

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

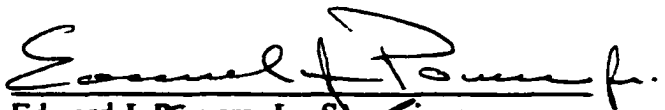
UMI

**A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600**


Copyright
by
Dwight David Daniel
1997

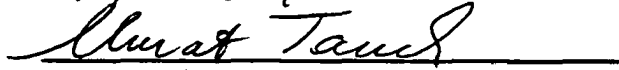
NeaSEL
Network and Software Engineering Laboratory


**Approved by
Dissertation Committee:**


Edward J. Powers, Jr. Supervisor


Stephen A. Szygenda, Supervisor


K. Suzanne Barber


Murat Tanik


Martin D. F. Wong

NeaSEL
Network and Software Engineering Laboratory

by

Dwight David Daniel, BES, MSE, MS

Dissertation

**Presented to the Faculty of the Graduate School of
the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of
Doctor of Philosophy**

The University of Texas at Austin

December 1997

UMI Number: 9824908

**Copyright 1997 by
Daniel, Dwight David**

All rights reserved.

**UMI Microform 9824908
Copyright 1998, by UMI Company. All rights reserved.**

**This microform edition is protected against unauthorized
copying under Title 17, United States Code.**

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

Dedicated to my dear wife Ati

Acknowledgments

First and foremost, I would like to express my gratitude to my principal supervisor, Dr. Stephen A. Szygenda, for his guidance and support throughout all phases of this and the other projects we pursued in the course of my studies. His keen insight combined with an appropriate prod at key times kept me on a productive track. I also wish to thank Dr. Barber, Dr. Powers, Dr. Tanik, and Dr. Wong for serving on my Ph.D. committee and for their useful suggestions at the time of my proposal.

A topic as complex as a methodology for an integrated system laboratory can only be successful if it based on extensive research into very technical realms. I was very fortunate to have Ed Gillen and his remarkable research team, particularly Jesus Campos, of the IBM Austin Technical Library to assist me. His team was able to ferret out even the most esoteric technical detail and obtain the original source wherever it was published.

Special thanks also goes to my dear friend, Bill Watson. His wry sense of humor helped me to maintain a rational approach to my educational pursuit. Additionally, he could always produce a snippet of code that would solve what I felt was the impossible and keep me moving in the right direction.

Finally, I wish to express my lasting gratitude to my friends, my family, and Fuzzy for tolerating this long undertaking.

NeaSEL
Network and Software Engineering Laboratory

Publication No. _____

Dwight David Daniel, Ph.D.
The University of Texas at Austin, 1997

Supervisors: Edward J. Powers, Jr. and Stephen A. Szygenda

This dissertation contributes a methodology, to the network and software engineering community, for designing an integrated systems laboratory. The systematic procedures presented permit any organization, university or industry, to create a multi-use, teaching and research, integrated network and software engineering laboratory.

The power of this methodology is twofold. First, it serves as a sound guideline for the creation of the facility and helps to eliminate the frustration of unworkable ideas. Second, the application of this methodology creates a single laboratory in which a wide range of teaching and research activities can be carried on, thus eliminating many specialized laboratories.

This dissertation further presents a case study of a large scale implementation of the procedures. The result is a laboratory, the Network and Software Engineering Laboratory (NeaSEL) at the University of Texas at Austin.

Table of Contents

List of Tables	xi
List of Figures	xii
Chapter 1: Introduction	1
1.1 Historical Perspective	2
1.2 Software Simulation vs. Experimental Laboratory.....	7
1.3 Why is a New Architecture Needed?.....	9
1.4 Absence of General Methodology and Architecture	13
1.5 Requirements.....	13
1.6 Summary.....	13
Chapter 2: Design Goals	16
2.1 Core Set of Problems.....	17
2.2 Design Principles	18
2.3 Principles of Learning.....	21
Chapter 3: Fundamental Design Principles	23
3.1 Typical System Unit	23
3.2 Host Naming Convention	26
3.3 Compute Islands.....	34
3.4 Numbering Convention within Islands.....	38
3.5 Capabilities.....	41
3.6 Connection or Isolation	42
3.7 Physical vs. Electrical Isolation Procedures.....	45

3.8	Summary	47
Chapter 4: Architecture		49
4.1	Network Topology	49
4.2	Topology of Choice: Switched.....	51
4.3	Design Constraints.....	53
Chapter 5: Case Study, NeaSEL		57
5.1	Overview.....	57
5.2	Compute Islands.....	58
5.3	Topology and Protocol.....	60
5.4	Wiring Infrastructure.....	60
5.5	Multiple NICs per Workstation.....	62
5.6	TCP/IP Addresses	62
5.7	Design Constraints.....	67
Chapter 6: Research Topics for NeaSEL		70
6.1	System Administration Class.....	72
6.2	Cluster Computing	75
6.3	Distributed Computing	78
6.4	Combination: Cluster & Distributed Computing.....	81
6.5	Client/Server Computing	84
6.6	Viruses	87
6.7	Network Security Issues.....	90
6.8	Network Performance.....	93
6.9	Network Parameters.....	95
6.10	Ethernet Switch vs. Ethernet Hub	98
6.11	Data Striping across Networks	100
6.12	NIC Performance with Downloaded Protocol	102

6.13	Protocol Analysis & Lightweight Protocols.....	104
6.14	Study of Dynamic Network Load Balancing	107
6.15	SPECmark Analysis.....	110
6.16	Compression Techniques	113
6.16	Summary	115
Chapter 7: Validation of the Methodology		116
7.1	Third Party	116
7.2	Capabilities	117
7.3	Automated Tests	119
7.4	Ease of Use	119
7.5	Research Facility	122
7.6	Teaching Facility	124
7.7	Typical Experiments	125
7.8	Conclusion	134
Chapter 8: Components of the Methodology		135
8.1	Component Boundaries	135
8.2	Applying Individual Components	136
Chapter 9: Future Research		138
Chapter 10: Conclusion		140
Appendix A: Validation Code Samples		141
A.1	FTP Test	141
A.2	Network Performance Test	149
A.3	Xfract	161

Appendix B: Case Study Implementation Details	163
B.1 Layout	163
B.2 Wiring	165
B.3 Electrical Power.....	169
B.4 Color Coding.....	170
B.5 Patch Cables	170
B.6 Workstations	172
B.7 Label Equipment.....	175
B.8 Patch Panel <u>inside</u> the Laboratory.....	177
B.9 Documentation	178
B.10 Use of Tag Outs.....	181
B.11 Configuration Sameness.....	182
B.12 Extensive Back-up Capability.....	183
B.13 Cordless Phone	184
B.14 Miscellaneous Equipment.....	184
Appendix C: Manuals	185
C.1 <i>NeaSEL User's Manual</i>	185
C.2 <i>NeaSEL Unique Installation Guide</i>	186
C.3 <i>Research Topics for NeaSEL</i>	187
C.4 <i>NeaSEL Research Presentations</i>	188
C.5 <i>Lessons Learned from NeaSEL Buildup</i>	188
Glossary	189
Bibliography	202
Vita	208

List of Tables

Table 1. Addressing Range vs. Manufacturer and NIC.....	64
Table 2. Address Range of Special Use Devices	66

List of Figures

Figure 1.	Industry Trends.....	15
Figure 2.	NIC Identification Convention.....	31
Figure 3.	NeaSEL Compute Islands.....	37
Figure 4.	Numbering within Islands.....	40
Figure 5.	Network Topologies.....	50
Figure 6.	General Layout of NeaSEL.....	59
Figure 7.	Raceway Attachments.....	61
Figure 8.	Network Connectivity by Machine.....	63
Figure 9.	Composition of NeaSEL.....	74
Figure 10.	Cluster Computing.....	77
Figure 11.	Distributed Computing.....	80
Figure 12.	Combination Cluster/Distributed Computing.....	83
Figure 13.	Client/Server Computing.....	86
Figure 14.	Virus Infection.....	89
Figure 15.	Typical Firewall.....	92
Figure 16.	Network Performance.....	94
Figure 17.	Network Parameters with a Sniffer.....	97
Figure 18.	Switch versus Hub.....	99
Figure 19.	Network Striping.....	101
Figure 20.	Downloading Protocols to Adapters.....	103
Figure 21.	Locations of Checksums.....	106
Figure 22.	Dynamic Load Balancing.....	109
Figure 23.	Sample of SPECfp95 Results.....	112
Figure 24.	Compression Algorithm Testing.....	114
Figure 25.	ftp Transfers over 10 Mbit Ethernet.....	127
Figure 26.	ftp Transfers, Ethernet vs. ATM.....	128
Figure 27.	ftp Transfers over Sockets.....	129
Figure 28.	Validation of Network Striping.....	133
Figure 29.	Raceway Attachments.....	166
Figure 30.	System Attachment to Wall Jacks.....	166
Figure 31.	Typical System Placard.....	176
Figure 32.	Typical Informational Table.....	179

Chapter 1: Introduction

The recent, explosive growth of the Internet and related information and network access services has demonstrated alarming deficiencies of applicable knowledge (*i.e.*, experience versus theory) in network performance, network security, distributed computing, large-scale client/server implementations (*e.g.*, World Wide Web) and wide-area software engineering. The deficiency lies in the fact there is seldom cross-development or cross pollination between the two disciplines of software engineering and network engineering. While many discreet examples of this statement can be demonstrated, one of the easiest to show is Object Oriented Programming (OOP) and high speed network development.

Companies are looking at OOP to reduce software costs and to remove schedule bottlenecks caused by the software development cycle. While there are many advantages to OOP, one major disadvantage is that objects have to be moved across networks. This can be a significant network load, particularly if OOP is fully implemented. References and texts on OOP gloss over this fact [1-3].¹ Unfortunately, the same can be said of most network engineering texts. Both, a major deficiency.

As a result of this and many other examples, our networked world is requiring a new paradigm, that of interdisciplinary inquiry by both network and software engineering at the theoretical level and at the research level. This

¹ Only the IBM reference [3] slightly explores network loading. This is done because the reference is an actual product where the customer must know the effects before purchasing it. Even then, a general analysis of the subject is not given.

dissertation addresses the latter by developing a methodology for designing an integrated systems laboratory. Stated another way, my dissertation is a systematic approach to designing and building a totally integrated network and software engineering laboratory that fully blends research with teaching.

While it's directed to a research and teaching laboratory, the approach also helps to address problems at the theoretical level by providing a single, general purpose methodology and facility that promotes and simplifies a myriad of topics which can be used to validate theoretical research. In fact, this is a strongpoint of the methodology: the capability to do numerous diverse studies in a single facility, therefore, eliminating the requirement for a series of single use facilities.

