AXIOMATIC DESIGN AND NETWORK PERFORMANCE ANALYSIS FOR APPLICATIONS IN HOME-BASED HEALTH CARE

by

JASON DOUGLAS HINTERSTEINER

S.B., Mechanical Engineering
Massachusetts Institute of Technology, 1996

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the
Massachusetts Institute of Technology
February, 1998

© 1997 Massachusetts Institute of Technology
All rights reserved.

Signature of Author

Department of Mechanical Engineering
December 16, 1997

Certified by

Kai Yeung Sunny Siu
Assistant Professor of Mechanical Engineering
Thesis Supervisor

Accepted by

Ain A. Sonin
Chairman, Departmental Graduate Committee
To Mom and Dad, for everything,
and to Nancy, for being there.
AXIOMATIC DESIGN AND NETWORK PERFORMANCE ANALYSIS FOR APPLICATIONS IN HOME-BASED HEALTH CARE

by

JASON DOUGLAS HINTERSTEINER

S.B. Mechanical Engineering
Massachusetts Institute of Technology, 1996

Submitted to the Department of Mechanical Engineering on December 16, 1997 in partial fulfillment of the requirements for the Degree of Master of Science in Mechanical Engineering

ABSTRACT

Home-based health care systems of the future will be designed to enable patients to live independently with the aid of electromechanical devices, without the presence of expensive on-site health care professionals. Nevertheless, there will still exist a need for a patient’s health to be monitored in real-time, so that a sudden deterioration of a patient’s condition can be responded to immediately. Such monitoring will ultimately save lives, prevent or minimize medical complications, and thereby ultimately lower health care costs.

Utilizing the resources and rapid growth of communication networks such as the Internet, constant monitoring of home-based patients is achievable. A “telenursing system” has been designed which can obtain real-time sensor data from patients and their home environments and transmit it over the network to a central monitoring facility staffed by nurses and medical technicians. The system also incorporates network applications, such as video conferencing as well as remote access to patient medical records, in order to facilitate remote diagnosis and early detection of medical problems. The axiomatic design approach was used to create the telenursing system design.

To provide network quality of service to medically-related data, the system design incorporates the use of RSVP, an emerging Internet protocol that reserves resources along the network path from senders to receivers. A simulation model of RSVP has been developed which incorporates two different methods (unicast and multicast) for reserving network resources. This model can determine the practicality of RSVP for use in large-scale network applications where real-time interactive data transmission is essential.

Thesis Supervisor: Dr. Kai Yeung Sunny Siu
Title: Assistant Professor of Mechanical Engineering

- 3 -
# Table of Contents

Abstract 3  
Table of Contents 4  
Table of Figures 7  
Table of Tables 8  
Biography 9  
Acknowledgments 10

## Chapter 1: Introduction

1.1 Home-Based Health Care 14  
1.2 Guaranteeing Quality of Service (QoS) 16  
1.3 Telenursing System 17  
1.4 Network Simulation Model 18  
1.5 Previous Work  
  1.5.1 Monitoring System Design 19  
  1.5.2 RSVP Simulation Models 20

## Chapter 2: Network Quality of Service

2.1 Fundamental Networking Issues  
  2.1.1 Throughput 24  
  2.1.2 Delay 25  
  2.1.3 Fairness 26  
2.2 QoS Control Services  
  2.2.1 Controlled-Load Service 31  
  2.2.2 Guaranteed Service 33  
2.3 Overview of RSVP  
  2.3.1 Reservation Styles 36  
  2.3.2 Using RSVP with QoS Control Services 39  
    2.3.2.1 Sender-TSPEC 40  
    2.3.2.2 Adspec 40  
    2.3.2.3 Flowspec 41  
  2.3.3 Control Messages 41  
    2.3.3.1 Path Messages and Path State 42  
    2.3.3.2 Resv Messages and Reservation State 43  
    2.3.3.3 Teardown Messages 44  
    2.3.3.4 Error Messages 45  
    2.3.3.5 Confirmation Messages 45  
  2.3.4 Traffic Control 46
2.4 QoS Routing
   2.4.1 Unicast Routing: OSPF 48
   2.4.2 Multicast Routing
      2.4.2.1 IGMP 51
      2.4.2.2 MOSPF 52
   2.4.3 QoS Routing 53

CHAPTER 3: AXIOMATIC DESIGN OF THE TELENURSING SYSTEM 57

3.1 Overview of Axiomatic Design 58
   3.1.1 Problem Definition 59
   3.1.2 Creative Process 60
   3.1.3 Analytical Process
      3.1.3.1 The Independence Axiom 61
      3.1.3.2 The Information Axiom 64
   3.1.4 Ultimate Check 65

3.2 Design of the Telenursing System 65
   3.2.1 Modular Interface with Home-Based Sensors (DP1) 70
   3.2.2 External Application Interface (DP2) 73
   3.2.3 Network Interface (DP3) 77
   3.2.4 Computer-assisted Health Care Practitioner (DP4) 84
   3.2.5 Graphical User Interface (DP5) 88
      3.2.5.1 Window-based GUI 89
      3.2.5.2 Security Module 90
      3.2.5.3 Network Module 91
      3.2.5.4 Sensor Module 92
      3.2.5.5 External Application Module 92

3.3 Overall System Design 93

CHAPTER 4: NETWORK SIMULATION MODEL 97

4.1 Overview of OPNET 97

4.2 Establishing Reservations 98
   4.2.1 Multicast-Interactive Reservations 99
   4.2.2 Unicast-Interactive Reservations 100

4.3 Simulation Model 102
   4.3.1 Establish a Session ID 105
   4.3.2 Sending a Path Message 106
   4.3.3 Sending a Resv Message 108
   4.3.4 Sending Data Messages 111
   4.3.5 Ending the Reservation 112
CHAPTER 5: CONCLUSION 119

5.1 Future Work on the Telenursing System Design 120
5.2 Future Work on the Simulation Model 121

APPENDICES:

A: Specification of the TSPEC 123

B: Process Model Descriptions 125
   B.1 Host Reserve Application (Host_rapp) 125
   B.2 Host Unicast Non-Reserve Application (Host_uapp) 129
   B.3 Host RSVP Daemon (Host_rsvpd) 131
   B.4 Host Multicast Daemon (Host_igmpd) 135
   B.5 IP Network Layer (Rsvp_IP) 138
   B.6 Router RSVP Daemon (Router_rsvpd) 140
   B.7 Router Multicast Daemon (Router_igmpd) 144
   B.8 Router Transmission Queue 146

C: Packet Formats 149
   C.1: Host Process Model Formats 149
   C.2: RSVP Daemon Process Model Formats 150

D: RFCs and Internet-Drafts 155

E: List of Abbreviations 156

ENDNOTES:

Chapter 1 159
Chapter 2 160
Chapter 3 161
Chapter 4 162
### Table of Figures

**Chapter 1:**
- 1-1: Schematic diagram of the telenursing system.  

**Chapter 2:**
- 2-1: Throughput characterization.  
- 2-2: Example of queuing delay.  
- 2-3: The fairness problem for a DQDB network.  
- 2-4: Example of flow merging in an FF-style reservation.  
- 2-5: Schematic diagram of RSVP in a host and router.  
- 2-6: Schematic diagram showing different router types.

**Chapter 3:**
- 3-1: Information analysis graph.  
- 3-2: Schematic diagram of cable modem internet connection.  
- 3-3: Tree diagram for the telenursing system (FRs).  
- 3-4: Tree diagram for the telenursing system (DPs).  
- 3-5: Module-junction structure diagram for the telenursing system.  
- 3-6: Flow chart diagram for the telenursing system.

**Chapter 4:**
- 4-1: Multicast-interactive reservation model for two nodes.  
- 4-2: Unicast interactive reservation model.  
- 4-3: Telenursing center host node (ethernet).  
- 4-4: Telenursing center gateway node (ethernet).  
- 4-5: Backbone router.  
- 4-6: Telenursing center subnetwork configuration (ethernet).  
- 4-7: Network topography for connected telenursing subnetworks.  
- 4-8: Network topography of a cable-modem network.  
- 4-9: Sample metropolitan telenursing system network topology.  
- 4-10: Sample rural telenursing system network topology.

**Appendices:**
- B-1: State transition diagram for the host reserve application.  
- B-2: State transition diagram for the host unicast non-reserve application.  
- B-3: State transition diagram for the host RSVP daemon.  
- B-4: State transition diagram for the host IGMP daemon.  
- B-5: State transition diagram for the IP network layer.  
- B-6: State transition diagram for the router RSVP daemon.  
- B-7: State transition diagram for the router IGMP daemon.  
- B-8: State transition diagram for the router transmission queue.
# Table of Tables

## Chapter 2:
- 2-1: Reservation styles available in RSVP. 37
- 2-2: Reservation requests and aggregate reservation states. 39

## Chapter 3:
- 3-1: Core functional requirement for the telenursing system. 66
- 3-2: Top-level functional requirements. 66
- 3-3: Decomposition of interface with home-based sensors (DP1). 70
- 3-4: Decomposition of interface with external applications (DP2). 74
- 3-5: Decomposition of interface with network (DP3). 79
- 3-6: Information content of networking technologies for the home. 81
- 3-7: Decomposition of computer-assisted telenurse (DP4). 85
- 3-8: Decomposition of computerized agents (DP42). 86
- 3-9: Decomposition of graphical user interface (DP5). 89
- 3-10: Decomposition of security module (DP52). 90
- 3-11: Decomposition of network module (DP53). 92
- 3-12: Junction types and relationships for system architecture diagrams. 95

## Chapter 4:
- 4-1: Format of Path State Block (PSB). 107
- 4-2: Format of Reservation State Block (RSB). 110

## Appendices:
- B-1: Application parameters for the host reserve application. 127
- B-2: Application parameters for the host unicast non-reserve application. 127
- B-3: Action taken when a packet arrives from the host’s RSVP daemon. 130
- B-4: Action taken upon packet arrival from a host reserve application. 132
- B-5: Action taken upon packet arrival from the IP network layer. 134
- B-6: Application parameters for the host’s IGMP daemon. 136
- B-7: Application parameters for the IP network layer. 139
- B-8: Action taken upon packet arrives from the IP network layer. 142
- B-9: Application parameters for the router’s IGMP daemon. 145
BIOGRAFHY

Jason D. Hintersteiner received a Bachelor of Science in Mechanical Engineering from the Massachusetts Institute of Technology in June, 1996. As an undergraduate, he specialized in control systems, working on projects involving robotics, digital control, computer-numerical control, and error compensation. He also has experience with the design and prototype implementation of an educational controls laboratory. As a graduate student, he has served as a teaching assistant for an undergraduate controls course, and his research has incorporated networking and systemic design techniques.

His current research interests include the design of complex systems, focusing on the identification of adequate design processes, determination of the optimal role of software control and compensation within the design of complex mechanical systems, and construction of tools to help integrate technologies from various engineering and scientific disciplines into system design.
ACKNOWLEDGMENTS

The work described in this Master's thesis is part of ongoing research for a comprehensive home-based health care system under development by the Total Home Automation and Health Care / Elder Care Research Consortium. This Consortium operates under the auspices of the Mechanical Engineering department at MIT.

I would like to acknowledge and thank Professor Kai Yeung Sunny Siu, along with the d'Arbeloff Laboratory for Information Systems and Technology and the MIT Mechanical Engineering department, for providing the funding, facilities, and equipment necessary for this project. I have gained substantial experience, and I am very appreciative of this opportunity to apply my mechanical engineering background to a field of research that normally is excluded in conventional mechanical engineering projects.

I would also like to acknowledge Professor Nam P. Suh, for providing me with the inspiration and the impetus to use the axiomatic design approach for the design of the telenursing system.

In addition, I would like to acknowledge my family and friends for all of their support and encouragement.
CHAPTER 1: INTRODUCTION

The health care industry is a rapidly growing segment of today’s economy. People’s life expectancies are higher, and as the baby-boom generation of the 1940’s and 1950’s gets older, the demand for health care will continue to grow dramatically. In addition, people are more aware of and concerned about their health and health care today than ever before.

One issue affecting the health care climate is that the population as a whole is becoming more committed to health and more in tune with health care. Healthier diets, increased exercise and elimination of harmful behaviors such as smoking are in turn leading to a healthier population overall.[1]

Health care costs, however, are skyrocketing, which is creating considerable interest in developing systems and products to provide high quality health care at lower costs. For patients who are both financially self-sufficient and reasonably capable of taking care of themselves, home-based care has been a viable option for many years. Home-based care, however, is very expensive, and as it becomes more pervasive in the next few years, new technologies must be developed to make home health care systems both more affordable and more sophisticated.

The Total Home Automation and Health Care Research Consortium, within the auspices of the MIT d’Arbeloff Laboratory for Information Systems and Technology, is devising a holistic approach to the problem of home-based health care by developing models of both the human patient and the home environment. By integrating the vast amount of information that these sophisticated models require, new health care applications are being created.[2] Among the systems currently under development by the Consortium are sensors which can gather a great deal of information about the patient’s
health status and his or her surrounding home environment. Once sensor data is acquired, it must be transmitted, in real-time, to health care practitioners in a centralized monitoring facility, so that the patient can be frequently, if not constantly, monitored.

The goal of this Master's thesis is to design a *telenursing system* -- a communications system that is capable of transmitting real-time medical data generated at a patient's home to a centralized monitoring facility. This system will reduce overall costs by not requiring the presence of on-site health care practitioners in patient homes, while providing a fast-response mechanism for the early detection of medical problems and emergencies. The system is also capable of connecting to distributed external information sources, including doctors, hospitals, medical databases, and medical information libraries. *Figure #1-1* shows a schematic diagram of the system.

![Schematic diagram of the telenursing system. Several patient homes are connected, via a communication network such as the Internet, to monitoring centers, hospitals, and on-line medical databases.](image-url)
Because of the time-sensitive nature of medical data, the development of the telenursing system requires challenging networking problems to be overcome. The information exchanged between the patient and the telenursing center includes data from various sensors located on the patient and around the home as well as video and audio conference data, so that the patient and medical practitioners can readily communicate. In current networking applications, however, bandwidth is limited and delay through a network is often inconsistent. Since the timely and accurate receipt of such data may often prove crucial in saving lives, an adequate quality of service (QoS) must be provided by the network in order to guarantee certain throughput, limit or minimize delay, and maximize the fairness of access to shared network resources.

The use of the Internet to provide communication between the patient's home and health care practitioners is very desirable, since the network is ubiquitous, reasonably inexpensive, and capable of carrying a large amount of data. However, the Internet was originally designed for bulk transfer of data, and has proven to be inadequate for many real-time applications. Thus, extensive research is underway to develop methods of transporting real-time traffic over the Internet by providing QoS guarantees.

The rest of this chapter introduces the research conducted in the development of the telenursing system. Section 1.1 provides some background material on home-based health care. Section 1.2 introduces the challenges in providing service quality guarantees for network applications. Section 1.3 provides an overview of the telenursing system design. Section 1.4 discusses issues arising in the development of a network simulation model of the telenursing system. Section 1.5 outlines some of the previous work done in the development of similar technologies.
1.1 HOME-BASED HEALTH CARE

Patients who are suitable to receive care at home must be self-sufficient, though they may require assistance in doing ordinary day-to-day tasks, and may also need assistance in monitoring their health conditions. At home, however, a patient is able to maintain a certain degree of independence while living in a comfortable and familiar environment.

Not only is home care popular with consumers because it can keep patients out of more expensive nursing homes, it's increasingly endorsed by medical insurers as well. As policies become more favorable [to home care], more people will gravitate to it.[3]

Typically, a part-time or full-time caregiver will help the patient with day-to-day tasks, such as getting in and out of bed, going to the bathroom, cooking, cleaning, etc. While less expensive than a hospital or nursing home stay, home-based care as it typically exists today can still be costly. In many cases, doorways must be widened so that wheelchairs can pass through, lifts must be installed so that the patient can get up and down the stairs, and other devices must be placed in the home to enable the patient to continue to live there.

Even if such mechanical devices and home structural alterations are not necessary, a large portion of home medical care costs go towards hiring a caregiver to monitor a patient's health and to assist in the patient's daily activities. Since the home health care industry is unregulated, the quality of the caregiver cannot be assured, and as a result there are often cases of fraud and neglect committed by the very people who are supposed to be providing care:

Each day, millions of elderly and disabled Americans open their doors to people hired to take care of them. Too often, they're letting in
thieves, thugs and scam artists. In the rush to embrace home health care as a better alternative to nursing homes, some of the nation’s most frail and vulnerable residents end up victims -- unable to defend themselves in their own homes, too afraid or confused to speak out. And there is almost no place for them to turn for protection.

Federal and state rules that govern care in nursing homes don’t apply in private homes. Criminals can easily slip into home care; reports of theft, abuse and billing fraud are rising sharply. Mostly, the damage goes undetected, unreported and uncorrected... Although experts believe as many as 7 million patients pay for home care, most have no assurance that their care givers aren’t convicted criminals or incompetent...

To be sure, troublemakers are a small minority in home care, where legions of care givers do good work, often for low wages. It's far easier to find cases where home care aides have saved lives than ruined them. But the industry is growing so fast -- this year Americans will spend $36 billion on home care -- and demand for workers is so fierce, that bad people are getting into it at alarming rates.... As the nation's population ages, the problem will only get worse.\[41\]

In order to decrease costs and ensure that patients are receiving quality care, there is considerable interest in developing technologies which can autonomously monitor a patient’s health status and assist the patient in day-to-day tasks. The goal is to remove, or at least to reduce drastically, the dependency on human caregivers by implementing available technology:

At the same time that consumers are becoming more health-smart, "smarter" technology is also emerging. Expert systems -- computer systems with decision-making capabilities -- will make the best medical knowledge available to caregivers, as well as to consumers in their homes. The systems will also be able to present the material in ways that reflect the users’ learning styles and levels of knowledge. Expert systems will also... be used in homes within the next 10 years and will compete with some of the work caregivers perform. A variety of personal body function-monitoring tools will also be available.\[5\]

While many companies are already providing home health care products and services,\[6\] they typically do not provide a comprehensive approach to home-based medical care.
Furthermore, systems that do transfer data between a patient’s home and a centralized monitoring facility typically only transfer a limited amount of data over the telephone network, or will transmit data in bulk (i.e. not in real time). Therefore, there is a substantial need for systems which can transfer larger amounts of real-time medical data from a patient's home to a centralized monitoring facility. Most health care companies still rely upon on-site caregivers to provide care, utilizing emergency response systems, such as wearable beepers or voice-activated systems, to maintain a direct connection to health care providers.

1.2 Guaranteeing Quality of Service

There are several networking issues that arise from the development of the telenursing system. The nature of this application requires that large amounts of data be transmitted in very short periods of time. As the telenursing system is scaled upwards to cover entire regions, the whole country, or even the entire world, the demand that will be placed on its supporting communication network will drastically increase.

In order to satisfy the demands of the telenursing system, a guarantee of the quality of service that the network will grant to real-time data transmissions must be provided. To support specific levels of service quality on the Internet, which was originally designed only to provide “best-effort” service, protocols have been and are continuing to be developed to reserve network resources for individual data streams. While there are no established and proven ways, at present, to provide resource reservation on the Internet, the problem is undergoing extensive research in the networking industry.
In *Chapter 2*, a detailed background is provided of the technology that has been developed to attempt to provide QoS guarantees over the Internet. This technology includes the creation of QoS control services, which specify the type of service qualities the network is capable of, as well as the development of new network protocols designed to provide these QoS control services. The main protocol discussed is RSVP, or Resource Reservation Protocol. This protocol is designed to maintain information within the memory of network routers that control the reservation of resources for unicast and multicast applications. In order to accomplish this, the protocol must establish a path for the data through the network and reserve resources along that path for the data flow.

### 1.3 Telesnursing System

In order to monitor home-based patients successfully from a centralized monitoring facility, a system must be designed which can reliably and consistently obtain, transmit, and process real-time medical data from a patient. In the patient’s home, the system must aggregate all of the relevant data from the assortment of sensors located on the patient and around the home, and transmit it to the centralized monitoring site. At the telenursing center, the system must be able to gather the data received from the patient’s home, interpret it, monitor the data for any abnormalities, and send relevant information back to the patient’s home. Since these transmissions contain vital information about the health of the patient, it must be received in real-time with a guaranteed maximum delay in order to be pertinent. The overall functional requirements of this system are as follows:

- monitor sensor data in real-time;
- incorporate applications which facilitate home-based care;
- transmit sensor and communication data over a network with a specific quality of service for bandwidth and latency;
interpret sensor data and act accordingly;
be user-friendly to both medical personnel and patients.

In Chapter 3, a proposed conceptual design for the telenursing system is presented, along with the axiomatic design approach used to generate it. Even though the design outlined here is still in the conceptual stage, several fundamental issues emerge. While the implementation of the system and the detailed design of many of its sub-components are beyond the scope of this thesis, the conceptual design provides a clear understanding of the network requirements this application will ultimately need.

The axiomatic design approach ensures that the system design is generated in a logical and coherent manner from its inception, and that the design addresses all of the system's functional requirements. In addition, axiomatic design is also very useful in alerting the designer to potential problems that may be encountered during the implementation of such a system, as well as methods for cleanly and smoothly incorporating design changes. Alternative techniques to systemic design often involve a great deal of trial-and-error ad hoc approaches which can waste time and money, as well as lead to an inferior product or system which does not adequately address the functional requirements of the system.

1.4 Network Simulation Model

In Chapter 4, a simulation model is presented which has been developed to investigate the applicability of the QoS networking technology discussed in Chapter 2 in meeting the network requirements of the telenursing system. Specifically, the simulation model uses RSVP to reserve network resources along a path from senders to receivers. The simulation model has been created in the OPNET modeling environment, etc.
incorporates the core control messages and procedures required to establish a reservation on network nodes between senders and receivers. The telenursing applications (i.e. sensors, video conferencing, etc.) are modeled as sources and sinks for network data, which first generate requests for specific QoS requirements before transmitting the data itself. By manipulating these models, a wide range of potential network applications for the telenursing system can be evaluated.

1.5 PREVIOUS WORK

The telenursing system presented in this thesis contributes to two formerly unrelated areas of current engineering research. First, it provides a good case-study example of applying axiomatic design to large-scale systems which have subcomponents that do not fall within the more conventional scope of mechanical design. Second, it is a large-scale network application, and as a result forms the basis of a good case-study example to address performance and behavior issues of the RSVP protocol. The following two subsections outline the current state of work in these research areas.

1.5.1 Monitoring System Design

While there is an extensive array of products and services that are made available by the medical technology industry, there is a lack of monitoring systems which comprehensively integrate new technologies into home-based patient care. One of the only examples of a system currently available on the market which begins to meet the challenges involved with high-tech home-based patient monitoring systems is called the HANC Network. This system provides a home-based computerized station capable of processing information from integrated equipment that measures such things as heart rate.
and blood pressure. Should a medical problem arise, the system can place a telephone
call to a computer at a central monitoring facility and transmit the relevant data. This
system, however, has two notable limitations. First, the use of analog phone lines to
establish the communications link between the home and the monitoring facility prevents
the use of the system to transmit high bandwidth medical data, such as real-time video
streams. As a result, the only video-based information transmitted by this system is still
video images. The use of analog phone lines also constrains the number of simultaneous
connections to different patients that the monitoring center can maintain. Second, the
system requires active participation from the patient, making the system impractical for
mobility-impaired patients.

To date, there are no references available indicating that axiomatic design has
been used in the development of any patient monitoring system. Since the initial
development of the technique, it has mainly been used on projects in academia involving
mechanical design. An overview of axiomatic design is provided in detail in Chapter 3.

1.5.2 RSVP Simulation Models

While individual RSVP testbeds have been developed for the initial design of the
protocol, no major studies to date have been published which investigate how well this
protocol will perform in large-scale network applications. The only simulation model
publicly available to address performance analysis has been developed by the Networking
and Simulation Lab, C3I Center at George Mason University.\(^{(11)}\) The goal of this project
was to create a simulation of IP multicast utilizing RSVP and QOSPF (a proposed QoS
routing protocol discussed in Chapter 2), in order to analyze the performance and
behavior of these protocols under various network topologies. Their study was primarily
concerned with the performance of QOSPF as a routing protocol, and as a result their
model of RSVP was not sufficiently detailed for use in the simulation presented in this
thesis. While the simulation model presented in Chapter 4 is loosely based upon the
model from George Mason University, the RSVP protocol model itself has been almost
completely rewritten from scratch. Accordingly, the model developed here provides a
clearer picture of how the protocol behaves when used in large-scale network
applications.
This page intentionally left blank.
CHAPTER 2: NETWORK QUALITY OF SERVICE

Data acquired from real-time applications, such as medical sensors or video conferences, have a very limited timespan of usefulness. If the information is not received within a certain amount of time, it becomes worthless. Because of this, the Internet has not been widely used to transmit critical real-time data, as it was designed only to provide a best-effort service. Such service quality makes no assurances as to when the data will reach its destination.

