RL-TR-94-151
Final Technical Report
August 1994

ADVANCED TECHNIQUES FOR ANALYSIS AND EVALUATION OF ROBUST PROTOCOLS

Clarkson University and Boston University

Robert A. Meyer and David A. Perreault

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

[illegible]

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

APPROVED: Charles Meyer

CHARLES MEYER
Project Engineer

FOR THE COMMANDER: [illegible]

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE August 1994	3. REPORT TYPE AND DATES COVERED Final Aug 92 - Nov 93

4. TITLE AND SUBTITLE
ADVANCED TECHNIQUES FOR ANALYSIS AND EVALUATION OF ROBUST PROTOCOLS

5. FUNDING NUMBERS
C - F30602-92-C-0074
PE - 62702F
PR - 4519
TA - 22
WU - PH

6. AUTHOR(S)
Robert A. Meyer*
David A. Perreault**

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Clarkson University* (Electrical & Computer Eng. Dept.)
Potsdam NY 13699
Boston University** (Microprocessor Research Laboratory)
Boston MA 02215

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3BC)
525 Brooks Road
Griffiss AFB NY 13441-4504

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-151

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Charles Meyer/C3BC/(315) 330-1880

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The objective of this effort was to study new algorithms for robust communication network protocols and to develop techniques for analysis and evaluation of these protocol algorithms. These techniques are necessary to provide an objective basis on which to judge the effectiveness of these new algorithms for future C3I Air Force networks. A primary contribution of this work was to assist in the transfer of technology from on-going contract research efforts to the Rome Laboratory Network Design Facility. This report describes the testbed environment used during this study, presents an example of a typical protocol algorithm study, and illustrates the analysis and evaluation methodology with this example protocol. Sample experimental results that were used to identify certain problems with this protocol are also included.

14. SUBJECT TERMS
Protocols, Simulation, Robust, Communication networks

15. NUMBER OF PAGES
28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Table of Contents

List of Figures

Advanced Techniques for Analysis and Evaluation of Robust Protocols

Abstract

The objective of this effort was to study new algorithms for robust communication network protocols and to develop techniques for analysis and evaluation of these protocol algorithms. These techniques are necessary to provide an objective basis on which to judge the effectiveness of these new algorithms for future C3I Air Force networks. A primary contribution of this work was to assist in the transfer of technology from on-going contract research efforts to the Rome Laboratory Network Design Facility.

This report describes the testbed environment used during this study, presents an example of a typical protocol algorithm studied, and illustrates the analysis and evaluation methodology with this example protocol. Sample experimental results which were used to identify certain problems with this protocol are also included.

1. Introduction

The C^3I networks of the future must be able to operate at high performance levels under very adverse conditions. Although significant research has been conducted over the past several years on packet switched networks, this previous work has not focused on the specific environment expected as these networks are extended into theater operations involving military conflicts. In order to address this need, Rome Laboratory has established a testbed within the Network Design Facility in which experimental research is being performed with the objective of testing and evaluating packet-switched network protocols in highly dynamic networks.

An objective of Rome Laboratory's research program is to develop the necessary advanced technology for support of high speed, theater level packet-switched networks. This objective is achieved through a combination of in-house research and contract research projects directed toward various specific aspects of this technology development effort. Each of these projects brings experts in the various subareas of networking technology together to study the problems in the light of current technology and to make advancements in those subareas which are found to be limiting factors in achieving the goal of high performance C^3I networks. One of these efforts was a project entitled EDMUNDS (Evaluation and Development of Multimedia Networks in Dynamic Stress) under contract to SRI International. This effort, which concluded in October, 1993, developed three new protocols f.. routing and flow control in packet switched networks subject to highly dynamic stress. Our work has focused on two major tasks. First, we have conducted an analysis and evaluation of these three algorithms. Second, working with the Rome Labs engineers we have developed a sound methodology for experimental testing of these and similar algorithms using the testbed tools developed by SRI. These tools were available within the Rome Labs Network Design Facility and within our own Distributed Systems Laboratory at Clarkson.

In this report we first describe the testbed environment in which this study was conducted. Although this testbed was designed and implemented by SRI, we provided guidance in its design by using and evaluating early versions of these testbed tools and making recommendations for improvements and enhancements. Having appropriate, easy to use tools is a key factor in developing an effective methodology for protocol test and evaluation. Second, we discuss the protocol algorithms and summarize our analysis of these. Third, based on one of the example protocols, we illustrate the evaluation process and show how a problem was identified using the testbed tools. Conclusions and suggestions for continued evaluation of these protocols complete the report.

2. Protocol Development and Simulation Testbed

The design of new protocols in packet switched networks typically involves at least three distinct software development steps: algorithm testing, protocol evaluation, and final implementation on operational hardware. This process typically suffers from the inefficiencies of porting between different software environments at each of these different phases of development, and from the likelihood of introducing new errors at each step. In this section we describe a new software testbed which supports an integrated environment for the development of protocols by providing smooth transitions from initial testing of core algorithms, to comprehensive protocol evaluation, and finally to implementation on a platform suitable for easy porting to a high-performance packet switch.

The architecture of this testbed has been developed in several stages. As experience with each stage was gained, we and the Rome Labs engineers who were using it provided feedback to SRI and recommended changes for its improvement. In this report we present a "snapshot" of the testbed based on its status during the majority of the time in which this work was done. We also give a brief description of the changes which have taken place since then.