This dissertation is two documents in one. First, it is a journey of discovery into the development of the methodology. Second, it presents a case study of an actual physical implementation of the methodology on a large scale at the University of Texas at Austin.

No matter the direction, no matter the document, the first step begins with a historical perspective.

1.1 Historical Perspective

Before creating a methodology and architecture, one must first establish that a deficiency exists in some part of a fundamental framework. The best approach is to look at a historical perspective and answer the questions: what is missing, why are a new methodology and architecture needed, and why is my

approach the best solution? The following sections will address these questions and guide the reader into the initial framework.

1.1.1 Industry Trends

It is well recognized that the personal computer (PC) has revolutionized information technology. In less than 15 years, we have gone from centralized computing to ubiquitous, high-powered stand-alone systems. Today, we have the following trends:

1. Processor capability, MIPS, is doubling approximately every 15-18 months[4].
2. Costs of processors are dynamically dropping to the point of being considered inexpensive.

Initially, the purpose of computers was to compute, to process data. We are now in the second revolution: how does one get information to a stand-alone system so a user can process the information to the user's benefit? This is a significant difference. First, we are no longer processing raw data to get an answer—we are sending information for analysis. Note, this information may be very large, for example access to a library or topological map information. Second, we now have the compute power to do sophisticated analysis if we have the information.

Because we are processing information, the following trends are emerging:

3. Client/server relationships are increasing in importance. [37, 57]
4. Futurists are talking about a world that is completely networked. Regional Bell Operating Companies (RBOCs) and other utility providers are working to provide information access to everybody, access that is equivalent to our current telephone access.
5. The preceding points are motivating people to develop higher speed networks with larger bandwidths.

These trends are prompting a systems view of the world.

6. People are concerned with issues of security. For example, what is a network virus? How do I prevent virus propagation? What are firewalls? Other security and privacy issue are also causing concerns?
7. Because there are literally thousands of powerful systems readily available, the question is: "How can a superuser or single application tap this compute power during off-shift?", that is how can it utilize "Sleeping MIPS"? Would the owner want to giveaway or perhaps sell his compute power?

With all these computer systems available and relatively cheap, network concerns are starting to dominate computer engineering. This is rapidly leading to a new rule in the industry:

Rule: Software developers can no longer ignore network parameters and their dynamics when developing code and applications.

The converse is also true:

Rule: Network engineers must understand the applications that are sending data across the networks.

1.1.2 A University's Role

A university has a dual role. First, it is a repository of knowledge, which it uses to educate people in facts and how to think. Second, it is a research institution, which it furthers knowledge by pushing the frontiers. Finally, for a university to be at the forefront of knowledge, an additional requirement is that research findings must be pushed back into the education process.²

The dual roles of an institution combined with the previously noted industry trends have dictated that for an institution to be in the vanguard of the computer education advance it must combine software engineering and network engineering into an integrated curriculum at the systems level. They can no

²This sentence appears to be stating the obvious and be simplistic in nature. As will be demonstrated, this seemingly straightforward statement added unmeasurable difficulty to the birth of the methodology and the laboratory because of the requirement to integrate students with researchers.

longer be separate course structures. This understanding caused, Dr. Szygenda (my adviser), Eric White (a fellow researcher), and me to propose a family of topics under one research umbrella that would combine both software engineering and networking into an integrated structure. This series ran from dynamic network load balancing; to high fidelity, discrete, large entity digital simulations; to distributed logic simulation using a massive number of heterogeneous processors.

Seemingly independent, these topics had common threads. First, they were heavily dependent on network topologies, network performance, and other variables. Second, software and network issues had to be integrated to have a successful conclusion. Third, a real network of multiple topologies was required for the research. Fourth, engineering statements normally require proof, which implies measurements, further implying access to an experimental laboratory.

The conclusion was that access to a general purpose, integrated software engineering and network laboratory would be required for any of the topics. Obviously, the creation of such a facility would be a large undertaking. The initial goal was to find such a laboratory and then copy the architecture. A search of the University of Texas at Austin showed that there were no general purpose network labs on site. There were very specialized ones but none that met the criteria of integrating software engineering with network engineering. Since the University of Texas at Austin did not have such a laboratory, the clear direction was to duplicate a methodology and existing laboratory from another university or from industry. The search for such a laboratory led directly to the reason for this dissertation.

1.2 Software Simulation vs. Experimental Laboratory

From the preceding, it is obvious our research team was looking for a laboratory and not a software simulation procedure to validate the topics under our research umbrella. This was not an arbitrary decision but came after much deliberate thought. Before continuing with a historical search, let us digress into a discussion of software simulation versus experimental laboratory and explain why we chose to pursue an experimental laboratory approach over software simulation.

The debate of software simulation versus experimental laboratory has been with scientists and engineers since the advent of computers and software. Each has advantages and disadvantages. Software simulators seem to be more cost effective because they do not need a facility or expensive equipment. The "experiment" can be run at any time and the direct cost is only that of system time. A quick comparison shows that laboratories are the reverse.

While this contrast can be greatly expanded, even this quick analysis decisively showed that we could not use a simulator for validation of topics under our research umbrella. It is a falsehood that simulators don't need a facility or expensive equipment (a laboratory). That is only true once the simulator itself has been validated from results produced in a laboratory. If one is simulating known and previously validated topics, then one can have full faith in the answers from the simulator. The problem is if one is doing research into a wide range of unknown and yet to be defined topics, as our research umbrella

was planned to do, one cannot be assured the simulator is predicting correct results. The result is that one needs a laboratory to validate the simulator. Since one has to perform experiments anyway for validation, our research team felt that having access to a simulator would not give us any advantage.

While this in itself might have prevented us from pursuing a simulation solution, there is another disadvantage that must be addressed. A very flexible simulator, one that can address a wide range of topics, from micro issues (like packet size) to macro issues (cluster computing O/Ss) inherent with network analysis, comes with a major disadvantage, that of steep learning curve. The more powerful the simulator, the more complex the syntax. It takes time to learn the simulator.

In industry, steep learning curves are somewhat acceptable because it is assumed the employees will stay. For universities, there is a fixed and never ending turnover rate. Since there is no universally recognized network simulator, students and researchers could be spending considerable time learning something that will have no applicability upon leaving the university.

This statement should not be taken to imply that our research team was totally against learning simulation. Just the contrary, it was expected that students and researchers working under our umbrella have already had simulation experience. Our concern was the intense learning effort required for a tool that might not be widely applicable. Furthermore, working with real networks is directly applicable to future experience because networks, their analysis, and their techniques never radically change, they evolve. These reasons led us to a focused pursuit of a laboratory solution.

1.3 Why is a New Architecture Needed?

From contacts with local industry (IBM, Southwestern Bell, & GTE) and with other universities (University of Illinois, Texas A&M), we learned that a general purpose software engineering and network laboratory was not known. Known labs were homogeneous in equipment or did not integrate software engineering with networking at a systems level. Examples were the ATM laboratory at Texas A&M, the FDDI Interoperability Test Laboratory at the University of New Hampshire [5], and an ATM switch compliance laboratory at Southwestern Bell Telephone Research Institute in Austin, Texas. All of these were very functional labs but none met our initial design criteria of an integrated software engineering and network laboratory.

1.3.1 World-wide Search of Research Facilities

Many universities have designed their own microprocessor based workstations for teaching different hardware and software topics associated with microprocessors, but these hardware based laboratories are not being utilized to teach computer networks. Except for Reiss [6], articles and textbooks stress software simulation of networks for undergraduate laboratory teaching [7-11].

For example, Finkel and Chandra [11] created a software simulator that helps students to understand and do projects with the first two layers of the OSI model: physical and data link, with a hybrid layer that includes all the others.

This is an excellent tool because packet loss, error correction routines, and the like can be simulated. But, it has no hardware component, *i.e.*, there is no laboratory. Also, because it is at such a low level, it is impossible to do systems level research such as cluster algorithms with different network topologies.

Other authors at various universities have started a hardware approach to teaching networks but again always at the lowest level. For example, Deware & Seti [12] at the University of New South Wales, Kensington, Australia, used highly specialized interface boards to teach networking at the network interface card (NIC) level. This is an admirable approach for teaching NIC engineers but falls short of our general requirement for a systems approach.

Two universities, City University of New York [13] and the University of Cantabria, Spain [14], have integrated software and hardware with a laboratory into their curriculums, but in both cases, have stayed at the lowest level of hardware (for example, buffers [13] or microprocessor programming [14]).

These courses train electrical engineers, but there were no attempts to integrate software engineering with networking. Of all the approaches, only two attempted to combine software engineering and network engineering, and those were, surprisingly, over a decade apart. The first was a course taught at Dartmouth College by Sherman and Mark [15]. Sherman and Mark developed a laboratory and a curriculum but did not develop a full architecture nor was the laboratory open for research. Due to budgetary constraints the equipment was totally homogeneous (all Apple machines) with a totally homogeneous network (AppleTalk). There was no expandability provision for new technologies, different workstations, or different O/Ss [16-18]. This was a very good first

attempt at a network laboratory but fell far short of our needs of a full methodology and a fully integrated software engineering and networking laboratory at the systems level.

Over a decade later, Mengel and Bowling at the University of Arkansas [19], created a network laboratory. Even though this laboratory was created twelve years later, it still suffered from many of the flaws of the original attempt at Dartmouth. They used a single network topology, Ethernet, and the laboratory was exclusively for network related topics, no software engineering. Also, the laboratory was solely oriented to teaching and not research or a combination.

Again, a good laboratory for a specific purpose, teaching, but it fell far short of our needs of a full methodology and a fully integrated software engineering and networking laboratory at the systems level.

Even the latest articles in engineering education journals, one of which is a general overview of trends in network education, propose only a framework or a simulation for a network engineering laboratory. A fully integrated software engineering with network engineering laboratory seems to be deemed too complex to contemplate. [20, 21]

1.3.2 Peculiar Point: Industry

A peculiar point is that all the published articles report work done at universities, not in industry. This is somewhat expected because companies normally do not publish as much as universities and because companies like to

keep their work secret so as to not inform their competitors. What is strange is that there is no mention of an integrated software engineering and network engineering laboratory at the systems level in even the standards committees, such as ANSI or IEEE.

Since the standards committees are public and they need testing to validate future standards, one could confidently expect to see something referenced in the various reports—if such a facility existed in industry. This lack of comment can only mean that our required laboratory does not exist in industry.

1.3.3 Restrictions of Problem Types

All of these discovered labs suffer from the fact that they arbitrarily bound the levels of problems that can be studied. Examples of problems falling outside the bounds and being arbitrarily restricted are:

- study of distributed computer algorithms over long distances, *e.g.*, continental delays
- how to mix heterogeneous systems with heterogeneous network topologies with heterogeneous O/Ss
- how to utilize unused MIPS
- economic trade-offs of various topologies
- development of cluster computer algorithms.

