TECHNICAL RESEARCH REPORT

Object Oriented Hybrid Network Simulation

by J. Baras, G. Atallah, R. Karne, A. Murad, K.D. Jang

CSHCN T.R. 94-2
(ISR T.R. 94-41)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Object Oriented Hybrid Network Simulation

A Functional Description

Version 1

Dated: September 16, 1994

John Baras, George Atallah, Ramesh Karne,

Ahsun Murad and Kap Do Jang

Contents

1. Introduction

1.1. Motivation

Twenty years ago, when computers were more the exception than the norm in the world of office automation, most personal computers were used as stand-alone devices for simple tasks like word processing, and the media for exchange of information was still paper. At that time, most serious computing was done on large mainframes, each with many terminals, working almost independently of each other, and computer networking was not a problem. Today, with personal computers becoming increasingly powerful, and cost-effective, more people are turning to using a larger number of less powerful (and less expensive) machines to satisfy their computing demands. With the availability of such a wide variety of computing resources and peripheral devices, a need to exchange information between various computers arose. Computers, with the ability to be used as flexible communication tools, have become an integral part of todays information hungry society.

In a networked environment, a computer is a powerful tool for information gathering, organization, storage and disbursement. A communication network represents the backbone of most major computing systems today, and is most often a critical factor in the performance of the entire computer system. Careful design and analysis of the communication network can often lead to cost-effective solutions that satisfy the communication needs of an organization. On the other hand, a poorly designed communication system will result in costly investments in maintenance, constant monitoring, trouble-shooting and eventual replacement.

While many tools exist for the design and analysis of communication network systems, there exist none powerful enough to deal with the complex systems into which communication networks have evolved. Most simulation tools are capable only of simulating small, unrealistic, simple communication networks. Many different network design principles exist today, and even these change constantly as further research into new network designs, and analysis of current network systems shows which networking paradigms are effective for satisfying the networking needs of today. What is needed is a flexible tool that is powerful enough to simulate the widely varied and complex communication networks commonly found in many organizations today, and yet one, that is capable of evolving as networking needs and design paradigms change.

1.2. The Network Simulation Project

The network simulation project aims at the development of exactly such a communication network simulation package, one that is capable of simulating the varied and highly complex communication networks of today and tomorrow. With the availability of such a powerful network simulation product, network designers shall be able to gauge the performance of communication networks under varied technical considerations like choices of various network topologies and protocols, allocation of resources, various routing algorithms, and flow control and fault management policies. Work-load management is an important part of network design, and the simulation will allow the incorporation of complex real-world applications onto the network, a feature that may also be used to test the performance of applications in a network environment.

To achieve such functionality and flexibility, an object-oriented approach will be used in the design of the software. Various physical and logical network entities will be designed independently, each with a well defined interface with the other components of the network. As new network concepts come into play, the software can be adapted to simulate them by simply adding the required functional component to the simulation software, producing a simulation package that grows with the needs of the user.

The software will incorporate an ergonomically designed graphical user interface, which allows the user to create a network topology, assign resources, and select various network protocols, flow-control and fault management policies. The user may also assign the workload on the network. (For the expert user, and for the simulation of large, complex networks, where the interaction through a GUI becomes unwieldy, an alternate user-interface that uses network simulation configuration files will be provided.)

The simulation is structured into three phases: the pre-execution phase, the execution phase, and the post-execution phase. During the pre-execution phase, the user interacts with the simulation software (either through the GUI, or through files) to create a network scenario to be simulated. During the execution phase, the network is simulated: source applications generate packets of data, which are transported across the network (through links and nodes) and delivered to sink applications. As the simulation proceeds, various network performance statistics are generated and stored in an object-oriented database. As part of the post-execution phase, this database is accessed, and the desired network performance parameters are computed and presented to the user (in the form of performance graphs and tables). As a visual verification of the correctness of the simulation, the user is able to view an animation of the simulation, with the key transactions (like transfer of packets) being displayed dynamically on the graphical user-interface.

At its lowest level, the object-oriented design of the network describes a network as made up of nodes and links. The nodes represent the computing resources of the network, and the links represent point-to-point communication channels. Through the process of inheritance, more and more specialized network components like mobile nodes, ATM switches, and broadcast and multicast channels are developed from pre-existing network components.

It is hoped that the paradigm of object-oriented programming incorporated into every component of the simulation will result in a simulation tool that is both powerful and flexible, with new network components added and older, outdated components deleted, as our view of communication in a networking environment changes.

1.3. Objective

The objective of the Network Simulation project is to develop a communication network simulation capable of evaluating the performance of heterogeneous real-world applications under the influence of the following different technical considerations:

1. **Network Workload**
   (a) Types of Applications
      - Image
      - Data
      - Voice
      - Video

   (b) Statistical Properties of Applications
      - Total number of Users
      - Emitting time
      - Packet Types, and Distributions

2. **Network Algorithms**
   (a) Network Protocols
   (b) Routing Algorithms
   (c) Flow Control

3. **Network Topology**

(a) Resource Allocation

(b) Bandwidth Allocation

(c) Connectivity

4. **Network Management**

(a) Fault Management

- Detection
- Tracing
- Resolution

1.4. Purpose of this Document

This document is a functional specification of the Network Simulation Project. It provides a detailed description of the software package in terms of both structure and functionality, and shall serve as an abstract specification of the software.

1.5. Organizaton of this Document

Personal Note: This will change later to reflect the actual structure of the document.

This document is structured as follows: Section 2 lays down the goals of the Network Simulation Software in terms of the kinds of network simulation scenarios that we would like to simulate. Section 3 provides a high-level description of the software, showing how the simulation is broken up into the pre-execution, execution and post-execution phases, and the major components of each. Section 4 provides a detailed functional specification of the physical network components: basis network components like links and nodes, and advanced network components(which are derived from these) like mobile nodes, broadcast channels, switches, etc.. Section 5 provides a functional specification of the logical network components: the network communication protocols, routing and flow-control schemes, network management schemes, and fault management schemes, etc.. Section 6 provides a functional specification of the Graphical User Interface (GUI) which handles the interaction between the user and the simulation software. Finally, Section 7 concludes this specification.

2. An Overview of the Simulation

At the highest level of abstraction, the simulation may be divided into three distinct temporal phases:

1. Pre-execution Phase
2. Execution Phase
3. Post-execution Phase

The block-diagram of the simulation is as shown in Fig. 1.

2.1. Pre-Execution Phase

The pre-execution phase is the first stage in the execution of the network simulation software. Broadly, this stage represents all those actions that are required to be taken (by the user and the software) before the actual simulation of the network may be started.

2.1.1. The User Interfaces

The pre-execution stage begins when the user invokes the simulation software. There are two interfaces provided to the user: A Graphical User Interface (GUI) or a command-based interface. For the novice user with graphic capabilities, and for the creation and simulation of simple networks, the GUI provides a user-friendly interface to the software. However, there are a number of scenarios where the use of a GUI is not recommended: An expert user prefers a command-based interface because he/she can usually interact much faster through a keyboard; not all users have graphics capabilities, and more important, there does not exist a uniform standard for graphics platforms (though X-Windows is becoming a standard for Unix workstations); and finally, for complex simulation tasks, the GUI is not only slow, but very cumbersome (any graphical representation of a network with hundreds of nodes will be necessarily messy).

In this specification, we shall assume that the user may use either interface description to interact with the simulation. More importantly, we shall require that the two be entirely interchangeable in the sense that any simulation invoked on-line may then be analyzed using the GUI, and vice-versa, any simulation scenario created using the GUI may be stored in files, and restarted using on-line commands.

