);
simres;
end.-
```

IMAGE EVALUATION
TEST TARGET (QA-3)