The testbed consists of a set of software development, simulation, and data analysis tools in an integrated environment running on a Sun SPARC workstation. The testbed components include a Network Algorithm Programmer's Interface (NAPI), a Simple Network Algorithm Prototyping Simulator (SNAPS), an OPNET-based [OPNET (Optimized Network Engineering Tools) is a commercial network simulation package developed by MIL3, Inc.] packet level simulation platform, a set of automated data analysis tools, and a color graphical Network Visualization tool (NetViz). All of the data analysis and NetViz tools are compatible with and may be used with simulation data produced by both SNAPS and OPNET.

2.1 Protocol Development

The first step in the development of a new protocol is a highly interactive, iterative design process. This process involves specifying the basic algorithm for the protocol in some language and then testing it in a simplified simulation of the network environment. SRI created a simulation tool called SNAPS (Simple Network Algorithm Prototyping Simulator) for this purpose. SNAPS is a C language based simulation tool in which the interface between the user's specification of the protocol and the simulation package is a small set of C functions. This set consists of functions which may be called by the user's code (e.g. initialization, generate a packet, etc.) and functions which must be written to respond to calls from the simulator (e.g. packet arrival, packet received at destination, etc.). SNAPS provides a good high-level view of a protocol algorithm in operation, but is not detailed and uses a simplified network model. It assists the protocol designer in debugging the basic concept of a new algorithm and gives an approximate indication of the likely performance of an algorithm. It is not intended as a tool for making comparative judgments about the relative merits of two or more well developed protocols, nor is it intended as a tool for final verification of an algorithm's correctness.

In order to ease the transition from the SNAPS simulation to a more detailed simulation, SRI developed a common interface to both the SNAPS simulator and the OPNET simulator. This interface is called a Network Algorithm Programmer's Interface (NAPI). It consists of the following components:

(a) Environment - the simulation platform, mostly protocol independent,

(b) Core - the network protocol,

(c) Wrapper - the part of the environment which is protocol dependent,

(d) Libraries - common functions such as traffic generator and parameter file parser.

The interrelationships of these components of NAPI are illustrated in Figure 1 and Figure 2. Figure 1 shows the components used in conducting a SNAPS simulation. These include the protocol code - the core - which is represented by the innermost, light blue circle. This code interfaces with the simulation package via the wrapper functions, shown as the blue ring. The simulation code is represented by the outermost ring and is made up the SNAPS simulator and a set of library functions. The benefits derived from this architecture can be seen by examining Figure 2. This illustrates the components used in conducting an OPNET simulation. We note that the protocol code is identical and the wrapper interface functions are identical and the libraries are identical. Thus, in order to simulate a given protocol, only the simulator component need be changed. Similarly, if we were to consider NAPI for a