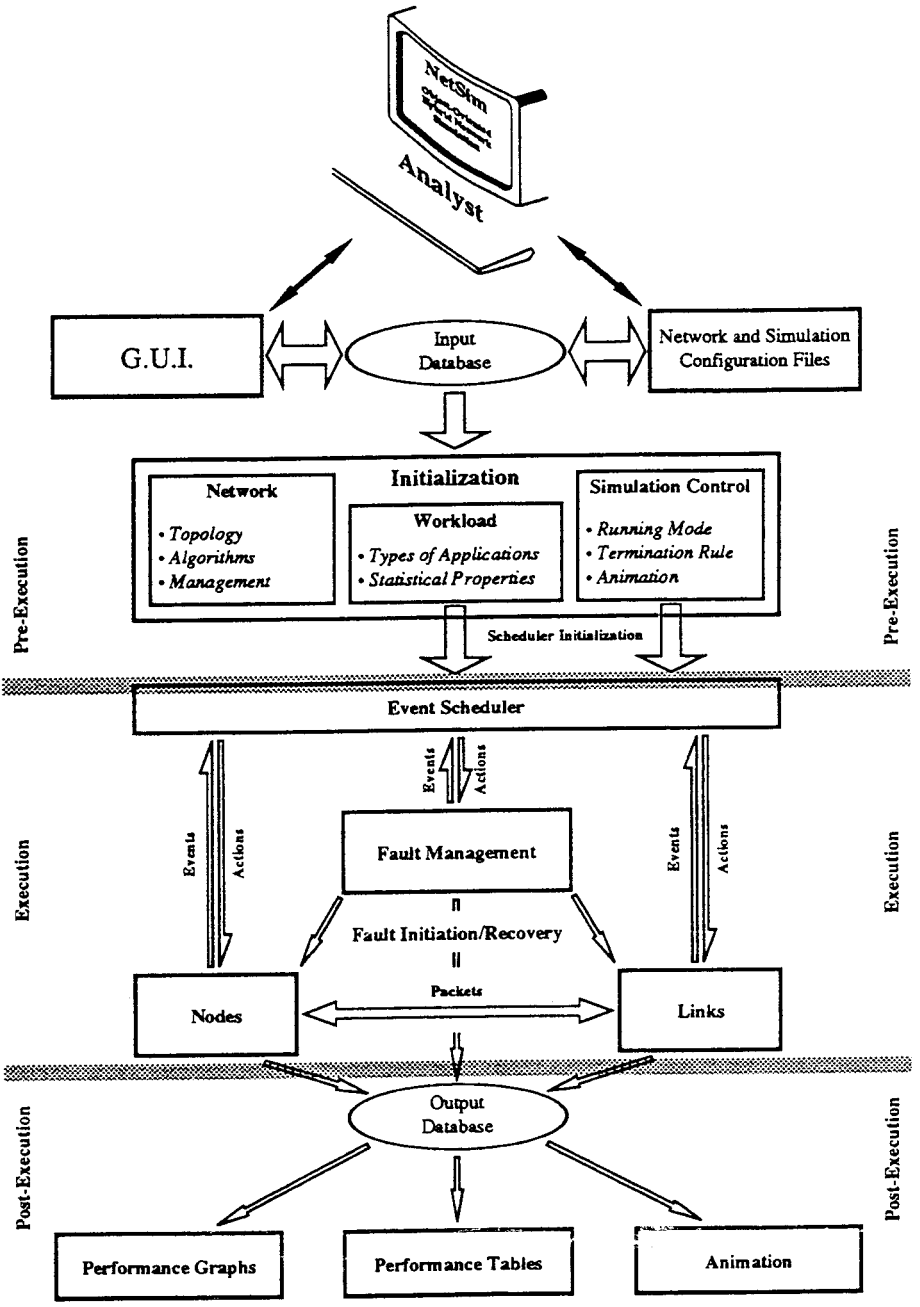

Fig. 1. Phases of Execution

2.1.2. Internal Database

Internally, an object-oriented database is used to represent the network and simulation configuration established by the user. The advantage of an intermediate database is that the GUI can be made to interact with it dynamically to update the database as the user changes the configuration information through the GUI. At the same time, user network and simulation configuration files can be easily entered into the database. The database therefore provides a common medium for unification of the two interfaces. (Incidentally, the same database is used to store all kinds of information generated by the program as it runs the network simulation, including computed performance parameters, etc..)

2.1.3. Initialization of the Simulation

The critical component of the pre-execution phase is the initialization of the simulation. This is divided into the following steps:

1. Network Initialization,
2. Workload Initialization, and
3. Simulation Control Initialization.

2.1.4. Network Initialization

The initialization of the network involves the following:

1. Creation of a network topology to be simulated. A network consists of nodes and links. The nodes represent the computing resources available to the network, and the links represent the communication channels that connect them together. Together, the two specify the physical structure of the network. Advanced network components (which are derived from links and nodes) like switches, hubs, mobile nodes, broadcast and multicast channels may also be available as network components. The user interacts with the interface to create a network topology by selecting the network nodes, and establishing connections between them.
2. Selection of network algorithms. While the physical network components describe the topology of the network, a complete specification of the network also requires the definition of the logical components: This involves the selection of a network protocol, the selection of a routing algorithm, a flow-control mechanism, and queueing logic etc.. Some network components and network algorithms are incompatible, and the initialization shall not be considered complete until these discrepancies are resolved.

3. Selection of network management strategies. This involves the choice of fault management strategies, and other network management policies.

2.1.5. Workload Initialization

The workload initialization involves the following:

1. Selection of the applications to be simulated. The simulation software shall allow the incorporation of any real-world application as long as it obeys a specified interface format. This is a major feature of the simulation software. In most other network simulation packages, the applications are defined only statistically, and do not perform real computations, and generate any real data. The ability to incorporate real network applications allows the simulation of these applications in a network environment (much like the environment it is intended to be run under), and evaluation of the network performance of these applications. (Of course, statistically defined applications may also be simulated using this package.)

2. Selection of the node(s) on which the application will be run. In the simulation, the node is a model of a computing resource available on the network. Each node may represent either a sequential programming environment (if the job queueing logic on the node is selected to be First Come, First Served (FCFS)), or a multiprograming (or process-sharing) environment (if a round-robin job queueing logic is selected).

3. Initialization of the application. Each application may require the user to input some parameters specific to the application, and a part of the interface with the application specifies how this may be done.

4. Statistical Definition of the application. Network applications running on nodes may be source applications, which periodically generate packets, sink applications, which periodically consume packets, source-sink applications (that do both), and simple computational applications (that run on the node, but neither generate, nor consume packets). For an accurate simulation of applications, the application must be statistically described to the simulation software. This includes the start time of each application, workload (in terms of Millions of Machine Instructions) that the application represents, the rate of generation/consumption of packets by the application, the number of packets generated/consumed etc.. The user may interact with the interface to set(or change) these parameters for each application.

4. Establish application connectivity. Source applications need to know the destination of the packets they generate. At the same time, the application that is to receive

these packets must be a sink application. The user interacts with the interface to establish this connectivity, and to check the sanity of these connections.

After the successful initialization of the application, the application is registered with the event-scheduler (which will be responsible for starting it up at the proper time.)

2.1.6. Simulation Control Initialization

The last component of the pre-execution phase is the setup of simulation control parameters. These parameters control the behavior of the execution phase, which is in turn controlled by the event-scheduler. The following need to be specified:

1. Simulation mode. There are two modes provided for simulation: Continuous, and Step. In the continuous mode, once simulation starts, it continues until either terminated, or paused by the user. In the step mode, simulation proceeds in a sequence of steps, pausing after execution of each critical event. The set of events to pause after, may be specified by the user.

2. Termination Conditions. These are a set of conditions that must be satisfied before the simulation will terminate. Simulation will also terminate if there are no more events to be executed, or if the user intervenes, and stops the simulation. Termination conditions are simple conditions like duration of simulation, number of events etc..

When the user has satisfactorily completed the above initializations (or valid default values exist), the pre-execution phase is complete. A network has now been created, and is ready to be simulated. The event-scheduler has also been initialized and is ready to assume control of the simulation. The actual simulation is performed during the execution phase.

2.2. Execution Phase

The beginning of the execution phase is marked by the invocation of the event-scheduler, and its termination by the relinquishing of control by the event-scheduler (either through user intervention—like hitting a control button—or when a simulation termination condition is satisfied).

2.2.1. Event-Driven Simulation and the Event-Scheduler

A network simulation is the simulation of a distributed processing system, where many communicating processes may be executing concurrently on different machines.

Simulation of continuous time systems has always posed a problem to computing systems that are inherently discrete. One popular technique is to implement an event-driven simulation. Event-driven simulation works on the idea that a system needs to be simulated only when something significant (an event) happens. For example, consider a task requiring no input/output being run on a sequential machine. The events required to simulate this system are: a task invocation event (which occurs at the time when the task is invoked), and a task completion event (which occurs when the task completes).

The other problem faced in the simulation of networks is that while networks are inherently distributed computing environments (with a number of things happening at the same time), the simulation is inherently sequential. To simulate distributed systems on a single processor, the real-time clock needs to be simulated. Events are queued in order of increasing time of occurrence, and are simulated one by one. Each event may generate a number of other events which will occur at a later time, and these events are also queued.

Events cannot however be chosen indiscriminately. In a real distributed processing system, a set of concurrently executing tasks may interact with each other through the use of shared resourced. This behavior cannot be simulated on a single processor system, unless the interaction with the shared resource is considered as an event. In that case, two interacting tasks must be represented by events that correspond to the non-interacting sub-tasks, and other events which correspond to the communication between the two tasks. This requires a careful definition of the events if they are to correctly simulate the network system.

The event-scheduler is the engine that implements the event-driven simulation. It consists of an event queue and an event simulation engine. Incoming events are queued in their temporal order of occurrence, and at each step, the event-scheduler picks the most recent event and simulates it. This is repeated until either there are no more events to be simulated, or if a termination condition is satisfied (which is implemented by inserting an appropriate termination event into event queue), or at the user's intervention.

The functions performed during the execution phase are divided into the following broad categories. (This is only a conceptual categorization of the actions; in practice, each action to be simulated will require the execution of many (or all) of these functions.)

1. packet generation and consumption,

2. packet transport,
3. packet routing,
4. flow-control, and
5. fault management.

2.2.2. Packet Generation and Consumption: Applications

The network can be conceptually divided into the packet generation and consumption mechanism (the applications), and the packet transport mechanism (the nodes and links). This section describes the applications and their interface to the network.

The network simulation package is designed to be able to simulate various different kinds of applications, which are representative of the complex real-world applications running on actual networked systems. An application is an object-oriented class whose action is defined by the user, within a few interface constraints.

To the network, an application is defined in terms of the interface to the network. There are two kinds of interfaces that an application may present to the network: a source application interface, and a sink application interface. An application may present more than one application interface to the network (e.g., an application that is both a source and a sink will present a source-application interface, and a sink-application interface). The source-application interface defines the mechanism for the injection of packets into the network, and the sink-application interface defines the mechanism for removal of packets from the network.