These are real world problems that need immediate solutions.

1.4 Absence of General Methodology and Architecture

Another discovery in the search was, that not only was there no laboratory meeting our requirements, there also was no methodology or architecture to even create such a facility. There was no theory in place. This meant the architecture with a specification had to be created to generate the methodology before our general purpose laboratory could be built.

1.5 Requirements

These findings of this chapter can be summarized in two sentences.

1. There is a general need for a totally heterogeneous laboratory in systems types, network topologies, and O/Ss which totally integrates software engineering and network engineering and fully blends teaching with research.
2. Before such a laboratory can be created, an architecture with a specification has to be created to generate the methodology for the laboratory.

1.6 Summary

This chapter listed the industry trends (summarized in Figure 1) that are driving institutions to a new paradigm of teaching and research, that of fully integrating software engineering with network engineering.

Even though there is a requirement for a new paradigm:

- **There was no general networking laboratory known in the world that can pursue multiple topologies.**
- **There was no software engineering and network engineering, fully integrated laboratory at the systems level.**
- **There was no architecture or methodology to develop or employ such a laboratory.**

The result is that even before pursuing any dissertation under our research umbrella, a tool (here laboratory) had to be created, without prior art, and implemented. More importantly, there was no architecture or specification to use to create the research and teaching laboratory; therefore:

1. **a vacuum exists in architecture, methodology, and implementation of a software engineering and networking integrated laboratory at the systems level.**
2. **industry trends are requiring such a laboratory.**

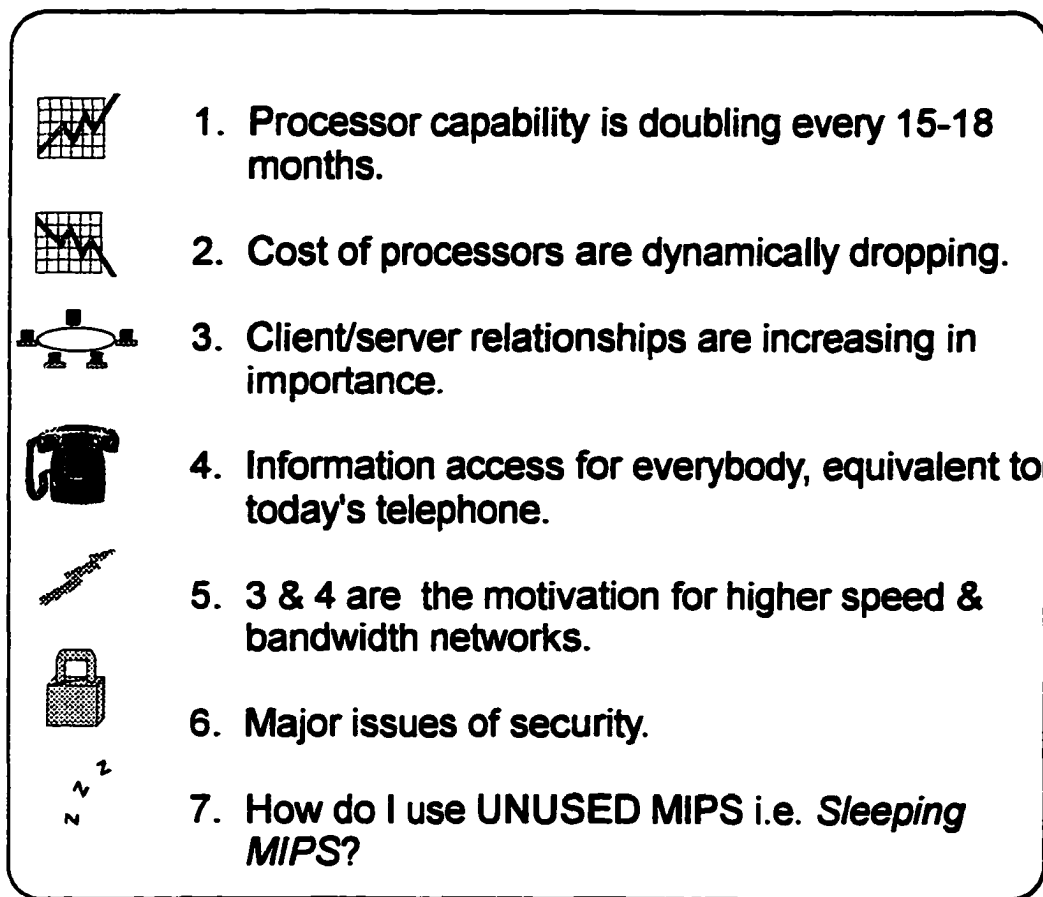


Figure 1. Industry Trends

Chapter 2: Design Goals

As demonstrated in the previous section, two fundamental requirements are being faced by industry and academia:

1. There is a requirement for a totally integrated network and software engineering laboratory to do systems level studies.
2. Research must be fully blended with teaching in this facility.

Investigation through literature, from industry, and from educational institutes showed there was no such facility to copy; furthermore, there was no blueprint to follow to build such a facility.

It must be remembered that our research team's original goal was to build a laboratory so that topics under our research umbrella could be pursued and later validated. Since there was no blueprint and given the breadth of our research umbrella,³ it became immediately obvious that a general purpose, multi-use, teaching and research facility would have to be designed and built. This was the genesis of this dissertation.

³ Remember, two of our umbrella topics were dynamic network load balancing, which is a packet study on networks, and digital logic simulation over a large number of heterogeneous systems, which is a software engineering issue. A very wide breadth.

2.1 Core Set of Problems

Before developing a blueprint, a core set of problems had to be catalogued to test the validity of the methodology as it was being developed. If the methodology is to be truly general purpose and multi-use, combining teaching and research for an integrated network and software engineering facility, it should handle not only the problems in our research umbrella but also other teaching subjects. Combining topics from our research umbrella with studies from the current course catalog, yielded the following list:

- performance studies across different network topologies
- network viruses
- dynamic network load balancing
- distributed simulation
- distributed software applications
- CASE tools & methodologies
- system security issues
- network security issues.

Even though these are very specific topics, they cover a wide spectrum of requirements that can strain an incomplete architecture. Reading the list, one can see there are network problems (performance, viruses, *etc.*); software engineering problems (CASE tools, system security issues); and combined problems (distributed simulation and distributed software applications). Because of this broad brush, the list provides an excellent testing device for the methodology.

2.2 Design Principles

Before developing the methodology, the foundations and the base assumptions must be clearly identified and documented for future reference.

2.2.1 Software Simulation vs. Experimental Laboratory

This has already been explained and the choice is experimental laboratory.

2.2.2 Open Laboratory or Dedicated Laboratory

An open laboratory means that it is free for use for not only teaching and research, but students can use the equipment for homework, e-mail, *etc.* Because there will be a large amount of experimentation occurring, the last thing wanted is extraneous traffic on the network, such as a student answering his mail. Such an occurrence could mean that experiments would not be reproducible nor repeatable; therefore, a dedicated laboratory must be the answer.

2.2.3 Network Isolation or World-wide Connectivity

To do software engineering or network research, the experimental network must be isolated from the production network. Why? By its very definition, a production network must be reliable and constantly available. A research network is just the reverse. Experimenters are going to crash the

network from time to time. This means the methodology and therefore the laboratory must, for the most part, be isolated from the production network.

Does this mean it must be always isolated from the production network? No. There are times when students and researchers must connect to nodes external to the experimental network, for example, to download code, view manuals using a browser, or to collaborate with colleagues. Therefore, both isolation and universal connectivity must be allowed.

The methodology must allow for ease of connectivity of the experimental network to, or isolation from, the production network. Furthermore, because we will be experimenting with network protocols, it is essential that we be able to disconnect from the campus backbone at anytime for any length of time. Therefore, users can research various topics without endangering the campus network, topics such as injecting viruses to penetrate firewalls.

2.2.4 Single Facility

As previously noted, there are many separate facilities that do individual topics. The missing entity is a general facility that does a large subset of problems, either research or classes. The developed methodology must be general purpose and allow concurrent experiments. This is a powerful concept. The goal is to allow a universal set of problems in one facility. Since many experiments are cross purposes to each other, this means isolation and controls must be put in place to prevent interference.

2.2.5 Automated Testing

Because of the complexity of tests foreseen for this methodology, the procedure developed must allow for simple, automated tests. This means tests must be able to be run from scripts or simple programs. Furthermore, to be an effective methodology, the system must allow automated tests to be run by any level of student, with minimal guidance and supervision.

Normally, automated testing is a requirement of industry and not of universities. Manufacturers run proof or performance tests of long duration; whereas, universities run short labs or class experiments. This distinction is not necessarily true today. There are cases, even under our research umbrella, that would require long duration testing. If so, it is illogical to expect researchers to sit and watch running equipment for the sake of watching equipment, therefore, the requirement for automated tests. Also, automated tests allow the tests to be run late at night which will free the laboratory for other uses during the day.

Let's look at one example of where this capability would be a necessity in a research environment, that of dynamic network load balancing. Once a balancing algorithm has been developed, how does one test it? It only makes sense to test on a loaded network that varies with time where the algorithm can pick the most appropriate topology for sending. This is the very essence of the requirement for automated testing.

2.3 Principles of Learning

Since the methodology is to be both for teaching and research, it is appropriate to digress and look at some principles of learning. It is well known that many people, especially engineers, learn considerably better if they can see the physical phenomena rather than just conceptualizing it with applied mathematics. As an example of this statement, consider congestion on networks and how certain topologies react. The response of many networks to congestion can be precisely shown through mathematics, but the ideal situation is for a student to go to a laboratory after seeing the theory and see the phenomena, with the resultant reactions, on a piece of test equipment. The concept becomes reality.

While this is obvious, let's expand the thought into abstract learning areas such as network topologies. It is very difficult to understand the topologies because the actual wiring is done in walls and connected through wiring closets, which is normally hidden from the students. To the student and researcher, wiring topology, whether switched, bus, *etc.*, is always the same. It is a cable from a system unit to a jack in the wall, completely hidden and unknown.

To overcome this problem and to teach students different topologies, one professor [21] in his laboratory course even mounted the cable topologies to wall boards so students could physically see the topologies as they ran the experiments. While there are practical disadvantages to this, it is an ideal way to teach the topologies and give students a better understanding of the realities involved.

Another difficult example is how to teach and show different compute models, such as client/server, clustering, or distributed computing. Since all that is happening is code running in a machine, it is not obvious to the student which model is executing. This issue is complex and will be addressed further in a latter section.

Chapter 3: Fundamental Design Principles

The goal for this dissertation is to develop a methodology that will create a general purpose, multi-use, combined teaching and research, integrated network and software engineering laboratory at the systems level. This is an ambitious goal; and as such, it needs to start with the fundamentals. It would be all too easy to obtain many systems and network components, throw them into a room, somehow connect them to some type of network, and declare it a laboratory. With no methodology or architecture, this would be doomed to failure because it literally would be a bunch of systems thrown into a room.