There is considerable interest, however, in using the Internet for transmitting real-time data. The Internet is widespread across the globe, is relatively inexpensive for an individual user to use, and has the capacity to transmit large amounts of data within short periods of time. In addition, considerable work has been done in the past few years in multicasting (i.e. transmitting data from several senders to several receivers). In order to transmit real-time data, however, the Internet must gain the additional functionality of providing quality of service (QoS) from senders to receivers when required to do so.

The design of mechanisms to add this functionality to the Internet is presented in this chapter, in order to provide background material on the QoS networking technology utilized by the telenursing system design presented in Chapter 3. In Section 2.1, the issues which create the need for QoS control services are presented. In Section 2.2, the QoS control services that have been defined by the Internet Engineering Task Force (IETF) are introduced. In Section 2.3, an overview of RSVP is presented, and in Section 2.4, QoS routing technology designed to work with RSVP is discussed.
2.1 Fundamental Networking Issues

Since network resources are finite, methods must be created to share these resources between many users. As a result, several issues emerge which must be addressed when developing such methods. These issues include throughput, delay, and fairness, as discussed in the following subsections.

2.1.1 Throughput

The throughput of a network describes the speed at which data can travel along links between two nodes. This is usually measured in terms of bandwidth, and describes how much information can be transmitted per unit time. Since Internet links are shared, the throughput of any particular data flow may only be a fraction of the total bandwidth capacity of the link, as shown in Figure #2-1.

![Source Receiver Diagram]

Figure #2-1: Throughput characterization. The amount of data per unit time from a particular stream may only be a fraction of the link’s total bandwidth.

While the total amount of bandwidth devoted to an individual data stream is an issue, network designs are typically more concerned with the ratio between the total amount of bandwidth a particular data stream desires and the bandwidth capacity of the link. For example, obtaining a 64 kB/sec share of bandwidth on a 1.5 MB/sec link is much easier than obtaining the same bandwidth share on a 128 kB/sec link.
2.1.2 Delay

Even if throughput is high, a data packet can be delayed as it traverses the network. Several mathematical models exist for predicting the latency (i.e. delay) that packets will endure under various network configurations.[1] The primary sources of delay in a network can be characterized as follows:

- **Propagation**: This characterizes the amount of time it takes for a data packet to propagate along a link. This is determined by comparing the time the first bit is transmitted by the source to the time the last bit is received by the receiver. This delay is typically small if not negligible, as it usually results from the speed of light limitation of signals traversing electrical or optical media. When using satellite links or other links which traverse large distances, however, the effect of this type of delay can become significant.

- **Processing**: Once a node receives a packet, it incurs a delay as it is processed and routed to the queue of the correct outgoing link. Assuming that routers possess fast processors and that the routing algorithms are not overly complex, this delay is also typically small if not negligible.

- **Queuing**: Once a packet is processed, it is placed in a queue to await transmission along an outgoing link. If the node is very busy, or if the outgoing link has less bandwidth than the incoming link (as shown in Figure #2-2), the packet can wait for a relatively long time in the queue. If a queue fills up to its capacity, any further packets attempting to enter the queue will be dropped (i.e. lost), which can be very detrimental to some applications,
such as bulk data transfer. In most common network topologies, queuing delay is much larger than propagation and processing delay.

![Diagram](image)

**Figure #2-2:** Example of queuing delay. Data packets may be queued for a long time when the outgoing link has a much smaller bandwidth than the incoming link.

The total delay $T$ in a network is often given by application of Little’s Law:

$$T = \frac{N}{\lambda}$$

Here, $N$ is the number of packets being serviced (i.e. a measure of the throughput), and $\lambda$ is the Poisson packet arrival rate. This relationship shows that delay is proportional to the ratio of the queue size over the packet arrival rate.

### 2.1.3 Fairness

Many network topologies will tend to favor some traffic sources over others, even though this favoritism is not always obvious. It is desirable, therefore, to design networks with routing and queuing algorithms which will distribute the resources of the network in a reasonably fair manner. Since fairness issues may not be immediately obvious in a network design, it may be difficult to detect problems before the network is implemented.

A good example showing the fairness problem is shown in **Figure #2-3**. This example shows a distributed queue dual bus (DQDB) network, which is a configuration often used in metropolitan area networks. Each unidirectional bus sends packets in 53 byte "slots". When the slot contains data, a "busy" bit is set. The problem with this
scheme is that nodes close to the head of a given bus get higher priority, so that nodes further along the path have a larger delay. The solution adopted for this problem is for a node to set a "request" bit in one frame going in the opposite direction whenever it has more data to send, so that the upstream nodes will release an empty slot.

Figure #2-3: The fairness problem for a DQDB network. Fair access to network resources are affected by the network topology and the routing algorithms.

When the propagation delays are large, however, this scheme fails. For a 50 km bus between two neighboring locales, approximately 60 slots occur between the two nodes. If both nodes are attempting to transmit large files and the upstream node starts first, it will saturate most of the network, since the downstream node will only receive one free slot in 120. Similarly, if the downstream node starts first, it will keep passing requests to the upstream node, which will force the upstream node to allow several empty slots through, leaving it with about 10% of the bandwidth in steady-state. The solution to this problem is to allow each node to fill only 90% of the slots to which it is entitled. This provides more idle slots for downstream nodes, and thus fewer requests for idle slots, so that the system eventually settles into a somewhat fair equilibrium state with a loss in efficiency of about 0.1/M, where M is the number of active nodes.

It is plainly apparent, therefore, that even in somewhat straightforward network configurations, fairness can be a significant issue.
2.2 **QoS Control Services**

While many applications may require certain quality of service (QoS) guarantees from the network, different applications will, in general, require different types of service quality. There is a range of potentially useful choices for end-to-end behavior (i.e. source to destination network performance), ranging from no dynamic QoS control to extremely precise and accurate feedback control of QoS control parameters. When designing a QoS control service, therefore, the first and most important task is to define the nature of the desired packet delivery service. Ultimately, applications requiring QoS control services will be concerned about three things: the achieved bandwidth, the delay that packets incur as they travel through the network, and the rate at which packets are lost or dropped because of congestion.

In order to develop QoS control services, the resources required by a network node to provide such services to data flows must be ascertained. To do this, general and service-specific parameters must be identified, along with the relationship between these parameters and end-to-end behavior. It is therefore necessary to identify the control parameters characterizing a network element (i.e. a router, subnetwork, or end-node) that will be used to provide service quality guarantees. In general, these fall into two categories: *local* and *composed*. Local parameters reflect the condition of a single network element, while composed parameters maintain a running total of local values along the network path between the sender and that node. In addition, the functions which use these parameters as input and produce them as output must also be defined.
The implementation of a QoS control service must also specify rules for admission control (i.e. the criteria used as a basis for a network element to allow or deny particular reservation requests) and define what actions should be taken on flows not conforming to their allocated reservations. In addition, a QoS control service must specify rules concerning the renewal of requests and the merger of requests from different receivers for data from the same sources.

It is probable that the Internet will eventually provide support for several different types of QoS control services. As a result, the selection of an appropriate QoS control service will depend on the nature of the data being transmitted. While each QoS control service may have its own service-specific parameters, a core set of parameters are identifiable as necessary for all potential QoS control services: [3]

- **Non-Service-Node flag**: An indicator flag, set when at least one network element along the network path of the flow does not support the QoS control service. In practice, to be a service-aware node, a node must at least know how to identify and deny the QoS control service. Non-service-aware nodes will likely be older nodes on the network which do not know how to process QoS control messages. As a result, this flag would typically be set by a neighboring node which is aware of the non-service-aware node.

- **Number of Hops**: A count of how many network nodes (i.e. hops) the data must pass through from the source to the destination.

- **Available Path Bandwidth**: The amount of bandwidth available to the data flow. To maintain fairness, a node may only allow up to a specified maximum
percentage of its resources to be reserved for QoS data traffic, thereby still allowing some best-effort data traffic to get through.

- **Minimum Path Latency:** The minimal packet delay possible from the network element. This is a measure of the propagation and processing delay of the node, which can then used as a baseline for estimating the lower bound of the total delay along the path.

- **Path Maximum Transmission Unit (MTU):** The maximum size (in bytes) of an IP data packet that a node can send without having to fragment the data into two or more packets. If a node has to fragment a packet, more processing and propagation delay may occur, which may hamper the desired level of service quality and lead to a risk of one or more packets being lost. Usually, if an IP packet is broken into several fragments and one fragment is lost, the entire packet must be retransmitted.

- **Transmission Specification (TSPEC):** This describes the nature of the data traffic being transmitted by the sender. It is used in the reservation request, and a network element which grants the reservation request will expect data traffic flow to be less than or equal to the level stated in the TSPEC. A flow which violates its TSPEC may be subject to regulatory action. More information about the format of the TSPEC is provided in Appendix A.

There are currently two types of QoS control services that have been defined by the Internet Engineering Task Force (IETF), known as *Controlled-Load* service and *Guaranteed* service. These are discussed in the following subsections.
2.2.1 Controlled-Load Service

Under lightly loaded network conditions, best-effort service is adequate for many applications. When the network is lightly loaded, an individual packet will not spend a significant amount of time in a router's queue waiting to be transmitted to the next node in the path. Many current video and audio applications have been developed with buffering or smoothing functions that are capable of reasonably good signal reconstruction, even if a few packets are lost. Problems develop, however, when the network becomes heavily congested with traffic, since queues will fill up to capacity, leading to long queuing delays and dropped packets.

The goal of controlled-load service, therefore, is to provide an approximation of best-effort service under lightly loaded conditions, even when the network is suffering from heavy congestion. As a result, the percentage of dropped packets is very low, and the transmission delay is minimized. This service is designed for adaptive real-time applications, which work adequately over unloaded or lightly loaded networks but are incapable of adapting quickly enough when the network is overloaded. Controlled-load service is also designed for simplicity, as it only requires the core QoS control parameters specified above. As a result, this service does not guarantee specific values for allocated bandwidth, delay, or packet loss. Instead, it only promises lightly-loaded best-effort service. Because of the nature of this service, it has the potential to be useful even when non-service-aware nodes are a part of the data path, as long as these nodes do not become overly congested.

In order to ensure adequate bandwidth and packet processing resources, an admission control scheme is used. This scheme examines link bandwidth, router buffer
space, and router computational capacity. If a flow conforms to the TSPEC, it will not have any data bursts over the size specified by its TSPEC’s token bucket rate $r$ and token bucket depth $b$. In return, the flow will suffer little to no queuing delay, and little to no congestion loss.

If a flow violates its TSPEC, meaning that the amount of data sent in time $T$ is greater than $rT+b$, then the node must regulate the flow. In practice, most implementations should be ready to allocate more bandwidth than specified by the TSPEC token rate, since a burst of traffic above the token rate can fill a queue, and if the flow then returns to the token rate, the queue may never clear. (If the source is sending packets at less than the specified token rate, the extra time available can be used to clear out the queue.) In order to be fair to other well-behaved flows when a particular data flow is exceeding its specified TSPEC, the node should be able to provide QoS control services to other data flows as well as prevent significant interference with regular best-effort data traffic. The node has several options for regulating a nonconforming data flow, including dropping the excess traffic, relegating the excess traffic to best-effort service, or degrading the overall QoS control service for the data flow.

Several proposals have been suggested for an appropriate algorithm to use for admission control. One of the more straightforward ones is “measurement-based admission control”. Given a time measurement window $T_{\text{win}}$ broken up into an integral number $n$ sample periods $S$, the average load per sample is given as follows:

$$L_i = \frac{\Sigma \text{bytes receiving controlled-load service}}{S_i} \quad (i = 1, 2, ..., n)$$
After each measurement window $T = nS$, the highest load average $L$ that occurred during the window (i.e. $L = \max \{L_i\}$) is determined. A new data flow requesting controlled-load service will be admitted if the following condition is met:

$$L \leq \psi C - \kappa r$$

where $C$ is the link’s bandwidth capacity, $r$ is the token bucket rate of the new data flow requesting service, and $\psi$ is an aggregate loading factor which regulates the amount of resources dedicated to controlled-load traffic. This is done for fairness, so that at least some best-effort traffic can still use resources on the node. $\kappa$ is the flow effect factor, and indicates the assumption made by the node as to how much the flow will actually require the token bucket rate it has requested. When $\kappa = 1$, the node assumes that the flow will enter the node at a constant bit rate $r$. Ideally, a source will specify $r$ proportional to the expected short-term burst traffic, so that the actual rate will be less than $r$, and therefore $\kappa$ is less than 1. Upon admission of a new data flow, $L$ is automatically increased to $L+r$.

### 2.2.2 Guaranteed Service

Unlike controlled-load service, guaranteed service provides a guaranteed amount of bandwidth and a maximum expected delay over a specific path through the network. As a result, packets are guaranteed to arrive within a certain delivery time. This type of service is expected to be very useful for applications with firm real-time requirements, such as audio and video playback and real-time remote control applications. Under lightly loaded conditions, the actual delay may be much less than the guaranteed maximum delay. Unfortunately, every element along the path must support this service for it to be useful, since in general a non-service-aware node along the path will not
maintain the bandwidth and delay requirements for the flow. In addition to the general characterization parameters listed above, guaranteed service also utilizes several service-specific parameters, including:

- **Receiver Specification (RSPEC):** Once the receiver receives the TSPEC, it can generate an RSPEC to request a specific level of QoS that matches both the sender’s capabilities and the receiver’s network constraints. The RSPEC contains two parameters: the desired rate $R$ in bytes per second, and a slack term $S$ which indicates the amount of slack time (in microseconds) that nodes can use to reduce the service quality under overloaded conditions.

- **Exported Information:** Since, in general, the flow of data is not “fluidic” through the network, these parameters include a rate-dependent error $C$ as well as a rate-independent error $D$. Nodes will also export the cumulative totals for these errors $C_{tot}$ and $D_{tot}$, as well as sums between reshaping nodes $C_{sum}$ and $D_{sum}$.

When a flow exceeds its TSPEC, a policing algorithm will attempt to reshape the traffic until it conforms to the parameters listed in the TSPEC. This may happen at branch points on the multicast routing tree where the TSPECs from each branch are not identical, as well as at merging points where traffic from two or more sources for the same reservation may be merged. The reshaping process involves delaying the nonconforming traffic in a special buffer regulated by the token bucket and peak rate parameters in the TSPEC. Little actual delay is caused at the reshaping point, and data loss should not occur if the TSPEC is accurate and the reshaping buffer itself is not full.
In order to implement guaranteed service, the latency of the path must be determined. This is typically calculated by measuring the delay of the first packet through the network, and treating that quantity as an upper bound on the latency. Other calculations can be performed to make estimates of the queuing delays at each node. It would seem that more work needs to be done on this QoS control service, in order to predict more accurately the delays that will be encountered in the network.

2.3 **OVERVIEW OF RSVP**

The RSVP protocol is used by a host computer, on behalf of a particular application data stream, to request a specific quality of service (QoS) from the network.[10] This request is receiver-oriented, meaning that the receivers request the appropriate amount of network resources they need along the reverse of the data path (i.e. from the receiver to the sender). RSVP has the following attributes:[11]

- makes resource reservation for both unicast and multicast applications;
- makes simplex (unidirectional data flow) reservations;
- assigns the responsibility for initiating and maintaining the reservation state to the receiver (receiver-oriented reservations);
- allows support for dynamic multicast group membership changes and automatic adaptation to routing changes (maintains "soft-state" in the routers);
- defines interfaces to interact with current and future routing protocols (RSVP is not intended to be an independent routing protocol);
- keeps traffic control and policy control modules independent, to allow network administrators flexibility in specifying reservation policies;
- provides transparent operation through non-RSVP capable routers (backwards compatible), though resource reservation may not be possible;
- provides several reservation models, or styles, to fit a variety of applications;
- remains compatible with both IPv4 and IPv6.
The core information required for a reservation request in RSVP consists of a *Flowspec*, which describes the desired QoS, and a *Filterspec*, which indicates the set of data packets (i.e. the flow) to receive the QoS. In general, the *Filterspec* can select any arbitrary subset of packets based on the contents of the packet headers, though in practice the sender's IP address and the application's transport layer source port should be sufficient. The selection of senders to be included in a reservation request can be made via various reservation styles, as discussed in *Section 2.3.1*.

Due to abstraction layers in the design of the protocol, some of the core information objects carried in RSVP control messages are actually opaque to RSVP. They are instead interpreted directly by independent algorithms in the hosts and routers. This allows individual network administrators to have some flexibility in determining appropriate admission and policy controls. These objects are discussed in *Section 2.3.2*.

RSVP control messages making reservation requests flow upstream (i.e. in the reverse direction of the data flow) from receivers to senders, though some control messages carrying information about the flow and the state of the sender flow downstream (i.e. in the direction of the data flow) from senders to receivers. These control messages are discussed in *Section 2.3.3*.

At each intermediate node along the path, an incoming reservation request must be either accepted or rejected. This process of reservation approval is called *traffic control*, and is discussed in *Section 2.3.4*.

### 2.3.1 Reservation Styles

A receiver making a reservation has the option of explicitly stating a list of all of the sources from whom it wants to receive data traffic, or using a wild card to select
implicitly all of the sources transmitting data to a particular multicast group. In the explicit case, a \textit{Filterspec} must be specified for each sender, though the receiver has the option of creating a distinct reservation with each sender or of specifying a single reservation to be shared amongst the packets from all of the senders. In the wild card case, a shared reservation request is assumed, since an explicit list of senders is never specified by the receiver. As a result, there are three reservation styles available, as summarized in Table \#2-1 below. Because of the way these styles are defined, they are all mutually incompatible in terms of flow merging at intermediate nodes.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Sender Selection & Distinct Reservation & Shared Reservation \\
\hline
Explicit & Fixed-Filter (FF) Style & Shared-Explicit (SE) Style \\
 Wildcard & \textit{undefined} & Wildcard-Filter (WF) Style \\
\hline
\end{tabular}
\caption{Reservation styles available in RSVP.\cite{12}}
\end{table}

- **FF-style:** This reservation style creates distinct reservations for data packets from particular senders. This style is good for applications where the numbers of senders and receivers are small, and in applications such as video streams from a video conference where data may be coming from different sources simultaneously. Symbolically, a FF-style reservation can be represented by: \text{FF}(S_1(Q_1), S_2(Q_2),...), where \( S_i \) is a particular sender and \( Q_i \) is the sender's \textit{Flowspec}.

- **WF-style:** This reservation style creates a single reservation that is shared by flows from all upstream senders on a particular multicast group. It can be thought of as a shared "pipe" whose size is the largest of the resource requests from all of the receivers, regardless of the number of senders using it. This
type of reservation is automatically extended to new senders as they appear. As a result, this style is good for multicast applications where there may be a large number of receivers joining and leaving during a session, such as a "broadcast" of a live concert or sporting event. Symbolically, a WF-style reservation can be represented by: WF(*{Q}).

- **SE-style:** This style of reservation creates a single reservation shared by a selected list of upstream senders. This style is good for multicast applications, such as audio conferences, where there may be several senders though data is not likely to be transmitted simultaneously. Symbolically, a SE-style reservation can be represented by: SE([S_1, S_2, ...]{Q}).

For the telenursing system, FF-style reservations are likely to be the most applicable. In many cases, the only senders will be the computers used by the telenurse and the patient, whose IP addresses will be known. In cases where there are other participants, such as medical specialists or family members participating in a video conference, the IP addresses of these nodes can be obtained immediately before the reservation is made.

An example of how FF-style reservations are merged is given in Figure #2-4. In this example, the router has four interfaces a, b, c, and d, connected to three sources [S_1, S_2, S_3] and four receivers [R_1, R_2, R_3, R_4] as shown. Table #2-2 illustrates the reservation requested by each receiver, the aggregate reservation requested, and the ultimate reservation state that is established for each sender over each interface. For simplicity, the Flowspec parameters are specified as integral multiples of a base value Q.
Figure #2-4: Example of flow merging in a FF-style reservation.\textsuperscript{[13]}

<table>
<thead>
<tr>
<th>Interface</th>
<th>Reservation Request by Receivers</th>
<th>Aggregate Reservation Requested</th>
<th>Reservation State Established for Senders</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>none</td>
<td>none</td>
<td>FF (S1{4Q})</td>
</tr>
<tr>
<td>b</td>
<td>FF (S1{3Q+},S3{Q}) from R2</td>
<td>S1{3Q} S3{Q}</td>
<td>FF (S2{5Q})</td>
</tr>
<tr>
<td>c</td>
<td>FF (S1{3Q},S2{5Q}) from R1</td>
<td>S1{3Q} S2{5Q}</td>
<td>none</td>
</tr>
<tr>
<td>d</td>
<td>FF (S1{Q},S2{Q}) from R3</td>
<td>S1{4Q} S2{Q}</td>
<td>FF (S3{Q})</td>
</tr>
</tbody>
</table>

2.3.2 Using RSVP with QoS Control Services

The QoS control services described in Section 2.2 were specified independently from the design of RSVP, even though RSVP is expected to be able to provide these services. Because of this logical distinction, there are objects carried within RSVP.
control messages whose format is opaque to RSVP, leaving their contents to be read
directly by independent traffic control modules. There are three objects this applies to:
Sender TSPEC, Adspec, and Flowspec.\(^{[14]}\)

2.3.2.1 Sender TSPEC

The Sender TSPEC, as its name implies, is constructed by the sender and
describes the type of data traffic that the sender is generating. This represents the
maximum parameters that the sender is capable of using for its transmissions, though not
necessarily the actual characteristics of the flow. This object is carried in Path messages
(see Section 2.3.3.1) from the sender towards the receivers, and is never modified by any
of the intermediate nodes.

2.3.2.2 Adspec

The Adspec contains information about the properties of the data path, such as the
availability of specific QoS control services as well as general and specific QoS control
service parameters, along the path from the sender to the receivers. This information is
used by the receiving application in order to make decisions about what resources and
QoS control services to request. Like the Sender TSPEC, the Adspec is carried within
Path messages, though the information in the Adspec is used and modified by the
intermediate nodes in order to reflect an accurate picture of the aggregate network
resources along the path.

The initial Adspec is generated by the sender application, in order to specify its
own QoS control service capabilities. If the sender application does not support a specific
QoS control service, it will omit it from the Adspec. At each intermediate node, the
Adspec is sent to traffic control where each QoS control service listed is updated. Flags are set to indicate if a specific QoS control service is not supported by the node. If the node itself does not support QoS control, a global flag is set. (In practice, this global flag would typically be set by a neighboring node, since the non-service-aware node would, in general, be incapable of processing the Adspec.) The Adspec is returned to RSVP for transmission to the next node once traffic control has updated it.

When the Path message reaches the receiver, the Sender TSPEC and Adspec objects are evaluated. If flags are set indicating that specific QoS control services (or QoS control services in general) are unsupported, the rest of the Adspec data for those control services is considered inaccurate, since at least one node did not properly update the QoS control service parameters.