2.2.3. Packet Transport

The main task of a communication network is to provide a means of communication between different applications running on different parts of the network. In a digital network, this is usually done through the use of packets. Source applications generate data, which is packed into either variable length, or fixed length packets. These packets are transferred from node to node (using a routing algorithm) until the packet reaches the destination node, where they are consumed by a sink application. To fascilitate this transfer, additional information is tacked at the beginning (packet header) and end (packet trailer) of the packet. This information may be: the destination address, the source identifier, the type of packet, the priority of the packet, error control fields, etc..

A network protocol is a logical component of the network. The particular protocol chosen dictates the format of the packet, the contents of the various fields, and how they are to be

interpreted by the network. While the specific sequence of operations that are involved in the transport of a packet from a source to a destination are dependent on the network protocol, there are some common actions.

2.2.4. Packet Routing

Packet routing deals with the problem of selecting the path that a packet will take in transport from the source application to the sink application through the network. A number of different network routing algorithms exist with varying levels of sophistication and efficiency. The action of packet routing may be fundamentally described as follows: given a packet and its current node location in the network, decide the next node that the packet will be transmitted to, and the link that will be used.

In the case of circuit-switched networks, a virtual circuit is established between a communicating source and destination application, and remains for the entire duration of the time during which that pair of processes wish to communicate. All packets generated by the source application, to be sent to the destination application take the same path through the network.

In the case of packet-switched networks, a routing decision is made independently for each packet, and different packets from the same source application, travelling to the same destination application may take different paths.

Routing algorithms may also be classified as centralized (all routing decisions are made at a central node), or distributed (each node makes all routing decisions for the packets currently resident at that node).

Another classification of routing algorithms is as static: routing tables are computed at the time of conception of the network, and do not change through the life of the network, and adaptive: where routing tables are periodically updated to adapt to changes in the network, like failure and restoration of links and nodes, and in response to changes in traffic patterns.

Implementation of packet routing is as another object-oriented class, and interacts with the network through an interface to each node. Different routing algorithms may be implemented, but the interface remains unchanged. In each case, the routing interface at any node requires information about the current node location of the packet, the destination of the packet, the application identifier (which uniquely identifies the source-destination pair of communicating processes—for circuit switched networks), and the priority of the packet, and returns the next node to transfer the packet to, and the link

to be used. All actions required to make the routing decisions, and to keep the routing tables current, are handled internally by the class.

The various network routing algorithms are discussed later.

2.2.5. Flow-control

Flow control deals with the regulation of the amount of traffic that the network has to deal with. Both local flow control strategies and end-to-end flow control strategies are employed in an effort to prevent "traffic-jams" in the network. When the buffer length on a certain node exceeds safe levels, a fraction of the incoming packets may be discarded by the node in an effort to reduce the incoming traffic on the node. Another strategy for flow control is employed end-to-end, where applications are allowed into the network only when there are resources available to provide the level of service required by the application. The first form of flow control is implemented on each node, and appears as the first interface between the link and the node. For each frame being received by the node, the flow-control algorithm decides whether to accept or reject the frame. If the frame is rejected, and an ARQ protocol is in place, the frame will be subsequently retransmitted by the previous node. If no ARQ procedure is in place, the frame will be lost. (An end-to-end ARQ procedure may be employed as part of the transport layer to retransmit missing packets.)

End-to-end flow control requires negotiation between the network, and the application, to agree on a level of service that is both acceptable to the application, and can be provided by the network. Implementation of end-to-end flow control is not yet fully understood, and needs to be analysed in more detail.

2.2.6. Fault management

Fault management deals with the detection, tracing and resolution of faults that occur in the network.

One kind of fault in the network is the failure of network components. A statistical failure model is incorporated into the design of each network components: nodes and links. At any given time, a node or a link may fail, changing the topology of the network. Detection of link failures is the responsibility of the nodes that the link connects, and detection of node failure is the responsibility of the adjacent nodes. In each case, it is assumes that the nodes responsible for failure detection can perform such detection instantly. This is done through the availability of a failure bit on each network component, which is set whenever the component fails, and is reset when the component comes back up. If a

link fails, the packet it is currently transporting (if any) is lost, and subsequent use of the failed link is not allowed. If a node fails, all packets currently in the node are lost, and the node does not accept any more packets. All applications running on the node are halted until the node comes back up.

Fault tracing is not a concern in the event of component failure. Fault resolution consists of taking actions to adapt to the failure, mainly updating the routing algorithms. Based on the statistical fault model, failed links and nodes may come back up at a later time. This also results in a sequence of restoration actions, including updating the routing algorithms, and restating of DLC protocols if required.

Other kinds of faults, and the fault management strategies to handle them are not well understood at this time, and need to be analyzed in more detail.

2.3. Post-Execution Phase

The post-execution phase involves analyzing the data generated by the execution phase, and do the following:

1. Calculation of user-required network performance statistics
2. Generation of output files, containing tables of performance parameters
3. Generation of performance graphs using the GUI
4. Animation of the simulation just performed

2.3.1. Calculation and Display of Statistics

As the simulation is run, all relevent measurements are made at each node and link. These measurements are recorded in the database at periodic intervals. At the end of the simulation, the database contains a complete dynamic record of various parameters of the network in the repsent simulation scenario. When a user requires the computation of a particular network performance statistic, the database is queried as required to compute the relevent quantities.

A number of alternative formats are available to view the statistics computed: numerical figures, tables and graphs. Single quantities are displayed as numerical figures, whereas dynamic quanitities, and other multidimensional statistics are displayed either as graphs or tables, at the users choice.

2.3.2. Animation

Finally, the statistic recorded also contains a record of all the significant events that took place during the simulation, and the time at which each event took place. Significant events are: generation of packets by source applications, transport of packets by the links and nodes, and delivery of packets at the destination. Significant events also include network component faults occurrence, and correction. This statistic is sufficient to provide an animation display of the simulation on the grahical user interface. The user may view the animation in continuous mode, step-by-step, and also view selected portions of the simulation. When animation is desired from a time different from the start of the simulation, the initial state of the GUI is first computed by executing the animation from the beginning to the desired point. Animation then proceeds normally from this point onwards. All computation associated with the animation is done on the significant events; all other events are ignored as they do not illustrate key features of the simulation.

3. The User Interface

The User Interface controls the user's interaction with the software at both the pre-execution and the post-execution phase.

3.1. Pre-Execution Phase User Interface

At the pre-execution phase, the user interface shall provide the user, the capability to initialize the network topology, initialize various components, select different protocols and algorithms, and set up the control for the simulation. (See Fig. 2.) More specifically, the options provided shall include:

1. **Network Initialization**

    a. **Network Topology Creation.** The user shall be provided a drawing-board on which to construct a network prototype to simulate. Templates for various network components: nodes, links, switches, applications etc. will be provided, and the user can select the ones he wants to create his network. Links, which establish the connectivity between various nodes will be created by selecting the nodes that they connect.

    b. **Network Algorithms Selection.**

        - **DLC Protocols.** The user shall be allowed a choice of various DLC protocols: without retransmission, and with ARQ: stop and wait, go back

n, and selective repeat, with and without piggybacking of the ACKs. Sub-menues shall allow the initialization of parameters specific to the protocol chosen.

- **Job-Queueing Logic.** The user shall be able to simulate multi-processing and single-processing on nodes by selecting the queueing logic for the job-queue on nodes: first-come first-served, and round-robin.
- **Routing Algorithms.** Various centralized and distributed packet routing strategies: both virtual circuit, and data-gram routing strategies will be implemented, and this menu shall allow the user to select one.

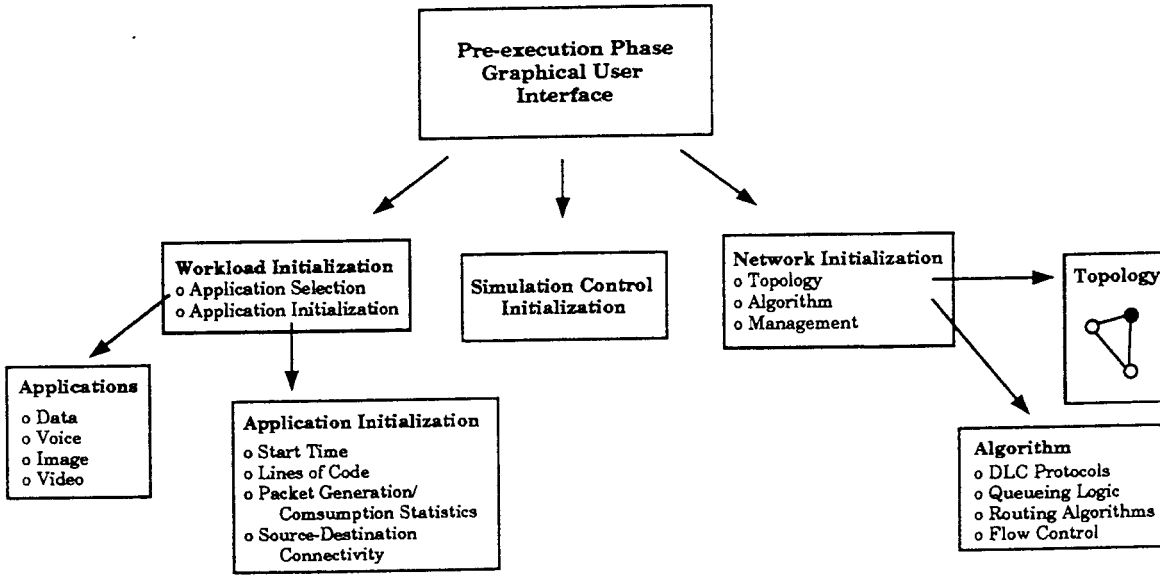

Fig. 2. The Pre-Execution Phase Graphical User Interface

- **Flow Control Strategies.** Various link based, and end-to-end flow-control strategies will be available for the user to choose between. A combination of more than one flow-control strategies will be allowed as long as they are compatible.

c. **Network Management.** This option is concerned mainly with fault management policies, and shall be described in detail later, after discussion.