This chapter will give the fundamental design principles behind the methodology and will show the deductive reasoning behind each principle. Once these principles have been developed, an architecture can then be completed.

3.1 Typical System Unit

Before proceeding, the requirements for systems used in this methodology must be defined:

- some number of heterogeneous units
- some number of homogeneous units
- must be able to use multiple network interface cards (NICs) simultaneously

- must be able to support and use multiple network topologies simultaneously
- must be able to support client/server, cluster and distributed computer paradigms.

Obviously this is not a complete machine specification; and just as obviously, it is very vague because one should not write a general methodology around one specific machine type. With that said, where did these requirements come from?

3.1.1 Homogeneity and Heterogeneity

The real world is a heterogeneous collection of machines and networks. Any architecture that is to study real world scenarios must accept heterogeneity and reject total homogeneity as too restrictive.⁴

This should not imply total heterogeneity is a requirement. Beyond a certain point, heterogeneity adds tremendously to administration overhead for very little gain in the learning experience. There is also a class of problems, such as cluster computing, that require homogeneity in systems.

These divergent requirements mean that both homogeneous and heterogeneous structures must be allowed simultaneously. Furthermore, even

⁴Most laboratory system administrators want total homogeneity in systems, O/Ss, network topologies, network components, *etc.* because it is much easier to perform administration. Learn how to administer one system and you can do all systems. Unfortunately, this is not real world.

though this requirement adds complexity, the administration must be simple or straightforward through well-documented procedures.

3.1.2 Multiple NICs of Multiple Topologies

Wide-ranging network research, from dynamic load balancing to analysis of complex network topologies can only be performed if each workstation allows the installation and configuration of multiple NICs. An example of this statement is today's client/server arrangement. [22, 31] Servers typically have multiple NICs of a fast network topology to maximize throughput to the client plus a low speed network topology for status (the term used for this is *heartbeats*). Three to five NICs are not unheard of for large servers.

This means the derived design methodology must transparently work with multiple NICs of any combination of multiple network topologies, simultaneously.

3.1.3 Compute Relationships

Today, there are four major compute relationships in the industry: parallel, distributed, cluster, and client/server. Parallel computing requires specialized equipment which is not apropos for a general purpose laboratory, such as being done with this methodology. Also, if the parallel equipment is correctly in place, the application will see a single system image. Virtually any methodology will work with a single system image, so parallel computing as an entity does not have to be considered as it relates to the methodology.

The other three are more complex through. Each have characteristics that are independent from the others, characteristics that are fully visible to the application and the network. A full and robust methodology would allow for client/server relationships, cluster computing, and distributed computing, either as individual compute models or in any combination. As already noted, since all that is happening is code running in a machine, it is not obvious to the student which model is executing. This complex issue will be addressed further in a latter section.

3.2 Host Naming Convention

The transmission of data between network attached systems is a process of address translations before the data can be transmitted. The name of the host to which one is sending data, such as *hal9000*, is converted to a network address, such as 129.45.36.188 in IP octets, which is further converted to a hardware address, such as 08 00 3E 30 25 07 if it's an Ethernet adapter.

This is a straightforward approach until one introduces multiple NICs into the systems. As a further complication, these extra NICs can either be on the same network or on multiple networks.

The question becomes how does one address, *i.e.*, send data to a specific NIC? To answer that, one must go back to the original definition of the addressing scheme to understand how to direct the information to the correct NIC in the correct machine. The original definition is that the addresses and “host” names belong to the NIC. This may seem like an abstract point but

without this abstraction one could not direct traffic to the appropriate NIC on the appropriate system.

For most users, this is at best a trivial point since they have a single NIC in a single machine. The problem arises here because I am trying to develop a methodology for a complex facility.

Since the goal of the methodology is to explore multiple NICs (same or different topologies) in the same machine, a naming scheme must be developed so that a user could directly and unambiguously address a specific NIC among one or more NICs in the same machine across multiple topologies. To understand the magnitude of the problem, consider the case of a small facility: thirty machines, three topologies, with four NICs per topology. Potentially, one would need $30*3*4 = 360$ unique names.

3.2.1 Why Unique Names?

Before describing the algorithms that were tried and then discarded for the final rule, the issue of why the names must be unique has to be addressed. Another way of stating this is, if experimenters never use the full capability (360 nodes in the example) why force unique names? Why not draw names from a pool and return them when finished?

There are three significant reasons, among many, why the pool concept will not work for this methodology. First, the systems' O/Ss must know the name at all times. If the names are being drawn from a pool, then there is considerable administrative overhead into putting that information into system

files. It can become an error prone burden which adds time to the experiment and detracts from the research or studies.

Users will be forced to keep track of all machine addresses at all times. This makes the running of experiments extremely difficult.

Third is the running of automated test scripts. If the student wants to re-run an experiment, then he must modify the original script. While this might seem insignificant, it can introduce an error and add wasted time to the effort. There is even a more dangerous problem with the automated scripts though. What if the student does rerun a script and forgets to change the name and the script works because another machine happens to have that name. The student would receive false data and might not know it. A laboratory must always be designed to prevent the logging of false data, especially if the user may not realize it.

A pool of names will not work.

3.2.2 False Starts

The naming convention is critical to the ease of use of the ultimate implementation of the methodology. Without a precise naming convention, automated test scripts cannot be run, users will not know which machines has what capabilities, *etc.* The importance of the naming convention is analogous to that of the word length in a computer. Before any computer system design can be started, the word length and its requirements must be fixed. Because of this

importance, it is necessary to describe the deductive reasoning that led from the many false starts to the final workable naming rule.

The first false start was to name the NICs randomly, after anything: beers, Star Trek characters, actresses, *etc.* [43-47] While this might be an acceptable practice for a single machine in a single room, it becomes unwieldy when there are a large number of machines. Returning to the previous example of 30 machines, in the same room, with potentially 4 network adapters per machine, with up to 3 network topologies, quick math showed that 360 names were needed for the facility. Totally impractical.

This meant the rule must be progression-like. The simplest rule with a progression is to name the systems with letters followed by some identifier for the NIC, such as "1", ".1", or the like. A simple rule, but impossible to relate to the hardware and the NICs. Expanding our previous example of 4 NICs and 3 topologies, does that mean the numbering convention is base 12, base 4, or base 3? This rule is easy to implement but virtually impossible for an experimenter to use.

3.2.3 The Naming Rule

After many false starts, the following rule was developed to guarantee unambiguous names for hosts per NIC per topology.

Hostname =

**lab || manufacturer || system || _ || network || _ || NIC
name identifier number topology number**

where || is the concatenation operator and is not part of the name

_ is the underscore character and is part of the name.

all characters are lowercase

Figure 2 shows this rule in detail.

This rule identifies the NIC by which machine it is in (manufacturer identifier and system number), what network type is attached, and what NIC in that machine is being addressed.

The underscore character, "_", was deliberately chosen to be a delimiter. This makes parsing commands much easier to use. For example, if one wants to determine which topology is being tested in a certain script, just capture the information between the underscores.

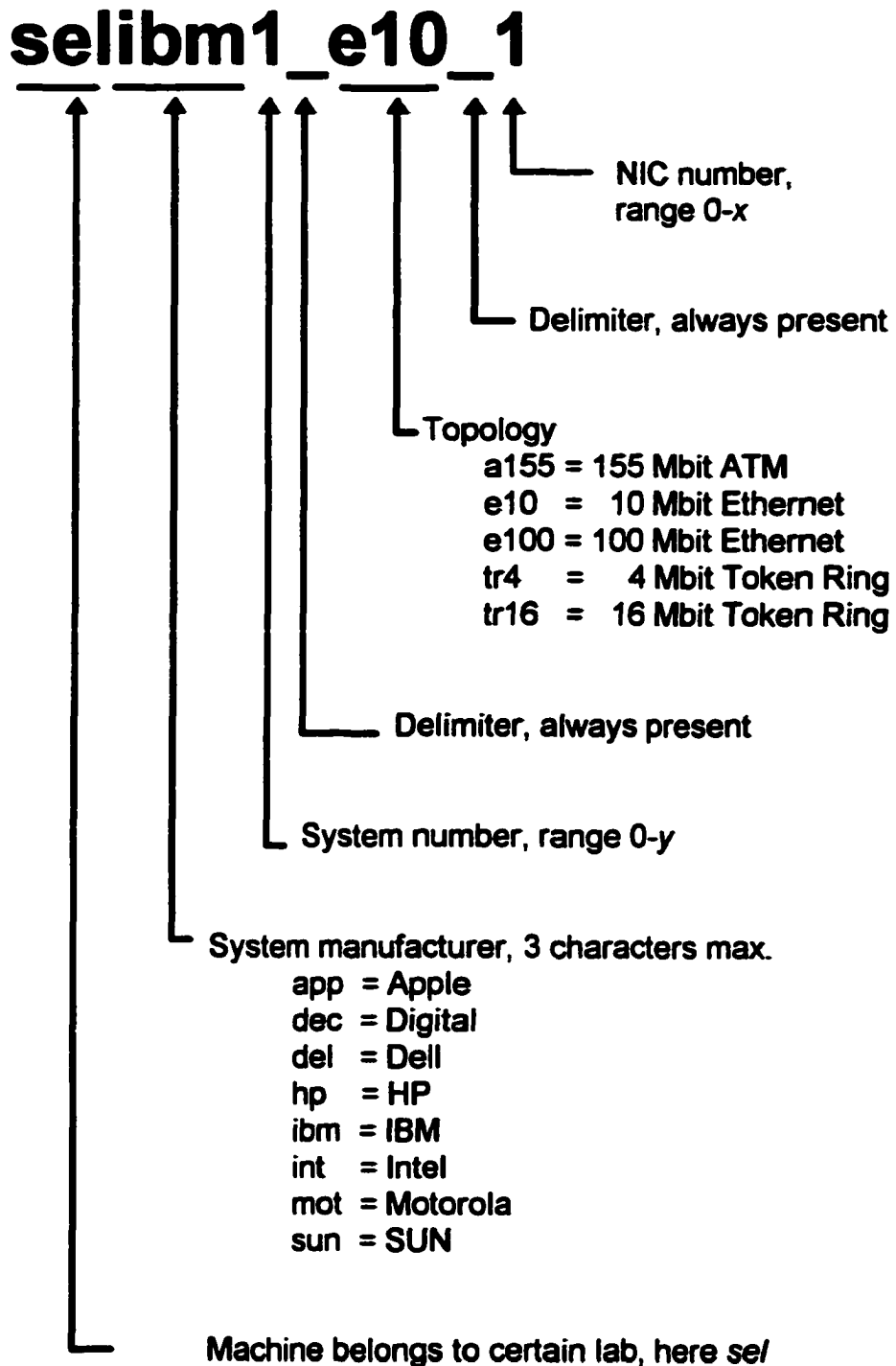


Figure 2. NIC Identification Convention

To help explain the rule, I will use real manufacturer names and topologies, for example `ibm` for IBM, `a155` for 155 Mbit ATM, instead of being abstract. In addition it is always good policy to identify the laboratory where these particular machines reside. For these examples, I will use the name `sel`.