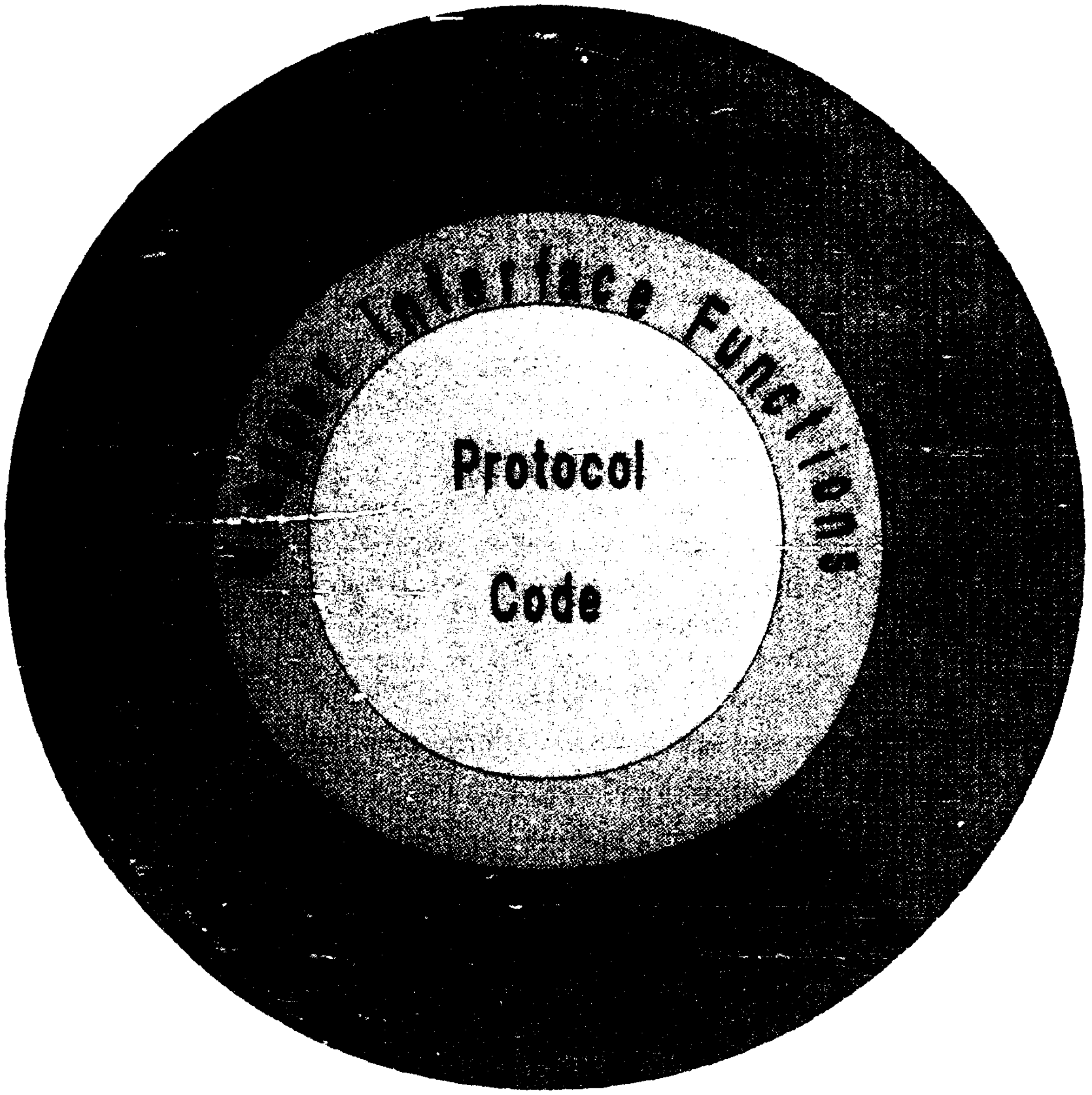

Figure 1 NAPI - SNAPS

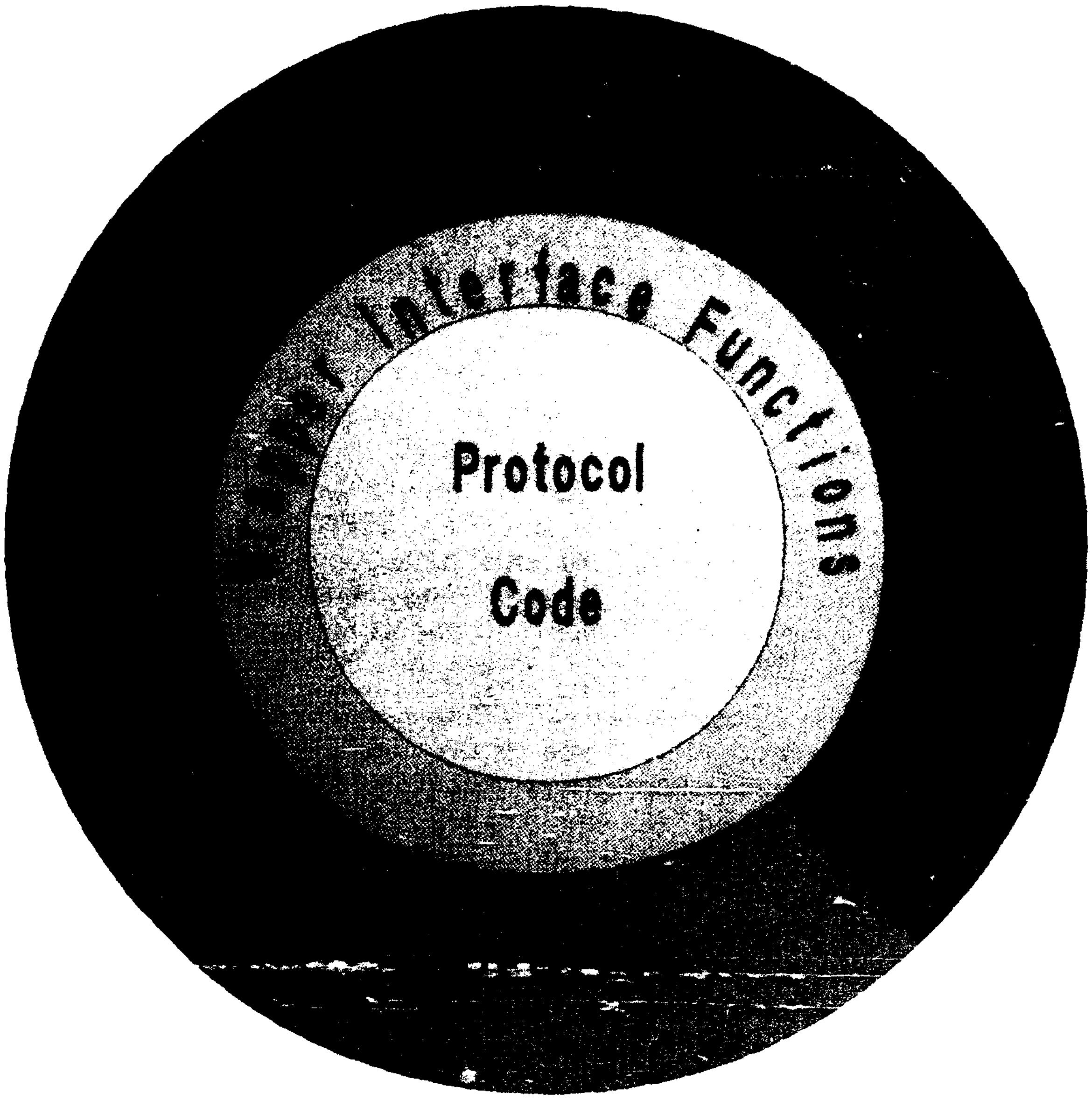

Figure 2 NAPI - OPNET

second (different) protocol algorithm, we would need to change only the core and parts of the wrapper functions. The libraries and environment portions remain the same for different algorithms.

Since the time during which our work was conducted, SRI has continued to improve this protocol development system. The current version is known as NAPI+, indicating that it is an object-oriented (C++) development environment. The major differences are that the protocol algorithm is specified in terms of object-oriented modules using C++ as the language, and the OPNET simulator has been replaced by an SRI developed C++ simulator. This simulator provides a comparable level of simulation as OPNET, but is more compatible with the object-oriented design philosophy of the protocol. The use of C++ for the protocol code enables this same protocol code to be ported directly to a multiprocessor emulation environment based on C++.

2.2 Protocol Simulation

As discussed in section 2, above, the work described in this report was done using the OPNET-based version of the simulator. OPNET (Optimized Network Engineering Tools) is a commercial software package developed by MIL3, Inc. OPNET is an open architecture network simulation system which provides a general purpose framework for a network designer to use in building specific simulation models of networks.