2.3.2.3 Flowspec

The Flowspec is generated by the receiver application and contains information about the type of QoS control service desired, the receiver’s TSPEC, and the receiver’s RSPEC (in the Guaranteed QoS case). The information travels upstream in Resv messages (see Section 2.3.3.2), and may be used and updated within the intermediate network nodes as it travels towards the sender. The Flowspec may need to be altered at merging points, where two or more receivers requesting the same flow from a specific sender will have their requests merged into one combined request to be sent upstream.\textsuperscript{13}

2.3.3 Control Messages

There are five types of control messages used by RSVP to establish and maintain reservation state, as outlined below.
2.3.3.1 Path Messages and Path State

*Path* messages are sent downstream along the same path followed by the data packets through the network. The information contained within these messages is used to establish "path state" in each node along the path. Path state is maintained by periodic *Path* messages. If no *Path* messages arrive at a given node after a set interval to refresh the path state, the path state will time out so that node resources do not remain allocated to inactive data flows. The information stored in the path state consists of the following:

- the unicast IP address of the previous node's interface (called the PHOP);
- the sender template (i.e. *Filterspec*) indicating the format of the sender's data;
- the *Sender TSPEC*, specifying the traffic characteristics of the data flow;
- the *Adspec*, specifying cumulative information about network resources from the source to the current node;
- the refresh period \( R \) that a node waits before generating new *Path* refresh messages downstream (set randomly between \( 0.5R \) and \( 1.5R \) at the node to avoid synchronization) -- this value can be changed node by node, as a smaller \( R \) speeds up adaptation to route changes in exchange for increasing overhead;
- the local state lifetime \( L \), which is set to \( 1.5(K+0.5)R \), where \( K \) is a small integer (e.g. \( K=3 \)) such that \( K-1 \) successive *Path* messages can be lost before the path state times out.

In addition to the information stored in the path state, *Path* messages may also contain integrity data to authenticate the message, as well as policy data for policy control algorithms used in traffic control. In multicast applications, *Path* messages will be copied at merge points to be sent out over multiple outgoing interfaces, though a different PHOP and *Adspec* may be specified for each interface.
When a new sender has data to transmit, it first sends an initial *Path* message to the receivers, notifying them to instigate a reservation request. QoS-Routing (described in *Section 2.4*) will determine the best route for this initial *Path* message through the network, which then becomes the path of the data flow.

### 2.3.3.2 Resv Messages and Reservation State

A receiver wishing to make a reservation request will generate *Resv* messages. These messages follow the reverse path of the data packets through the network, and are used to create and maintain "reservation state." Like path state, reservation state will expire if not periodically refreshed by *Resv* messages. The information stored in reservation state consists of the following:

- an identification of the session this reservation belongs to (contained within a *Session* object);
- the desired QoS (contained within the *Flowspec*);
- the current list of senders (contained within the *Filterspec*) in FF-style and SE-style reservations;
- the current list of senders (contained within a *Scope* object) in WF-style reservations (since the receiver does not maintain an explicit list);
- the IP address of the next node in the path (called the NHOP);
- the refresh period $R$ that a node should wait before generating new *Resv* refresh messages upstream (as defined above);
- the local state lifetime $L$ (as defined above).

In addition to the information stored in the reservation state, *Resv* messages may also contain integrity data to authenticate the message, policy data for policy control algorithms used in traffic control, and a flag indicating a request for confirmation of the
reservation, once it is established. Upon receipt of a Resv message for a new session (or upon receipt of a Resv message modifying an existing session), an intermediate node will pass the message through traffic control to determine whether the reservation request should be granted or denied.

In multicast applications, Resv messages may also be merged at nodes where two or more receivers are requesting traffic from the same data source. Each Resv message may contain a different NHOP, Flowspec, and Filterspec, which must be merged into one Resv message to be sent to the previous hop.[16]

2.3.3.3 Teardown Messages

When a sender has finished sending data, or when a receiver has finished receiving data, the host has the option of either allowing the reservation to expire after the time-out interval or explicitly tearing down (i.e. remove) the reservation. There are two types of teardown messages, PathTear and ResvTear, which delete path state and reservation state, respectively.

PathTear messages will delete the path state, and any resulting reservation state, along intermediate nodes. These messages are routed and processed like Path messages, except that any enclosed Sender TSPEC or Adspec data is ignored. A PathTear message will propagate through the network until it reaches all of the receivers.

ResvTear messages will delete the reservation state along intermediate nodes. These messages are routed and processed like Resv messages, except that any enclosed Flowspec or Scope data is ignored. In FF-style and SE-style reservations, any subset of the Filterspec can be torn down. A ResvTear message will propagate through the network until it reaches the sender or a merge point. At a merge point, due to the
reservation needs of other receivers, the *ResvTear* message will either not affect the upstream reservation state or require that the upstream reservation state to be modified. In the case where the upstream reservation state must be modified, the intermediate node will generate a modified *Resv* message to be passed upstream.

### 2.3.3.4 Error Messages

When a node has an error processing RSVP control messages, it must notify the offending application of the error. For sender applications, the node will send *PathErr* messages, which travel upstream using the path state to route the message to the previous node. For receiver applications, a similar type of message called a *ResvErr* is defined, which travels downstream towards the receiver using the reservation state to route the message to the next node(s). Each of these messages contain an *Error-spec* object, which specifies where the error occurred along with a code specifying the type of error.\(^{[17]}\) In FF-style reservations, each flow with an error will generate its own *ResvErr* message.

An intermediate node in the network also has the option of creating “blockade state,” in which it automatically blocks persistent requests from a particular receiver that it cannot handle. This safeguard ensures that smaller requests can still be processed, since the offending request is prevented from seizing additional network resources.

### 2.3.3.5 Confirmation Messages

When a receiver application requests a reservation, it will often want confirmation that the reservation has been established. To request this confirmation, a receiver can set a flag in the original *Resv* message. When the reservation state has been established up to the sender, or when the reservation is merged in such a way that the existing upstream
reservation at a merge point is adequate, a confirmation is returned to the receiver in a
*ResvConfirm* message. This message is routed back to the receiver host and forwarded
downstream node by node. It is structured exactly like a *ResvErr* message, except that the
*Error-spec* object only indicates the IP address of the node sending the confirmation.

This confirmation, however, is not a guarantee that the reservation state is
completely established, but rather a weaker assurance that the reservation is “probably” in
place. This results from flow merging; when an intermediate merge node sends a
*ResvConfirm* message to the receiver, the receiver will never get an acknowledgment as
to whether or not the reservation upstream of the merge point (i.e. between the merge
node and the upstream senders) is successfully established.

### 2.3.4 Traffic Control

The traffic control mechanisms used with RSVP are opaque to the protocol itself
and are instead handled by independent algorithms. Once the reservation is established,
all traffic control is maintained by information contained within refresh *Path* and *Resv*
messages. There are three places where traffic policing is important:

- **Network Edge:** The flow must be monitored at the point where it enters the
  network, in order to ensure that the sender is transmitting its data with
  characteristics that are less than or equal to those established by the
  reservation. If a sender is violating these parameters, the node has the option
  of degrading the total service quality for the entire flow, or relegating the
  excessive flow to best-effort service.
- **Source Merging Point:** When a receiver accepts data from multiple senders in a WF-style reservation, the flows from the different senders are merged together at intermediate nodes and are sent as a block to the receivers.

- **Branch Point:** When multiple receiver reservations are merged at a point, the node must be “smart” enough to establish a sufficient reservation upstream while providing the individual QoS reservations on its downstream interfaces.

A schematic diagram showing traffic control with respect to a RSVP-capable host and a RSVP-capable router is shown in *Figure #2-5*. Each node capable of providing QoS control services sends incoming data packets to a packet classifier, which determines the route and QoS for each packet. This is done through an interface to the local routing protocol, as described in *Section 2.4*. Then, the data packet is processed by a packet scheduler which determines, based on the particular outgoing interface, the schedule for forwarding the data to the next node.

![Schematic diagram of RSVP in a host and router.](image)

*Figure #2-5: Schematic diagram of RSVP in a host and router.* [15]
Before sending the data, however, the route must first be determined via Path messages and interfaces with the routing protocol in order to establish the reservation. At each node along the data path, the RSVP QoS control reservation request (i.e. a Resv message) is passed to two local decision modules. In admission control, the node’s current resources are evaluated to determine whether or not the node has sufficient capacity to supply the requested QoS. In policy control, the administrative rights of the application requesting the QoS reservation is examined to check whether or not the user of the application has adequate permissions to request the QoS control service. Accordingly, policy control may provide a good way to authenticate receivers of confidential data. If the request passes both admission and policy control, appropriate parameters are set in the packet classifier and scheduler to enable the QoS control service for that data flow. If either check fails, an error is returned (i.e. a ResvErr message) to the application that generated the request.

2.4 ROUTING QoS TRAFFIC

RSVP is not a routing protocol, but rather a protocol which works in conjunction with routing to establish and maintain QoS routes through the network. In order for RSVP to work effectively, the routing protocol it interacts with must be capable of discerning information about the available resources of the other nodes in the network and then using that information to establish the best routes through the network. Since RSVP was also designed with multicast routing in mind, the routing protocol used with RSVP must also be capable of routing packets addressed to multicast groups.¹⁹

While protocols have been designed to handle multicast traffic, QoS multicast routing is still a field being heavily researched. One protocol that has been proposed is
QOSPF, which is based upon QoS-aware extensions to the MOSPF protocol used in multicast routing. To describe these protocols, an understanding is also useful of the unicast routing protocol, known as OSPF, which they are both based upon. The following subsections outline the differences between these routing protocols.

2.4.1 Unicast Routing: OSPF

OSPF (Open Shortest Path First) is a routing protocol designed for unicast best-effort traffic, and is based solely on examination of the destination IP address. As a result, data packets do not require further encapsulation by the routing protocol. Each router maintains an identical database which lists the active interfaces and reachable neighbors of each node. Each router uses the database to compute the shortest path tree via a Dijkstra algorithm with itself as the root. Each link has a “cost” associated with sending data over it, and the shortest path is defined as the path with the least total cost. The cost may be in terms of the ratio of the total bandwidth currently in use to the link’s total bandwidth capacity, the link’s expected propagation delay, or even the financial usage price of the link. In the case where each link cost is approximately equivalent, the shortest path is simply defined as the path containing the least number of nodes between the source and destination. In order to keep the databases and the shortest path trees updated, each router sends link state advertisements (LSA) to the other routers on the network describing the current status of its own connections to its immediate neighbors. Since the shortest network path may change over time, the path taken by data packets between a source and a destination may also change accordingly.

In order to be scalable to large networks such as the Internet, OSPF can be sectioned off into areas. These areas may, for example, correspond to IP subnetworks.
This minimizes the amount of routing traffic, as well as facilitating smaller computation times for the shortest path trees. The internal topology of an area is invisible to routers outside of the area. In order to connect different areas together, area border routers (ABR) (i.e. gateway routers) are used. These routers maintain separate databases for each area they are connected to, and connect directly with backbone routers. An example topology showing different subnets connected by area border and backbone routers is shown in Figure #2-6.

![Diagram of router types](image)

Figure #2-6: Schematic diagram showing different router types (local subnet, area border, and backbone routers).

If the destination of a packet is within the same area as its source, the packet is routed along the shortest path tree from its source to its destination. If the destination is in a different area than the source, the packet must first be routed along the shortest path from the source to an area border router, then transmitted across the backbone routers to the area border router of the destination subnet, and then routed along the shortest path from that area border router to the receiver.
2.4.2 Multicast Routing

Multicast routing is more complicated than unicast routing. A separate IP address space (class D, ranging from 224.0.0.0 to 239.255.255.255) has been reserved for multicast traffic. These IP addresses do not correspond to physical hosts, but rather serve to define "multicast groups" that can be allocated dynamically as needed. Individual hosts can utilize a group management protocol such as IGMP, which is discussed in Section 2.4.2.1, to subscribe to these multicast groups. The purpose of such a protocol is to map group addresses with host IP addresses, and inform routers of which groups to route over which interfaces.

In order to apply OSPF to multicast traffic, several changes to the routing protocol become necessary. As a result, a new protocol based on OSPF, known as MOSPF, has been developed. This protocol is discussed in Section 2.4.2.2.

2.4.2.1 IGMP

The Internet Gateway Management Protocol (IGMP) is designed to provide group management services for multicast applications by maintaining group membership information lists that specify which hosts are currently members of which multicast groups. This protocol works in conjunction with a multicast routing protocol such as MOSPF, to keep routers updated as to which groups should be routed along which interfaces.

In a host, the local IGMP-daemon will accept requests from applications to join a particular group $G_i$. The daemon will update the local group membership list, and forward a join request (known as a "response") for group $G_i$ to the local router, assuming that another application on the host is not already a member of that group. (In some
implementations, including the simulation model presented in Chapter 4, a message may also be sent to the network layer to indicate that it should begin to accept packets for that group.) The local IGMP-daemon will act similarly for requests to terminate group membership by deleting the corresponding entry from the local group membership list.

An IGMP-capable router will periodically send queries to hosts and other routers to maintain a current list of which groups it should be forwarding along which interfaces. If a host receives a query for a particular group it is a member of, it will send a response to the router and the group after a random time interval. During this interval, if a host receives a response for the query from another host on the subnet, it cancels its own response. This is designed to minimize response traffic, as a router only needs to be aware that at least one host on that subnet is requesting traffic from that group. When a router receives a response along a particular interface, it will update its group membership list to indicate that a host on that interface is requesting membership to that group, and it will send a response to all other interfaces to indicate the change. Similarly, the router will delete a group from its membership list when it no longer receives any responses to queries for that group after a specified time-out interval.

2.4.2.2 MOSPF

The MOSPF (Multicast Open Shortest Path First) routing protocol is based heavily upon OSPF, though significant changes have been made.\[22\] The most noticeable alteration is that routing is based on the (source, multicast group destination) IP address pair instead of just the unicast destination IP address.

Because the potential combination of multicast sources and destinations is very large, the shortest path tree is calculated on demand, rather than in advance as in OSPF.
The Dijkstra algorithm is still used, though the tree is constructed with the root being at the packet’s source (i.e. intermediate routers do not use themselves as the root of their shortest path trees). As a result, the number of copies of packets going to receivers in close proximity to each other is minimized, since the number of paths used by the multicast data traffic flow is minimized. In order to facilitate this, a group-membership LSA is also periodically generated from IGMP data and sent to neighboring nodes.

2.4.3 QoS Routing

In order for RSVP to successfully and intelligently reserve resources along a path for a QoS data flow, the underlying routing protocol must be aware of the availability of network resources and the QoS requirements of the data flow (specified by the TSPEC), in addition to the network topology when the shortest path tree is being calculated. To meet these needs, a routing protocol based upon OSPF and MOSPF, called QOSPF (Quality of Service Path First), has been developed. Several changes have been made, which are represented by the addition of three new LSAs:

- **Link Resource Advertisements (RES-LSA):** These LSAs indicate the link resources available at each router, including the largest amount of resources available for reservation traffic on each interface, as well as a link delay metric. Like Router-LSAs in OSPF, they are flooded through the entire area, though RES-LSA are used instead of the Router-LSA to calculate the shortest path route, since QoS parameters are now included in the cost metric. These advertisements are generated whenever the network topology, delay, or available bandwidth changes.
Resource Reservation Advertisements (RRA): These LSAs describe a router’s reservation for a particular flow on its interfaces. It indicates the resources used by the flow (as indicated by RSVP Path and Resv messages), and is made available to other routers for use in their shortest path tree calculations. In FF-style and SE-style reservations, an RRA is transmitted for each sender, while in WF-style reservations, RRAs use a wild card as their IP sender address.

Deterministic Area Border Router Advertisements (DABRA): These LSAs are used between area border routers to facilitate multicast reservation traffic across different areas. When a source area computes routes to area border routers, not all of the area border routers may be used. The DABRA is intended to notify the destination area of which area border routers to use as the root for their shortest path tree calculations.

The Dijkstra algorithm used in QOSPF is also modified from the best-effort Dijkstra algorithm used in OSPF, as it is inefficient to compute the complete tree for the area once the source-to-destination route is determined. Any new additions to the destination list or any changes to the LSAs will trigger a recalculation. The modified Dijkstra algorithm works as follows:

1. Move the closest candidate router (V) from the candidate list to the shortest path tree.
2. Examine neighbors of V to include in the candidate list. This includes looking up the RES-LSA and RRA to determine the outgoing links L, which correspond to a neighboring candidate W. The candidate is eligible if a
reservation for the flow already exists (in which case it should be marked “reserved”) or there is enough free bandwidth available.

3. Eliminate \( W_i \) from the candidate list if any of the following are true: it is already on the shortest path tree, if the Router-LSA indicates a unidirectional link,\(^{24}\) if the LSAs have expired for \( W_i \) (indicating that \( W_i \) may be down), or if \( W_i \) is not capable of handling multicast or QOSPF data.

4. Assign the delay metric. This is done by using the link with the least delay, if multiple links between \( V \) and \( W_i \) exist. The delay from the source to \( W_i \) is the delay from the source to \( V \) plus the delay from \( V \) to \( W_i \).

5. Add \( W_i \) to the candidate list if it is not already included, or modify the entry on the candidate list if the calculated delay is less than the one already listed in the table.

6. Using the delay as the cost metric, select the shortest route and repeat the process until the candidate list is empty.

QOSRP also has the potential to perform route pinning, which means that the current route is used even when a better route becomes available. The advantage of this is that data transmissions do not incur interruptions in the traffic flow by frequently changing routes through the network. Conversely, however, data transmissions will not take advantage of more optimal routes when they become available. To utilize route pinning with the modified Dijkstra algorithm, links with preexisting reservation state status are marked “reserved”, and such candidates are chosen over unmarked candidates, even if an unmarked candidate has a smaller delay. There is currently a debate over whether or not route pinning or partial route pinning (pinning along sections of the path) is suitable for implementation with QoS routing. The matter is currently unresolved.\(^{25}\)

Another proposed modification to the Dijkstra algorithm is called explicit routing (EROSPF).\(^{26}\) In this scheme, only the source router (or the area border router if the
source is in a different area) calculates the route through the network, and forwards the result to the other nodes along the path through an Explicit Routing Advertisement (ERA). Since the other nodes along the path do not perform the Dijkstra calculation, the mechanism becomes scalable to a large number of senders in a network, as it reduces the number of other types of LSAs (such as RRAs and DABRAs) that need to be transmitted. However, this may lead to suboptimal routes, since information is not updated as quickly.
CHAPTER 3:

AXIOMATIC DESIGN OF THE TELENURSING SYSTEM

In Chapter 2, an overview was provided of current networking technologies under development to transmit real-time data traffic over the Internet. Such work, however, is meaningless unless there are practical applications which can utilize the technology to its fullest potential. This chapter presents the design of a telenursing system, which integrates real-time network technology into a patient care system for monitoring home-based patients from a centralized facility. The telenursing system is capable of satisfying several different requirements, and can be expanded in scope in the future.

At the core of the telenursing system are health care practitioners known as telenurses, who monitor, via computer, home-based patients from a telenursing center. This center can be located either in a hospital or in a separate facility. In order to monitor a patient successfully, the telenurses must have access to several sources of information, including:

- data from sensors located on the patient’s body and around the patient’s home;
- video conference equipment to allow telenurses to talk to patients, doctors, and medical specialists;
- medical records indicating the health history of the patient;
- medical reference knowledge from distributed medical databases.

The design of such a large-scale application is complex, as there are numerous technical details and potential implementation conflicts that the designer must consider. Although the design presented here is conceptual and the actual construction of this system is
beyond the scope of this thesis, suggestions have been incorporated to anticipate implementation problems, as well as to present a clear and coherent system design.

The design of the telenursing system has been generated by a technique known as axiomatic design. This technique uses a set of axioms, theorems, and related corollaries to facilitate the development of good system designs. While this approach can initially take longer than conventional design techniques, it guides the design process in a logical and scientific manner. As a result, design errors and omissions are detected relatively early, when they are both easier and less expensive to correct.

Axiomatic design is based on the concept that there exist fundamental principles in design. Therefore, it provides a unique design methodology based on the absolute referent (i.e. the design axioms) for the synthesis and analysis of a design. It provides design evaluation criteria which enable the designer to eliminate poor design solutions at every stage of decision making, and search for new ideas promptly. They also enable the designer to select the best among those proposed. It incorporates the concept of structured hierarchy, not only in the functional domain but also in the physical and process domains.¹

A brief summary of the axiomatic design process is outlined in Section 3.1. The design of the telenursing system, using this approach, is presented in Section 3.2, and an overview of the full design is provided in Section 3.3.

3.1 AXIOMATIC DESIGN PROCESS

Design is usually considered to be a creative art which cannot be broken down into scientific principles. Despite several attempts to establish guidelines for the design process, such as the use of Gantt charts, House of Quality charts, and other similar tools, good designs have typically resulted only after extensive experience:

In the absence of a scientific basis, human intellectual endeavors ranging from fine arts to engineering are performed subjectively in the realm of creative activity. Since the output of such activities cannot be
understood rationally in the absence of commonly accepted criteria, they are treated as such. What this really means is that we can appreciate the outcome of the intellectual endeavor but do not understand the process that produces the outcome, and [therefore] cannot quantify the results.\[2\]

The axiomatic design approach was originally developed in the late 1970's in an attempt to quantify the design and manufacturing process into a set of axioms, theorems, and corollaries which can be used either to generate new designs or to analyze and improve existing ones. Over the past 20 years, axiomatic design has helped to formalize the design / manufacturing process, and has gained a significant measure of interest within the corporate community.\[3\] Current research into axiomatic design techniques includes the application of axiomatic design to large-scale systems, such as the one presented in this chapter.

In order to generate good designs, the axiomatic design technique breaks down the design process into stages:

Design involves four distinct aspects of engineering and scientific endeavor: the problem definition from a “fuzzy” array of facts and myths into a coherent statement of the question; the creative process of devising a proposed physical embodiment of solutions; the analytical process of determining whether the proposed solution is correct or rational; and the ultimate check of the fidelity of the design product to the original perceived needs.\[4\]

These design stages are outlined further in the following subsections.

3.1.1 Problem Definition

Defining the problem -- figuring out what tasks need to be accomplished -- is the most difficult part of any design. While a customer will specify “tasks” that the design should be capable of addressing, these are not always clearly defined and may often possess inherent conflicts or be biased by hype, public perception, and previous
approaches to similar problems. In order to define the problem properly, the needs of the customer must be broken down into a series of functional requirements (FRs) which specify, in plain language, the tasks which the design needs to accomplish. FRs are usually phrased as verbs, since they describe what needs to be done.

When generating a list of functional requirements, it is important to do so in a solution-neutral environment. If problems are defined in terms of the methods that can be used to solve them, it may limit the ability of the designer to see the nature of the problem clearly. As a result, the designer may develop a solution which is suboptimal or unnecessarily complex. As a result, FRs should state the problems to be solved, and should not incorporate preconceived notions of what their solutions will look like.

3.1.2 Creative Process

Once the design problem has been defined as a set of functional requirements, creative ideas are generated by brainstorming. In order to come up with a wide variety of potential solutions, the designer must continue to work in a solution-neutral environment. If the designer enters the process with a preconceived idea for a solution, potentially better approaches may be ignored. The potential solutions are referred to as design parameters (DPs) and are usually phrased as nouns, since they describe how the tasks are accomplished.

Often, DPs can be decomposed into new sets of functional requirements. A hierarchical structure of FRs and DPs then develops in a design, where high level DPs form the basis for new FRs at a lower level. In essence, the lower level FRs determine what tasks need to be done to satisfy the higher level DP. The design of the telenursing system outlined in Section 3.2 possesses such a hierarchy.
3.1.3 Analytical Process

Once FRs and DPs are generated, some logical procedures must be followed to determine which DPs should be selected and how these DPs relate to FRs. This process is the core of axiomatic design. There are two fundamental axioms which are used to determine the benefits of choosing certain DPs over others, known as the Independence Axiom and the Information Axiom, as explained in the following subsections.

3.1.3.1 The Independence Axiom

*Maintain the independence of the functional requirements (FRs).*[^5]

This axiom states that the DPs and the FRs should be related in such a way that a design change in particular DPs should only affect corresponding FRs. In other words, each FR should be modifiable by a DP which is independent of the other FRs. If this axiom is violated, it will become impossible to modify part of the design without affecting the performance of other parts. As a result, design changes may be very difficult to implement, since a change in one part will result in potentially detrimental changes to other parts.

To measure how well a particular design (i.e. a specific set of DPs) satisfies the Independence Axiom, *design matrices* are used to show the relationship between the FRs and DPs in a design. In some design cases, physical values or functions can be placed in the design matrices to define the relationship explicitly. In general, however, it is usually sufficient to show qualitatively that a non-zero relationship exists between FRs and DPs, since numerical or functional values may not always be physically meaningful.