First example, one wants to communicate to the second 155 Mbit ATM adapter in the IBM #2 machine. What is the address? Using a numbering origin of 0 (to be explained in the next section) and following the naming progression shown in Figure 2, yields:

```
sel
  ibm
    2
      -
        a155
          -
            1
```

The complete name is `selibm2_a155_1`.

Another, Motorola machine #1, second 10 Mbit Ethernet adapter (`en1`), yields:

```
selmot1_e10_1
```

Very straightforward. This convention will generate guaranteed unique names.

3.2.4 Special Use Machines

Special use machines, such as routers, printers, *etc.*, could have the same naming convention. Another mechanism is if the facility has a few special devices then the manufacturers' identifier could be replaced by their function. For example, consoles are con, servers are serv, *etc.*

3.2.5 Powerful Naming Convention

The rule finally chosen is very simple and in that lies its power. It is very simple to use and learn, unambiguous and fully extensible. First, it allows a concise definition of multiple NICs in the same machine, even if they are different network topologies. Second, it is easily extensible to different network adapters, different system manufacturers, and even future undefined network topologies. By a simple definition of a new abbreviation for a manufacturer or for a new network topology, one can make major changes to the implementation without reinventing new host names. New devices can be easily integrated into the current structure with only insignificant changes. Old scripts will work with new hardware additions. This is a tremendous asset for experimenters and programmers.

3.3 Compute Islands

The next concept that must be addressed is how to organize the layout of the integrated facility. The simplest answer is to just place the systems on tables in a matrix pattern. For example, if there are $x * y$ system units, then place the machines on the tables in an $x \times y$ pattern. Simple, but a usability failure.

In this pattern, how would a user determine which machines are the servers in a client/server arrangement? Which machines can be clustered? Which machines are available for distributed computing with continental delays? Obviously in a rectangular pattern there is no way to easily make a determination of the compute relationship.

One of the requirements of this methodology is that users: students, researchers, and faculty, must easily make this determination. From the learning principles sections, the best way to make that determination is visually.

This means the compute relationships of cluster, distributed, and client/server must be visually portrayed to the user. What is unique to each of these compute paradigms that can be visually portrayed? Cluster computing is normally a set of homogeneous machines close together. Distributed computing is a set of largely independent and heterogeneous machines far enough apart that delays become important. Client/server has at least one machine, the server, surrounded by a series of clients.

The requirement for heterogeneity and homogeneity plus the characteristics of the compute paradigms guides us to organize the layout of the facility into distinct physical entities which will be termed compute islands. Each island should be populated with a significant number of nearly identical workstations by the same manufacturer. This will provide the homogeneity. Different vendors should be used to supply machines for different islands. This will be the heterogeneity. On each island, one machine will be designated as a server.

These islands should be physically separated but all inter- and intra- connected by LANs.

This straightforward physical separation allows both cluster computing and distributed computing to take place in the same room. By separating the machines into islands by vendor, users naturally understand the capabilities of each island and logically think in the terms of cluster versus distributed computing without being explicitly told.

Intra-island connection will give cluster computing. Inter-island gives distributed computing. Since the islands are physically separated, users naturally think in terms of delays. With the insertion of some delay mechanism between islands, real world delays, from slight to intercontinental, can be simulated in the facility (actually same room). This is a very powerful concept. Students will naturally think in terms of delays between islands. For example, island 1 can be city A and island 2 can be city B, all very natural. Also, the island concept fosters the thought of compute clusters widely separated which is another research area.

To this point, this discussion has deliberately ignored client/server computing. By designating one machine on an island as a server and designating the rest as clients, users naturally think in terms of client/server computing. By using a server on another island, the effects of heterogeneity can be introduced into the study. Again, all very naturally.

Also, by having all the islands intra and inter-connected, students can look at the effect of network outages, in a very simple context because cables can be disconnected which give a visual indication of what is broken.

Figure 3 shows the island concept.

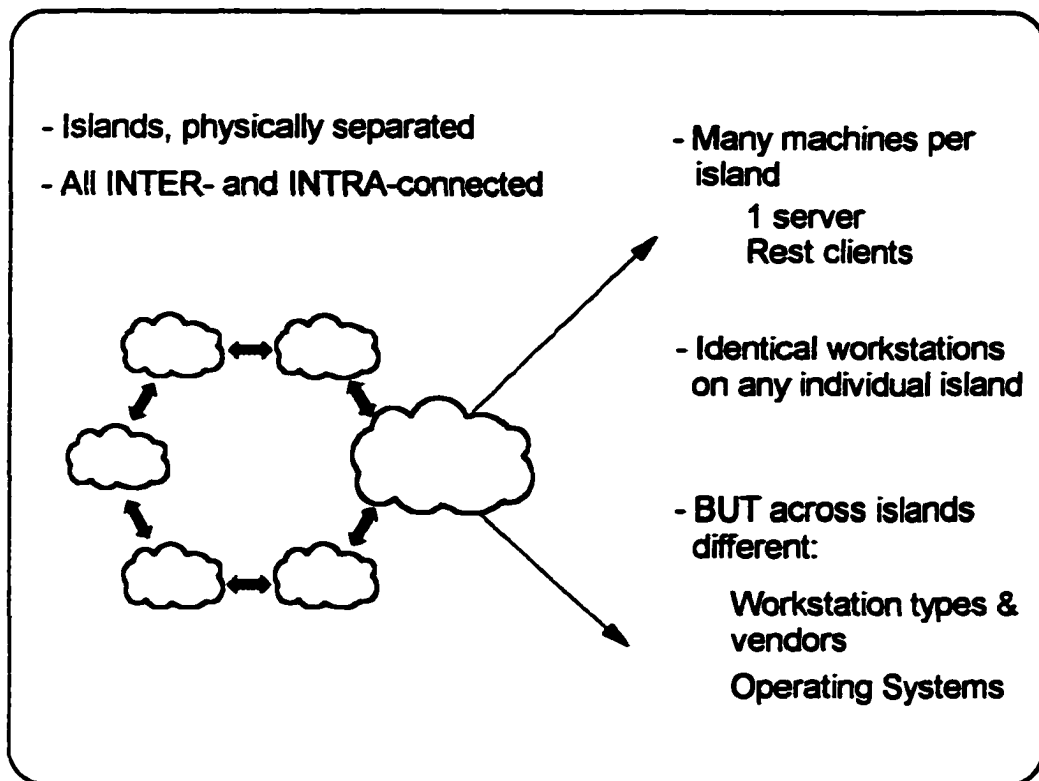


Figure 3. NeaSEL Compute Islands

3.4 Numbering Convention within Islands

The next methodology question is how does a user identify machines in an island? This is necessary so a user can immediately and unambiguously determine, for example, where the server is. Before explaining the algorithm, the numbering convention used within system units and O/Ss must be understood. In virtually all cases from all vendors, the origin for all numbers is 0, not 1. Another way of stating this is the first of anything: system units, NICs , interfaces, *etc.* is always zero. This means the first system unit number on an island must be 0 to provide consistency across all items that use a number.

As an example of this consistency, consider that the first LAN adapter under UNIX is termed and fixed at *en0* or *lan0*. It would be illogical to name the first system unit *ibm1* (for example) but refer to the first adapter in that unit as *en0*, hence the first system unit is *ibm0*

With the origin now understood, let's assign each location on an island a number. As one faces the equipment on an island, location numbers increase from left to right and continue in a horseshoe path around the island (see Figure 4). Numbers start at 0.

Assign the server for the island to position 0. If there is a second server, then assign it to the last position (in the figure that would be position 5).

Obviously, this numbering convention is utter simplicity. It was so chosen because it is an integral part of a very concise host naming convention.

Once this naming convention is understood, a user can determine which machine, which topology, and which interface of that particular topology he wants to use without resorting to a table lookup of names or memorizing a series of very complex names.

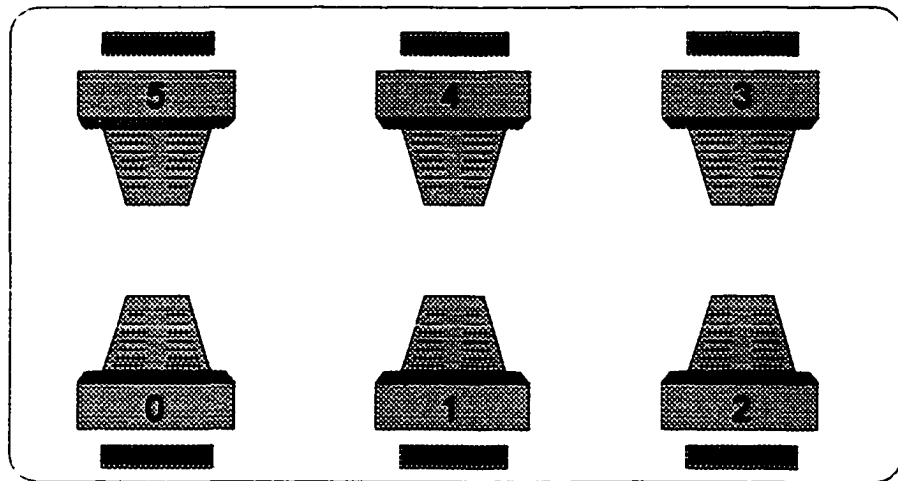


Figure 4. Numbering within Islands.
(Top view, bottom of table is facing entrance door, assuming 6 machines)

3.5 Capabilities

Before going further, the effects of the numbering convention combined with the compute island concept plus the numbering convention must be discussed.

These effects are synergistic. To validate this statement, consider the following examples.

With a minimum amount of training an experimenter instantly knows how to address machines. If the person needs to communicate with a server on the Sun island (for example), 155 Mbit ATM connected, with only one NIC in the box, the name is `selsun0_a155_0`. If a researcher is working on a DEC (for example) but wants to use an IBM cluster for computation, then he knows he must access machines of `selibmi_NIC-information`. If a student needs to study distributed computing with a city delay, all he has to do is put a delay device⁵ between the two islands he wants to study.

The power of these items is that they hide the complexity of the procedure but accentuate the concepts. This is as it should be for a facility with a design goal of ease of use.

⁵The delay can be caused by a formal delay line or by interposing another system that is in a spin cycle for the required time period.