OPNET provides several editing environments, predefined functions that perform common networking tasks, and capabilities for network display, data collection and analysis. Models may be represented graphically, and are specified by code written in the C language. The protocol algorithm is modeled as an OPNET process. In addition, there are processes to model a traffic generator and a link manager. For a typical protocol algorithm which is distributed over all nodes of the network, these processes are then inserted into each node model. The nodes are then connected together to form a network model.

A simulation requires two input files. The script file specifies network connectivity, link capacities and link dynamics. The environment file specifies values for simulation parameters, eg. length of simulation, output file, etc., protocol parameters, eg. smoothing values, threshold levels, etc., and information specifying what data to log. As the simulation executes, the selected results are output to the log file. The log file is then used for post-simulation data analysis.

An integrated set of data analysis tools are available to be used on the output log file. These tools are written using the Unix PERL language (Practical Extraction and Report Language), a language useful for scanning text files to filter out the specific data of interest, and the gnuplot plotting programs. Data collected on the log file is filtered by a PERL script, prepared for the plot software, and then displayed and/or printed. This approach has several advantages over traditional simulation programs which only plot requested data as the simulation is being run. It allows the user first to view the results of a complete simulation run and then re-examine the same data at a specific point of interest, perhaps stepping through the log file in small increments of simulated time. Since the computational effort to perform the simulation is usually much greater than that to do data analysis, there is a significant benefit if a single simulation run may be used for several different analyses. The only costs incurred from requesting more data to be logged than is actually needed for one analysis is the increased size of the log file and some increase in the simulation time. When several different aspects of a simulation are to be analyzed, the log file approach has proven be a useful technique. Other advantages include the ease with which batch simulations may be run during non-peak time, thus permitting better utilization of computing resources.

In addition to the data analysis and plotting tools, there is an animated color graphics network visualization tool (NetViz). NetViz is written in C++ and provides a graphical user interface using the X-window/Motif environment. NetViz may be used in one of two modes. In the first mode, NetViz reads the output log file and effectively "replays" the simulation, allowing the user to display graphic representations of traffic routing, queue sizes, expected delays and other parameters of interest. In the second mode, NetViz may be run in real-time with the simulation, reading the log file as it is produced. Thus while not actually a part of the simulator, it gives the user an interface essentially equivalent to a real-time interactive simulation. The visualization of qualitative data which is not readily analyzed by the analysis and plotting tools is an important component of the testbed and greatly enhances the usefulness of the testbed in protocol analysis. Visualization of events in the network simulation is also important for demonstration purposes.

3. Protocol Description

In this section we describe a typical example protocol used in this research. This protocol was developed by SRI under the EDMUNDS program as described previously. It is the first of a family of protocols specifically designed for packet networks operating under the extreme conditions which are typically found in theater operations. This family is called the Secure Tactical Internet Protocol (STIP), and STIP1 is the first version to be analyzed and evaluated in the testbed. Development of STIP2 and STIP3 were in progress while we did most of the work for this effort.

STIP1 [1] is a multimedia network protocol in two senses. STIP1 is intended to support transport of multimedia or multiple information services, such as voice, video, or data. It is also intended to utilize multiple transmission media, or links having a diversity of characteristics such as bandwidth, delay, vulnerability to jamming, etc. STIP1 is specifically designed to be robust in highly dynamic tactical operating environments. These environments are characterized by link states (up or down) which may change rapidly in the presence of jamming, and by frequent topology changes due to node mobility and/or node destruction and reconstitution.

STIP1 is a distributed distance vector algorithm, which means that each node uses the information supplied by its neighbors as the basis for its actions. This is in contrast to link state algorithms which use global information about the entire network. Using neighbor data improves the response to localized problems and avoids difficulties which arise from the use of obsolete information in the event of long delays from distant parts of the network.

STIP1 uses thresholding to limit state updates to those instances in which significant changes have taken place. This reduces the overhead traffic and results in a smoother response to network changes by ensuring that a short term change in the network state does not produce oscillations in the response. It also averages delays over time so that delay information to more distant nodes is averaged more than that used for nearby nodes.

The basic cycle of the protocol is repeated as frequently as possible and is called an epoch. During each epoch the link and routing variables are measured and, if necessary, re-calculated. If thresholds have been met and a variable needs to be updated, an update packet is generated at this time and sent to a node's neighbors. Examples of variables computed during each epoch are link probabilities, link delays, effective link capacities, queue sizes, optimal flow rates for each link and destination, expected delays and queue thresholds.

Link probability is used to estimate the probability that a packet transmitted across a link will result in a corresponding ack being successfully received back. The estimate is based on the ratio of packets acknowledged to those transmitted; it is smoothed over short time periods. The raw delay across a link is calculated as the sum of three quantities: transmit time, propagation delay and the delay due to potential retransmissions. Effective link capacity is calculated as the product of link probability and actual capacity.

The link scheduling algorithm decides which destinations will be served by a specific link during an epoch. Each link has a scheduler that works independently of all others, and each node contains a queue for traffic to each destination. The pivotal parameter in the scheduling process is the queue threshold. The queue threshold, theta(l,d), is computed for each link, destination pair. For a given link, l, and destination, d, it represents the point in the queue for destination d, such that it is better for a packet to wait for a link other than l. When a link becomes available for the next packet transmission, first priority is given to any acknowledge packet (Ack packet) for a destination with an Ack packet above theta(l,d). After that, packets are selected on the basis of position in the destination queue for the next destination in the link schedule. The link schedule is computed on a frame basis which ensures that during the current frame, each destination gets a fair share of the information flow across this link. The link transmission algorithm gets the next packet eligible for sending when a link notifies the algorithm that the link is free. It also passes incoming packets to the packet processing function.