2. **Workload Initialization.**

   a. **Application Selection.** This menu will alllow the user to select various kinds of real-world applications that have been programmed into the software. It shall be possible for the user to implement other applications and include them in this menu, allowing the simulation of a wide variety of applications in the network environment.

      - **Data.** The class of data applications shall include various file transfer and remote terminal simulation scenarios, with user-assignable parameters.
      - **Voice.** This includes applications to simulate real-time digitized voice transfer.
      - **Image.** This includes various image applications, including progressive coding, and image data-base querying systems etc..
      - **Video.** The most demanding class of applications, this includes real-time video-data transfer applications.

   b. **Application Initialization.** This is an application-specific menu that shall allow the user to initialize all the parameters specific to the application, and the parameters related to the integration of the application into the network. This shall include:

      - **Start Time.** This is the initialization of the start time of the application—the time at which the application-job is inserted into the job-queue of the node. (The actual time at which tha application will start depends on the load on the node at that time.)
      - **Lines Of Code.** This is a measure of the equivalent workload presented by the application to the node.
      - **Packet Generation/Consumption statistics.** For source applications, this menu shall require the user to initialize the time of first generation of packets, the average rate of generation of packets, packet length, etc.. For sink applications, this shall involve the work required at the receipt of each packet.
      - **Source-Destination Connectivity.** This menu shall allow the user to establish the connection between a source application running on some node, and the corresponding destination application running

on another node on the network. This is required so that packets generated by the source application shall carry the correct destination address. Also, in the case of virtual-circuit routing, this action shall initiate the set-up of a virtual circuit between the pair of applications, which shall be established at the time of start-up of the source application.

3. **Simulation Control Initialization.** Thjis menu shall allow the user to set up the various parameters that control the execution phase of the simulation, including stop time, running mode of the simulation, and enabling of animation information generation (which may result in a significant reduction in simulation speed, but shall generate data that may later be used to view an animation of the simulation.

3.2. Post-Execution Phase User Interface

At the Post-execution phase, the main functions of the user interface are to interact with the database, to display various statistics gathered during simulation. The user shall be able to select the parameters to view, the viewing format: tables and graphs, where appropriate. The user may also view an animation of the simulation just performed. Controls shall be provided to start the simulation at any stage of the simulation, and vary the speed of animation. See Fig. 3.

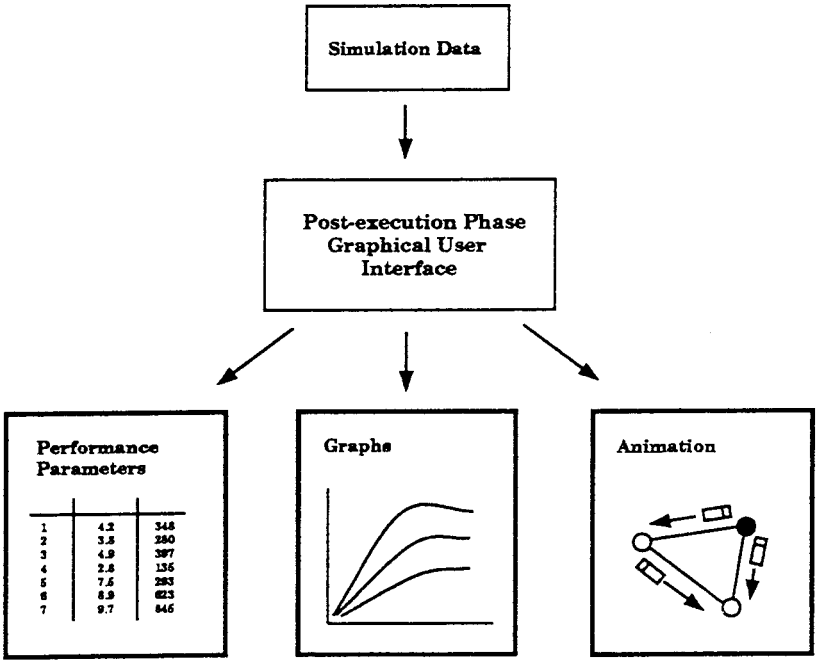

Fig. 3. The Pre-Execution Phase Graphical User Interface

4. Network Modeling

4.1. Node

The node is a basic network component, and is used to represent network nodes. In a network simulation, a node is connected to other nodes through links. Each node may also have a number of applications running on it, which represent the workload on the network. The node basically receives jobs in its job queue. A job may either be a packet received through a link or one of the source applications, or it may be a (packets from either links or applications, or the applications running in its receiver queue. The node processes each packet in turn according to some queueing logic, and then if the packet has reached its destination node, passes it on to the correct application, or if not, then it transmits it to some other node over a link. The basic structure of the node is as shown in Fig. 4(a)-(c).

4.1.1. Structural Components of the Node

The basic node consists of the following components:

1. **The Receiver Section:** This consists of the receiver buffer and the receiver. As packets arrive at the node (either packets received by the node from incoming links, or locally generated packets), they are queued into the receiver buffer. Additionally, as each packet is inserted into the receiver buffer, a `receive_packet` job is inserted into the job queue. (Upon execution, this job shall correspond to the execution of the receiver by the CPU.) When the CPU picks up a `receive_packet` job from the job queue, the receiver module is executed. This is the server for the receiver buffer, and upon execution removes the packet from the receiver buffer. (The CPU then calls the Packet Routing function to figure out where this packet is to be sent.

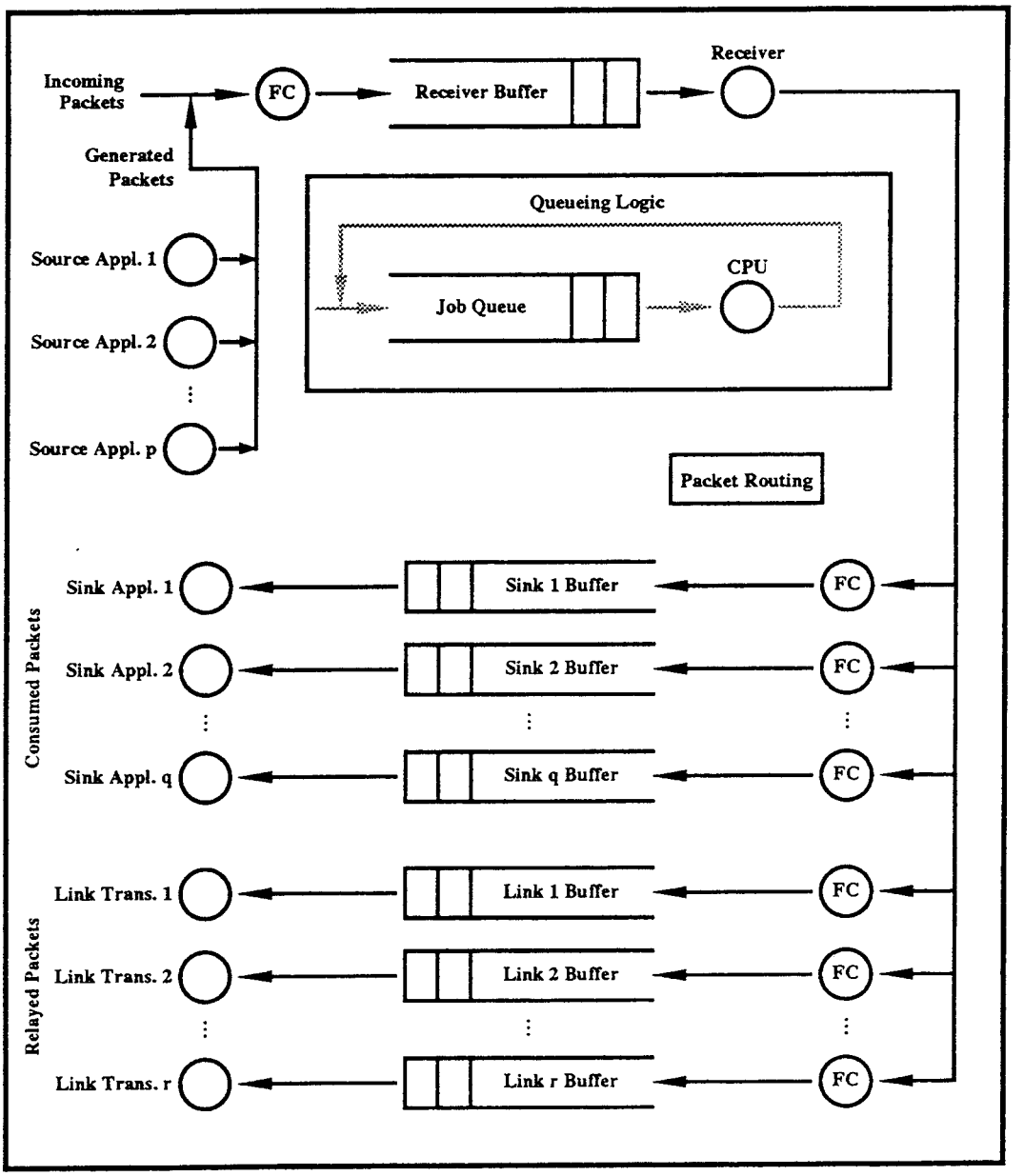

Fig. 4(a). Structure of Nodes. Black arrows indicate flow of Packets.