3.6 Connection or Isolation

To give full capability to the created laboratory and to be able to perform the wide range of anticipated experiments, it was determined that the facility must allow for ease of connectivity of the experimental network to, or isolation from, the production network. A further requirement is that the facility can be disconnected at anytime for any length of time.

This seems very easy to do but the requirement for connectivity and isolation makes the methodology very difficult. The problem stems from the fact that much of the world is network oriented and communication procedures rely on a connection. For example, where does one get the password file? How are external addresses resolved if there is no network connection?

Because this is a very complex issue it would be too easy to become too abstract and miss the essence of the problem. To help remove this abstraction, let's pick a specific problem under a specific protocol, that of name resolution under UNIX. Even though I have picked a specific instance of the problem, it is important to note that all protocols under all O/S have some form of this problem.

Returning to the specific example, to resolve host names versus TCP/IP addresses, most UNIX systems look to three places:

- internal cache tables
- external name server

- /etc/hosts file

These three sources of name resolution all serve the same purpose but attack the problem by different methods. The cache table stores the most recent address resolution. It has a limited capacity but it is fast. /etc/hosts is a file stored on the individual machine that has address resolutions, can be of any size, but can have out-of-date information. The external name server is normally very large, slower response than the other two, but is always updating itself so the information is always current.

The O/S will look at each one but in what order. Obviously, the host will always look to its internal cache table first. The real question is which order, name server then /etc/hosts, or /etc/hosts then name server, is the name resolved next. Since a name server look-up implies network traffic, the order is critical for precise, repeatable, performance measurements. The default search order for UNIX is name server then /etc/hosts, so something must be done to change this.

The goal of the methodology is to be able to run independently of external networks, which includes external name servers. Since the default search order is name server then /etc/hosts and if there is no access to the name server (*i.e.* a disconnected network), then the system will hang because no name resolution can be done. If the first choice in the search path is not available, UNIX does not go to the second choice. So the search order of the name server to /etc/hosts is not appropriate for this methodology.

A simple solution might appear to not specify a name server at all. UNIX would always look at /etc/hosts file. If no match, then the routing fails. While

this approach would work for experimentation, it would not work for any application that must go to an external network, for example Web browsing, downloading code, *etc.* Since some work will go outside the room, a name server must be allowed. The only solution is to change the default search order for name resolution.

This explanation was only for name resolution. The same discovery process must be applied to password resolution, command resolution, and the like. This means that for all systems to work properly under this methodology they must have their resolution orders defined in a certain.

There is no universal syntax for this modification because each O/S does configuration slightly different for each protocol. This means considerable research must be done in very specific areas to configure the systems correctly.

3.7 Physical vs. Electrical Isolation Procedures

A network and software engineering experimental laboratory must have an inherent capability to be removed from the campus network at any time for any duration. The real question is what is the mechanism to do this? Today, there are two. The first is by programmable network devices and the second is one is a mechanical one: disconnect the cable from the jack.

Current network devices such as switches and routers have capability to control their traffic flow. They can either filter some of all of the traffic. With a graphical users interface (GUI), many are straightforward to program and they give a lot of statistics. In a stable environment such a production environment, these are very desirable features.

In an experimental environment, everything is changing. This would require somebody to be constantly programming the device. No matter how simple the programming procedure is, errors will occur. While this is a time waster, this is not the true disadvantage. Unless there is a monitor attached to the device and that monitor is looking at the device's characteristics, there is no visual way of knowing if a connection exists to the campus backbone. The lack of a visual indication is a major detriment in an experimental laboratory.

Cables are very simple to use. If the cable is not plugged into the jack then a connection does not exist. It is simple, visual, and no programming is involved.

For a facility as potentially complex as this methodology will allow, the most reliable disconnect is the simple cable. Use cables to connect to the campus backbone and for inter and intra-island connection.

3.8 Summary

This chapter introduced three powerful concepts that are the heart of the approach to the methodology for designing an integrated systems laboratory.

1. A naming convention

- **that directly and unambiguously addresses a specific NIC among one or more NICs in the same machine across multiple topologies**
- **fully extensible to future network technologies, system manufacturers, *etc.***
- **fully supports automated test scripts**
- **easy to learn, easy to use, easy to visualize.**

2. Compute island concept

- **allows the student to visualize what compute paradigm is being used**
- **easily modified into different paradigms**
- **visualizes and therefore makes easy the conceptual implementation of combination computer paradigms.**

3. Numbering Convention within Islands

- **part of the simple, flexible naming convention**
- **unambiguously places the server by name**
- **an easy mechanism to help bring machines together in a cluster**

The chapter also raised two points that must be considered implementing the methodology

1. Resolution order for connectivity to or isolation from the network

- **resolution order of commands, names, passwords must always be internal first then external, this will guarantee the facility will work both connected to and isolated from the external network**
- **the syntax to change this order is different for each O/S for each protocol**
- **must be checked because defaults could be the wrong order.**

2. Physical vs. electrical isolation procedures

- **use cables: simple, visual, and no programming is involved**
- **programmable network devices: should not be used because no simple visual indicator of connection status.**

Chapter 4: Architecture

Once the design principles have been defined, the only other items to complete the methodology is the choice of network topology and design constraints.

4.1 Network Topology

Figure 5 illustrates the general network topologies in use today. Obviously, each has its advantages and disadvantages. The point-to-point topology is the fastest with the least delay but only allows connections for two machines. A bus allows many machines to connect but throughput varies greatly depending on the load. A ring guarantees a certain throughput which does not vary with load but this means a lightly used network actually becomes inefficient. A switch allows many simultaneous conversations, but until recently, could only handle voice traffic because the electronics could not process connections fast enough to handle LAN traffic. Furthermore, many topologies did not lend themselves to be converted to a switched topology.

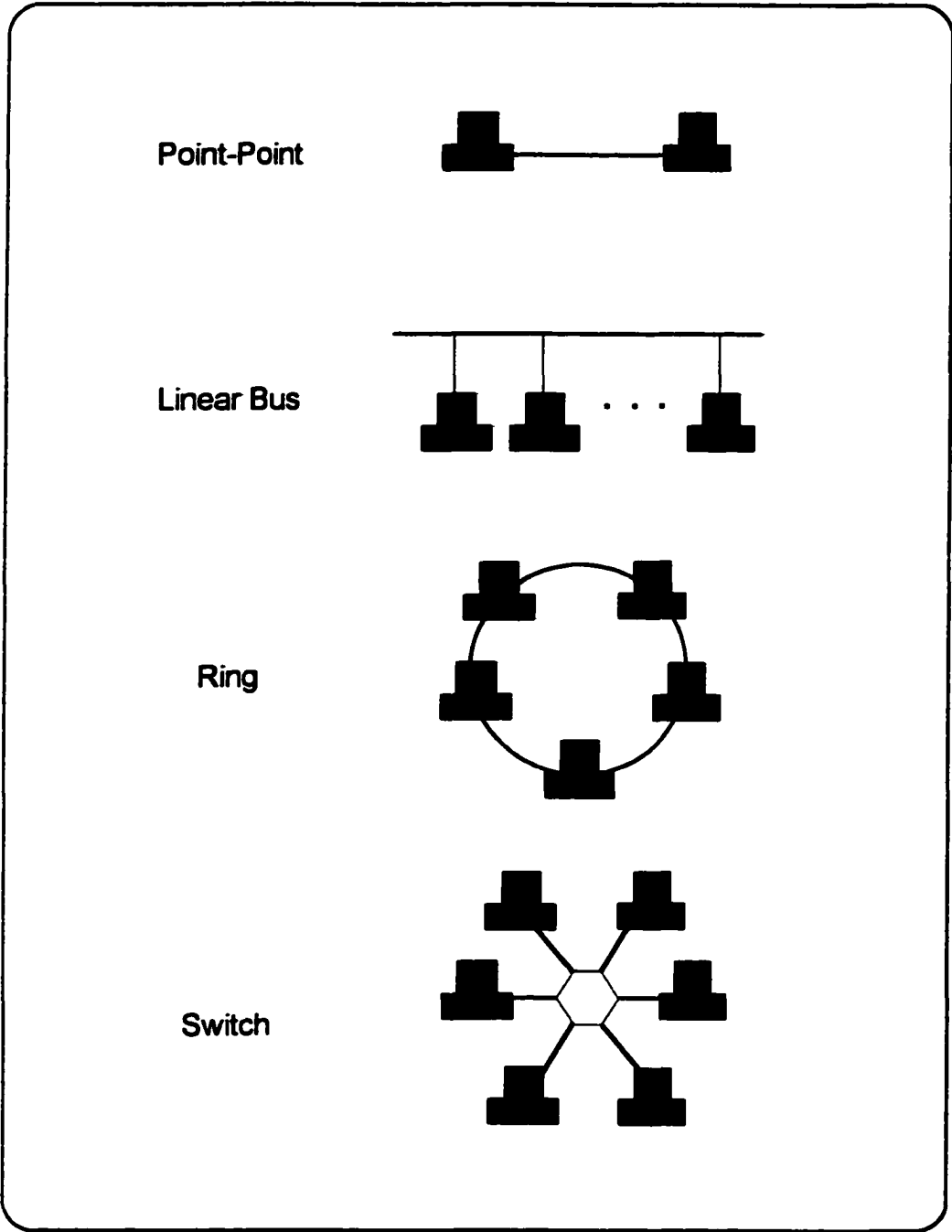


Figure 5. Network Topologies

4.2 Topology of Choice: Switched

4.2.1 Current Status of Switches

Switches logically convert each pair of users into a point-to-point connection. This yields the best performance for a network topology in terms of throughput, with three caveats. First, the switch must be able to handle numerous, simultaneous connections. Second, the set-up time of the switch (address conversion and the like) must be small when compared to the bit rate of the line. Third, the original topology of the network must lend itself to being remapped into a switched topology.

Within the last five to seven years, there has been a revolution in network switching and these three major problem have been solved. First, processors are readily available that will handle MFLOPS, with GFLOPS on the horizon. Second, processors, particularly microcontrollers, can handle just about any set-up time without an impact to the throughput. Third, most topologies, even the standard Ethernet, have been converted to switched topologies. [22]

4.2.2 Advantages and Mimicking Other Topologies

Switches give point-to-point connections, therefore, the fastest throughput. For experimentation purposes, it is always best to perform the experiment at the optimal or best point and use those results as a base line.⁶ It is

⁶For network performance measurements, *best* means the largest throughput or the least latency.

easy to slow a network down; it is virtually impossible to speed it up once installed. A switch provides the best performance with the capability to easily degrade the performance for other measurements.

Even with this advantage, a switch would be useless in a laboratory environment unless it could be made to mimic other topologies. Why? First, the whole world is not switch based. Next, there are very interesting problems that do not lend themselves to the fastest throughput in the real world. A prime example is distributed computing over satellite communications, where there is a very large delay between the nodes.