Conformance with the Independence Axiom is characterized by design matrices:
• **Uncoupled:** An uncoupled design matrix demonstrates a one-to-one relationship between FRs and DPs, and completely satisfies the Independence Axiom. As the following equation shows, any change to a particular DP in the uncoupled case will only affect its corresponding FR, without affecting any of the other FRs in the design.

\[
\begin{align*}
\{ \text{FR1} \} & = \begin{bmatrix} X & 0 & 0 \end{bmatrix} \{ \text{DP1} \} \\
\{ \text{FR2} \} & = \begin{bmatrix} 0 & X & 0 \end{bmatrix} \{ \text{DP2} \} \\
\{ \text{FR3} \} & = \begin{bmatrix} 0 & 0 & X \end{bmatrix} \{ \text{DP3} \}
\end{align*}
\]

• **Decoupled:** A decoupled design is characterized by a triangular matrix relationship. While a decoupled design expresses coupling between the FRs and DPs, it is still possible to modify the design safely, as long as the DPs are altered in a specific order. Sequential processes can often be represented by decoupled designs, since a particular FR can be satisfied only under the condition that the preceding FRs have already been satisfied. Most of the design matrices emerging from the design of the telenursing system are decoupled, since many of the processes involved are sets of sequential tasks. The following equation shows that a change in DP1 (to change FR1) will affect FR2 and FR3, though this can be compensated for by subsequently changing DP2, followed by changing DP3. As a result, decoupled designs satisfy the Independence Axiom, since functional independence can be maintained when design changes are strictly made in order.

\[
\begin{align*}
\{ \text{FR1} \} & = \begin{bmatrix} X & 0 & 0 \end{bmatrix} \{ \text{DP1} \} \\
\{ \text{FR2} \} & = \begin{bmatrix} X & X & 0 \end{bmatrix} \{ \text{DP2} \} \\
\{ \text{FR3} \} & = \begin{bmatrix} X & X & X \end{bmatrix} \{ \text{DP3} \}
\end{align*}
\]
• **Coupled:** A coupled design matrix is typically indicative of a bad design, since it is impossible to implement a change in a particular DP without altering several FRs in the process. As a result, even small design changes can lead to large problems, because these changes may adversely affect the performance of other parts of the design. In order to incorporate design changes, iteration between the changes to FRs and DPs is usually necessary.

\[
\begin{align*}
\{ \text{FR1} \} &= \begin{bmatrix} X & X & X \end{bmatrix} \{ \text{DP1} \} \\
\{ \text{FR2} \} &= \begin{bmatrix} X & X & X \end{bmatrix} \{ \text{DP2} \} \\
\{ \text{FR3} \} &= \begin{bmatrix} X & X & X \end{bmatrix} \{ \text{DP3} \}
\end{align*}
\]

These three cases are typically mixed together in real systems. For example, a "partially decoupled" design is a design which is mostly uncoupled, although two or more parameters would be decoupled. In this particular case, the order of any design changes would still have to be strictly followed for the decoupled elements, though the order of changes is irrelevant for the uncoupled elements. Many examples of partially decoupled designs emerge in the design of the telenursing system. Another type is a "partially coupled" design, which has two or more coupled components that relate to other components in an uncoupled or decoupled manner.

In the design of complex systems, partial coupling often exists between subsystems at the very highest levels of the design hierarchy, even when the subsystems themselves are uncoupled or decoupled. This coupling usually results from the fact that integrating two subsystems together requires information from both subsystems. As a result, the integration of subsystems is often a coupled process requiring iteration. Research is currently underway to determine whether such systemic coupling can be
detected and compensated for during the design stage of a large-scale system which incorporates several subsystems.\textsuperscript{6}

\subsection*{3.1.3.2 The Information Axiom}

\emph{Minimize the information content of the design.}\textsuperscript{7}

The Information Axiom provides a measure of how well a particular design (i.e. a particular set of DPs) satisfies the design goals (i.e. the FRs). The information content of a design is a probabilistic measure of the amount of information required for the design to achieve its tasks. In general, the more information a design requires, the lower its probability of success. Typically, high complexity or large costs will contribute to increasing the information content of a design.

Like the Independence Axiom, the Information Axiom can be used for both quantitative and qualitative analysis. A measure of information content is made by using probability density distributions. Functional requirements typically have a range of tolerances, referred to as a \textit{design range}. A probability density function of the performance of a particular DP, known as the \textit{system range}, is plotted against the design range. The amount of overlap between the design range and the system range is referred to as the \textit{common range}. The information content, as shown by the following equation, is the logarithm of the system range divided by the common range (in units of bits, when the logarithm is base-2). An example of this type of analysis is provided in \textit{Section 3.2.3}.

\[ I = \log_2 \left( \frac{\textit{system range}}{\textit{common range}} \right) \]

Instead of this quantitative tool, most of the design choices in the telenursing system are based on the more qualitative notion that they possess the least informatic.
content of the conceivable options. An in-depth information analysis was deemed to be premature at this point in the system design. However, more quantitative analysis of the information content may prove useful when this system is implemented and some of the subsystems are defined in greater detail.

3.1.4 Ultimate Check

The ultimate check of a design occurs when the design is built or implemented and people actually attempt to use it to accomplish specific tasks. Utilizing axiomatic design to generate an uncoupled or decoupled design helps to ensure that the design will be successful in accomplishing its desired goals. The axiomatic design process forces the designer to explicitly think about certain aspects and potential problems that otherwise may be overlooked, along with providing a measure of how well a design will satisfy its goals and how design changes should be implemented.[8]

3.2 Design of the Telenursing System

At the highest hierarchical level in the design of a comprehensive home-based health care system, a functional requirement exists for health care practitioners to monitor patients remotely from a centralized facility. The design parameter selected to satisfy this functional requirement is the telenursing system.

Axiomatic design tables, like Table #3-1, are used throughout this section to match FRs with DPs at each hierarchical level of the design. The functional requirement listed above is only one of several under the total scope of the home-based health care research consortium. The complete list of FRs for the entire home-based health care system is not provided here, though it is expected ultimately to become a decoupled or
coupled design, deriving from the fact that different subsystems must be mutually compatible.

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor home-based patients remotely from a centralized site</td>
<td>Telenursing system</td>
</tr>
</tbody>
</table>

As discussed in Section 1.3, there are five core functional requirements which the telenursing system needs to satisfy, as shown in Table #3-2.

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor sensor data in real-time</td>
<td>Modular interface with home-based sensors</td>
</tr>
<tr>
<td>Run external applications which facilitate home-based health care</td>
<td>Modular interface with external applications</td>
</tr>
<tr>
<td>Transmit real-time sensor and communication data over network with guaranteed service quality</td>
<td>Interface with Internet utilizing protocols which guarantee a quality of service (QoS)</td>
</tr>
<tr>
<td>Interpret and act upon data</td>
<td>Computer-assisted health care practitioner</td>
</tr>
<tr>
<td>Be easy to operate by medical personnel and patients</td>
<td>Graphical User Interface (GUI)</td>
</tr>
</tbody>
</table>

- **Monitor sensor data in real-time (FR1):** The overall home-based health care system calls for numerous sensors to be located on the patient’s body and around the patient’s home. These sensors need to be monitored in real-time, so that current knowledge of a patient’s health and environmental status is maintained. Since different patients may require different types of sensors, a
modular interface is necessary which can interact with any potential sensors used. This interface is discussed in Section 3.2.1.

- **Run external applications (FR2):** In addition to monitoring patients via sensors, the telenursing system must be capable of launching external applications which facilitate patient care. These applications include video conferencing for remote diagnoses, accessing medical databases to obtain the patient’s health history, and accessing medical web sites over the Internet that contain information about specific health conditions, drugs, and so forth. In addition, a framework is incorporated to facilitate the addition of new applications in the future. As a result, an interface is specified enabling the telenursing system to interact with external applications. This interface is discussed in Section 3.2.2.

- **Transmit sensor and communication data (FR3):** The information content of the sensors and external applications dictates the network requirements of the system. Since the sensor data and some application data (e.g. video conferencing) are collected in real-time, it is desirable to be able to transmit this information to the telenursing center in real-time in order for the data to be useful. The Internet is deemed to be the best solution to meet this need because of its low cost, high bandwidth capacity, and relative ubiquitousness. Although the Internet does not currently support real-time traffic very well, technologies are under development to facilitate real-time communication and provide quality of service (QoS) guarantees, as discussed in Chapter 2. This
technology is incorporated into the design of the telenursing system, as presented in Section 3.2.3.

- **Interpret and act upon data (FR4):** Once data is received by the telenursing center, it must be interpreted in order to determine what medical actions, if any, are required. This covers a wide range of applications, ranging from a rapid emergency response to a sudden health crisis, such as dispatching an ambulance when a patient has a heart attack, to collecting data over a period of time for analysis of a patient’s health trends. Another potential application is remote diagnosis, which would utilize video conferencing to facilitate an interactive conversation between the patient and/or one or more health care practitioners. While the complete set of medical applications that this system could potentially perform in the future cannot be determined at this time, a framework for incorporating these applications is provided, to enable the telenurse to work with computers, physicians, medical technicians, and on-line medical databases. This is presented in further detail in Section 3.2.4.

- **Easy to operate by patients and medical personnel (FR5):** While it is expected that patients and health care practitioners will require some training before using the system, the assumption cannot be made that these users will be experts in networking and medical sensor technology. As a result, it is important that the users interact with the system via a front end interface. This interface must contain modules which manipulate the sensors, external applications, and network transmission hardware and software, while
presenting information in an easily understandable manner to the system’s operators. Although the construction of this interface is beyond the scope of this thesis, the design makes several suggestions which address issues likely to emerge during the implementation phase. For example, the design encourages writing the interface in Java, because Java can easily access data over the Internet, use graphical displays, and port to several different types of computer platforms and architectures. The functional design of the GUI is presented in Section 3.2.5.

The design matrix shows a partially coupled system, and provides a good example of the problem of integrating subsystems in a large-scale system design. Aside from external applications (FR2), the other FRs in the system are dependent on both their own corresponding DP and the successful implementation of the previous DPs. This is clearly demonstrated by the graphical user interface (DP5), which operates on top of the other parts of the system, and is therefore affected by all of the FRs. The external application interface will have some minor coupling with the telenursing interface. This coupling is unavoidable, since the external applications used in the system are defined to some extent by the needs of the telenurse to interpret and act upon patient data. In order to alter the telenurse’s abilities to interpret and act upon patient data, some iteration with the suite of external applications available may be required.

\[
\begin{align*}
\{ FR1 \} & = \begin{bmatrix} X & 0 & 0 & 0 \end{bmatrix} \{ DP1 \} \\
\{ FR2 \} & = \begin{bmatrix} 0 & X & 0 & X \end{bmatrix} \{ DP2 \} \\
\{ FR3 \} & = \begin{bmatrix} X & X & X & 0 \end{bmatrix} \{ DP3 \} \\
\{ FR4 \} & = \begin{bmatrix} X & X & X & X \end{bmatrix} \{ DP4 \} \\
\{ FR5 \} & = \begin{bmatrix} X & X & X & X \end{bmatrix} \{ DP5 \}
\end{align*}
\]
Implicit within all of these functional requirements is the need for each subsystem to be robust, meaning that the overall system should continue to function, at least partially, if specific components malfunction. This robustness, however, is more easily accounted for within the design of each subsystem than as a separate functional requirement. Some systemic robustness also emerges from ensuring an uncoupled or decoupled system design through the axiomatic design approach. The following subsections decompose the design parameters outlined above.

3.2.1 Modular Interface with Home-Based Sensors (DP1)

It is impossible to predict the full spectrum of sensors which may be utilized in the future to assist in home-based health care. Nevertheless, the telenursing system needs to be able to incorporate any type and combination of sensors which may prove useful in the care of a patient. It is important, therefore, that the telenursing system have a standardized (i.e. modular) interface for medical sensors, so that real-time data concerning the health of the patient can be easily acquired. The functional requirements for this subsystem are listed in Table #3-3.

Table #3-3: Decomposition of interface with home-based sensors (DP1).

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintain compatibility with any type of potential sensor</td>
<td>Generic interface to interact with external sensor software provided by sensor manufacturers</td>
</tr>
<tr>
<td>Read multiple sensors simultaneously</td>
<td>External sensor polling programs run in parallel, via real-time operating system</td>
</tr>
<tr>
<td>Store data for transmittal</td>
<td>Allocation of PC memory</td>
</tr>
<tr>
<td>Check data for errors or problems</td>
<td>Data interpretation detecting medical emergencies and sensor malfunctions</td>
</tr>
</tbody>
</table>
• **Sensor Compatibility (FR11):** Sensor manufacturers will create proprietary methods to read real-time data from their sensors into a computer, which will most likely be through a combination of hardware boards and software programs installed in the computer. It is also conceivable that sensor manufacturers will provide packages integrating a suite of home-based sensors, which include software programs and hardware components that read the sensors and return their current values. A generic interface is required, however, to specify the types of inputs required by an application to poll sensors, as well as the types of output the application should expect in return.

• **Read Multiple Sensors (FR12):** If there are only a few sensors that are being polled by the system, the delay incurred by reading each sensor in order (i.e. in series) would most likely be negligible. In the general case, however, there may be a large number of sensors, and the delay could potentially be significant enough to violate the requirement that the sensors be monitored in real-time. As a result, it is necessary that each sensor be accessible simultaneously (i.e. in parallel). This can be handled by using sensor hardware in the computer to read the sensors in parallel, in conjunction with real-time multiplexing of the computer's resources by its operating system. In order to handle this task properly, the computer must run an operating system capable of both performing real-time data acquisition and analyzing real-time data. One such operating system is Windows NT, when implemented with a real-time kernel patch.\(^9\)
• **Store Sensor Data (FR13):** Once information is gathered from the sensors, it must be stored in RAM so that the data can then be packetized and transmitted over the network. Accordingly, the computer should have adequate RAM (96 MB or greater) and a fast processor speed (233 MHz or greater) so that it is capable or running several applications and collecting real-time data simultaneously.\(^{[10]}\)

• **Check Data for Errors or Problems (FR14):** It is desirable that the home computer act as the first line of defense in case of a medical emergency. Therefore, the computer should have the capability of performing a quick analysis of the data to ensure that the sensor readings are within a critical range. This analysis should also be sufficiently intelligent to determine when it is receiving "noise" from the sensors (e.g. a negative value from the patient’s heart rate sensor), which may indicate that the sensor is malfunctioning or has been switched off. Should either an error or an emergency occur, the system could run manufacturer-provided diagnostic software to gather more information and then transmit this information to the telenursing center for further analysis. Since this type of check is dependent upon the particular sensors used, more specific development of this feature must wait until the telenursing system is implemented.

Each functional requirement in this subsystem is dependent upon preceding requirements, resulting in a decoupled design.
3.2.2 External Application Interface (DP2)

The external application interface is responsible for providing a framework to launch commercial applications included with the telenursing system. The interface must keep track of which applications are running, so that the appropriate network requirements can be determined. While it is possible that some applications could be written directly into the system, it will probably be simpler to utilize commercial applications, even if these applications require minor modifications to make them compatible with the telenursing system.

The implementation details concerning the integration of commercial applications with the overall telenursing system are beyond the scope of this thesis, since they depend on what types of applications are selected, as well as the perceived application needs of the telenurses for performing their jobs (DP4). Nevertheless, the framework provided in this section is necessary to determine, in general, the network requirements that the system will require, as well as to represent the full capabilities of the system. As a result, the design presented in this section is not intended to be comprehensive, but rather to form a basis for integrating external applications.

Three applications -- remote face-to-face communication, access to distributed patient information, and access to distributed medical knowledge -- are currently foreseen to be desirable to include with the telenursing system. It is likely, however, that more applications will be added in the future. As a result, a standard interface is required to
facilitate future expansion of this subsystem. The functional requirements, as currently specified, are outlined in Table #3-4.

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 Enable remote face-to-face communication</td>
<td>Commercial video conference application</td>
</tr>
<tr>
<td>22 Enable access to distributed patient medical information</td>
<td>Commercial database application</td>
</tr>
<tr>
<td>23 Enable interface with general medical information</td>
<td>Commercial web browser application</td>
</tr>
<tr>
<td>24 Standardize interface with applications</td>
<td>Generic application template via COM objects</td>
</tr>
</tbody>
</table>

- **Remote Face-to-Face Communication (FR21):** In order to achieve remote face-to-face communication, video conferencing equipment can be used. This is a very important application, because it allows the patient to establish contact at any time with the telenurse as well as with physicians, medical specialists, and family members. This also enables remote diagnoses of medical problems, as well as allowing real-time discussions between patients, telenurses, physicians, and family members. To enable video conferencing, the software and hardware used will most likely be an adaptation of a commercial product, such as PictureTel or Intel’s Proshare. Most commercial video conference applications typically utilize 128 kB/sec ISDN lines or 1.5 MB/sec ethernet lines. As a result, the bandwidth capacity of the network used for the telenursing system (FR3) should be a minimum of 128 kB/sec. Modifications may be required to permit multicast video conferencing as well
as to integrate the video conferencing package with the rest of the telenursing system. As a result, further decomposition of this DP is necessary for implementation.

- **Distributed Patient Medical Information (FR22):** In order to monitor the patient properly, the telenurse will require access to databases that can track the patient’s health status and compare it with both the patient’s health history and contemporary medical standards. As a result, the total information required may be stored in several distributed databases on the Internet. The medical database interface will launch an external database program, such as Microsoft Access, and will incorporate data from distributed network databases, to maintain an up-to-date picture of the patient’s health status. In all likelihood, the database programs themselves will have some capacities in this area, though modifications to the database software may be required. As a result, further decomposition of this DP is necessary for implementation.

- **General Medical Information (FR23):** In addition to medical databases, a great deal of information concerning various health conditions, drug usages, and so forth can be found on the World Wide Web. While some of these web pages may be maintained by the telenursing center, it is likely that a great deal of informational web sites will be maintained by unrelated medical organizations. Many of these pages may also utilize Java applets and security encryption schemes. As a result, the telenursing system must incorporate an interface capable of launching a Java-capable and security-capable web
browser, such as Netscape or Internet Explorer. While it is unlikely, the commercial web browser selected may require minor modifications to make it compatible with the rest of the telenursing system, resulting in further decomposition of this DP to implement this application.

- **Standardize Interface (FR24):** A standard template is required to provide a framework for designing application interfaces. This will ease the integration of current applications and facilitate the inclusion of new applications in the future. In general, this interface is responsible for incorporating external commercial applications into the telenursing system software. The template must allow for command-line options to specify preferences, as well as incorporate a way to export the network requirements of the application, so that they can be accounted for by the network interface (DP3). To accomplish this, the template could make use of the Component Object Model (COM), which is an emerging binary code standard developed by Microsoft Corporation. By transforming external applications into COM objects, they can easily be embedded into the telenursing software package:

The Component Object Model (COM) is a platform-independent, distributed, object-oriented, system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents), ActiveX (Internet enabled components), as well as others... [COM] is not an object-oriented language, but a standard... Language, structure, and implementation details are left to the application programmer. COM... [specifies] an object model and programming requirements that enable COM objects (also called COM components, or sometimes simply objects) to interact with other objects. These objects can be within a single process, in other processes, even on remote machines. They can have been written in other languages, and may be structurally quite dissimilar. That is why
COM is referred to as a binary standard -- it is a standard that applies after a program has been translated to binary machine code.\textsuperscript{[11]}

The application interfaces themselves are uncoupled from each other, though they will dictate the necessary capabilities of the application template, making this a partially decoupled design.

\[
\begin{align*}
\{ \text{FR21} \} &= \{ X \ 0 \ 0 \ 0 \} \{ \text{DP21} \} \\
\{ \text{FR22} \} &= \{ 0 \ X \ 0 \ 0 \} \{ \text{DP22} \} \\
\{ \text{FR23} \} &= \{ 0 \ 0 \ X \ 0 \} \{ \text{DP23} \} \\
\{ \text{FR24} \} &= \{ X \ X \ X \ X \} \{ \text{DP24} \}
\end{align*}
\]

3.2.3 Network Interface (DP3)

Once information is obtained from sensors and external applications, it must be transmitted to its destination. Since this will involve the transmission of large quantities of data in real-time, there are several technological challenges to be overcome, since the technology to do real-time data communication over the Internet is still being developed. Despite this, the Internet has many advantages, such as a wide support-base in research and development, reasonably low cost, and omnipresence. In addition, for non-real-time applications (e.g. web page queries), the Internet can provide access to information sites maintained by external groups and organizations that may be of interest to patients.

In order to evaluate which networking techniques and technologies should be used for the telenursing system, an examination must first be made to determine the system's networking requirements. There are other networks besides the Internet which can be used to connect the patient’s home to the telenursing center, including telephone networks utilizing either analog phone lines (POTS) capable of 33.6 kB/sec bandwidth or digital phone lines (ISDN) capable of 128 kB/sec bandwidth. The Information Axiom,
however, can be used to demonstrate qualitatively why the Internet is the best networking choice. Video conferencing, which is expected to utilize the largest amount of bandwidth in the system, is primarily done over ISDN in current applications. However, ISDN is limited because it is expensive to implement over long distances and on a large scale, as well as being incapable of handling high quality video images which may require more than 128 kB/sec bandwidth, even when video compression schemes are utilized. It is also reasonable to assume that the connection between the patient's home and the telenursing center may need to occur very frequently (i.e. several times a day) if not continuously, which could potentially saturate a telephone network. This behavior would violate the Information Axiom, since the amount of information (i.e. costs) required would escalate as the system was scaled upwards to include many users or to cover a wide geographical area. Building a private network to connect the telenursing center to a patient's home would also violate the Information Axiom, since the high cost of building and maintaining such a network would be unacceptable, especially as the network is expanded. As a result, using the Internet seems to be the most feasible option.

In order to make use of the Internet, the system design must first consider how patient homes and the telenursing center will be connected. Once this is established, the volume and types of data which must be transmitted at a particular instant must be examined in order to determine the necessary QoS before data transmissions can be made. This type of connection must also work in both directions, since both the patient's home and the telenursing center can be sources for some types of data (e.g. video conferencing). The functional requirements for the network interface are summarized in Table #3-5.
Table 3-5: Decomposition of interface with network (DP3).

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 Connect patient homes to Internet</td>
<td>Cable modem connection to Internet</td>
</tr>
<tr>
<td>32 Connect telenursing center to Internet</td>
<td>High-capacity network link to Internet</td>
</tr>
<tr>
<td>33 Calculate desired QoS</td>
<td>Algorithm which polls applications and determines the necessary QoS</td>
</tr>
<tr>
<td>34 Establish / refresh QoS connection</td>
<td>RSVP control messages to reserve network resources</td>
</tr>
<tr>
<td>35 Transmit / receive data</td>
<td>Encapsulation of data into packets sent across reserved portion of network</td>
</tr>
</tbody>
</table>

- **Connect Patient's Home to Internet (FR31):** An Internet service provider (ISP) is capable of connecting a patient’s home to the Internet. While the telenursing center can become its own ISP, it will probably be easier to arrange Internet service to patient homes through commercial providers. Most ISPs currently only support up to 28.8 kB/sec on an analog modem connection, and even an ISDN line will only support up to 128 kB/sec. Accordingly, emerging cable modem technology seems to be a good choice for patient connectivity. This technology enables cable television companies to supply Internet service through their existing coaxial cable wiring, leading to larger bandwidth capacities at only a slightly higher cost. There are some limitations, however, due to the fact that the bandwidth is asymmetric: 1.5 MB/sec bandwidth entering the home, but only 300 kB/sec exiting the home. This can still prove useful for many applications, however. For example, the patient would be able to download non-interactive information,
such as web-based educational material, at very fast speeds while maintaining an interactive link to the telenursing center.

![Information analysis graph](image)

**Figure #3-1: Information analysis graph. The various technologies are rated on their ability to handle data traffic requiring bandwidth over 100 kB/sec.**

In *Figure #3-1*, a comparison is made between the bandwidth capabilities of various networking technologies for the home and how well each can transmit data traffic with bandwidth requirements over 100 kB/sec, such as data traffic from video conferencing. A probabilistic measure for each Internet connection technology states how much bandwidth it is capable of obtaining at any instant of time, and is a function of the level of network congestion. (For simplicity, a uniform probability density is assumed, though in actuality this probability density function is more complex.)

The cable modem has the largest common range, and therefore the smallest information content. The information content of these network technologies is presented in *Table #3-6*. Currently, the cost of a cable modem connection in the Boston, MA area is about $50 per month, though this price will likely decrease as the service area expands. A schematic diagram of this connection is shown in *Figure #3-2*. 
Table #3-6: Information content of networking technologies for the home, in relation to ability to handle bandwidths over 100 kB/sec.