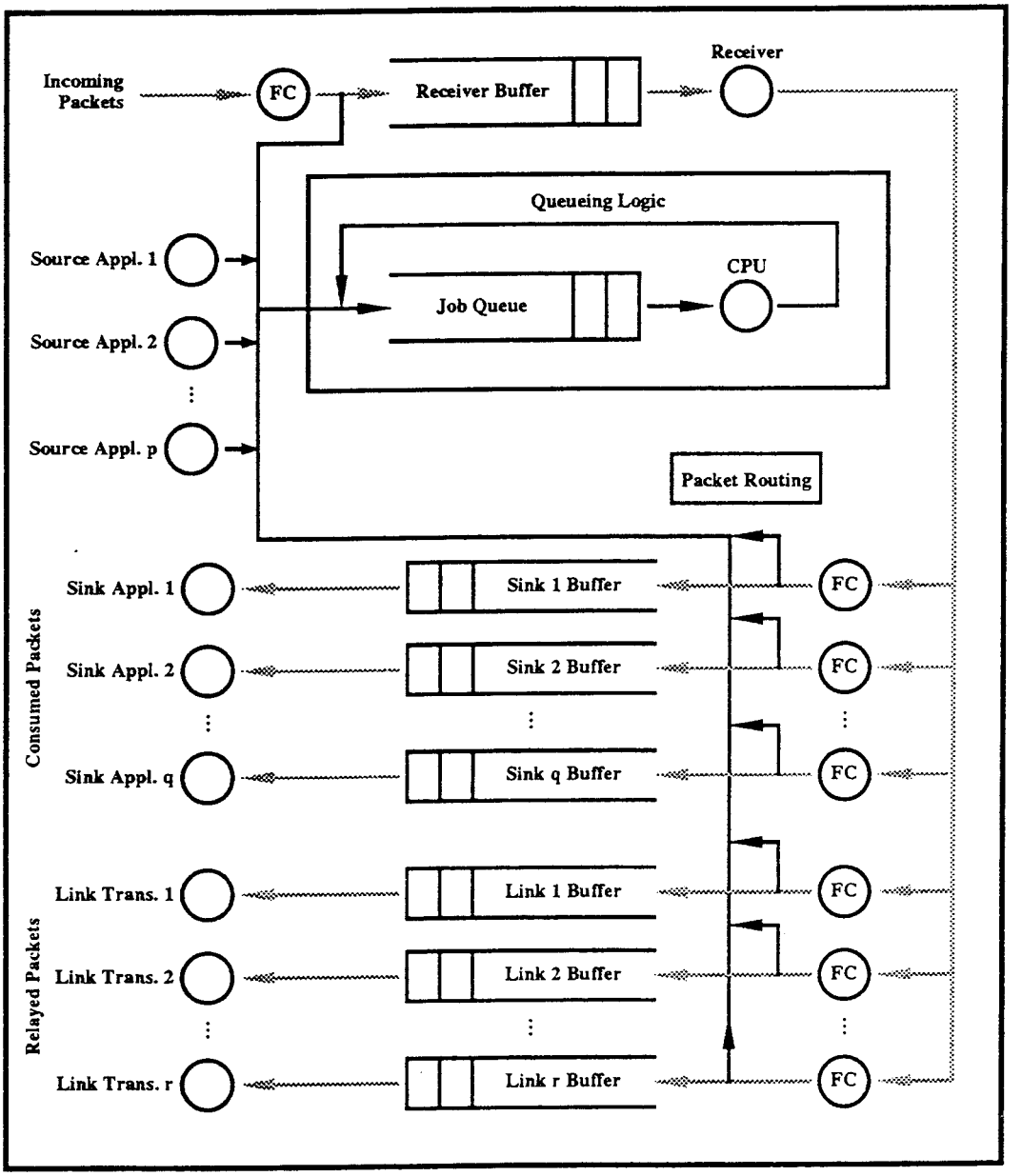

Fig. 4(b). Structure of Nodes (continued). Black arrows indicate queueing of Jobs.

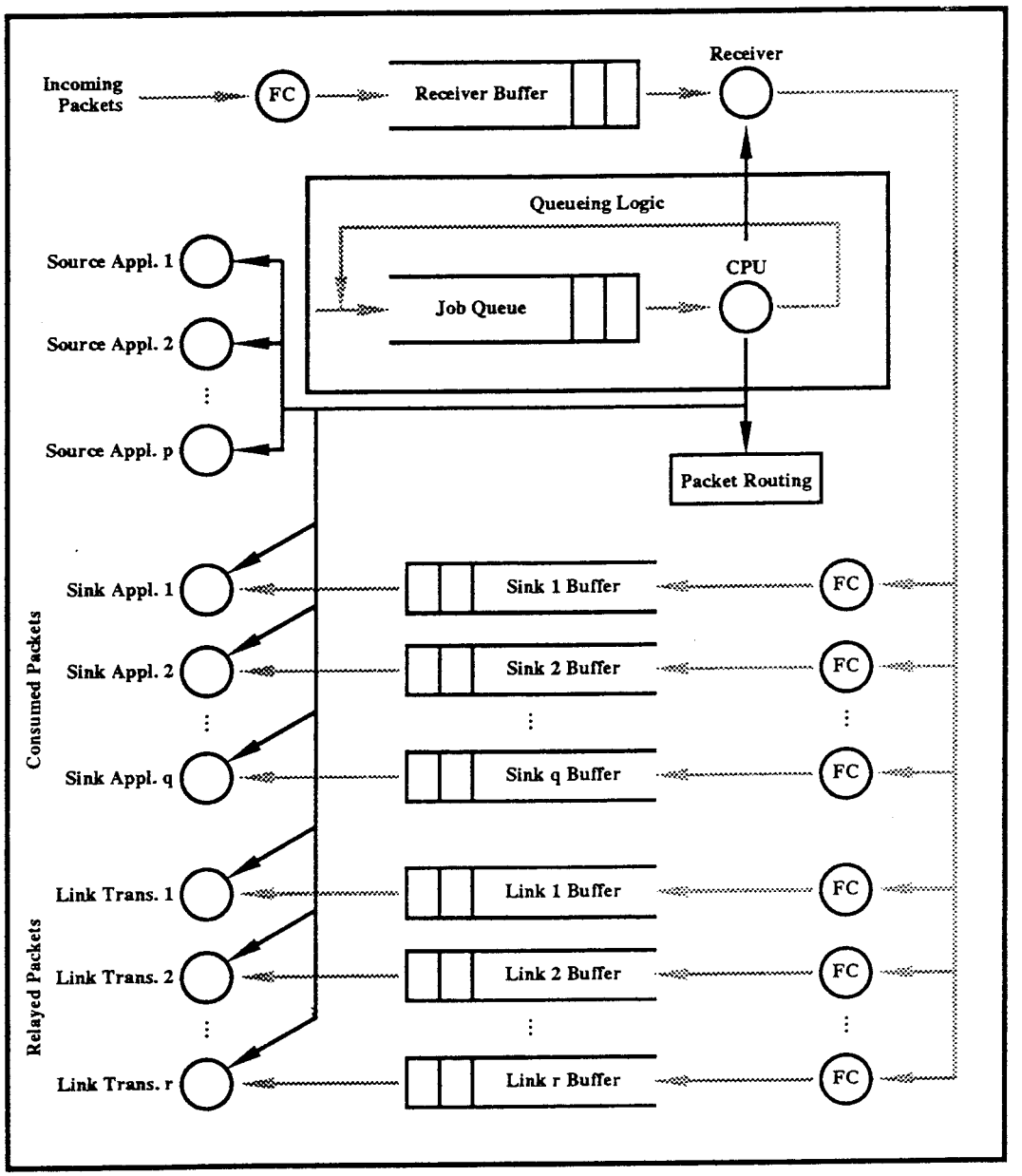

Fig. 4(c). Structure of Nodes (continued). Black arrows indicate execution of Jobs by CPU.

2. **The Job Execution Section:** This section models the computing engine of the node. It consists of a job queue and the job-server, the CPU. Each job simulates a software task to be executed by the CPU. These tasks are: Source and Sink Applications running on the node, the receiver, the transmitters (one for each outgoing link), and finally the packet-routing algorithm. Each of the following constitute a single job: receiving a packet, transmitting a packet, generating a packet, consuming a packet, and performing packet routing for a single packet. A number of different queueing logics may be implemented on the job-queue: first-come first-served (FCFS), which simulates a single processor machine running in uni-tasking mode; round-robin, which simulates a single processor machine running in multitasking mode, and non-interrupting priority-based FCFS and round-robin (where jobs are queued according to their priority, and an incoming job with higher priority does not interrupt on-going service to a lower-priority job).