The next group of algorithms deal with the packet processing. Functions that fall under this heading include enqueueing, retransmissions and queue size. Enqueueing deals with placing packets in their queues according to their priority. There are five packet priority levels(from highest to lowest): acks for traffic, acks for updates, acks for probes, updates and traffic. These priorities are based on size. The acks for larger packets have highest priority to avoid retransmissions of the large packets. Acks as a class have higher priority for two reasons. The first is that in order to be able to handle fast network dynamics, acks must be received quickly to minimize their timeout window. The second reason is that acks are small, so their transmission will not cause the delay for updates or traffic to increase significantly. The retransmission function reinserts into the appropriate queue traffic or update packets for which no acks have been received during the timeout window. They are placed just ahead of any other packet of the same type already in the queue. This maintains the first come first served property.

At the network level, algorithms to calculate link flows, queue thresholds, expected delay and routing DAG (directed acyclic graph) are performed. These processes use expected delay information received from a node's neighbors in combination with local link properties such as delay and capacity to compute the flows for each link and destination as well as the expected delays form this node to each destination. The flow is the information rate (in bits/sec) at which packets destined for a destination are transmitted over a link. The optimal flows are computed so that the sum of expected delays to the next node for the last packet currently in the queues is minimized. As described above, the queue threshold is the position in a queue for a particular destination at which it is equally good for a packet to use a secondary link immediately as it is to wait for the best link to be available. This is computed once the flows have been computed.

The expected delays are taken into account when computing the flows. These values are time averaged progressively so as to average them more as they are propagated farther away from a node. The expected delay information received by a node from its neighbors is used in combination with the characteristics of local links, such as capacity, to compute expected delay from this node to each destination as well as the flows [bits/sec] for each link and destination.

A DAG is the set of links that are valid for routing to get to a specific destination. These links are chosen in a manner that avoids routing loops, making use of the reported expected delays. To maintain a stable DAG in a dynamic network, the changes in expected delays must meet certain thresholds before they can trigger a change in the DAG.

4. Protocol Analysis and Evaluation

Simulations of the STIP1 protocol were conducted in the Network Design Facility at Rome Labs by the Rome Labs engineer. Performance and vulnerability testing were conducted by exercising the algorithm under various test scenarios. More than fifty simulation runs were performed. As this is the first release of the software, interaction between the Rome Labs engineer and contractor engineer was frequent in order to debug operation and clarify documentation. As a result, efforts initially were in a trial and error mode. Initial experiments concentrated on testing all the analysis tools and simulation options to get a better understanding of how they all worked. A few anomalies were encountered that were easily corrected. A static simulation, with traffic but no jamming, is the usual starting point in a series of experiments. This creates what should be a steady state performance baseline to which subsequent runs can be compared. Most experiments have been done using the ten node network shown in Figure 3. The number of traffic streams, traffic rates, and jamming patterns are the main input properties changed among experiments. In this section we discuss how the protocol simulation is used to analyze the protocol and evaluate its performance.

Performance of a protocol algorithm must be measured in terms of several different factors. In the development of these new protocols, SRI defined a performance measure based on throughput, delay, reliability, and fairness [2]. The simulation testbed tools include a capability for directly computing this performance measure for each simulation run. It is not possible to use this statistic as an absolute measure of the effectiveness of a specific protocol. Rather it is useful for making comparisons with other protocols or with the performance of one protocol under a variety of different jamming scenarios. However, our experience at this point suggests that a *single number is not adequate in many cases.* In order to fully understand a protocol, one needs to look at more detailed measurements. The simulation testbed tools provide the necessary capabilities to do this.

For example, for each traffic stream (a set of packets with a specified source node and destination node) we have found it very useful to examine end-to-end delay. Each packet that is received at the destination experiences a delay from the point of first transmission by the source node. This delay is called the end-to-end delay. For a typical simulation run, Figure 4 shows a plot of end-to-end delay for packets in the stream from source node 8 to destination node 5. Each "dot" on this plot represents a single packet. It is clear from this data that when a set of links was jammed at time t = 20 to 26 sec., the end-to-end delay of most packets in this stream increased by factors of 3 to 5. We also note that after jamming, the delay quickly returned to a "normal" level.

The example in Figure 4 further illustrates why just looking at single performance numbers is not sufficient to understand protocol operation. Later in this scenario, at time t = 226 sec., a second set of links was jammed, and again we see a short term increase in delay. What is most interesting is that in this case the delay after jamming returns to a new level which is actually lower than the original. To understand why this happens one must examine the other traffic streams both before and after the jamming at t = 226 sec.