<table>
<thead>
<tr>
<th>Network Technology</th>
<th>Max. Bandwidth (kB/sec)</th>
<th>Information Content (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>POTS (Plain Old Telephone System)</td>
<td>33.6</td>
<td>$\log_2(33.6/0) = \infty$</td>
</tr>
<tr>
<td>ISDN (Integrated Services Digital Network)</td>
<td>128</td>
<td>$\log_2(128/28) = 2.193$</td>
</tr>
<tr>
<td>Cable Modems (coaxial cable / fiber network)</td>
<td>300</td>
<td>$\log_2(300/200) = 0.585$</td>
</tr>
</tbody>
</table>

Figure #3-2: Schematic diagram of cable modem Internet connection.¹¹⁵

- **Connect telenursing center to Internet (FR32):** A high-capacity connection from the telenursing center to the Internet is required because of the need to connect to several patients simultaneously as well as to have access to other information available on the Internet, such as medical databases and web sites. The exact level of Internet service depends upon the size of the center, or more specifically on the number of active connections the center expects to maintain simultaneously. Currently, the design calls for the telenursing center to be its own private subnet using 10 MB/sec ethernet technology, with a 100 MB/sec or greater link to the Internet. If higher capacity links become more readily available in the future, they should be used instead.¹⁶
• **Calculate desired QoS (FR33):** Before data transmission can commence, the sender must determine the type and quantity of data being sent, and use this information to determine the desired QoS for the transmission. Applications that do not require a QoS guarantee (i.e. non-real-time data traffic such as web page queries) can skip DP33 and DP34 and send data traffic directly using UDP or TCP. Each real-time application (i.e. each application requiring QoS for its data traffic, such as video conferencing) must generate its own TSPEC specifying its current networking requirements, and then an algorithm must merge these TSPECs to determine the total QoS requirements. The algorithm must also ensure that the total desired QoS falls within the bandwidth and latency constraints of the available Internet connection, as well as possess the ability to recalculate the QoS requirements quickly when changes occur in the types of sensor and external application data being transmitted.

• **Establish / Refresh QoS Connection (FR34):** Once the desired QoS is calculated, one or more connections must be established (or refreshed if the connections already exist) between the patient's home and the telenursing center. These connections can be accomplished with RSVP, in conjunction with QoS routing techniques, to reserve resources between the patient's home, the telenursing center, and any other receivers of the information (i.e. consulting physicians or other medical specialists). Since RSVP is receiver-oriented, the patient's home must first send a Path message to the telenursing center indicating a QoS connection is desired, which will result in
the telenursing center sending a $Resv$ message to establish the reservation. If video conferencing is desired, the node originating the request will also need to send a $Resv$ message to establish the reservation in the other direction, since in this case the patient's home and the telenursing terminal are both acting as a sender and a receiver. Examples of both types of connections are provided in the simulation model, and are explained in further detail in Section 4.2. Given the types of data that the telenursing system is expected to transmit, controlled-load service and a FF-style reservation should be adequate.\textsuperscript{18} As new QoS control services emerge or as new applications are added to the system, the type of service and style of the reservation may be altered.

- **Transmit / Receive Data (FR35):** Once a resource reservation is established over the network, the data can be encapsulated into IP packets and transmitted. In addition to the standard IP headers, each packet must also include appropriate headers indicating for which applications the data is destined. In the video conferencing case, it must be assured that reservation state exists in both directions, since each node will act as both a sender and a receiver. When a sending application has finished, the sending node should indicate a change to the QoS reservation request, to reflect the termination of that application. This will reinitiate the QoS algorithm (DP33) and the QoS reservation refresh (DP34). If no other sending applications are active, a $PathTear$ message can be sent to terminate the reservation.\textsuperscript{19}
The Internet connections to patient homes (FR31) and the telenursing center (FR32) should be independent of each other. The other functional requirements are sequential processes that are ultimately dependent upon the capacity of the Internet connections. As a result, the design is partially decoupled:

\[
\begin{align*}
\text{FR31} & = \{ X \ 0 \ 0 \ 0 \ 0 \ \mid \ DP31 \} \\
\text{FR32} & = \{ 0 \ X \ 0 \ 0 \ 0 \ \mid \ DP32 \} \\
\text{FR33} & = \{ X \ X \ X \ 0 \ 0 \ \mid \ DP33 \} \\
\text{FR34} & = \{ X \ X \ X \ X \ 0 \ \mid \ DP34 \} \\
\text{FR35} & = \{ X \ X \ X \ X \ X \ \mid \ DP35 \}
\end{align*}
\]

3.2.4 Computer-Assisted Health Care Practitioner (DP4)

Once data characterizing the health of a patient is received at the telenursing center, it must be interpreted and acted upon. A telenurse monitoring a particular patient will be responsible for examining the data, detecting problems, and responding accordingly. The computer will be a very valuable tool for the telenurse, as it can be utilized to assist in interpreting and acting upon patient data.

While it is impossible to predict the full scope of the types of data to be examined or the range of actions the telenurse will be allowed to take, a framework can be developed to provide computer assistance to telenurses to help them perform their tasks, leaving the specifics of this assistance until implementation. The functional requirements of this subsystem, as they have currently been identified, are summarized in Table #3-7.
Table #3-7: Decomposition of computer-assisted telenurse (DP4).

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>41 Enable the telenurse to interpret data</td>
<td>Tools (databases, spreadsheets, etc.)</td>
</tr>
<tr>
<td>42 Allow computer to assist in data interpretation</td>
<td>Knowledge-based system agents</td>
</tr>
<tr>
<td>43 Establish connections w/ patients, doctors, etc.</td>
<td>Utilization of commercial applications (DP2) and network tools (DP3)</td>
</tr>
<tr>
<td>44 Dispatch medical help</td>
<td>Connection to local hospital and/or ambulance dispatcher</td>
</tr>
</tbody>
</table>

- **Provide tools to allow telenurse to interpret data (FR41):** The main task of the telenurse is to interpret patient data. Standard commercially-available software tools, such as databases and spreadsheet programs, can greatly facilitate this task. These tools can show trends in a patient’s data and compare it to the patient’s previous medical history and accepted medical standards. Some tools, such as databases, may be the same as those already integrated above (DP2), while others (e.g. applications that do not require a network interface) may be added by embedding them within an interface similar to the generic application interface (DP24).

- **Provide tools to allow computer to interpret data automatically (FR42):** In addition to the telenurse's interpretation of the data, it may be useful to have software programs which can, in conjunction with the software tools specified in DP41, automatically examine the data and use an inference engine to come up with interpretations and suggestions, based on a predefined set of rules. This type of system is known as a *knowledge-based system agent.* While such agents can offer medical suggestions and alert the telenurse to trends in
the data, it is important that the final judgment as to an appropriate response to the data come from a human operator, and not only from a computer program. Two classes of agents are conceivable as being useful in the telenursing system: *dumb agents* could be responsible for simple tests on the data in real-time, to ensure that there are no immediate problems, while *smart agents* could perform more complex analyses to interpret data trends and detect more subtle problems. These agents are summarized in Table #3-8.

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>421 Test data for problems, obscurities, etc.</td>
<td>&quot;Dumb agents&quot; to compare data against ranges, etc.</td>
</tr>
<tr>
<td>422 Interpret data trends, compare with medical knowledge and provide judgments</td>
<td>&quot;Smart agents&quot; which can learn and adapt over time</td>
</tr>
</tbody>
</table>

While the details of these agents are not specified here, it is logical to assume that smart agents (DP422) may utilize at least some properties of the dumb agents (DP421). As a result, this design is decoupled.

\[
\begin{bmatrix}
FR421 \\
FR422
\end{bmatrix} = \begin{bmatrix}
X & 0 \\
X & X
\end{bmatrix} \begin{bmatrix}
DP421 \\
DP422
\end{bmatrix}
\]

- **Establish contact with patient (FR43):** Most of the time, the patient will be in reasonably healthy condition. As a result, the telenurse need only continue to monitor the data in order to ensure that the patient’s health status does not change. From time to time, however, it may become necessary for the telenurse to contact individual patients to see how they are feeling, or
otherwise to discuss a patient’s condition with physicians, family members, and/or medical specialists. To accomplish this, the video conference interface and networking tools (DP21 and DP3) can be used. As discussed above, the telenursing center, the patient’s home, and any other participants in the conference act as both senders and receivers. Hence, every node must establish reservation state in both directions.\textsuperscript{[21]} It is also conceivable that, at some point in the future, the telenursing center may wish to send other information to patient homes. This is easily accomplished using the network tools described previously.

- **Dispatch medical help (FR44):** In emergency situations, it may become necessary for the telenurse to dispatch an ambulance or other medical assistance to the patient’s home. This can be accomplished by contacting appropriate emergency medical teams via the Internet or by other means, such as radio or telephone, to ensure that the patient receives immediate care.

Even though the interpretation of the data (FR41 and FR42) leads to an appropriate response (FR43 and FR44), the mechanisms provided to perform the interpretation and the response are uncoupled. As a result, the design is partially decoupled:

\[
\begin{align*}
\{ \text{FR41} \} & = \begin{bmatrix} X & 0 & 0 & 0 \end{bmatrix} \{ \text{DP41} \} \\
\{ \text{FR42} \} & = \begin{bmatrix} X & X & 0 & 0 \end{bmatrix} \{ \text{DP42} \} \\
\{ \text{FR43} \} & = \begin{bmatrix} 0 & 0 & X & 0 \end{bmatrix} \{ \text{DP43} \} \\
\{ \text{FR44} \} & = \begin{bmatrix} 0 & 0 & X & X \end{bmatrix} \{ \text{DP44} \}
\end{align*}
\]
3.2.5 Graphical User Interface (DP5)

Once the core technologies specified in the previous DPs are in place, an interface must be implemented in order to tie everything together into a package that can be understood and used easily by patients and medical personnel. This interface must also be robust enough to handle the collection of algorithms and applications which have been specified in previous sections.

It is also possible that, at some future point in time, this user interface may need to be integrated with other user interfaces on a higher hierarchical level of the system design. For example, it may become desirable for the patient’s computer to integrate control of other systems around the home, such as lighting, security, and heating. The system may also need to allow a medical practitioner to access these systems remotely from the telenursing center. While the current telenursing system does not include such capabilities, the interface must be expandable to new potential tasks.

This functional requirement can be satisfied by having a high level graphical user interface (GUI) which implements various modules that interact with the core control software. While the modules discussed in this section are discussed in terms of their functionality (i.e. what tasks they need to accomplish), their implementation will, in most cases, consist of a bridge between the GUI and the actual implementation software. For example, the network module is responsible for making functional calls, with the appropriate inputs, to the software daemons which actually establish the resource reservation, packetize the data, and transmit the data over the network. Table #3-9 summarizes the different types of modules that the system must currently provide.
Table #3-9: Decomposition of graphical user interface (DP5).

<table>
<thead>
<tr>
<th>Functional Requirements</th>
<th>Design Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>51 Guide user through appropriate steps and limit access to</td>
<td>Window-based GUI</td>
</tr>
<tr>
<td>core functionality</td>
<td></td>
</tr>
<tr>
<td>52 Ensure security of data</td>
<td>Security Module</td>
</tr>
<tr>
<td>53 Interface with network</td>
<td>Network Module</td>
</tr>
<tr>
<td>54 Interface with sensors</td>
<td>Sensor Module</td>
</tr>
<tr>
<td>55 Interface with external applications</td>
<td>External Application Module</td>
</tr>
</tbody>
</table>

The design is partially decoupled, since each GUI module may need to interact with preceding modules. These modules are outlined briefly in the following subsections.

\[
\begin{align*}
\text{FR51} &= \begin{bmatrix} X & X & 0 & 0 & 0 \end{bmatrix} \\
\text{FR52} &= \begin{bmatrix} X & X & X & 0 & 0 \end{bmatrix} \\
\text{FR53} &= \begin{bmatrix} X & X & X & X & 0 \end{bmatrix} \\
\text{FR54} &= \begin{bmatrix} X & X & X & X & 0 \end{bmatrix} \\
\text{FR55} &= \begin{bmatrix} X & X & X & 0 & X \end{bmatrix}
\end{align*}
\]

3.2.5.1 Window-Based GUI (DP51)

Since most users of this system will not be computer experts, it is very important that the interface is easy to use by patients and telenurses. As a result, a window-driven GUI is required which can walk the user step-by-step through the different modules contained within the interface. The GUI must interact with the modules specified above in previous DPs, as well as any modules which may be added in the future.

The GUI could be implemented in any object-oriented computer language, though the best choice currently available is Java. While a language such as C++ is more sophisticated, Java applications can easily interact directly with a network and are designed to be easily portable to several operating system platforms. Java is also capable of interacting with COM objects (DP24). While Java is still maturing at this point in
time, future developments to the language over the next few years should make the implementation of the GUI in Java a viable option.

3.2.5.2 Security Module (DP52)

Since medical information is private, safeguards must be included to prevent unauthorized access to patient data. Security issues must also be considered in all aspects of the design in order to prevent security loopholes from occurring. The areas that have been identified as requiring security, and possible mechanisms to handle such security, are summarized in Table #3-10. Further decomposition of these DPs will most likely be necessary in order to specify how these security mechanisms will be implemented.

Table #3-10: Decomposition of security module (DP52).

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>521 Secure nodes at telenursing center</td>
<td>ID cards, secure computer rooms, video cameras, security pads to lock down computers, etc.</td>
</tr>
<tr>
<td>522 Secure nodes at patient’s home</td>
<td>Security pads, etc.</td>
</tr>
<tr>
<td>523 Prevent users from disabling system</td>
<td>File and application permissions, passwords, auto-reboot and recover sequences, etc.</td>
</tr>
<tr>
<td>524 Secure data traversing network</td>
<td>Data encryption schemes, RSVP policy control, etc.</td>
</tr>
</tbody>
</table>

In the ideal case, the security mechanisms should remain independent of each other, leading to an uncoupled design.

\[
\begin{align*}
\begin{bmatrix}
FR521 \\
FR522 \\
FR523 \\
FR524
\end{bmatrix} &=
\begin{bmatrix}
X & 0 & 0 & 0 \\
0 & X & 0 & 0 \\
0 & 0 & X & 0 \\
0 & 0 & 0 & X
\end{bmatrix}
\begin{bmatrix}
DP521 \\
DP522 \\
DP523 \\
DP524
\end{bmatrix}
\end{align*}
\]
While the hardware security measures (e.g. video cameras, ID cards, etc.) will be controlled independently, the software security systems (i.e. DP513 and DP514) can be controlled through special windows provided in the GUI in conjunction with the computer’s operating system (e.g. utilizing the file security permissions of Windows NT). Access to these screens could be limited by an administrative-level password, to prevent willful tampering or accidental changes to the security permissions by ordinary users, including both patients and telenurses.

3.2.5.3 Network Module (DP53)

The network module is responsible for providing a bridge between the GUI and the networking tools provided in DP3. The core network technologies are embedded into the operating system of the computer as procedures and daemons. As a result, this module must accept configuration parameters quantifying the QoS requirements of the data from the sensors and external applications, use these parameters to compute the overall desired QoS, and provide this information to the RSVP daemon to establish the QoS connection. In addition, this module must interact with the embedded network protocols which encapsulate and transmit packets over the network. This module must also interact with the security module (DP52) in order to ensure data security as well as to prevent unauthorized alteration of important network parameters. The responsibilities of this module are summarized in Table #3-11.

Under normal usage conditions, the patient or the telenurse should not need to make any modifications to the network module. However, this interface is necessary for the initial setup and configuration of the telenursing system, as well as for any manual adaptation of network settings due to unusual network conditions.
Table #3-11: Decomposition of network module (DP53).

<table>
<thead>
<tr>
<th>Functional Requirements (FRs)</th>
<th>Design Parameters (DPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>531 Interact with security module</td>
<td>Function calls to the appropriate security procedures</td>
</tr>
<tr>
<td>532 Define QoS parameters</td>
<td>GUI screen to modify bandwidth and latency defaults</td>
</tr>
<tr>
<td>533 Control parameters related to network configuration settings</td>
<td>GUI screen to change network addresses, program locations, etc.</td>
</tr>
</tbody>
</table>

The design is partially decoupled, since both GUI screens (DP532 and DP533) depend upon the interface with the security module:

\[
\begin{bmatrix}
FR531 \\
FR532 \\
FR533
\end{bmatrix} =
\begin{bmatrix}
X & 0 & 0 \\
X & X & 0 \\
X & 0 & X
\end{bmatrix}
\begin{bmatrix}
DP531 \\
DP532 \\
DP533
\end{bmatrix}
\]

3.2.5.4 Sensor Module (DP54)

The sensor module provides a bridge from the GUI to the sensors. GUI windows should be provided which allow users to perform certain functions on the sensing equipment, including hardware tests, configuration, calibration, turning sensors on and off, specifying normal ranges, and so forth. In order to accomplish this, the sensor module interacts with the variables and functions specified in the generic sensor interface (DP11).

3.2.5.5 External Application Module (DP55)

The external application module provides a bridge from the GUI to the external applications. In structure, this module is very similar to the sensor module. Windows are provided which allow users to perform certain actions with external applications, such as launching the applications, specifying items such as command-line parameters, file
locations, application settings, and so forth. To accomplish this, the external application module interacts with the variables and functions specified in the generic external application template (DP24).

3.3 OVERALL SYSTEM ARCHITECTURE

Once the design of a complex system, such as the one presented in this chapter, is formalized by design matrices, it is useful to analyze the design graphically. These graphs can offer clues to implementation issues, as well as pointing out specific areas of the design which may require further development.

The first step is to generate a tree diagram which demonstrates the layout of the design decomposition. This is used to identify the leaves, or endpoints, in the system design. Assuming that the number of FRs correspond to the number of DPs on a particular hierarchy level, the tree diagrams should reveal the same pattern for the FRs and the DPs. This also provides a good check to ensure that all of the FRs were addressed by DPs in the decomposition. The tree diagrams for the telenursing system are provided in Figures #3-3 and #3-4.

The leaves of the tree can be transformed into independent modules, which represent the non-zero diagonal elements of the design matrices relating the FRs to DPs. These modules are then joined by junctions in a module-junction structure diagram, which represents the relationship between them (i.e. uncoupled, decoupled, or coupled). The module-junction structure diagram should roughly maintain the pattern of the tree diagram. This diagram is shown in Figure #3-5.
Once this diagram is specified, it can be used to generate a flow chart diagram, which shows how design information must flow through the system during implementation. Thus, it represents the order in which modules must be constructed in
order to properly satisfy the design requirements of the system. This diagram is shown in Figure #3-6. There are three types of junctions in the module-junction structure and flow chart diagrams, as specified in Table #3-12 below.

Table #3-12: Junction types and relationships for system architecture diagrams.

<table>
<thead>
<tr>
<th>Junction Type</th>
<th>Relationship</th>
<th>Flow Diagram Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S (Summation)</td>
<td>Uncoupled</td>
<td>Parallel summation of modules (order of operation does not matter)</td>
</tr>
<tr>
<td>C (Control)</td>
<td>Decoupled</td>
<td>Sequential processing of modules (order of operation is critical)</td>
</tr>
<tr>
<td>F (Feedback)</td>
<td>Coupled</td>
<td>Feedback loop of sequentially processed modules (iteration is required)</td>
</tr>
</tbody>
</table>

Figure #3-5: Module-junction structure diagram for the telenursing system design.
Figure #3-6: Flow chart for the telenursing system design.
CHAPTER 4: NETWORK SIMULATION MODEL

In the design of any large scale system, simulation analysis of key components can be useful in determining the system’s capabilities and limitations. For the telenursing system, one of the most important components to be analyzed is the behavior of the network connection between the patient’s home and the telenursing center. As a result, a network simulation of RSVP that models the behavior of the network to be used in the telenursing system is useful in measuring the overall performance of the system.

Using the OPNET modeling environment, a model of the network technology used in the design of the telenursing system has been developed. This model incorporates the basic features of RSVP required to make the protocol usable as well as traffic generators to simulate the expected types of data traffic. Section 4.1 describes the OPNET modeling environment. Section 4.2 explains the methodology used in creating reservation state for unicast and multicast applications. Section 4.3 outlines how reservations are established in the model itself by walking through simulation events. Section 4.4 discusses expected network and subnetwork topologies.

4.1 THE OPNET SIMULATION ENVIRONMENT

The Optimized Network Engineering Tools (OPNET) is a commercial simulation package manufactured by MIL3 of Arlington, VA. It utilizes a Discrete Event Simulation approach, which allows large numbers of closely-spaced events in a sizable network to be represented. OPNET models consist of network components interconnected by perfect links, which can then be degraded to represent more realistic link behavior.
The behavior of each component in the model is represented by a state-transition diagram. The process that takes place in each state is described by a program written in C, which makes OPNET-based models relatively easy to port to other modeling environments. It also enables considerable detail to be modeled in a large-scale network, making the models more representative of actual network behavior.

The state transition diagrams for the process models used in the telenursing system simulation are presented in Appendix B.

4.2 Establishing Reservations

In this simulation model, nodes may be both senders and receivers. This is very important for interactive applications, such as video conferencing, where reservation state must be established in both directions. RSVP was designed primarily for use with multicast traffic; therefore, many of the problems addressed in RSVP research and development deal with multicast issues. A method, however, has been developed here to use RSVP to reserve resources for a unicast connection as well. In the telenursing system, it may become desirable for some interactive applications to be unicast, because certain types of medical data will be of interest only to the patient and the telenurse. Keeping such interactive traffic outside of the bounds of multicast routing may be potentially desirable in terms of scalability and security issues.

Considering that interactive applications in the telenursing system can be either multicast or unicast, there are two reservation approaches incorporated into the simulation model. The first approach establishes a multicast-interactive reservation, where a node subscribes to a reservation group. Once the subscription is established, the node sends its data to the group, and receives data addressed to the group from other senders. The
second approach establishes a unicast-interactive bi-directional reservation between two nodes. Both reservation approaches assume a FF-style reservation to conform to the design of the telenursing system. These reservation approaches are explained in more detail in the following subsections.

4.2.1 Multicast-Interactive Reservations

In the multicast-interactive reservation approach, a node subscribes to a multicast group as both a sender and a receiver, and it transmits data traffic to and receives data traffic from other nodes subscribed to the group. It is important, therefore, that there be at least two subscribers to a particular multicast group before data traffic is sent, so that network resources are not wasted by a node sending data to itself. A schematic diagram demonstrating how a multicast-interactive session is established between two nodes is shown in Figure #4-1.

![Figure #4-1: Multicast-interactive reservation established between two nodes. This type of interaction must occur between every node subscribed to a multicast group.](image-url)
After a node subscribes to a new multicast group, it will send Path messages to the group periodically while it waits for another node to join. When a Resv message is received by the node in response to the Path message, data can then be sent to the group. Other nodes that join the multicast group later will detect senders from the periodic Path refreshes and subscribe accordingly. Figure #4-1 shows this procedure for two nodes sending data to a group simultaneously, though the process is the same for two nodes sending data asynchronously or for a node joining a preexisting group. In those cases, however, some hosts will transmit data while other hosts are still establishing reservation state. A receiver node must be aware of each sender explicitly, since the design of the telenursing system calls for FF-style reservations.

4.2.2 Unicast-Interactive Reservations

As in a multicast-interactive session, two nodes involved in a unicast interactive session must first establish reservation state in both directions. A schematic diagram of a unicast-interactive session being established between two nodes is shown in Figure #4-2.

In this type of connection, one node acts as the primary sender to initiate the interaction. The other node, upon receiving a unicast Path message from the primary sender, will send both a Resv message and a Path message in response. Upon receipt of the Resv message, the primary sender will begin to transmit data, and upon receipt of the other Path message, the primary sender will send a Resv message in response. Once this Resv message is received by the second node, it can begin to transmit its own data traffic.
Since two separate reservations are established, it is possible that the reservations will trace different paths through the network. As a result, ending a session requires both reservation states to be torn down independently. This can be accomplished either by the disconnecting node sending both PathTear and ResvTear messages simultaneously, or by the non-disconnecting node transmitting a PathTear message of its own in response to receipt of the PathTear message from the disconnecting node. This latter method has been implemented in the simulation model of the telenursing system, and is described in Section 4.3.5.
4.3 Simulation Model

This section discusses the procedures implemented in the simulation model to establish, send data across, and tear down a reservation. Each node included in the model is comprised of several process models, which are outlined in detail in Appendix B.