Mimicking is an area where a switch shines for laboratory use. A switched topology can imitate virtually all topologies. For example, topologies with extreme delay between nodes, such as satellite communications, can be simulated by the use of delay lines.⁷ A bus structure, such as classical Ethernet, can be recreated by placing all the systems on one leg of the switch. For a ring, just connect all the systems in a ring pattern on one leg of the switch.

4.2.3 Disadvantage

While there are significant advantages, a minor disadvantage is a switch cannot handle *broadcast* information. Switches handle *multicast* but not general broadcast.

⁷For those not familiar with delay lines, they do as their name states. The technology is very stable, mature, and quite inexpensive. Their use adds insignificant cost a laboratory.

For a laboratory this is not significant. The reason being there are topologies that do handle broadcast, and switches can be made to mimic those topologies in a laboratory. It would only be a problem in a production network where topologies are not free to change.

4.3 Design Constraints

Because of the complexity of the methodology, there are really two types of constraints: those rigidly defined by standards committees and those based on good design practices.

Rigid constraints are easy to define. They just require diligent research into the publications of the various standards bodies. One such rigid constraint is that the host name of a system when on an IP network must be 63 or less characters.[23] Constraints based on experience though are much more difficult to find and define. Consider two cases of this statement: number of machines per compute cluster and number available for distributed computing.

For cluster computing it has been demonstrated that interesting problems can be tackled when there are three or more machines interconnected on a high speed network. [25-29] For less than three machines, the increase in compute horsepower does not outweigh the cost in O/S complexity.

A different observation can be made for the number of machines employed in a distributed computing situation. Here interesting problems occur with twenty-five or more machines. This is a statement of complexity of O/S,

applications applicable to distributed computing, and time of flight on the network. [30-37]

Finally, mathematicians dislike the numbers one and two and this equally applies for computer networks. Too many realistic problems have been hidden by using only one or two systems.

4.3.1 Infrastructure

Systems per compute cluster, x :	$3 \leq x \leq 100$
Systems available for distributed computing, d	$25 \leq d \leq 1000$
Number of cluster islands, y :	$3 \leq y \leq 10$
Servers per island, s :	$1 \leq s \leq 3$
Number of manufacturers represented, m :	$m \geq 5$
Total number of machines, tot :	$tot = x * y$

4.3.2 System Unit Properties

Adapter slots per machine, n :	$n \geq 3$
Memory, mem :	$mem \geq 64 \text{ MB}$
Disk space, $dspace$:	$dspace \geq 2.0 \text{ GB}$

4.3.3 Network & NIC Constraints

Network topologies to be represented, <i>nt</i> : (low, medium, and high speed)	$nt \geq 3$
All network topologies, <i>nt_i</i> , are independent	$\forall nt_i, nt_i \cap nt_j = 0$
Average number of NICs per host, <i>nic</i> :	$3 \leq nic \leq 5$
IP addresses required, <i>addr</i> :	$addr = nt * nic * tot$
Number of subnets required, <i>subnet</i> :	$subnet = nt * nic$
Constraints on subnets, not allowed:	all 0's or all 1's
Private network not allowed	No addresses in range 10.0.0.0 172.16.0.0 192.168.0.0

4.4.4 Host Naming Convention

How hosts are named depends on what protocol is being used on the network. IP was chosen for this analysis. The reason: IP has now become ubiquitous and is a good representation of the problems associated with host names.

Host names on an IP network have the form of:

hostname.nth_level...fourth_level.third_level.second_level.top_level

An example of this convention is *hal9000.eeddept.ut.edu*, where *edu* is the *top_level* name. While there is no limit to the number of sublevels, *n* (only a practical limit of typing by the human user), there are constraints defined by the RFCs:

Number of characters per fully qualified name	$numchar \leq 255$
Number of characters per IP hostname, <i>hn</i> :	$hn \leq 63^8$
Number of characters per second level name	$sname \leq 12$

⁸ Note, 63 is a maximum. Many O/S's will not support the maximum. Also, many O/S's still prefer to use the old standby of 8 characters as a maximum.

Chapter 5: Case Study, NeaSEL

The completed methodology was implemented on a large scale as the Network and Software Engineering Laboratory, NeaSEL, built in the Electrical and Computer Engineering Department at the University at Texas at Austin, Texas. Because of the large scale of implementation, it would be very easy to get lost in the details. To prevent that, this chapter will give an overview of the implementation. Specific details, such as wiring, color coding, patch panel connectivity, *etc.* will be given in Appendix B.

5.1 Overview

Since NeaSEL was to be a physical validation of the methodology and architecture, every attempt was made to adhere to the methodology exactly. There were three design principles on which the methodology was based: host naming convention, compute island concept, and host numbering within islands. Figure 6 shows how the islands were implemented at the date of this dissertation.

Specifically, there six machines per island, five islands, for a total of thirty machines. NeaSEL had three wiring topologies: 10 Mbit Ethernet, 100 Mbit Ethernet, and 155 Mbit ATM. The 10 Mbit Etherent and 155 Mbit were actively pursued in testing. The 100 Mbit Ethernet was not pursued because of initial lack of equipment.

5.2 Compute Islands

NeaSEL has five areas that contain compute equipment. Compute equipment is segregated by manufacturer on a compute island so there is homogeneity in hardware within an island and heterogeneity across islands. For example, all IBM PowerPC 40Ps are located on island 1; all DEC Alphas on island 4. As of the date of this dissertation, NeaSEL has the following islands:

- IBM RISC System/6000 40P
- DEC Alpha 3000 Model 300LX
- Motorola PowerWorks RISC PC PCTMT604-100
- Intel Pentium 90 MHz
- Sun Ultra 1 Model 140 & 170E.

The mix of workstations is powerful and are cost-effective. Each workstation has been fully configured with a minimum of 2.0 GB hard disks, 64+ MB of memory, and L2 cache. Any of these items is easily removable if the experiment calls for it.

The O/Ss of these machine represent a major cross section of personal computers to UNIX workstations.

Through the network, islands can be isolated from one another or, conversely, connected into a complete network at any time.

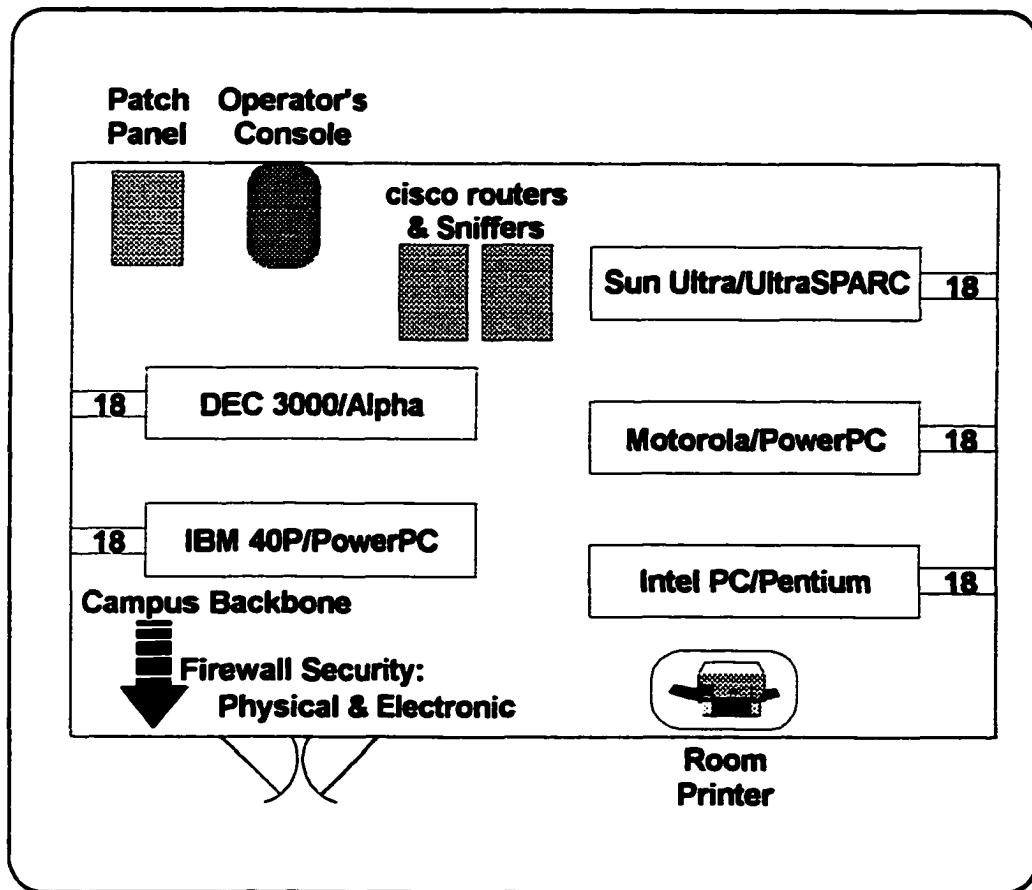


Figure 6. General Layout of NeaSEL

5.3 Topology and Protocol

Currently, NeaSEL has two fully switched topologies: 10 Mbit Ethernet and 155 Mbit ATM. Two Kalpana EtherSwitch Pro16's (now cisco) are used for 10 Mbit Ethernet. Two Fore Runner ASX 200 Switches are used for the 155 Mbit ATM. In each topology, the switches are ganged to give a single switch image. The result is each test system is on an individual port on a single switch image.

The protocol being tested is IPv4. This protocol was chosen for initial testing because it is the only universal network protocol. It works over all network types (ATM, Ethernet, token ring, *etc.*) on all systems (Intel, Sun, IBM), all models, and all O/Ss (UNIX, NT, OS/2, *etc.*).

5.4 Wiring Infrastructure

All wiring in the room, including patch cables, are CAT 5. All wiring is enclosed in raceways attached to the walls. To achieve the flexibility of NeaSEL and to maintain firewall security, all wiring is done through a patch panel that can be physically and electrically isolated from the rest of the campus.

There are seven points on the raceways for attachment by systems to the network (Figure 7). Each connection point on the raceway provides 18 RJ-45 connectors, six each for three different network topologies.