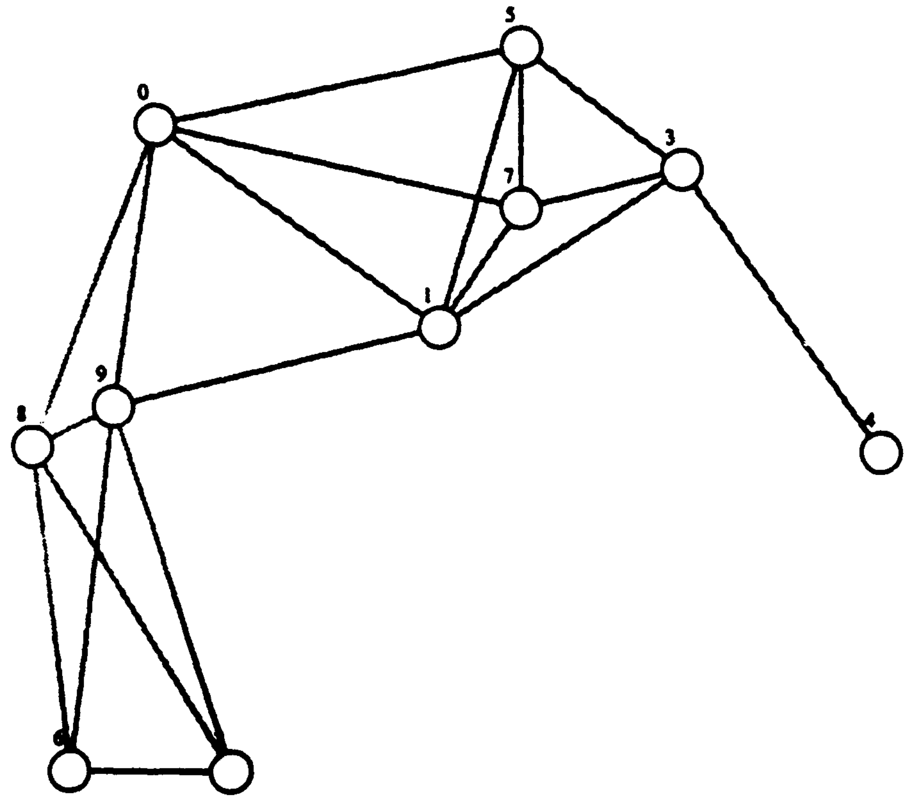

Figure 3 Ten Node Experimental Network

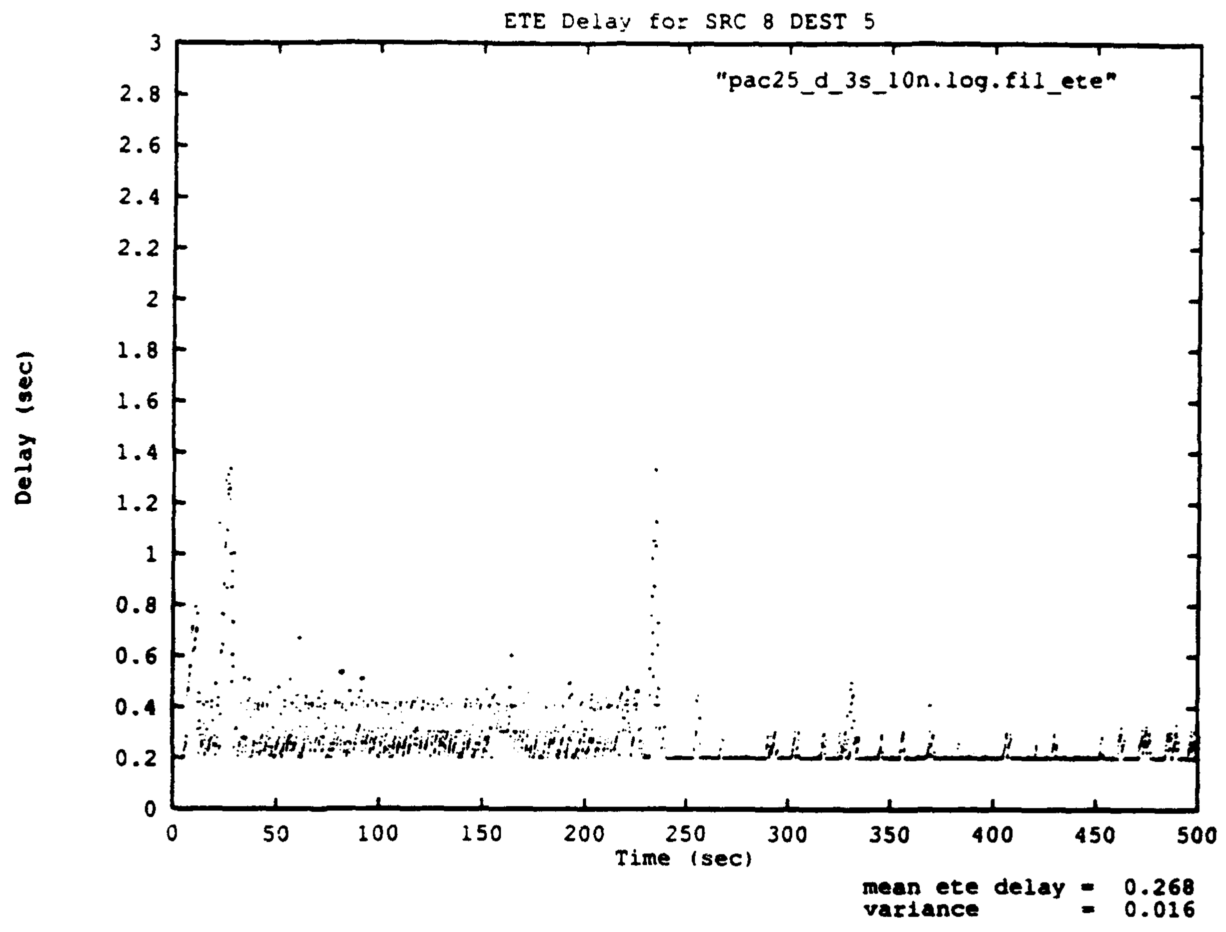

Figure 4 End-to-End Delay for Source 8 to Destination 5

The other traffic streams in this example were source 6 to destination 3, and source 0 to destination 4. End-to-end delay plots for these streams are shown in Figures 5 and 6. These indicate that the reduction in delay for stream 8-to-5 has taken place at the expense of increased delay for streams 6-to-3 and 0-to-4. By calculating the mean end-to-end delay over the "steady-state" time intervals between jamming events and after the second jamming event, it is possible to get a quantitative measure of this effect. We computed these delay times by simply using the same tool which produced Figures 4, 5, and 6, but excluding all times except the specific time interval of interest. This is a good example of the u[illegible]ulness of the log file approach. These delay measurements are shown below in T[illegible]I.