The process begins in a host node at the node's application layer. The host node for the telenursing center (i.e. a host connected to an ethernet subnet) is shown in Figure #4-3. Two applications capable of establishing reservations are provided (one for unicast and one for multicast). An application that wishes to send and/or receive messages through a reservation must first obtain a session ID from the local RSVP daemon. Once this is accomplished, the application can begin to send Path messages.

![Figure #4-3: Telenursing center host node (ethernet).](image)
In the case of a multipoint-interactive session, a subscription to a particular multicast group is made through the IGMP daemon, and messages are addressed to the multicast group. In the unicast case, messages are addressed to the receiving node.\textsuperscript{21}

The *Path* message is first routed to the subnet's local gateway, where the incoming *Path* message is examined and checked against any existing path state in the node. A telenursing (i.e. ethernet) gateway is shown in *Figure #4.4.*

![Figure #4-4: Telenursing center gateway node (ethernet).](image)

If new path state has been created or the information in the old path state has been updated, the *Path* message is routed to the outgoing interface which most closely leads to the destination address. For multicast packets, copies of the message will be sent out
over all outgoing interfaces which are subscribing to the multicast group. A message may pass through several gateway and/or backbone routers in this fashion, until it reaches its destination subnet. A backbone router is shown in Figure #4-5.

Once the Path message is received by another host, that node will send a Resv message along the reverse network path followed by the Path message in order to establish or update reservation state within each of the router nodes. Once the reservation state is established, data can be transmitted. When a sender application wishes to terminate its session, it will send a PathTear message. This PathTear message will propagate through the network, removing path state and corresponding reservation state from the router nodes.

The following subsections outline this procedure in more detail, by tracing the Path, Resv, Data, and PathTear messages through the node and process models. State
transition diagrams and an in-depth explanation of the process models in this simulation are provided in Appendix B.

4.3.1 Establishing a Session ID

When the application initially loads, it schedules a probabilistically-determined interrupt to join a new session. This interrupt occurs at least 10 seconds into the simulation run, in order to allow the process models in the network to load initial configuration data. When the interrupt is triggered, the application will determine whether or not a session ID has been assigned to it. If one has not been assigned, a destination address is selected at random from a uniform distribution, and a Session Call is sent to the local RSVP daemon, indicating the destination IP address and matching port. In the multicast case, a multicast group is selected. In addition, the node initializes a Sender Directory used for keeping track of messages from different senders and enters its own characteristics in the first slot, in order to discriminate any of its own data traffic from data traffic sent by other nodes. In the multicast case, the application will also send a Join Request to the IGMP daemon with the selected group ID, so that the node will receive data traffic from that multicast group if it is not doing so already.

When the local RSVP daemon receives a packet from the application, it first must determine what type of message is being received. For a Session Call, the IP address and port information of the destination are obtained from the packet, the lowest available session ID number is assigned, and the address and port information are stored in a Session Directory for future reference. At this point, the local RSVP daemon returns a Session Event Upcall to the application, indicating the value of the assigned session ID.
When the application receives a packet from the RSVP daemon, it must determine the type of upcall message being received. For a Session Event Upcall, it extracts the session ID and prepares to send a Path message.

4.3.2 Sending a Path Message

A Path message sent by the application is triggered either by the receipt of a session ID from the RSVP daemon or from a self-scheduled refresh interrupt, as described below. If the application is a sender, it will make a Sender Call to the local RSVP daemon, which includes the session ID and the default sender values for the desired bandwidth and delay. In addition, interrupts are scheduled for sending refresh Path messages and for terminating the session.

When a Sender Call arrives in the RSVP daemon, the information contained in the message, along with information stored in the Session Directory, is used to create a Path message. Information is also stored in a local Sender Directory to later identify the session. The previous node’s IP address (PHOP) and logical interface handle (LIH) are set to the host’s IP address. Once the Path message is composed, it is encapsulated into an IP datagram and sent to the network layer, where the packet is then transmitted to the local gateway router.

When the packet reaches a gateway or backbone router, it is passed to the RSVP transport layer daemon for processing, where the message is extracted from the IP datagram and identified. The information contained in the Path message is compared with information stored in the Path State Block (PSB), in order to determine if path state already exists for this sender. The format of the PSB is provided in Table #4-1. If an entry in the PSB exists, the PHOP, LIH, and TSPECs are examined to see if any
information has changed. If no matching PSB entry is found, a new entry is created from the information contained in the Path message. If a new entry is created or an old entry is updated, the Path message is sent out along all relevant outgoing links. This process is repeated within all backbone and gateway nodes that the Path message passes through from its source to its destination(s).

Table #4-1: Format of Path State Block (PSB) within RSVP routers.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender Descriptor</td>
<td>Structure</td>
<td>Contains the sender IP address and port, along with the sender's bandwidth and delay capacity.</td>
</tr>
<tr>
<td>PHOP</td>
<td>IP Address</td>
<td>Previous Hop Address (address of previous node).</td>
</tr>
<tr>
<td>LIH</td>
<td>IP Address</td>
<td>Logical Interface Handle (usually previous node).</td>
</tr>
<tr>
<td>RSBList</td>
<td>List</td>
<td>RSB indices listing the receivers subscribing to the sender indicated by the PSB entry.</td>
</tr>
<tr>
<td>RSBListLen</td>
<td>Integer</td>
<td>Current length of RSBList.</td>
</tr>
<tr>
<td>Interface</td>
<td>Integer</td>
<td>Incoming interface ID for RSVP messages from this sender.</td>
</tr>
<tr>
<td>TC Bandwidth</td>
<td>Double</td>
<td>Total amount of bandwidth allowed for this sender along its incoming interface.</td>
</tr>
</tbody>
</table>

When a Path message is received at a subscribing host’s RSVP daemon, the Path message is examined and the relevant sender information is sent to the appropriate application in a Path Event Upcall. Each application is identified by a unique port number contained within all RSVP messages. When the application receives the Path Event Upcall, the relevant information is extracted and the sender directory is updated accordingly.
4.3.3 Sending a Resv Message

Since the applications considered in this simulation model can act as both senders and receivers, a host node can transmit both Path and Resv messages. As described in Section 4.2, the first Resv message sent by a node is in response to its receipt of a Path message. Hence, the application sends a Reserve Call to the local RSVP daemon in response to a Path Event Upcall. The Reserve Call contains the Session ID, reservation style (either FF, SE, or WF), a flag indicating whether or not a reservation confirmation is desired, and the current list of senders along with the bandwidth and delay requested for each in the case of a FF-style reservation.

Upon receiving the Reserve Call, the RSVP daemon reads in all of the information from the message, creates a Resv message with a complete flow descriptor, and keeps a copy of the flow descriptor in a Reserve Directory. The Resv message is then encapsulated into an IP datagram and transmitted to the local gateway.

When a Resv message reaches a gateway or backbone router, it is passed to the RSVP transport layer daemon for processing. Information from the Resv message is extracted, and several steps are taken. First, the existing PSB is examined to see whether or not the senders listed in the Filterspec of the Resv message are registered with the node. If a sender does not have a matching PSB entry, it is removed from the Filterspec. Once the PSB is examined, a quick check of the traffic control block (TCB) is made. The TCB keeps track of the total amount of bandwidth and delay resources reserved on a particular network interface. This check is used to ensure that the bandwidth and delay requested for each sender do not violate the bandwidth and delay restrictions on the incoming interface used by the sender’s data traffic. If any resource violation occurs, the
sender is discarded from the Filterspec. Resource violation occurs if any of the following are true:

- the requested bandwidth is greater than the bandwidth achievable by the sender, as specified by the sender in the sender template;

- the requested minimum delay is less than the minimum delay achievable by the sender, as specified in the sender template;

- the maximum bandwidth available over a link is less than the sum of the total resources currently reserved on the link added to the new bandwidth request minus any bandwidth currently reserved for the sender;

- the requested minimum delay is less than the minimum delay achievable over the link.

Once the TCB is checked, the Reservation State Block (RSB) is examined to see if a reservation currently exists for the unicast receiver or multicast group. The format of the RSB is provided in Table #4-2. If no matching RSB entry is found (based upon the destination IP address and port for the traffic, along with the IP address of the next hop in the path), a new RSB entry is created. If a matching RSB entry is found, the flow descriptor is examined to see if there are any changes to the flowspec, indicating a change of desired resources for particular senders, or the Filterspec, indicating the addition of new senders to the list. If a new RSB entry was created or an existing RSB entry was modified, updated Resv messages are prepared. Every sender on a particular incoming interface (i.e. using the same PHOP) requiring an updated Resv message is listed in the flow descriptor of a new Resv message which is sent to the previous hop. In addition, the TCB is updated to reflect the resource reservation changes indicated by the updated Resv message. This process continues through all of the backbone and gateway routers until
the Resv messages reach the senders, or until no further reservation state updates are required.

Table #4-2: Format of Reservation State Block (RSB) within RSVP routers.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session Address</td>
<td>IP Address</td>
<td>The destination address (unicast or multicast) for this RSB entry.</td>
</tr>
<tr>
<td>Session Port</td>
<td>Integer</td>
<td>The application port number using this RSB entry.</td>
</tr>
<tr>
<td>Next Hop</td>
<td>IP Address</td>
<td>The next hop address in the path to the receiver.</td>
</tr>
<tr>
<td>Flow Descriptor</td>
<td>List</td>
<td>List of senders and the corresponding resources requested.</td>
</tr>
<tr>
<td>FDLen</td>
<td>Integer</td>
<td>Current length of the Flow Descriptor list.</td>
</tr>
<tr>
<td>Style</td>
<td>Integer</td>
<td>The reservation style used by this receiver (only FF-style is currently supported).</td>
</tr>
<tr>
<td>Resv Confirm Address</td>
<td>IP Address</td>
<td>Address to send a reservation confirmation message once complete reservation state is established (currently unsupported).</td>
</tr>
</tbody>
</table>

When the Resv message arrives at a sender node’s RSVP daemon, the flow descriptor information from the Resv message is used to compare the IP addresses and ports of the senders listed in the Filterspec against the local IP address and port. If there is a match, the corresponding bandwidth and delay request is extracted and sent to the appropriate application in a Resv Event Upcall, along with the reservation style type.

When an application receives a Resv Event Upcall, the information it contains is incorporated into a Reservation Directory. This information is not currently used, but is recorded for potential future expansion of the model. In addition, a flag is set to inform the application that at least one Resv message has been received, so that data can be transmitted.
4.3.4 Sending Data Messages

Once an application has received a Resv message, it can begin to transmit data. No data packets are transmitted by a sender node until at least one Resv message has been received. When the application is ready to transmit data, it sends a Data Call to the local RSVP daemon, which includes a mock data block as well as the session ID. The length of a data block and the frequency at which new data blocks are created are adjustable, in order to reflect the desired amount of traffic to be sent (e.g. a video conference application transmitting at 128 kB/sec may create 64 data blocks, each of length 2 kB, per second). A simulation probe records the fact that a new data packet has been sent, and an interrupt is scheduled to send another data packet. In the local RSVP daemon, the Data Call is used to create a Data message, which is then encapsulated into an IP packet and transmitted to the local gateway router.\footnote{71}

Once the Data message reaches a gateway or a backbone router’s RSVP transport layer, the PSB is searched to ensure that sender state does indeed exist for the data’s source node, and that there is at least one receiving node with an RSB entry established. If this is the case, the Data message is sent unmodified to the next node, duplicated as necessary in the multicast case. This process continues through all of the gateways and backbone routers through which the message passes, until reaching the receivers.

When the Data message is received by a host’s RSVP daemon, the data block and the sender information are extracted from the IP datagram and sent in a Data Event Upcall to the application layer. In the application, this information is extracted, and a counter is incremented in the Sender Directory to indicate that data have been received from that sender. In addition, a simulation probe indicates that data have been received.
4.3.5 Ending the Reservation

When an application first begins to transmit *Path* messages, it sets a probabilistically-determined interrupt to end its session. When the interrupt occurs, all reservation states (i.e. sender and receiver) established for the current session ID are terminated. A *Release Call* is then sent to the local RSVP daemon indicating the current session ID, and a *Terminate Request* is also sent to the local IGMP daemon in the multicast case. In addition, the total amount of data received is recorded by a simulation probe.

When the *Release Call* arrives at the local RSVP daemon, the session ID is interpreted. Currently, only *PathTear* messages have been implemented, though the Reserve Directory within the RSVP daemon is also erased if the session ID indicates an existing entry in the Reserve Directory. If an entry for the session ID exists in the Sender Directory, a *PathTear* message is composed from the information in the Sender Directory, encapsulated into an IP datagram, and transmitted to the local gateway router. Then, the Sender Directory is also erased.

When the RSVP transport layer in a gateway or a backbone router receives a *PathTear* message, the PSB is examined for a match. If a matching PSB entry is found, all of the RSBs which hold reservation state for this sender are examined, and the corresponding sender is removed from their flow descriptors. If an entry in the RSB is empty as a result (i.e. there are no more senders listed), the RSB entry is removed. Once this is accomplished, the TCB entry for the sender’s incoming interface is updated, so that resources are no longer reserved for that sender, and the matching PSB entry is also erased. The *PathTear* message is then forwarded along all relevant outgoing interfaces.
This process is repeated in each gateway or backbone router until the PathTear message reaches the receiver(s).

When the PathTear message is received by a receiver's RSVP daemon, the sender IP address and port are sent to the application layer in a PathTear Event Upcall, so that the application is aware that it should not expect to receive any more data from that sender. When the PathTear Event Upcall is received by the application, it erases that sender information from its Sender Directory. In unicast-interactive sessions, receipt of a PathTear also triggers the node to send its own PathTear for the session, in order to remove the reservation state in the reverse direction.

4.4 Network Model

Within the OPNET modeling environment, nodes are comprised of a combination of process models, subnets are comprised of a combination of nodes, and networks are comprised of a combination of subnets. This allows the network engineer to view the model from several different abstraction levels. By examining the subnets and the top-level network in this simulation model, an overall view of the network topology of the telenursing system can be discerned.

For the telenursing center, several terminals can be envisioned, where the telenurse at each terminal would be responsible for monitoring four or five patients simultaneously. Figure #4.6 shows a possible subnet configuration for the telenursing center, using five telenursing terminals. Such a center would be capable of monitoring between 20 to 25 patients.
To increase monitoring capacity, either more computers can be added to this subnet, or more subnets can be added. In this way, a company providing telenursing services could establish small regional centers within neighborhoods, hospitals, and nursing homes. These regional centers could communicate with each other using the same network technology already established in the design of the telenursing system, as described in Chapter 3. An example of a network topology connecting four telenursing center subnets to each other is given in Figure #4-7. This type of configuration would be emblematic of a regional telenursing center which would be able to monitor approximately 100 patients simultaneously.
The network model for cable-modem networks is more complex than the model for ethernet networks. An example of a standard cable-modem network is shown in Figure #4-8. Each end-node in this diagram represents a patient's home. These nodes are connected by a combination of coaxial cables and fiber-optic cables to backbone routers with high-capacity network links to the Internet.
There was insufficient time during this project to integrate the RSVP model developed here with the cable-modem model included in OPNET 3.0. As a result, while the ethernet RSVP model was tested for simple network configurations to ensure the accuracy and feasibility of the model, more work is required in order to use the simulation model to obtain meaningful results about the ultimate network performance of the telenursing system.

Nevertheless, the existing simulation model can be used to show the overall network topology of the telenursing system. In Figure #4-9, an example telenursing system topology is shown for the Boston metropolitan area. Here, several telenursing centers (as shown in Figure #4-7) are interspersed with the regional cable modem network (as shown in Figure #4-8), and are connected to each other by Internet backbone routers. The telenursing system topology for all major cities in the United States could be laid out in a similar fashion. For more remote areas, regional cable-modem networks could be connected to the Internet via satellite network links to telenursing centers located in metropolitan areas, as shown in Figure #4-10.

Figure #4-9: Sample metropolitan telenursing system network topology. Here, a telenursing system topology is presented for the Boston metropolitan area.
The network topologies presented above provides a good model for testing the performance of the telenursing system for different regional areas, once the RSVP model developed in this chapter can be integrated with the cable-modem network model. In order to accomplish this within the OPNET modeling environment, simulation probes can be set in various nodes located in different sections of the network, so that various statistics, such as network throughput, delay, duration of sessions, group membership, and total data transmitted can be measured and analyzed.

Figure #4-10: Sample rural telenursing system network topology. Here, satellite backbone links connect the remote cable-modem area to telenursing centers in a metropolitan area.
This page intentionally left blank.
CHAPTER 5: CONCLUSION

The field of home-based health care is likely to generate rapid commercial growth and increased public interest over the next several years. In order to meet the emerging demand for high quality and low cost health care, new technologies must be developed. The d’Arbeloff Laboratory for Information Systems and Technology has undertaken a mission to contribute to the development of health care technology by forming a consortium of research groups and corporate health care providers to create a comprehensive home-based health care entity. This thesis has presented the design of one part of this entity, to be used for monitoring patients remotely from a centralized facility. The telenursing system presented in this thesis will enable health-care providers to detect patient medical problems and crises in a timely manner, without enduring the expense of providing on-site health care practitioners. The use of this telenursing system should lead to decreased costs and increased health care provider productivity, which will ultimately improve the quality of home-based health care.

Since the telenursing system is defined by a large number of functional requirements that may change over time, it is important to understand how the functional requirements are impacted by the design decisions that have been made. The axiomatic design technique presented in Chapter 3 provides a methodology for mathematically and graphically managing design decisions. This technique enables the systems engineer to perceive and understand the complex relationships between the design parameters chosen and the functional requirements the design must address. The use of axiomatic design to generate the design of the telenursing system has outlined specific solutions to achieve the system’s tasks, as well as highlighted areas, such as the proposed network strategy to
transmit real-time data over the Internet, which require further research and understanding before the telenursing system can be implemented.

The simulation model developed in Chapter 4 serves to provide a better understanding of the overall behavior and performance of the telenursing system, since these are significantly affected by the behavior and performance of RSVP to provide QoS network connections over the Internet. As the telenursing system expands to cover a large number of patients over a wide geographical area, the performance of the network connection will highly dictate the feasibility and usefulness of this system as a tool to monitor home-based patients.

Since it is impossible to predict the full range of functional requirements that the system may perform in the future, it would be presumptuous to claim that the telenursing system design presented in this thesis is the “best” possible design for this application over the full range of its life. However, the axiomatic design approach shows the existing relationship between the functional requirements of the system and the design parameters chosen to satisfy them. If future designers wish to change design parameters or add new functional requirements, they will already have the framework required to make design changes without adversely affecting the overall system. As a result, the telenursing system is robust enough to handle expansion in both expected and unexpected ways.

5.1 FUTURE WORK ON THE TELENURSING SYSTEM DESIGN

Since the implementation of the telenursing system is beyond the scope of this thesis, the design of this system is ultimately on a conceptual level. When the system is actually implemented, many issues will have to be addressed. Throughout the design decomposition presented in Chapter 3, many areas were highlighted as requiring further
development, including the construction of sensor, application, and network interfaces with the real-time operating system, along with the creation of a window-driven GUI to provide a front-end user interface. In addition, a medical knowledge-base will need to be established, so that patient information as well as general medical knowledge will be available online. Once all of this is completed, patients and medical practitioners will also have to be trained in order to use the system to its full potential.

The use of axiomatic design to generate the design of the telenursing system is intended to facilitate future development. The axiomatic design approach itself is also an evolving methodology. As tools are added to the technique to further expand its ability to be used in the development of large-scale systems, future expansion and adaptation of the design of the telenursing system will also become easier.

5.2 Future Work on the Simulation Model

The simulation model presented in Chapter 4 provides a good intuitive understanding of how QoS network technology will impact the network performance of the telenursing system. Unfortunately, the model is not sufficiently developed to provide meaningful results about the network behavior of the telenursing system, since it still requires RSVP to be integrated into the model of cable-modems so that the performance of patient home network connections can be analyzed. In addition, only the basic elements of RSVP were implemented in the model; less essential elements of the protocol, such as error messages, reservation confirmation messages, and interfaces with QoS routing technologies, were omitted. A more complete model must be developed, so that the telenursing system network model has access to all of the features of the RSVP protocol.
It is probable that new technologies will emerge in the future which will supplement or even surpass the design solutions presented in this thesis. This is especially likely in the field of QoS networking. The specifications of RSVP as presented in Chapter 2 are still being developed and defined, and as a result they continue to undergo radical changes. While a simulation model of RSVP has been developed here, and individual RSVP testbeds have been built elsewhere, RSVP has not been implemented on a large scale, and hence it is still ultimately unknown how practical this protocol will be for large-scale QoS network applications.

Since changes and improvements in QoS network technology are likely to be made in the future, the axiomatic design decomposition of the telenursing system provided in this thesis will enable future engineers to understand the logic behind design decisions. As a result, changes in QoS network technology can easily be incorporated into the telenursing system. These changes will ensure that the telenursing system remains a viable option for providing a superior level of home-based health care.
APPENDIX A: SPECIFICATION OF THE TSPEC

The most common way to specify a TSPEC to characterize the data traffic rate through a node is to use a token bucket filter. This means that a data packet must first obtain a virtual "token" from a "bucket" before being allowed to transmit. By setting the token rate $r$ and the token bucket depth $b$, the rate at which packets are transmitted can be regulated. In general, a flow will conform to its TSPEC if the amount of data sent in time $T$ does not exceed $rT+b$. Most TSPEC implementations also include the predicted peak rate of transmission $p$, the minimum packet size $m$ (packets smaller than $m$ are treated as if they are of size $m$ by the node), and the maximum packet size $M$ ($M \leq$ MTU along the path).

In the multicast case, many receivers can request data traffic from the same source. Each receiver, however, may have different QoS requirements, and maintaining separate reservations for each receiver is unscalable to a network consisting of a large number of receivers. As a result, it is necessary to define a method to order and merge flows, so that the reservation state created for a particular data flow incorporates the greatest QoS requirements specified by any of the receivers. Since each TSPEC contains multiple independent parameters, it is necessary to create a merged TSPEC which describes the "largest" requirements of all of the flows.

Two TSPECs are ordered if one can completely substitute for another. In other words, TSPEC A is considered to be greater than or equal to TSPEC B when all of the following conditions are true:

1. $r_A \geq r_B$ (token bucket rate of A $\geq$ token bucket rate of B)
2. \( b_A \geq b_B \) (token bucket size of \( A \geq \) token bucket size of \( B \))

3. \( p_A \geq p_B \) (peak rate of \( A \geq \) peak rate of \( B \))

4. \( m_A \leq m_B \) (minimum packet size of \( A \leq \) minimum packet size of \( B \))

5. \( M_A \geq M_B \) (maximum packet size of \( A \geq \) maximum packet size of \( B \))

When two TSPECs are unordered, only some of the above conditions are true. In this case, a merged TSPEC must be created at the node to describe the characteristics of the combined flow that is desired.

\[
\text{Merged TSPEC} = \max\{r_i\}, \max\{b_i\}, \max\{p_i\}, \min\{m_i\}, \max\{M_i\}
\]

There are also other comparative computations that can be done on TSPECs, depending on specific tasks the router may try to accomplish. These include:

- **Least Common TSPEC**: This is similar to a merged TSPEC, but is used strictly within the node (i.e. no parameters are passed on to another node).

  \[
  \text{Least Common TSPEC} = \max\{r_i\}, \max\{b_i\}, \max\{p_i\}, \min\{m_i\}, \max\{M_i\}
  \]

- **Sum of TSPECs**: This is used for computing the shared TSPECs of a number of flows.

  \[
  \text{Sum of TSPEC's} = \sum\{r_i\}, \sum\{b_i\}, \sum\{p_i\}, \min\{m_i\}, \max\{M_i\}
  \]

- **Minimum TSPEC**: In the ordered case, the smaller TSPEC is the minimum. If unordered, the minimum TSPEC is a merged TSPEC with the smallest token bucket rate, peak rate, minimum packet size, and maximum packet size, but the largest token bucket depth.

  \[
  \text{Merged TSPEC} = \min\{r_i\}, \max\{b_i\}, \min\{p_i\}, \min\{m_i\}, \min\{M_i\}
  \]
APPENDIX B: PROCESS MODEL STATE DIAGRAMS

This appendix provides a detailed description of the process models that have been created and/or developed for the network simulation model presented in Chapter 4.