3. **Source Applications:** The node is also capable of running a number of source applications. Source applications are those that are capable of generating packets. Each source application is registered with the event-scheduler during pre-execution, which causes the source-application to be started up at the right time. Upon startup, the source application inserts a `packet_generation` job into the job-queue, which corresponds to the work required to be done to generate the first packet. Execution of this job by the CPU causes the generation of a single packet, and if the application has further packets to transmit, the insertion of a similar job into the queue. This process continues until the application has exhausted its supply of packets, at which time the source application is said to have terminated.

4. **Sink Applications:** The other kind of application that the node can run is sink applications. Each sink application simulates an application that is capable of consuming packets. When a packet is received for the corresponding sink-application, the packet is inserted into the corresponding sink buffer (there is one buffer for each sink application). This also causes the insertion of a `packet_consumption` job into the job-queue. Execution of this job by the CPU causes the removal of a single packet from the sink buffer. This packet is passed on to the sink application.

5. **Transmitter Sections:** A number of links may be connected to each node. The node requires a transmitter section for each link. Each transmitter section consists of a transmitter buffer and a transmitter. Packets to be transmitted over a particular transmitter buffer are queued into the corresponding transmitter buffer. The enqueueing of each packet also causes the insertion of a `transmit_packet` job

into the job queue. Upon execution of such a job, the transmitter removes a single packet from the corresponding receiver queue, and attempts to transmit it over the corresponding link.

4.1.2. Functional Specification of the Node

This section is a functional specification of the node. The actions performed by the node may be divided into the following parts: Packet Generation (by Source Applications), Packet Reception, Packet Routing, Packet Transmission, and Packet Consumption. Other functions of the node (related to fault-management and flow-control) are not specified at this time. Before we discuss this, we shall provide a functional specification of the Job-Execution Section.

4.1.3. Job-Execution Section

As mentioned earlier, the job-execution section consists of the job queue, and the job-server (the CPU). The basic entity of the Job-Execution Section is the job. Each job has the following attributes:

1. Start time t_{start},
2. Number of instructions I_{total},
3. Priority P,
4. Number of instructions already executed by CPU I_{done},
5. Action Routine `<function>`.

4.2. Structure of the Link—The Physical Layer

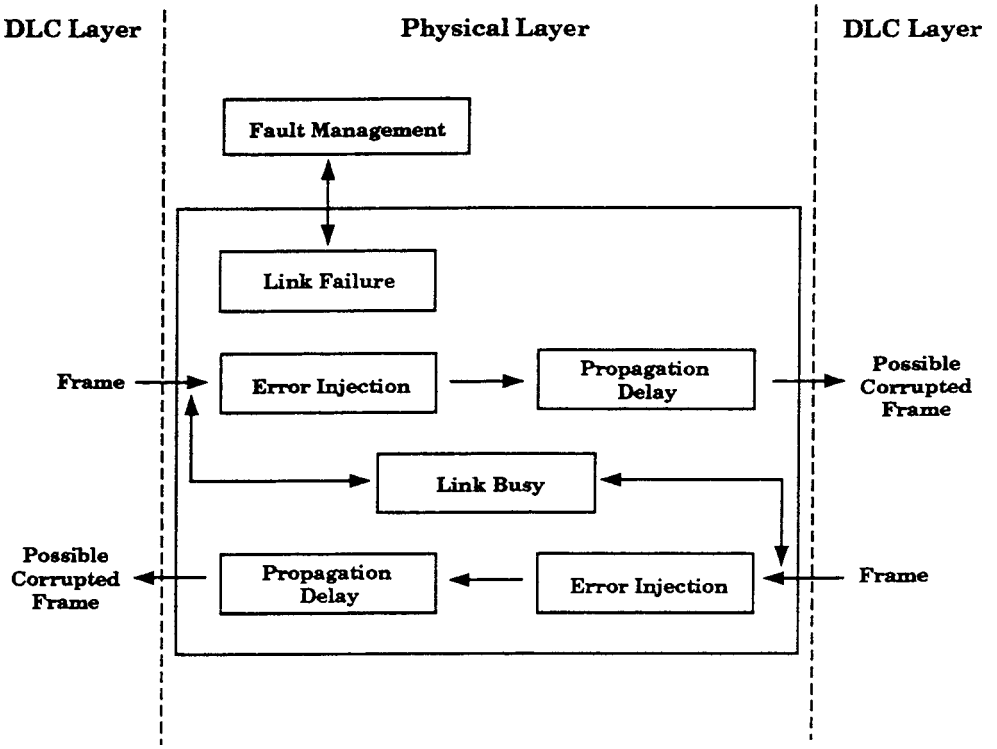

Fig. 4. Structure of the Link: Physical Layer

4.3. Data Link Control Layer

This section gives a functional specification of the Data Link Control (DLC) layer, which is the second layer of the OSI model. The role of the DLC layer is usually to convert the unreliable bit-pipe provided by layer 1 into a higher-level, virtual error-free channel for sending packets asynchronously, in both directions over the link.

The DLC layer is implemented through Data Link Control modules present at each end of the bi-directional link as peer processes.

The input to the DLC layer is a stream of packets at each end of the link. The DLC module at the sending end adds a header and a trailer to the packet, resulting in a longer string of bits called a frame. Some of the bits detect transmission errors, some request transmission in the event of errors, and some detect the frame boundaries. In some network protocols, the DLC does not retransmit frames in the event of errors. Packets with errors are simply dropped, and retransmission may be attempted on an end-to-end basis.

In this section, we discuss the structure of the DLC layer: the modules at each end of the link, and the various DLC algorithms that may be implemented.

4.3.1. The DLC Module

The structure of the DLC module is as shown in Fig. 5.

The DLC module at each end consists of a transmitter section and a receiver section. The working of each section is broadly as follows: The DLC layer accepts a packet from the Network layer (layer 3), converts it into a frame, and stores it in the transmitter buffer, which is of a fixed size (the size depends on the ARQ protocol being implemented). Frames are transmitted on a first-come first-served basis, and received by the DLC module on the other side. Frames in error are discarded. The DLC module on the receiving side, sends acknowledgements (ACKs) for the frames received, in accordance with the protocol being implemented. These ACKs are also frames, which travel on the same link in the other direction. When the transmitter receives confirmation of receipt of a frame, the frame is flushed from the buffer, and another frame may be accepted into the DLC layer. If confirmation of receipt of a frame is not received by the transmitter within a certain time, the frame is usually retransmitted. The DLC layer usually preserves packet sequence, and provides error-free communication through error-detection and retransmission, except for some DLC layer protocols where these functions are performed at a higher level. A number of different DLC layer protocols exist, which we shall describe in the following subsections.

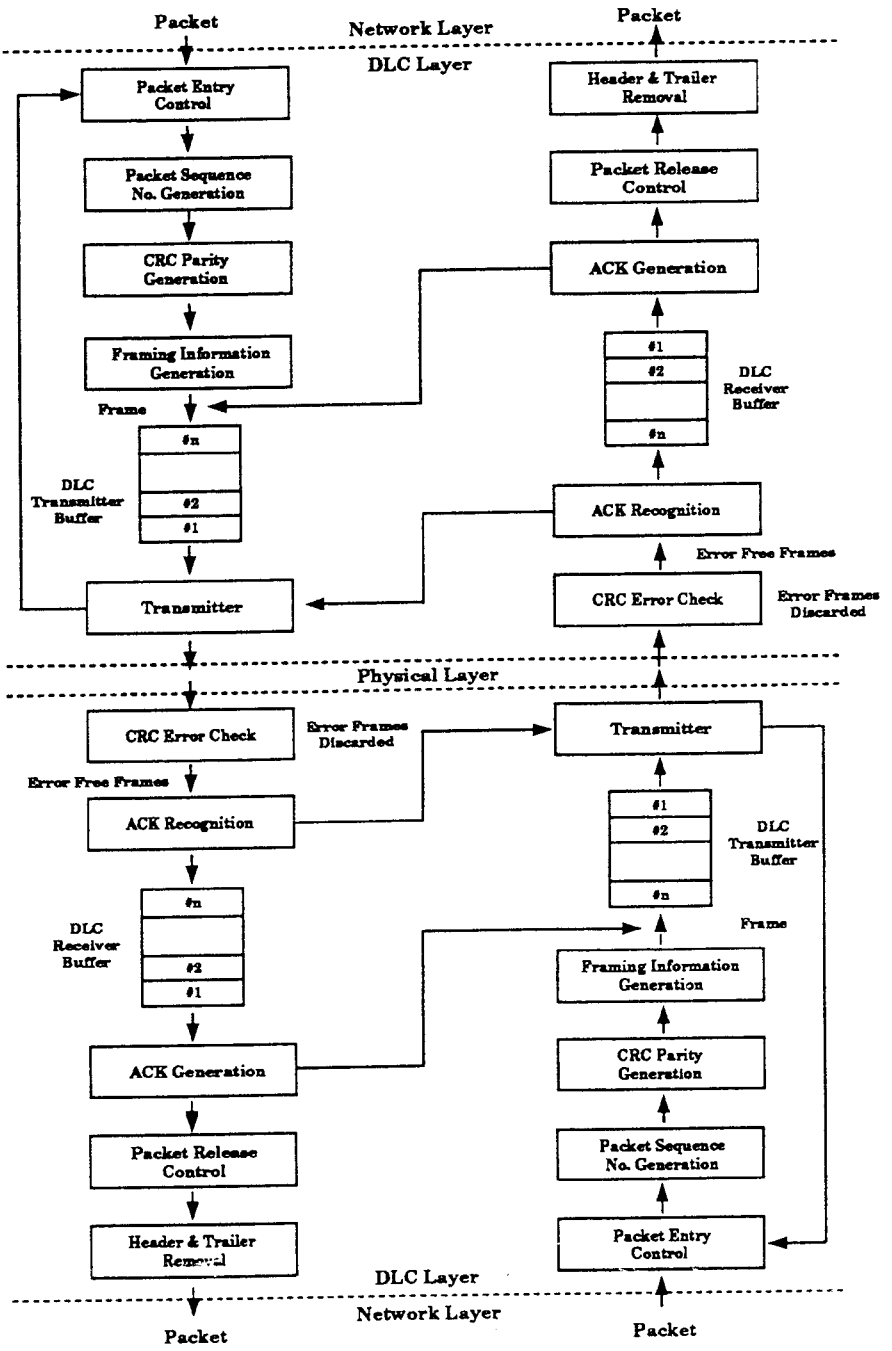

Fig. 5. The DLC Module

4.3.2. DLC Protocol with no Retransmission

In a very simple DLC layer protocol, error frames are simply discarded, and no retransmission is attempted. Retransmission of missing packets may be implemented end-to-end at the higher transport layer, if required. In this case, the structure of the DLC modules is as shown in Fig. 6.