Stream	100 - 200 sec. (after first jamming event)	350-450 sec. (after second jamming event)
8-to-5	0.287	0.210
6-to-3	0.320	0.327
0-to-4	0.349	0.379

Mean End-to-End Delay (in seconds)

Table I

As is evident from the data, the mean delay for stream 8-to-5 was reduced by 0.077 sec. while the mean delay for stream 0-to-4 and 6-to-3 was increased a total of 0.037 sec. Thus it appears that the second jamming event results in the overall network being in a better state with respect to total mean delay. We have observed similar behavior in other examples as well.

Based on the theory behind the algorithm's development, this sort of anomaly should not occur. It seems that during the first time interval the algorithm is failing to find a true minimum delay solution. Using the NetViz tool to examine detailed routing information during each of these time intervals suggests that minor differences in routing decisions lead to these results. We investigated possible causes and have concluded that the only significant difference between the algorithm in theory and practice is the use of threshold limits which prevent continuous updates to each node's neighbors. We believe the thresholding may be the cause of this behavior.

In other experiments in which several jamming sessions took place, the first jamming had the greatest effect on end-to-end delay. We believe this also occurred [illegible]ue to the change in routing described earlier. Continually jamming only one node

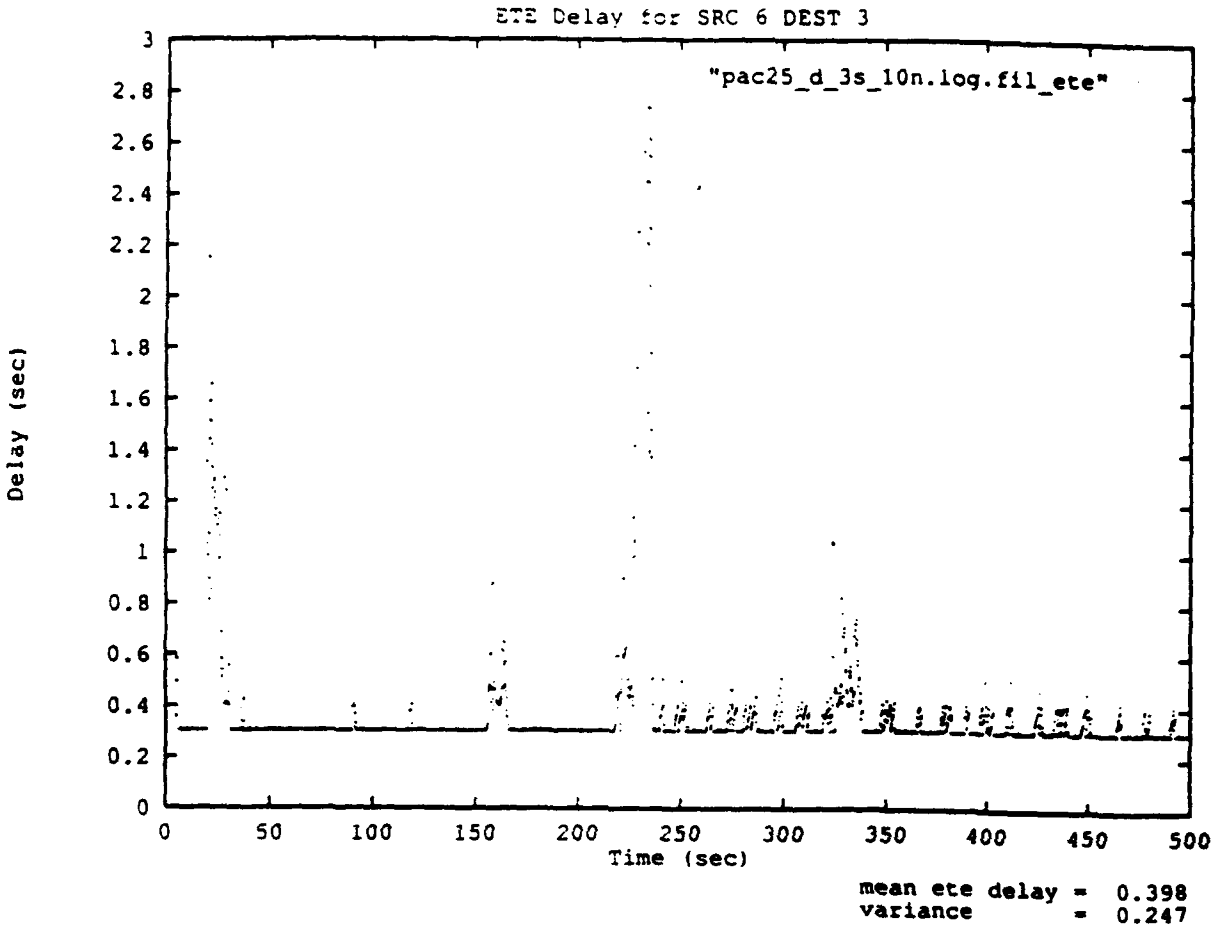

Figure 5 End-to-End Delay for Source 6 to Destination 3

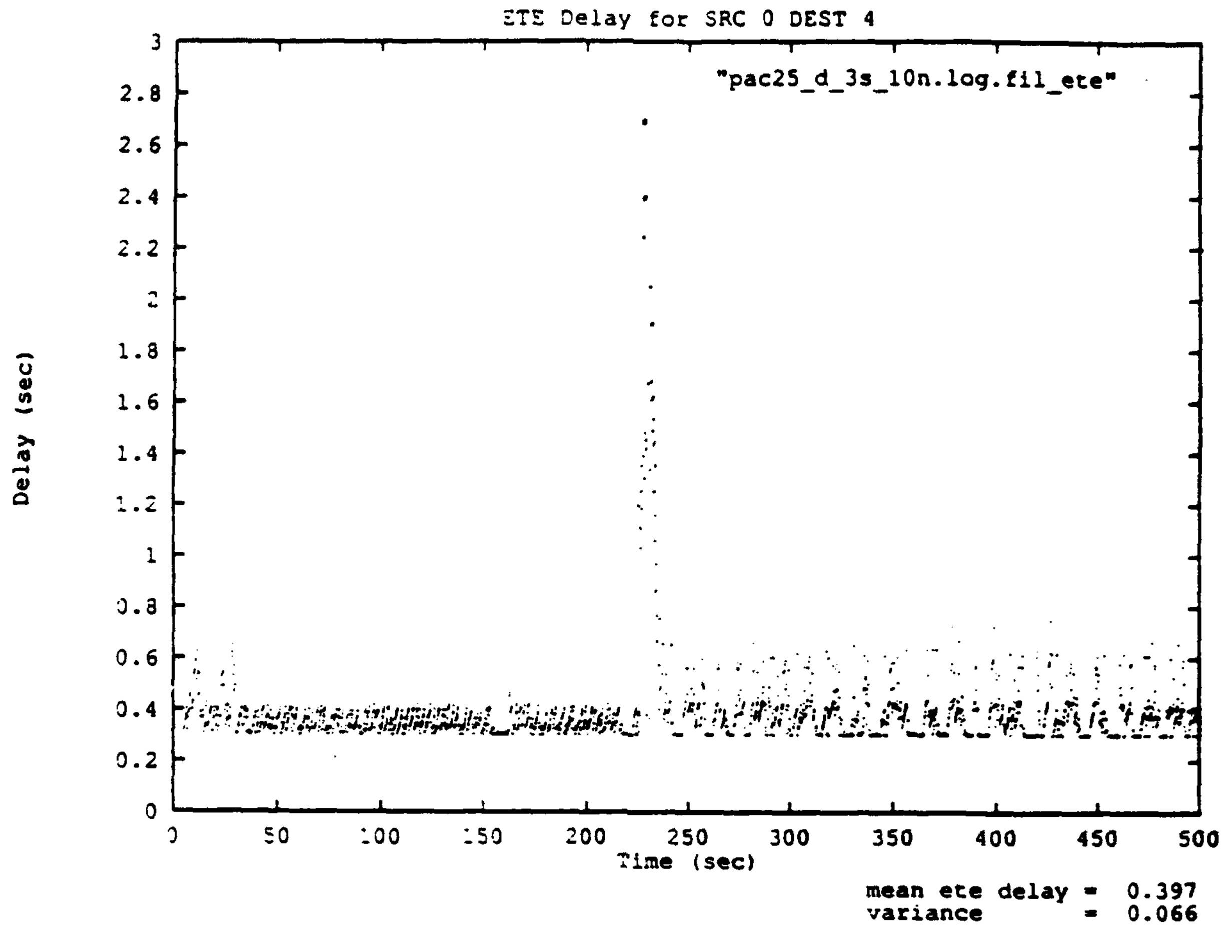

Figure 6 End-to-End Delay for Source 0 to Destination 4

did not continually raise delay for traffic initially traversing that link. Jamming for an extended period of time (10 sec) resulted in behavior very close to a long jam (5 sec) with delays either increasing very slightly or decreasing.