The process models which are explained in this appendix include the following:

- Host Reserve Application (Host_rapp)
- Host Unicast Non-Reserve Application (Host_uapp)
- Host RSVP Daemon (Host_rsvpd)
- Host Multicast Daemon (Host_igmpd)
- IP Network Layer (Rsvp_IP)
- Router RSVP Daemon (Rou_Rsvp)
- Router Multicast Daemon (Router_igmpd)
- Router Transmission Queue (Rsvp_pc_fifo)

The other process models included in this simulation, namely the IP address resolution process model (ip_arp) and the ethernet models (eth_mac, eth_defer), have not been modified from the versions provided by OPNET 2.5A.

B.1 HOST RESERVE APPLICATION (HOST_RAPP)

This process model simulates an application seeking reserved resources, and is in charge of creating and destroying reservations as well as generating data traffic. The state transition diagram for this process model is given in Figure #B-1. This process model implements both the multipoint-interactive and the unicast-interactive reservation models described in Section 4.2. Only FF-style reservations are made, since this style is considered to be the most practical for video and other medically-related data traffic. Each application keeps its own table of senders, indicating which nodes are sending to the session. In this way, the application can discriminate between data traffic from each source node in the Filter Spec of the Resv message.
This process model contains the following states to perform its tasks:

- **Init State**: This state, entered at the beginning of the simulation, is in charge of initializing variables and loading the distributions used for determining session inter-arrival time, session duration, and selection of the group ID (multicast case) or destination node (unicast case). In addition, an interrupt is scheduled to join the first session. This interrupt occurs at least 10 seconds after the beginning of the simulation, to allow all of the process models to load properly. This state also loads in parameters specified at the node level, as listed in Table #B-1.

- **Idle State**: This state is the default state when the process is not undertaking any other tasks.
Table #B-1: Application parameters set at the node level for the host reserve application.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID</td>
<td>App_ID</td>
<td>Unique ID for application model</td>
<td>1</td>
</tr>
<tr>
<td>Session Minimum</td>
<td>SessMin</td>
<td>Minimum Session Time</td>
<td>5 sec.</td>
</tr>
<tr>
<td>Session Maximum</td>
<td>SessMax</td>
<td>Maximum Session Time</td>
<td>60 sec.</td>
</tr>
<tr>
<td>Data Inter-Arrival Time</td>
<td>DataLat</td>
<td>Period of data packet transmission</td>
<td>1 sec.</td>
</tr>
<tr>
<td>Session Inter-Arrival Time</td>
<td>SessLat</td>
<td>Time distribution seed for subscribing to new sessions</td>
<td>10 sec.</td>
</tr>
<tr>
<td>Sender Node</td>
<td>Sender</td>
<td>Indicates application is sender</td>
<td>TRUE</td>
</tr>
<tr>
<td>Receiver Node</td>
<td>Receiver</td>
<td>Indicates application is receiver</td>
<td>TRUE</td>
</tr>
<tr>
<td>Multicast Node</td>
<td>Mcast</td>
<td>Indicates application is multicast</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

- **Arr State:** This state is entered when there is a packet arrival from a lower layer (i.e. the host’s RSVP daemon -- see Section B.3). This state is in charge of determining what type of packet has arrived, and directing the action accordingly. Table #B-2 lists the upcall types that can be received.

Table #B-2: Action taken when a packet arrives from the host’s RSVP daemon.

<table>
<thead>
<tr>
<th>Upcall</th>
<th>Action Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>This packet contains the session ID assigned by the host’s RSVP daemon.</td>
</tr>
<tr>
<td></td>
<td>Records the new session ID and triggers a <strong>Sender Call</strong>.</td>
</tr>
<tr>
<td>Path</td>
<td>A <strong>Path</strong> message has been received. Records the appropriate information</td>
</tr>
<tr>
<td></td>
<td>(source address and port, bandwidth, delay) in the sender table and</td>
</tr>
<tr>
<td></td>
<td>triggers a <strong>Reserve Call</strong> and a <strong>Sender Call</strong> (in the unicast-interactive case).</td>
</tr>
<tr>
<td>Resv</td>
<td>A <strong>Resv</strong> message has been received. Reads in the appropriate information</td>
</tr>
<tr>
<td></td>
<td>(style, bandwidth, delay), determines the proper data transmission</td>
</tr>
<tr>
<td></td>
<td>parameters, and triggers a <strong>DataCall</strong> to send data messages.</td>
</tr>
<tr>
<td>PathTear</td>
<td>A <strong>PathTear</strong> message has been received. Reads in the sender’s IP address</td>
</tr>
<tr>
<td></td>
<td>and port, and deletes the sender from the <strong>Sender State</strong> list.</td>
</tr>
<tr>
<td>ResvTear</td>
<td>A <strong>ResvTear</strong> message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>PathErr</td>
<td>A <strong>PathErr</strong> message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>ResvErr</td>
<td>A <strong>ResvErr</strong> message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>ResvConf</td>
<td>A <strong>ResvConf</strong> message has been received. This indicates to the application</td>
</tr>
<tr>
<td></td>
<td>that a reservation request was confirmed. (Currently unimplemented.)</td>
</tr>
<tr>
<td>Data</td>
<td>A data packet for the session was received. Reads in the sender’s IP</td>
</tr>
<tr>
<td></td>
<td>address and port, and records the data as received in the <strong>Sender State</strong></td>
</tr>
<tr>
<td></td>
<td>list.</td>
</tr>
</tbody>
</table>
• **Join State:** Transition to this state occurs by a scheduled interrupt when it is time for the application to join a new session. This interrupt is scheduled either at the beginning of the simulation or when the last session is terminated. This state selects a group ID from a distribution (in the multicast case) or selects a unicast destination address (in the unicast case) and creates a *Session Call* to request a session ID from the host’s RSVP daemon. If the destination is multicast, this state also sends a join request to the host’s IGMP daemon (see *Section B.4*). In addition, the *Sender Directory* is initialized, and an interrupt is scheduled to terminate the current session.

• **Send-Tx State:** This state is entered either upon the receipt of a Session ID from the local RSVP daemon or by a scheduled refresh interrupt. If the application has a session ID and is a sender application, it creates a *Sender Call* and sends it to the local RSVP daemon, where the information is used to create a *Path* message. This state also schedules interrupts for sending refresh *Sender Call* messages.

• **Reserve-Tx State:** This state is entered either upon the receipt of a *Path* message from the local RSVP daemon or by a scheduled refresh interrupt. If the application has a session ID and is a receiver application, it creates a *Reserve Call* and sends it to the local RSVP daemon, where the information is used to create a *Resv* message. This state also schedules interrupts for sending refresh *Reserve Call* messages.
Data-Tx: Transition into this state occurs either upon the receipt of a *Resv Upcall* from the local RSVP daemon or by an interrupt to send data. If the application is subscribed to a session and is designated as a sender, it will create data and encapsulate it into a packet to be sent to the local RSVP daemon. An interrupt to send another data packet is also scheduled.

Terminate: This state is entered either upon the receipt of a *PathTear* message (in the unicast-interactive case) or by an interrupt indicating it is time for the application to leave the current session. This state sends a *Release Call* to the RSVP daemon, terminates the group membership (in the multicast-interactive case) by sending a terminate request to the local IGMP daemon, and schedules an interrupt for joining a new session.

B.2 HOST UNICAST NON-RESERVE APPLICATION (HOST_UAPP)

The purpose of this process model is to provide background traffic on the network, so that a more realistic performance of RSVP applications can be measured. The state transition diagram for this process model is shown in *Figure #B-2*. The traffic generated by this process model is unicast UDP. Because a high degree of sophistication is not necessary for this purpose, the process model interacts directly with the IP layer (see *Section B.5*), without the use of an intermediate UDP transport layer.

The following states are used by this process model to perform its tasks:
Figure #B-2: State transition diagram for the host unicast non-reserve application (Host_uapp).

- **Init State:** This state, entered at the beginning of the simulation, is in charge of initializing variables, determining the local IP address, loading the distributions for selecting unicast IP addresses used for addressing packets, and scheduling an interrupt to send data. This interrupt is scheduled for at least a minimum time (specified on the node level) after the beginning of the simulation, to allow all of the process models to load. This state also loads in parameters specified at the node level, as listed in Table #B-3.

### Table #B-3: Application parameters set at the node level for the host unicast non-interactive application.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Data Time</td>
<td>InitTime</td>
<td>Minimum time to wait before sending initial packet</td>
<td>5 sec.</td>
</tr>
<tr>
<td>Data Inter-Arrival Time</td>
<td>DataIat</td>
<td>Time between data packets being sent</td>
<td>1 sec.</td>
</tr>
</tbody>
</table>
**Idle State:** This state is the default state when the process is not undertaking any other tasks.

**Pk-Arr State:** This state is entered when there is a packet arrival from the IP layer, meaning that a UDP unicast packet destined for this node has been received. The receipt of a data packet is recorded in a local statistic, and the data packet is destroyed.

**Data State:** This state is entered by a scheduled interrupt, and indicates that a unicast UDP packet should be sent. A destination IP address is selected randomly and mock data is created and encapsulated into an IP packet, which is then sent to the IP layer. A local statistic records the transmission of a data packet, and an interrupt is scheduled to send the next data packet.

### B.3 HOST RSVP DAEMON (HOST_RSVPD)

This process model is the host's RSVP daemon. The purpose of this process is to route appropriate control and data messages back and forth between the application and the IP network layer. The state transition diagram for this process model is shown in Figure #B-3. Each of the RSVP control messages, and the objects that they contain, are coded as nested packets (see Appendix C), in order to easily facilitate access to specific information contained within the control messages.

The following states are contained within this process model:

**Init State:** This state, loaded at the beginning of the simulation, initializes all variables and lists, as well as obtaining the local IP address of the node.
Figure #B-3: State transition diagram for the host's RSVP daemon (Host_rsvpd).

- **Idle State:** This is the default state when the process is not undertaking any other tasks.

- **Arr State:** This state is entered when a packet arrives, either from a reserve application or from the IP layer. This state determines the type of message being received and routes it to the appropriate state for processing. *Tables #B-4 and #B-5* below shows the types of messages that can be received.

### Table #B-4: Action taken when a packet arrives from a host reserve application.

<table>
<thead>
<tr>
<th>Call</th>
<th>Action Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>This packet contains a request for a new session ID. The request is routed to the <strong>Session State</strong>.</td>
</tr>
<tr>
<td>Sender</td>
<td>The application is making a request for a sender call. The request is routed to the <strong>Sender State</strong>, to send an appropriate <em>Path</em> message.</td>
</tr>
<tr>
<td>Reserve</td>
<td>The application is making a reserve request call. The request is routed to the <strong>Reserve State</strong>, to send an appropriate <em>Resv</em> message.</td>
</tr>
<tr>
<td>Release</td>
<td>The application wishes to terminate its current session. The request is routed to the <strong>Release State</strong>, to send an appropriate <em>PathTear</em> message.</td>
</tr>
<tr>
<td>Data</td>
<td>The application wishes to send data. The request is routed to the <strong>Data State</strong>, to encapsulate and send data to the IP layer.</td>
</tr>
</tbody>
</table>
• **Session State:** This state reads in the appropriate information from the
  *Session Call* (destination IP address and port), and calls a function to assign
  the next available session ID. It then passes this session ID back to the
  application in a *Session Upcall*.

• **Sender State:** This state reads in the appropriate information from the sender
  call, updates the *RsvpSender Directory* to keep track that this node is a sender,
  and creates a *Path* message which is sent to the IP layer.

• **Reserve State:** This state reads in the appropriate information from the
  *Reserve Call*, creates the flow descriptor list, and then creates a *Resv*
  message which is sent to the IP layer.

• **Release State:** This state reads in the appropriate information from the
  *Release Call*, and sends an appropriate *Tear* message. For sender
  applications, a *PathTear* message is created, and the fields are filled from the
  *RsvpSender Directory*. This message is then sent to the IP layer. (*ResvTear*
  messages are currently unimplemented.)

• **Data State:** This state reads in the session ID for the data packet, determines
  the appropriate destination address from the session state, and then
  encapsulates the data into an IP packet which is sent to the IP layer.

• **Path State:** The sender IP address, port, bandwidth, and delay requested are
  extracted from the incoming *Path* message and passed to the application in a
  *Path Upcall*. 
Table #B-5: Action taken when a packet arrives from the IP layer.

<table>
<thead>
<tr>
<th>Message</th>
<th>Action Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>A <em>Path</em> message has been received, and is processed in the <em>Path</em> state.</td>
</tr>
<tr>
<td>Resv</td>
<td>A <em>Resv</em> message has been received, and is processed in the <em>Resv</em> state.</td>
</tr>
<tr>
<td>PathTear</td>
<td>A <em>PathTear</em> message has been received, and is processed in the <em>PathTear</em> state.</td>
</tr>
<tr>
<td>ResvTear</td>
<td><em>ResvTear</em> message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>PathErr</td>
<td>A <em>PathErr</em> message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>ResvErr</td>
<td>A <em>ResvErr</em> message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>ResvConf</td>
<td>A <em>ResvConf</em> message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>Data</td>
<td>A <em>Data</em> message has been received, and is processed in the <em>IncData</em> state before being sent to the application layer.</td>
</tr>
</tbody>
</table>

- **Resv State**: The style, bandwidth, and delay characteristics corresponding to the receiver's entry in the filter spec of the incoming *Resv* message are passed to the application in a *Resv Upcall*.

- **ResvConf State**: This state examines an incoming *ResvConf* message and extracts the necessary information. (Currently unimplemented.)

- **PathTear State**: This state examines an incoming *PathTear* message and removes the corresponding sender from the *RsvpSender Directory*. The sender IP address and port are passed up to the application layer in a *PathTear Upcall*, so that the application knows it should no longer expect to receive data from that source.

- **ResvTear State**: This state examines an incoming *ResvTear* message and extracts the necessary information. (Currently unimplemented.)
• **Error State:** This state examines an incoming *PathErr* or *ResvErr* message, extracts the appropriate information, and takes the necessary action. (Currently unimplemented.)

• **IncData State:** This state passes the data, along with the sender’s IP address and port, to the appropriate application in a *Data Upcall*.

### B.4 HOST MULTICAST DAEMON (HOST_IGMPD)

This process model is in charge of maintaining current multicast application group membership lists, using the IGMP protocol. The state transition diagram for this process model is given in *Figure #B-4*.

![Figure #B-4: State transition diagram for the host IGMP daemon (Host_igmpd).](image)

The host’s IGMP daemon works by determining which multicast groups a host wishes to receive, and sending subscription messages for that group to the local gateway node. It also periodically responds to queries from nearby gateways to inform them which groups it currently wants to receive.
This process model contains the following states to perform its tasks:

- **Init State:** This state initializes the local application table and distributions for sending responses to router queries (see Section B.7), and obtains the local IP address. In addition, parameters specified at the node level are loaded, as listed in *Table #B-6.*

*Table #B-6: Application parameters set at the node level for the host IGMP daemon.*

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Groups</td>
<td>MaxGrps</td>
<td>Total number of multicast groups in the simulation</td>
<td>10</td>
</tr>
<tr>
<td>Number of Applications</td>
<td>MaxApps</td>
<td>Total number of multicast applications in the simulation</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Idle State:** This is the default state when the process is not undertaking any other tasks.

- **Splitter State:** This state determines whether the incoming packet is from the IP layer, meaning that it is a query or a response to a query, or from a multicast application, meaning that it is a request to establish or terminate a connection to a group. It will route the packet to the Data_Decap State or App_Inf State, accordingly.

- **Data-Decap State:** This state reads in the incoming IGMP packet from the IP layer. If the packet is a query from an IGMP router daemon (see Section B.7), a table is checked to see if this host is currently a member of any groups. If it is, interrupts are scheduled to send responses for each group the host is subscribed to. If the packet is not a query, it is a response to a query from another local node, and the scheduled response for that group is canceled (so
that a gateway or router sending a query only receives one response per group per local subnet).

- **App-Inf State:** This state receives requests to establish or terminate a connection to a particular multicast group from the applications on this node. If it is a request to join a multicast group, a table is updated to indicate that this application is now a member of this group. If the node is not currently subscribed to this group (from another local multicast application), it sends a **Join Request** to the local gateway. If an application sends a request to terminate membership for a session, the group membership for that application is erased from the table. No request to terminate a session is sent to other nodes -- when the gateway next posts a query for that group, the host node will ignore it, since it is no longer a member. If no other local nodes are on that group, the gateway will not receive any response, and will therefore stop sending packets for that multicast group to the subnet. If a group subscription is established or fully terminated (i.e. no applications are still subscribing to the group), a local statistic is written to the IP layer, indicating that it should pass up or destroy, respectively, any incoming packets for that group.

- **Response State:** This state is entered when a scheduled interrupt for sending a response comes due. If the node is still a member of that group, it will send a response to the gateway or router which initiated the query.
B.5 IP Network Layer (Rsvp_IP)

This process model acts as the IP layer in hosts, gateways, and independent routers. The state transition diagram for this process model is provided in Figure #B-5. The model is based upon the IP model included with OPNET 2.5A, though it has been modified in order to accommodate multicast routing.

![State transition diagram for the IP Network layer (Rsvp_IP).](image)

This process model contains the following states to perform its tasks:

- **Init State**: This state initializes the local routing tables and parameters, obtains the local IP addresses for each interface, and loads in the parameters specified at the node level, as listed in Table #B-7.

- **Idle State**: This is the default state when the process is not undertaking any other tasks. If an interrupt from the IGMP layer is received, the process enters the Grp-Member State. (The interrupt from IGMP has low priority, and as such is only acknowledged once the process model is not otherwise engaged.)
Table #B-7: Application parameters set at the node level for the IP network layer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Rate</td>
<td>service_rate</td>
<td>The service rate used to process packets (large enough to handle several streams simultaneously)</td>
<td>10000 pkts/sec</td>
</tr>
</tbody>
</table>
| Router Type   | RouType   | Type of node connected to this IP layer                                     | 0 = Host
                                                          | 1 = Gateway
                                                          | 2 = Backbone                      |
| Net ID [0-4]  | netnum[0-4] | Network number for interface [i]                                            | Specified at network level         |
| Node ID [0-4] | nodenum[0-4] | Node number for interface [i]                                               | Specified at network level         |
| Nets          | MaxNets   | Number of networks in simulation                                            | 10                                 |
| Nodes         | MaxNodes  | Number of nodes in simulation                                               | 6                                  |

- **Grp-Member State:** This state is entered upon the receipt of a statistical interrupt from the IGMP daemon. Information from this interrupt is used to update the group membership table. If the group number received is not currently activated in the membership table, it is activated, and the node will begin to accept packets addressed to that multicast group. Similarly, a group number received that is currently activated will be removed from the membership table, meaning that the node should ignore packets addressed to that multicast group.

- **Arrival State:** This state is entered when there is a packet arrival. If the packet is unicast or is a multicast packet for a group to which the node subscribes, it attempts to place it into one of the three subqueues. The subqueues are ranked by priority: RSVP packets go into the top subqueue, IGMP packets go into the next subqueue, and all other packets (i.e. unicast) go into the bottom subqueue.
• **Svc-Start State:** In this state, the service time required for the packet (equal to the inverse of the service rate specified at the node level) is calculated, and an interrupt to finish service on the packet at that time is scheduled.

• **Svc-Comp State:** This state is entered by an interrupt scheduled in the Svc-Start State, and performs all of the necessary actions to route a packet to its next destination. First, it determines whether or not the packet should be passed to a higher layer module for processing. All RSVP and IGMP packets that have come into the node from another node, as well as unicast UDP packets that are addressed to this node, are passed up to the appropriate transport layer. If the packet is not passed to the higher layer (i.e. UDP packets addressed to another node or packets received from a higher layer), it routes the packet to the appropriate outgoing streams, based upon information contained in the unicast or multicast routing tables.

**B.6 Router RSVP Daemon (Rou_Rsvp)**

This process model is the RSVP daemon active on a router or gateway. This model sets up appropriate path state and reservation state on the router, and passes along appropriate information to the next nodes (or previous nodes) in the data path. The state transition diagram for this process model is given in *Figure #B-6.*
This process model contains the following states to perform its tasks:

- **Init State:** This state initializes all of the state block lists, obtains the local IP addresses, and determines the number of incoming and outgoing links.

- **Idle State:** This is the default state when the process is not undertaking any other tasks.

- **Arr State:** This state is entered when a packet arrives from the IP layer. This state determines the type of message being received and sends it to the appropriate state for processing. *Table #B-8* shows the types of messages that can be received.

- **Path State:** This state examines the existing Path State Block (PSB) and looks for a match with the incoming *Path* message. If a matching PSB entry is found, the previous hop address, logical interface handle, and sender
TSPECs are examined to determine if a Path refresh message is required. If no matching PSB entry exists, a new PSB entry is created, and an updated Path message is formed and sent to the IP layer.

Table #B-8: Action taken when a packet arrives from the IP layer.

<table>
<thead>
<tr>
<th>Message</th>
<th>Action Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>A Path message has been received. The message is routed to the Path State for processing.</td>
</tr>
<tr>
<td>Resv</td>
<td>A Resv message has been received. The message is routed to the Resv State for processing.</td>
</tr>
<tr>
<td>PathTear</td>
<td>A PathTear message has been received. The message is routed to the PathTear State for processing.</td>
</tr>
<tr>
<td>ResvTear</td>
<td>A ResvTear message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>PathErr</td>
<td>A PathErr message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>ResvErr</td>
<td>A ResvErr message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>ResvConf</td>
<td>A ResvConf message has been received. (Currently unimplemented.)</td>
</tr>
<tr>
<td>Data</td>
<td>A data packet for the session has been received. The message is routed to the Data State for processing and transmission to the next node(s).</td>
</tr>
</tbody>
</table>

- **Resv State:** This state first examines the existing PSB to ensure that all senders listed in the filter spec are registered with the node. Any senders that do not have corresponding PSB entries are removed from the filter spec. (This should also generate a ResvErr message, though Error messages are not currently implemented in this simulation model.) After the PSB is examined, a check with the TCB is made to ensure that the new reservation request will not violate the bandwidth and delay capacities of the sender node or the links on this node. Any request in violation will be removed from the filter spec (and an appropriate ResvErr message should be generated). At this point, the RSB is examined, to see if reservation state already exists for this unicast or multicast destination. If a matching RSB entry is found, any changes to the
next hop address, logical interface handle, and flow descriptor are made and a
Resv refresh message is generated for each relevant outgoing interface (i.e. the
interfaces along which the requested data traffic enters this node). If no
matching RSB entry exists, a new entry is created. These Resv refresh
messages are then sent to the IP layer.

- **PathTear State:** This state looks for a match between the sender listed in the
  PathTear message with information stored in the PSB. If a matching PSB
  entry is found, all RSB entries which list this sender in their flow descriptors
  are examined, and reservation state for the sender is removed from the RSB.
  If an RSB entry is empty as a result (i.e. has no more senders listed), the entry
  itself is also removed. At this point, traffic control is updated to reflect the
  end of the reservation, and the PSB entry is removed. Then, updated
  PathTear messages are passed to the next nodes in the path to the receiver(s).

- **ResvTear State:** This state is currently unimplemented, though it is included
  as a place holder for future development of the model.

- **ResvConf State:** This state is currently unimplemented, though it is included
  as a place holder for future development of the model.

- **PathErr State:** This state is currently unimplemented, though it is included
  as a place holder for future development of the model.

- **ResvErr State:** This state is currently unimplemented, though it is included
  as a place holder for future development of the model.
• **Data State:** This state ensures that a PSB entry for the data source exists and that it is conforming to its TSPEC. Also, a check is made to see that at least one RSB entry exists for this data flow. Once the PSB and RSB are checked, the data is reincapsulated and forwarded to the IP layer.

### B.7 Router Multicast Daemon (Router_igmpd)

This process model is in charge of maintaining the multicast routing tables on gateways and routers. The state transition diagram for this process model is given in *Figure #B-7*.