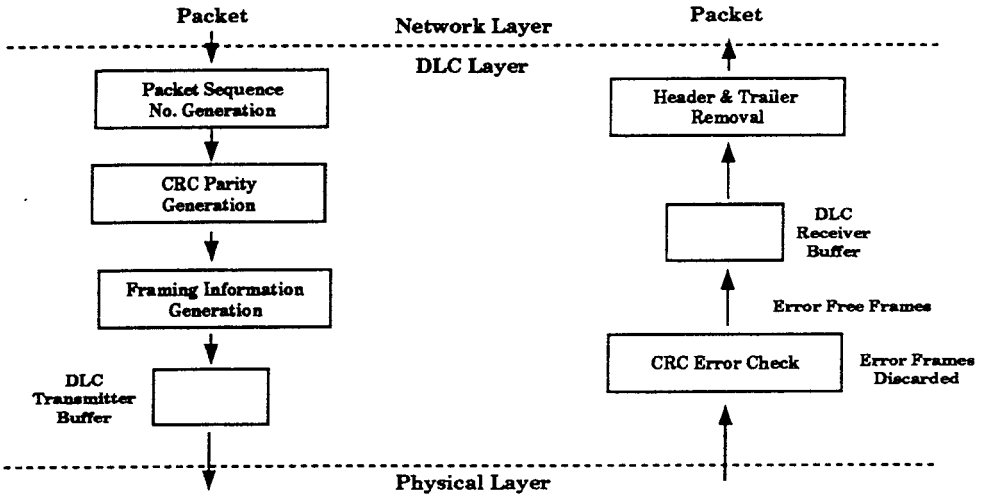

Fig. 6. The DLC Module—No Retransmission

The algorithm implemented at each transmitter is as follows:

1. Get next packet from Network layer; if no packet is available, wait.
2. Convert packet into frame: add framing and CRC field.
3. Transmit frame.
4. Go to 1.

The algorithm implemented at each receiver is as follows:

1. Receive frame from Layer 1; if no frame is available, wait.
2. Check frame for errors; if in error, discard frame, and go to 1.

3. If error-free frame, strip off extra fields from frame, and convert to packet.
4. Release packet to network layer.
5. Go to 1.

This is a very simple protocol, requiring no retransmission of packets. Depending on the error-quality of the link, a significant number of packets may be lost.

4.3.3. Stop and Wait ARQ

This is the simplest of the ARQ protocols requiring retransmission of erroneous frames. In this case, the structure of the DLC modules is as shown in Fig. 7.

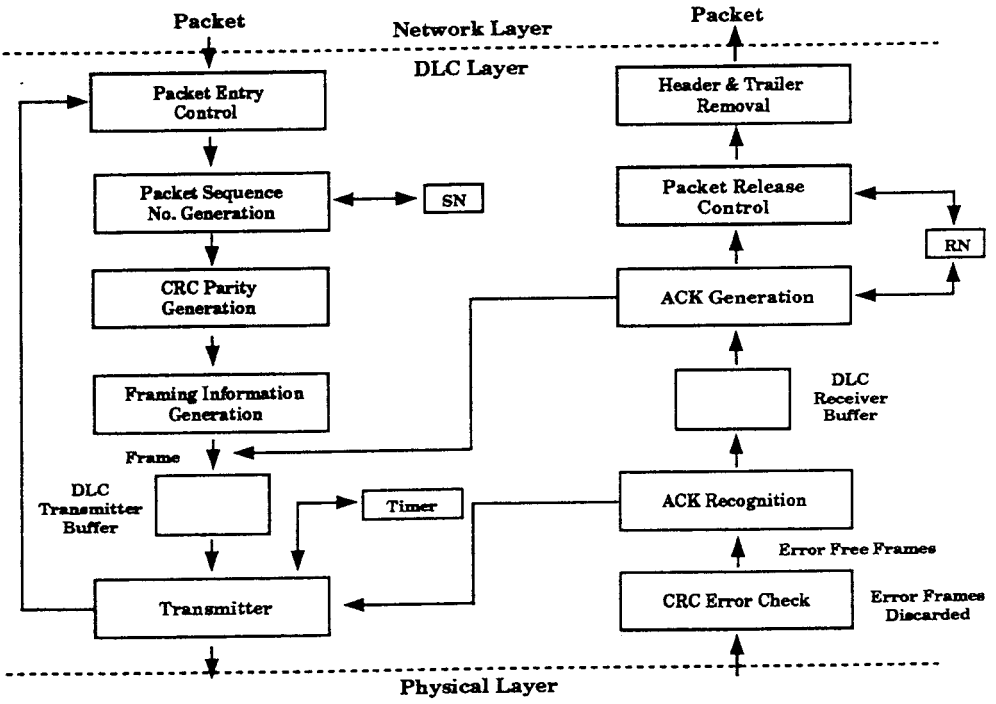

Fig. 7. The DLC Module—Stop and Wait ARQ

The algorithm implemented at each transmitter is as follows:

1. Set $SN = 0$.
2. Get next packet from Network layer; if no packet is available, wait.

3. Convert packet into frame: add framing and CRC field, and a sequence number field with sequence number *SN*.
4. Transmit frame, and start timer.
5. If ACK with request number $RN > SN$ is received, set $SN = RN$, and go to 2.
6. If time-out occurs, go to 4.

The algorithm implemented at each receiver is as follows:

1. Set $RN = 0$.
2. Receive frame from Layer 1; if no frame is available, wait.
3. Check frame for errors; if in error, discard frame, and go to 2.
4. If frame has sequence number $SN \neq RN$, discard frame, and go to 2.
5. Otherwise, if frame has sequence number $SN = RN$, strip off extra fields from frame, convert to packet, and release to network layer.
6. Otherwise, if frame is an ACK, give ACK to transmitter.
7. Increment *RN*.
8. Send ACK with request number *RN*.
9. Go to 2.

This is a simple ARQ protocol, but is not commonly used because of its inefficient use of the communication channel. In the event of a large round-trip delay, a lot of time is spent waiting for the ACK.

It should also be noted that it is sufficient to send *RN* and *SN* modulo 2 for correct working of the protocol.

4.3.4. Stop and Wait with Piggybacking of the ACK

This algorithm is conceptually the same as above, except that ACKs, instead of being transmitted as separate frames, are piggybacked onto frames being sent in the reverse direction. In the event that there is no traffic in the reverse direction, this strategy would cause traffic in the forward direction to stop too. To prevent this, ACKs are released after a finite delay, as individual frames, if there is no traffic in the meanwhile. In this case, each frame contains a field for both the sequence number, *SN*, and the request number *RN* of the packet being waited for. The algorithm implemented at each transmitter is as follows:

1. Set $SN = 0$.
2. Get next packet from Network layer; if no packet is available, wait.
3. Convert packet into frame: add framing and CRC field, and a sequence number field with sequence number SN.
4. Transmit frame, and start timer.
5. If ACK with request number $RN > SN$ is received, set $SN = RN$, and go to 2.
6. If time-out occurs, go to 4.

The algorithm implemented at each receiver is as follows:

1. Set $RN = 0$.
2. Receive frame from Layer 1; if no frame is available, wait.
3. Check frame for errors; if in error, discard frame, and go to 2.
4. If frame has sequence number $SN \neq RN$, discard frame, and go to 2.
5. Otherwise, if frame has sequence number $SN = RN$, strip off extra fields from frame, convert to packet, and release to network layer. Also extract RN from frame, and give ACK with request number RN to transmitter.
6. Otherwise, if frame is an ACK, give ACK to transmitter.
7. Increment RN.
8. Generate ACK with request number RN. Start timer. If a frame is to be transmitted in the reverse direction, piggyback the ACK onto it; otherwise, if time-out occurs, send ACK.
9. Go to 2.

The piggybacking strategy may be used with any of the ARQ startegies, requiring independent ACKs, and is implemented exactly as above, and so shall not be discussed further.

4.3.5. Go back n ARQ

In the stop-and wait strategy, the transmitter is idle while waiting for the receipt of an ACK. In the Go back n strategy, effort is made to make the ARQ protocol more effifient by transmitting subsequent packets while waiting for the arrivial of the ACK. The structure of the DLC module is as shown in Fig. 8.