5. Conclusions and Future Work

A set of experiments with a new packet network protocol has been described. The preliminary evaluation of the protocol has shown that although in theory it should find a unique optimal set of routing paths with minimal end-to-end delays, there appear to be cases in which there are multiple "nearly equally good" solutions. Perturbations resulting from short term link outages may cause the algorithm to move from one solution to another. We believe this is a result of the thresholding which is used to reduce the overhead traffic of reporting every state change to all neighbors. This hypothesis should be investigated by additional experimentation. Also we note that the STIP1 protocol is only the first version in the STIP family, and more sophisticated protocol algorithms (STIP2 and STIP3) have since been delivered. Simulations should be run on these as well as on a baseline protocol to be used for comparison. By comparing the results of running the same experimental conditions across the three STIP protocols and the baseline, an accurate picture of each level of sophistication will be acquired.

Bibliography

[1] D. Beyer, J. Hight, R. Ogier, and D. Lee, "Secure Tactical Internet Protocol 1 (STIP1)", Technical Report ITAD-8558-TR-93-31, SRI International, Menlo Park, CA, Feb. 24, 1993.

[2] D. Lee, D. Beyer, and R. Ogier, "EDMUNDS Network Environment Profile: Benign and Adversarial Conditions, and Benchmark Scenarios," Technical Report ITAD-8558-TR-91-23, SRI International, Menlo Park, CA, March 1991.

[3] J. Hight, E. Costa, D. Lee, R Ogier, and J. Wong, "Evaluation and Development of Multimedia Networks in Dynamic Stress (EDMUNDS): Final Technical Report", Technical Report ITAD-85558-FR-93-277, SRI International, Menlo Park, CA, October 1993.

*U.S. GOVERNMENT PRINTING OFFICE: 1994-510-117 -50037