![State transition diagram for the router IGMP daemon (Router_igmpd).](image)

*Figure #B-7: State transition diagram for the router IGMP daemon (Router_igmpd).*

This module acts very similarly to the host IGMP daemon (see Section B.4), though it is also responsible for periodically transmitting multicast queries to neighboring nodes. Responses received from other routers and individual hosts will enable the gateway to maintain a multicast table indicating which groups should be forwarded along each of the router's outgoing interfaces. Since gateways and routers do not have an
application layer, the node itself does not subscribe to any groups, though it will receive and forward packets belonging to any group.

This process model contains the following states to perform its tasks:

- **Init State:** This state will obtain the local IP addresses, schedule an interrupt for sending a multicast routing query, and initialize a table indicating the queries sent out and responses received. Also, a parameter set at the node level is read in, as listed in Table #B-9.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Symbol</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Groups</td>
<td>MaxGrps</td>
<td>Total number of multicast groups in the simulation.</td>
<td>10</td>
</tr>
</tbody>
</table>

- **Idle State:** This is the default state when the process is not undertaking any other tasks.

- **Incoming State:** This state is entered upon the arrival of an IGMP packet from the IP layer. Three types of messages can be received: queries, responses to queries, and new Join Requests. However, since this daemon sends out general queries (see below), responses to queries and new Join Requests contain the same information, and are considered to be equivalent by the model. If the message is a query, the node will send individual unicast responses (i.e. Join Requests) back to the node which made the query for each group that is requested by at least one neighboring node. If a group is not requested by any neighbors, then no response is sent for that group. If the incoming message is a response to a query (i.e. a Join Request), the group is
added to the multicast table for the incoming interface, if it is not already listed. Also, the query response table is reset, to indicate that a response has been received for that particular group.

- **Send-Query State:** This state is entered upon a scheduled interrupt, when it is time for the node to send a multicast routing query. First, the node will determine if outgoing links are still subscribed to the same groups as when the last query was sent. This is done by determining whether or not a response was received to the last query for each group on each incoming link. If a response was not received, then the node or nodes connected to that link have terminated their subscriptions to the group, and the multicast routing table is updated so that multicast messages are no longer routed along those links. Once this bookkeeping is settled, a query is sent to all of the surrounding nodes to ask for responses for all groups to which neighboring nodes are subscribed.

**B.8 Router Transmission Queue (Rsvp_pc_fifo)**

In order to transmit messages properly along point-to-point links within the network between routers, some assurance must be made that only one message is using the link at any particular time. To accomplish this, each gateway and backbone router includes a queue in front of the packet transmitter which stores incoming IP packets and transmits them once the link is available. The state transition diagram for this queue process model is provided in *Figure #B-8*. This model is based upon the pc_fifo queue model included with OPNET 3.0, though it has been modified in order to make the queue
a "push" queue (i.e. it will send the message to the transmitter as soon as it detects a free link) instead of a "pull" queue (i.e. a queue which waits for an external request for the next packet before sending it).

This process model contains the following states to perform its tasks:

- **Init State**: This state is loaded at the beginning of the simulation, and is used to load local variables and set local parameters.

- **Idle State**: This is the default state when the process is not undertaking any other tasks.

- **Ins-Tail State**: This state is entered upon the arrival of a packet from the IP layer. The packet is inserted at the tail (i.e. bottom) of the queue.

- **Send-Head State**: This state is entered either from the Ins-Tail State, if the transmitter is free when a packet comes into the queue, or from the Idle state,
upon receipt of an interrupt from the transmitter indicating that the link is now available. The packet at the head (i.e. top) of the queue is sent to the transmitter.
APPENDIX C: PACKET FORMATS

This appendix lists all of the packet formats created for the simulation analysis. This includes a description of each packet format along with a description of the data contained within each field. Section C-1 contains packet formats used by the host process model. Section C-2 contains packet formats used by the RSVP daemon process models.

C.1: HOST PROCESS MODEL FORMATS

- **SessionCall:** Used to create a session call from host_rapp to host_rsvpd.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestNet</td>
<td>integer</td>
<td>Destination network address.</td>
</tr>
<tr>
<td>DestNode</td>
<td>integer</td>
<td>Destination node address.</td>
</tr>
<tr>
<td>DestPort</td>
<td>integer</td>
<td>Destination port address.</td>
</tr>
</tbody>
</table>

- **SenderCall:** Used to create a sender message from host_rapp to host_rsvpd.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionId</td>
<td>integer</td>
<td>Current Session ID.</td>
</tr>
<tr>
<td>BandWidth</td>
<td>double</td>
<td>Desired transmission bandwidth.</td>
</tr>
<tr>
<td>Delay</td>
<td>double</td>
<td>Desired transmission delay.</td>
</tr>
</tbody>
</table>

- **ReserveCall:** Used in reserve call messages from host_rapp to host_rsvpd.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionId</td>
<td>integer</td>
<td>Current Session ID.</td>
</tr>
<tr>
<td>Style</td>
<td>integer</td>
<td>Reservation style (FF, SE, WF).</td>
</tr>
<tr>
<td>ResvConf</td>
<td>integer</td>
<td>ResvConf desired flag.</td>
</tr>
<tr>
<td>NoFDpk</td>
<td>integer</td>
<td>Total flow descriptor elements (&lt;= 5).</td>
</tr>
</tbody>
</table>

- **ReserveFD:** Sender information contained in a ReserveCall packet.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenderNet</td>
<td>integer</td>
<td>Sender network address.</td>
</tr>
<tr>
<td>SenderNode</td>
<td>integer</td>
<td>Sender node address.</td>
</tr>
<tr>
<td>SenderPort</td>
<td>integer</td>
<td>Sender port address.</td>
</tr>
<tr>
<td>BandWidth</td>
<td>double</td>
<td>Desired bandwidth.</td>
</tr>
<tr>
<td>Delay</td>
<td>double</td>
<td>Desired delay.</td>
</tr>
</tbody>
</table>
- DataCall: Used to contain data sent from host_rapp to host_rsvpd.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionId</td>
<td>integer</td>
<td>Current Session ID.</td>
</tr>
<tr>
<td>Data</td>
<td>Structure</td>
<td>Mock data carried by packet.</td>
</tr>
</tbody>
</table>

- ReleaseCall: Used to send a release call from host_rapp to host_rsvpd.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionId</td>
<td>integer</td>
<td>Current Session ID.</td>
</tr>
</tbody>
</table>

- RSVP-APP-CNTRL: Encapsulates calls into packets to be sent from host_rapp to host_rsvpd.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppCall</td>
<td>Packet</td>
<td>Control Packet (one of the packet types listed above).</td>
</tr>
<tr>
<td>AppCallType</td>
<td>integer</td>
<td>Type of control packet contained.</td>
</tr>
</tbody>
</table>

- MC-APP-CNTRL: Packet sent to local IGMP stream to request a join or terminate for a particular multicast group.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRP_ID</td>
<td>integer</td>
<td>Group ID being joined / terminated.</td>
</tr>
<tr>
<td>Join</td>
<td>integer</td>
<td>Join flag (opposite of terminate flag).</td>
</tr>
<tr>
<td>Terminate</td>
<td>integer</td>
<td>Terminate flag (opposite of join flag).</td>
</tr>
</tbody>
</table>

C.2: RSVP DAEMON PROCESS MODEL FORMATS

- SessionUpcall: Packet sent to local application to indicate the new assigned session ID.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpcallType</td>
<td>integer</td>
<td>Upcall type (set to SESSION_EVENT).</td>
</tr>
<tr>
<td>SessionId</td>
<td>integer</td>
<td>New assigned session ID.</td>
</tr>
</tbody>
</table>
- **PathUpcall**: Packet sent to local application containing data from *Path* message.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpcallType</td>
<td>integer</td>
<td>Upcall Type (set to PATH_EVENT).</td>
</tr>
<tr>
<td>SenderNet</td>
<td>integer</td>
<td>Network address of sender.</td>
</tr>
<tr>
<td>SenderNode</td>
<td>integer</td>
<td>Node address of sender.</td>
</tr>
<tr>
<td>SenderPort</td>
<td>integer</td>
<td>Application port of sender.</td>
</tr>
<tr>
<td>BandWidth</td>
<td>double</td>
<td>Available bandwidth.</td>
</tr>
<tr>
<td>Delay</td>
<td>double</td>
<td>Available delay.</td>
</tr>
</tbody>
</table>

- **ResvUpcall**: Packet sent to local application containing data from *Resv* message.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpcallType</td>
<td>integer</td>
<td>Upcall Type (set to RESV_EVENT).</td>
</tr>
<tr>
<td>Style</td>
<td>integer</td>
<td>Reservation Style.</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>integer</td>
<td>Desired bandwidth for this sender.</td>
</tr>
<tr>
<td>Delay</td>
<td>integer</td>
<td>Desired delay for this sender.</td>
</tr>
</tbody>
</table>

- **DataUpcall**: Packet sent to local application containing data packet.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpcallType</td>
<td>integer</td>
<td>Upcall type (DATA_EVENT).</td>
</tr>
<tr>
<td>Data</td>
<td>Structure</td>
<td>Data Structure containing mock data.</td>
</tr>
<tr>
<td>SenderNet</td>
<td>integer</td>
<td>Source Network Address.</td>
</tr>
<tr>
<td>SenderNode</td>
<td>integer</td>
<td>Source Node Address.</td>
</tr>
<tr>
<td>SenderPort</td>
<td>integer</td>
<td>Source Port Address.</td>
</tr>
</tbody>
</table>

- **PathTearUpcall**: Packet sent to local application, containing *PathTear* data.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpcallType</td>
<td>integer</td>
<td>Upcall type (PATH_TEAR_EVENT).</td>
</tr>
<tr>
<td>SenderNet</td>
<td>integer</td>
<td>Source Network Address.</td>
</tr>
<tr>
<td>SenderNode</td>
<td>integer</td>
<td>Source Node Address.</td>
</tr>
<tr>
<td>SenderPort</td>
<td>integer</td>
<td>Source Port Address.</td>
</tr>
</tbody>
</table>

- **CommonHdr**: Packet with common header information in RSVP messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>integer</td>
<td>Current version of RSVP.</td>
</tr>
<tr>
<td>Msg_Type</td>
<td>integer</td>
<td>Type of RSVP control message.</td>
</tr>
<tr>
<td>TTL</td>
<td>integer</td>
<td>Current TTL of RSVP message.</td>
</tr>
</tbody>
</table>
• **Session**: Packet with session information in RSVP messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DestNet</td>
<td>integer</td>
<td>Destination network address.</td>
</tr>
<tr>
<td>DestNode</td>
<td>integer</td>
<td>Destination node address.</td>
</tr>
<tr>
<td>DestPort</td>
<td>integer</td>
<td>Destination application port.</td>
</tr>
<tr>
<td>Protocol_ID</td>
<td>integer</td>
<td>Protocol ID (RSVP = 46).</td>
</tr>
</tbody>
</table>

• **HOP**: Packet with PHOP/NHOP and LIH information in RSVP messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOP_Net</td>
<td>integer</td>
<td>PHOP/NHOP network address.</td>
</tr>
<tr>
<td>HOP_Node</td>
<td>integer</td>
<td>PHOP/NHOP node address.</td>
</tr>
<tr>
<td>LIH_Net</td>
<td>integer</td>
<td>LIH network address.</td>
</tr>
<tr>
<td>LIH_Node</td>
<td>integer</td>
<td>LIH node address.</td>
</tr>
</tbody>
</table>

• **TimeVal**: Packet with refresh rate information in RSVP messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh</td>
<td>double</td>
<td>Refresh period (in seconds).</td>
</tr>
</tbody>
</table>

• **FlowSpec**: Packet with flowspec (sender TSPEC) information.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BandWidth</td>
<td>double</td>
<td>Desired / requested bandwidth.</td>
</tr>
<tr>
<td>Delay</td>
<td>double</td>
<td>Desired / requested delay.</td>
</tr>
</tbody>
</table>

• **FilterSpec**: Packet with filter spec (sender template) information.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net</td>
<td>integer</td>
<td>Sender network address.</td>
</tr>
<tr>
<td>Node</td>
<td>integer</td>
<td>Sender node address.</td>
</tr>
<tr>
<td>Port</td>
<td>integer</td>
<td>Sender application port.</td>
</tr>
</tbody>
</table>

• **FlowDescriptor**: Packet containing a paired flowspec and filter spec.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filterspec</td>
<td>Packet</td>
<td>Filter Spec Object.</td>
</tr>
<tr>
<td>Flowspec</td>
<td>Packet</td>
<td>Flowspec Object.</td>
</tr>
</tbody>
</table>

• **IPAddress**: Packet format which specifies an IP network and node address.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net</td>
<td>integer</td>
<td>IP network address.</td>
</tr>
<tr>
<td>Node</td>
<td>integer</td>
<td>IP node address.</td>
</tr>
</tbody>
</table>
- **Scope**: Packet with scope information in Resv messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr[0-4]</td>
<td>Packet</td>
<td>Scope Address (5 total).</td>
</tr>
</tbody>
</table>

- **Style**: Packet containing reservation style information in Resv messages.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResvStyle</td>
<td>integer</td>
<td>Reservation Style (FF_Style = 10).</td>
</tr>
<tr>
<td>FDLen</td>
<td>integer</td>
<td>Number of Flow Descriptor Objects.</td>
</tr>
</tbody>
</table>

- **Path**: Path or PathTear message contained within an RSVP message.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonHeader</td>
<td>Packet</td>
<td>Common Header Object.</td>
</tr>
<tr>
<td>Session</td>
<td>Packet</td>
<td>Session Object.</td>
</tr>
<tr>
<td>RsvpHop</td>
<td>Packet</td>
<td>PHOP Object.</td>
</tr>
<tr>
<td>TimeVal</td>
<td>Packet</td>
<td>Time Values Object.</td>
</tr>
<tr>
<td>SenderDescriptor</td>
<td>Packet</td>
<td>Sender Descriptor (FlowDescriptor) Object.</td>
</tr>
</tbody>
</table>

- **Resv**: Resv message contained within a RSVP message.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonHeader</td>
<td>Packet</td>
<td>Common Header Object.</td>
</tr>
<tr>
<td>Session</td>
<td>Packet</td>
<td>Session Object.</td>
</tr>
<tr>
<td>RsvpHop</td>
<td>Packet</td>
<td>PHOP Object.</td>
</tr>
<tr>
<td>TimeVal</td>
<td>Packet</td>
<td>Time Values Object.</td>
</tr>
<tr>
<td>Scope</td>
<td>Packet</td>
<td>Scope Object.</td>
</tr>
<tr>
<td>ResvConf</td>
<td>Packet</td>
<td>Resv Confirm Object.</td>
</tr>
<tr>
<td>Style</td>
<td>Packet</td>
<td>Style Object.</td>
</tr>
</tbody>
</table>

- **Data**: Data message contained within a RSVP message.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonHeader</td>
<td>Packet</td>
<td>Common Header Object.</td>
</tr>
<tr>
<td>Session</td>
<td>Packet</td>
<td>Session Object.</td>
</tr>
<tr>
<td>RsvpHop</td>
<td>Packet</td>
<td>PHOP Object.</td>
</tr>
<tr>
<td>TimeVal</td>
<td>Packet</td>
<td>Time Values Object.</td>
</tr>
<tr>
<td>SenderDescriptor</td>
<td>Packet</td>
<td>Sender Descriptor (Flow Descriptor) Object.</td>
</tr>
<tr>
<td>Content</td>
<td>Structure</td>
<td>Data Content.</td>
</tr>
</tbody>
</table>
• **rsvp-pkfmt**: This is the default format for a RSVP control message sent to the IP layer.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg</td>
<td>Packet</td>
<td>Control message (e.g. Path, Resv, etc.)</td>
</tr>
<tr>
<td>Msg_Type</td>
<td>integer</td>
<td>Type of control message.</td>
</tr>
</tbody>
</table>

• **ip-dgram**: This is the default format for routing a message to the IP layer.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Packet</td>
<td>Message packet being sent.</td>
</tr>
<tr>
<td>outstrm</td>
<td>integer</td>
<td>Out stream that packet should be routed along.</td>
</tr>
<tr>
<td>prot_id</td>
<td>integer</td>
<td>Protocol ID.</td>
</tr>
<tr>
<td>orig_len</td>
<td>integer</td>
<td>Length of the full IP packet.</td>
</tr>
<tr>
<td>frag_len</td>
<td>integer</td>
<td>Length of the fragment (same as orig_len since IP fragmentation not used here).</td>
</tr>
<tr>
<td>frag</td>
<td>integer</td>
<td>Flag to indicate fragmentation (set to zero since IP fragmentation not considered here).</td>
</tr>
<tr>
<td>dest_net</td>
<td>integer</td>
<td>Destination network address.</td>
</tr>
<tr>
<td>dest_node</td>
<td>integer</td>
<td>Destination node address.</td>
</tr>
<tr>
<td>src_net</td>
<td>integer</td>
<td>Source network address.</td>
</tr>
<tr>
<td>src_node</td>
<td>integer</td>
<td>Source node address.</td>
</tr>
<tr>
<td>ttl</td>
<td>integer</td>
<td>Time to Live for IP packet.</td>
</tr>
</tbody>
</table>
APPENDIX D: RFCs AND INTERNET-DRAFTS

Many of the sources cited in this paper are either RFCs (Request for Comments) or Internet-Drafts. Due to the nature of the research covered in this thesis, many of the network technologies discussed are still under development, and therefore little to no reference material exists in published form. Where appropriate, the phrase “work in progress” has been used to indicate the sources which represent ongoing research.

Internet-Drafts are draft documents which outline ideas to be commented upon by the Internet community. They are only considered valid for six months after the publication date, and may be updated, replaced, or made obsolete by other documents at any time. When possible, the drafts used as references in this thesis were the most recent available, and many are still considered to be valid as of the date of this thesis. However, some drafts have not been updated recently by their authors, and so officially outdated drafts had to be used instead. All references in this thesis made to Internet-Drafts have been clearly labeled as such, and are provided to give credit for the ideas and technologies to the appropriate parties. These materials are not intended to be an official proclamation of current or future network technology.

RFCs usually represent network technology that is more standardized and accepted than Internet-Drafts, though remain open to the Internet community for minor comments and discussion. Typically, RFCs are former Internet-Drafts that have been approved and accepted by IETF working groups. As a result, they do not expire like Internet-Drafts, though are still not quite on a par with official publications.

RFCs and currently valid Internet-Drafts are available at http://ds.internic.net/.

- 155 -
**APPENDIX E: LIST OF ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABR</td>
<td>Area Border Routers</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>DABRA</td>
<td>Deterministic Area Border Routing Advertisement</td>
</tr>
<tr>
<td>DQDB</td>
<td>Distributed Queue Dual Bus</td>
</tr>
<tr>
<td>DP</td>
<td>Design Parameter</td>
</tr>
<tr>
<td>ERA</td>
<td>Explicit Routing Advertisement</td>
</tr>
<tr>
<td>EROSPF</td>
<td>Explicit Routing Open Shortest Path First</td>
</tr>
<tr>
<td>FF</td>
<td>Fixed-Filter</td>
</tr>
<tr>
<td>FR</td>
<td>Functional Requirement</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IGMP</td>
<td>Internet Gateway Management Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IS</td>
<td>Integrated Services</td>
</tr>
<tr>
<td>ISDN</td>
<td>Integrated Services Digital Network</td>
</tr>
<tr>
<td>LSA</td>
<td>Link State Advertisements</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>MOSPF</td>
<td>Multicast Open Shortest Path First</td>
</tr>
<tr>
<td>MTU</td>
<td>Maximum Transmission Unit</td>
</tr>
<tr>
<td>NHOP</td>
<td>Next Hop</td>
</tr>
<tr>
<td>OPNET</td>
<td>Optimized Network Engineering Tools</td>
</tr>
<tr>
<td>OSPF</td>
<td>Open Shortest Path First</td>
</tr>
<tr>
<td>PHOP</td>
<td>Previous Hop</td>
</tr>
<tr>
<td>POTS</td>
<td>Plain Old Telephone System</td>
</tr>
<tr>
<td>PSB</td>
<td>Path State Block</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>QOSPF</td>
<td>Quality of Service Path First</td>
</tr>
<tr>
<td>RAM</td>
<td>Read Access Memory</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>RES-LSA</td>
<td>Resource Link State Advertisement</td>
</tr>
<tr>
<td>RRA</td>
<td>Resource Reservation Advertisement</td>
</tr>
<tr>
<td>RSB</td>
<td>Reservation State Block</td>
</tr>
<tr>
<td>RSPEC</td>
<td>Receiver Specification</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource Reservation Protocol</td>
</tr>
<tr>
<td>SE</td>
<td>Shared-Explicit</td>
</tr>
<tr>
<td>TCB</td>
<td>Traffic Control Block</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TSPEC</td>
<td>Transmission Specification</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>WF</td>
<td>Wildcard-Filter</td>
</tr>
</tbody>
</table>
This page intentionally left blank.
ENDNOTES FOR CHAPTER 1


[6] A sample listing of the companies which provide home-based health care products and services can be seen online at: "http://www.yahoo.com/Business_and_Economy/Companies/Health/Home_Healthcare/"


[8] Several companies provide this type of service (i.e. on-site medical practitioners), including Cambridge Home Health Care (http://www.cambridgehealth.com), Active Home Health Care, Inc. (http://members.aol.com/activehhc/), Heritage Health Care Services (http://www.heritage-hcs.com), Health Care Horizons (http://www.ccs-i.com/hch/hch.html), et al.

[9] Many companies manufacture such emergency response systems. An example of this is Pioneer Medical Systems, Inc. (http://www.pioneermed.com/).


ENDNOTES FOR CHAPTER 2


[13] Braden, Zhang, et al, 1997. Based on Figure #4, page 13, and Figure #6, page 14.


[16] See Appendix A for more information about flow merging.


[18] Braden, R. et al. Figure #1, Pg. 5.

[19] Multicast group addresses are Class D IP Addresses. This is explained in further detail in Section 4.2.


[24] Resv messages must be passed along the reverse of the path. Even though the total amount of bandwidth available in the reverse direction is not important for unidirectional data flow applications, Resv messages must still be able to flow in the reverse direction.

[25] This debate has been ongoing for several months in the IETF community. Some of the debate is available in the archives for the rsvp@isi.edu discussion list.

ENDNOTES FOR CHAPTER 3


[6] This is the basis for a proposed Ph.D. research topic by the author.


[8] For a more comprehensive explanation of the axiomatic design technique, please refer to [Suh, 1990] and [Suh, 1996].


[10] A typical Pentium II / 233 MHz computer will come with 32 MB or 64 MB of RAM as of the date of this thesis.


[16] See Section 4.3 for more information about this type of network connection.

[17] See Appendix A for more information about merging TSPECs.

[18] See Chapter 2 for more information about QoS control services and reservation styles available through RSVP.

[19] See Chapter 4 for more information about how data and control messages are transmitted.


[21] The procedure for establishing reservation state in two directions is explained in Chapter 4.
ENDNOTES FOR CHAPTER 4

[1] This model is based loosely upon an RSVP and QOSPF model developed by the Networking and Simulation Lab at George Mason University. The original model is available at http://bacon.gmu.edu/qosip/, though their documentation was incomplete and their project was mostly concerned with measuring the performance of QOSPF. As a result, their RSVP model was not sufficiently developed for this project, though it did provide a good starting point for this research. As a result, substantial portions of the RSVP model presented in this thesis have been rewritten from the original version developed at George Mason University.

[2] Currently, all host nodes in the network model are connected to ethernet subnets. Hosts connected to a cable modem network should be added in the future to conform with the telenursing system design presented in Chapter 3. More information about the cable modem MAC is available at http://www.cablemodem.com/opnet/.

[3] More information about the ethernet MAC is available in Opnet’s online documentation, Chapter Eth.

[4] The IP address used in the model is of the form (500.Group_ID). The network address of 500 indicates to the simulation model that the packet has a multicast destination to the group corresponding to the node address.

[5] Any application in the model can either be a pure sender, pure receiver, or both a sender and receiver. The default is currently for all applications to act as both senders and receivers.

[6] Currently, only the FF-style of reservations is supported by the model, so an explicit list of senders and the corresponding bandwidth and delay requested is required for every Resv message.

[7] For unicast-interactive, this is the network indicated by the unicast network address. For multicast-interactive, this is a network with one or more nodes subscribing to the multicast group, as indicated by IGMP.

[8] This network model is provided in OPNET 3.0.