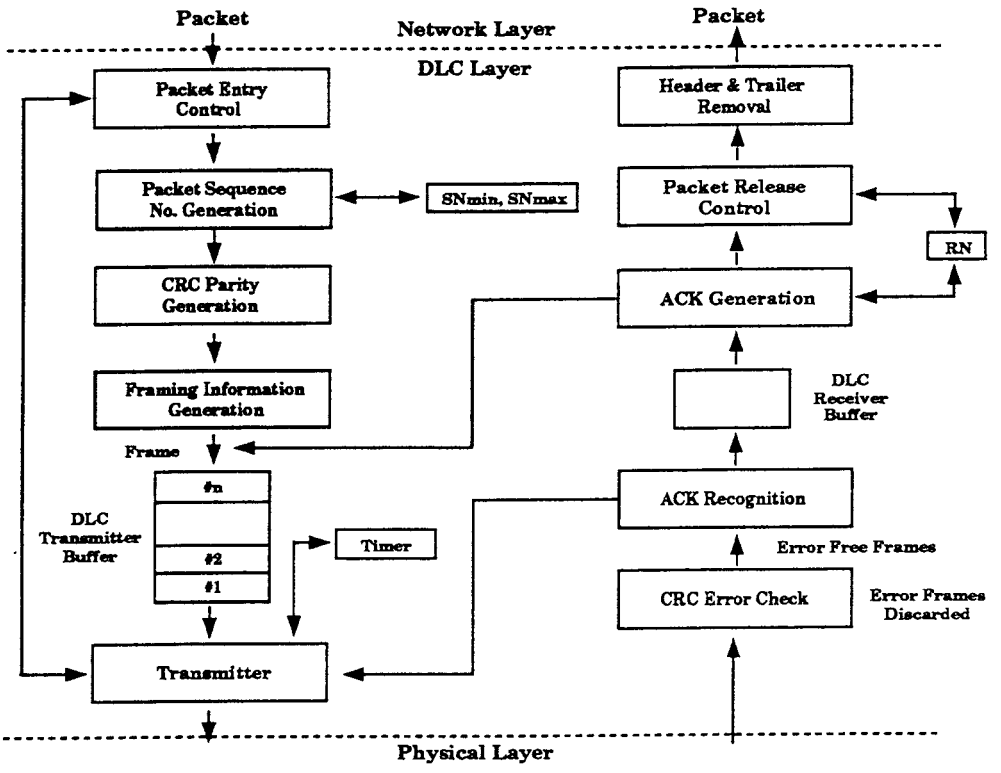

Fig. 8. The DLC Module—Go back n ARQ

The algorithm implemented at each transmitter is as follows:

1. Set $SN\text{min} = SN\text{max} = 0$.
2. Do 3, 4, or 5 in any order, when the corresponding condition is satisified.
3. *Condition:* $SN\text{max} < SN\text{min} + n$. *Action:* If a packet is available from the network layer, accept packet, add framing and CRC fields, and sequence number field assigned to $SN\text{max}$. Increment $SN\text{max}$.
4. *Condition:* An ACK is received with $RN > SN\text{min}$. *Action:* Increment $SN\text{min}$ to RN.
5. *Condition:* $SN\text{min} < SN\text{max}$, and no frame is being transmitted. *Action* Choose some SN: $SN\text{min} \leq SN < SN\text{max}$, and transmit frame number SN.

The algorithm implemented at each receiver is as follows:

1. Set $RN = 0$.

2. Receive frame from Layer 1; if no frame is available, wait.
3. Check frame for errors; if in error, discard frame, and go to 2.
4. If frame has sequence number $SN \neq RN$, discard frame, and go to 2.
5. Otherwise, if frame has sequence number $SN = RN$, strip off extra fields from frame, convert to packet, and release to network layer.
6. Otherwise, if frame is an ACK, give ACK to transmitter.
7. Increment RN.
8. Send ACK with request number RN.
9. Go to 2.

Note that the go back n ARQ protocol operates correctly when RN and SN are modulo m: $m > n$.

4.3.6. Selective Repeat ARQ

Another effort to make the stop-and-wait ARQ protocol more efficient results in the Selective Repeat protocol. The structure of the DLC modules is as shown in Fig. 9.

The algorithm implemented at each transmitter is as follows:

1. Set $SN_{min} = SN_{max} = 0$.
2. Do 3, 4, or 5 in any order, when the corresponding condition is satisified.
3. *Condition:* $SN_{max} < SN_{min} + n$. *Action:* If a packet is available from the network layer, accept packet, add framing and CRC fields, and sequence number field assigned to SN_{max}. Increment SN_{max}.
4. *Condition:* An ACK is received with $RN > SN_{min}$. *Action:* Increment SN_{min} to RN.
5. *Condition:* $SN_{min} < SN_{max}$, and no frame is being transmitted. *Action* Choose some SN: $SN_{min} \leq SN < SN_{max}$, and transmit frame number SN.

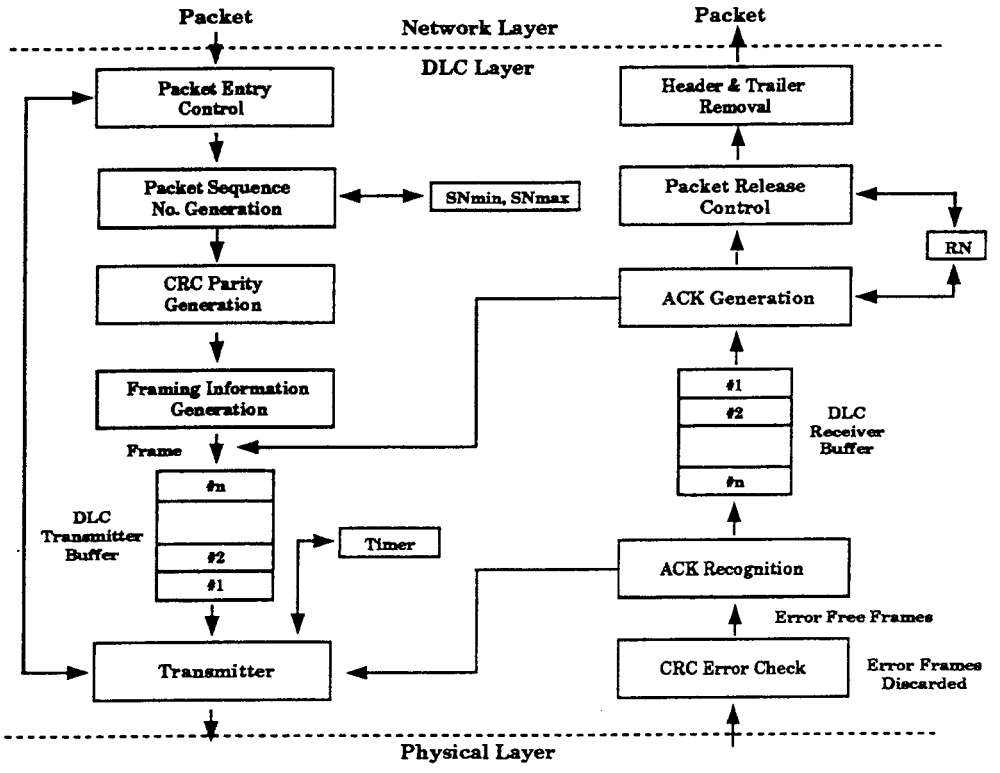

Fig. 9. The DLC Module—Selective Repeat ARQ

The algorithm implemented at each receiver is as follows:

1. Set $RN = 0$.
2. Receive frame from Layer 1; if no frame is available, wait.
3. Check frame for errors; if in error, discard frame, and go to 2.
4. If frame has sequence number SN: $RN \leq SN \leq RN + n - 1$, accept frame.
5. If $SN = RN$, strip off extra fields from frame, convert to packet, and release to network layer, increment RN.
6. Repeat 5, until all properly sequenced frames have been released to network layer, and RN is the number of the next missing frame.
7. Send ACK with request number RN.

8. Otherwise, if frame has sequence number outside range $RN \leq SN \leq RN + n - 1$, discard frame, and go to 2.

9. Otherwise, if frame is an ACK, give ACK to transmitter.

8. Go to 2.

Note that the selective repeat ARQ operates correctly with RN and SN modulo m: $m \geq 2n$.

Another implementation of the selective repeat ARQ protocol requires the ACK to include both RN, and a single bit that gives the receipt condition of each subsequent packet. Therefore, multiple ACKs are received each time, with little overhead, preventing retransmission of frames due to lost ACKs.

Personal Note: The following sections to be completed later as the project is better defined.

4.4. Network Layer

4.5. Transport Layer

5. Network Applications

5.1. Statistical Description of Applications

5.2. Source Application Interface

5.3. Sink Application Interface

5.4. Establishing Application Connectivity

6. Network Routing

6.1. Virtual Circuit Routing

6.2. Datagram Routing Routing

6.3. Shortest-Path Routing Algorithms

6.4. Optimal Routing Algorithms

7. Fault Simulation and Management

7.1. Simulation of Faults in Links and Nodes

7.2. Response to Network Component Failure

7.3. Response to NEtwork Component Restoration

8. Advanced Network Components

8.1. Mobile Nodes

8.2. Switches

8.3. Broadcast Channels

8.4. Multicast Channels

8.5. Specialized Channels

9. Advanced Network Algorithms

Personal Note: This section shall include discussion on implementation of various existing network standards.

10. Summary