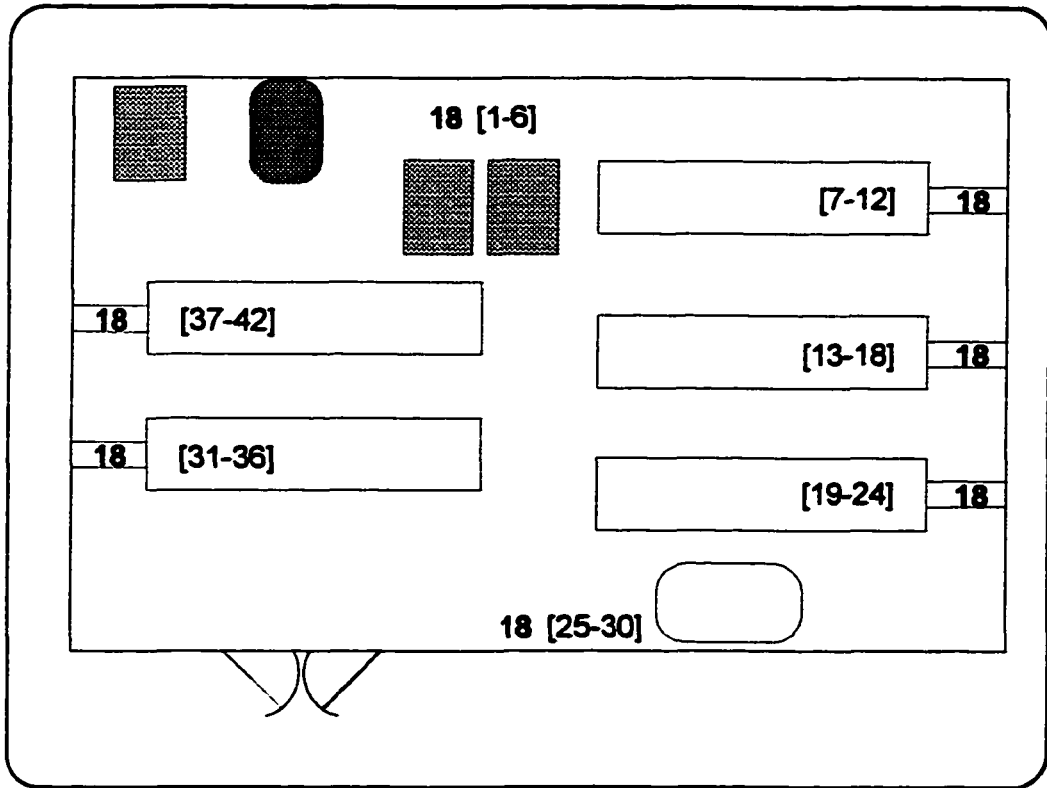


Figure 7. Raceway Attachments

The number 18 shows the locations of the connection points.
 The numbers in [] are the numbers assigned to the connectors.

5.5 Multiple NICs per Workstation

NeaSEL utilized full function workstations that allow multiple NICs. The workstations selected all run O/Ss that will support multiple NICs simultaneously. The switch topology allows full connectivity of multiple NICs. Figure 8 shows this capability.

5.6 TCP/IP Addresses

5.6.1 Subnets

NeaSEL has three subnets assigned to it:

- 146.6.148.0 10 Mbit Ethernet
- 146.6.149.0 100 Mbit Ethernet or ISDN
- 146.6.150.0 155 Mbit ATM.

These subnets were further divided into four subnets each. This allowed each system to have four independently addressable adapters.

- **Goals:**
 - Maximum connectivity with
 - Minimum number of physical changes

- **Each island has 18 connections**
 - All CAT-5, 155 Mbit lines
 - On the average, 3 connectivity options per machine **simultaneously**

- **Some examples:**

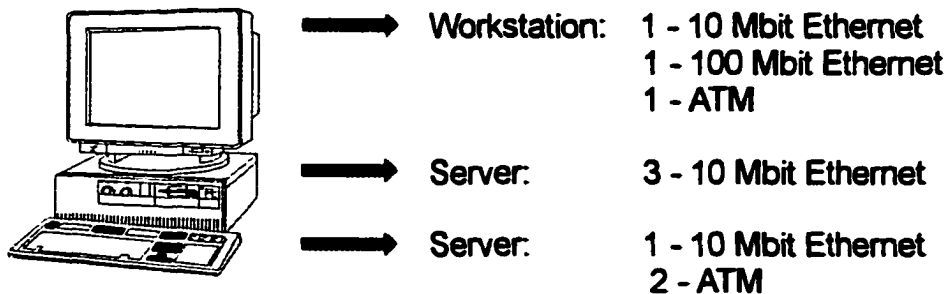


Figure 8. Network Connectivity by Machine

Table 1. Addressing Range vs. Manufacturer and NIC

System Manufacturer	NIC 0	NIC 1	NIC 2	NIC 3
IBM	200-209	130-139	70-79	01-09
Motorola	210-219	140-149	80-89	10-19
Sun	220-229	150-159	90-99	20-29
Dec	230-239	160-169	100-109	30-39
Intel	240-249	170-179	110-119	40-49

The primary NIC is assigned from the NIC 0 range. The second NIC is assigned from NIC 1. *etc.* Within a range, numbers are assigned sequentially. For example, the primary (first) NIC in IBM 0 will be 146.6.148.200, where we are using 10 Mbit Ethernet as the example topology. The second IBM machine, IBM1, is 146.6.148.201. A more complicated example is the second NIC for 10 Mbit Ethernet in IBM 3 will be 146.6.148.133.

5.6.2 Network Topology vs. Addresses

To change the address from one network topology to another, all one has to do is to change the subnet number. The host addresses will stay the same. This was a deliberate design point to make script writing much simpler.

Consider the following practical example. A researcher wants to compare ftp performance between 10 Mbit Ethernet and 155 Mbit ATM. Say the two host numbers are 200 & 235. The addresses are:

Ethernet to Ethernet	146.6.148.200 to 146.6.148.200
ATM to ATM	146.6.150.200 to 146.6.150.235

A simple change in the subnet number (easily done in scripts) is all that is required. The rest of your test script will work. Makes the logging of the performance numbers very simple.

5.6.3 Special Use Devices

Special use devices such as printers, switches, gateways, *etc.* have been assigned the following range:

Table 2. Address Range of Special Use Devices

NIC 0	NIC 1
190-199	180-189

5.7 Design Constraints

The architecture chapter ended with tables of requirements. This section will repeat those constraints plus show how they were implemented in NeaSEL. In all cases, NeaSEL equaled or surpassed the requirement.

5.7.1 Infrastructure

		NeaSEL
Systems per compute cluster, x :	$3 \leq x \leq 100$	6
Systems available for distributed comp., d	$25 \leq d \leq 1000$	30
Number of cluster islands, y :	$3 \leq y \leq 10$	5
Servers per island, s :	$1 \leq s \leq 3$	$1 \leq s \leq 2$
Number of manufacturers represented, m :	$m \geq 5$	5
Total number of machines, tot :	$tot = x * y$	$30 = 6 * 5$

5.7.2 System Unit Properties

		NeaSEL
Adapter slots per machine, n :	$n \geq 3$	$n \geq 3$
Memory, mem :	$mem \geq 64 \text{ MB}$	$mem \geq 64\text{MB}$
Disk space, $dspace$:	$dspace \geq 2.0 \text{ GB}$	$dspace \geq 2.0$

5.7.3 Network & NIC Constraints

		NeaSEL
Network top. to be represented, <i>nt</i> : (low, medium, and high speed)	$nt \geq 3$	3
All ntwk top., <i>nt_i</i> , are independent	$\forall nt_i, nt_i \cap nt_j = 0$	True
Average number of NICs per host, <i>nic</i> :	$3 \leq nic \leq 5$	3
IP addresses required, <i>addr</i> :	$addr = nt * nic * tot$	360
Number of subnets required, <i>subnet</i> :	$subnet = nt * nic$	4
Constraints on subnets, not allowed:	all 0's or all 1's	True
Private network not allowed	No addresses in range 10.0.0.0 172.16.0.0 192.168.0.0	True

5.7.4 Host Naming Convention

NeaSEL		
Number of char. per fully qualified name	<i>numchar</i> ≤ 255	True
Number of char. per IP hostname, <i>hn</i> :	<i>hn</i> ≤ 63	True
Number of char. per second level name	<i>sname</i> ≤ 12	6

Chapter 6: Research Topics for NeaSEL

It is given that individual research topics can be done in individual specialized laboratories. The power of the proposed methodology is that it creates a general purpose, multi-use, combined teaching and research, integrated network and software engineering laboratory at the systems level. The methodology will greatly facilitate a wide range of research topics in a single facility.. This statement must be validated to validate the methodology.

A list of topics was compiled for use in the validation procedure. These topics were largely taken from the original research umbrella and from other course work. They include the following:

- performance (for example, ATM vs. Ethernet)
- network viruses
- dynamic network load balancing
- distributed simulation
- distributed software
- CASE tools & methodologies
- system security issues
- network security issues.

Therefore, proof of concept and proof of implementation must include how these and other topics were investigated and what conclusion were drawn.

This chapter will supply the proof of concept and proof of implementation by expanding these general areas into sixteen specific topics, each with numerous

6.1 System Administration Class

6.1.1 Domain

NeaSEL is an ideal place to teach system administration. First, it is totally heterogeneous in system types, O/Ss, and network topologies (see Figure 2). In NeaSEL the student has to learn the principles of system administration and not just master one specific O/S.

Second, it is an ideal place to learn system administration because NeaSEL can easily be disconnected from the campus and still function. A student system administrator can experiment and not worry about typing errors (such as mistakenly typing the same IP address on two different systems). Furthermore, mistakes can be deliberately added to the exercise to teach network debug techniques.

Third, these students can study heterogeneous hardware & software effects, such as how portable are data files, portability of languages, libraries, *etc.*

Fourth, given NeaSEL's extensive back-up facilities, the laboratory can be returned to a basic state in a relatively short time.

Who would the audience be for this class? There are two possibilities:

1. A class in operating systems or administration. While the theory of operating systems can be learned, a necessary adjunct is: if the O/S cannot be administrated, then it is useless as an operating system.

subtopics. For each topic, there is a written description followed by a diagram that shows how the methodology facilitates the experiments. As already noted, this list is not meant to be all inclusive but only to show the breadth of capabilities inherent in the methodology.

It is worth repeating that the goal is show how all of these topics can be done in one facility. Certainly similar investigations can be carried out in specialized facilities but that is not the purpose of this methodology.

NeaSEL provides an ideal place for a proving ground.

- 2. Training Computer Center personnel. Bring the trainees into NeaSEL during off-hours and train them as a group. This is example of the school staff receiving benefit from a teaching facility on campus.**

6.1.2 Possible Studies

- 1. Since NeaSEL has both heterogeneous hardware and heterogeneous software, like industry, it provides a very good system administration experience.**
- 2. How do I move files between different systems? If conversions are required, what are the performance penalties?**
- 3. What languages and what libraries are truly portable?**
- 4. Security studies in how passwords are stored on the various systems.**
- 5. What security policies should an I/S shop implement in duration of passwords, file access authorizations, *etc.*?**
- 6. How does a system administrator check to see that password rules, file access authorization rules, and the like are being followed, particularly on heterogeneous system types?**
- 7. Problem determination: How do one determine {specific problem} on a network? One example of a specific problem is duplicate IP addresses. Many other problems can be injected into the exercises.**

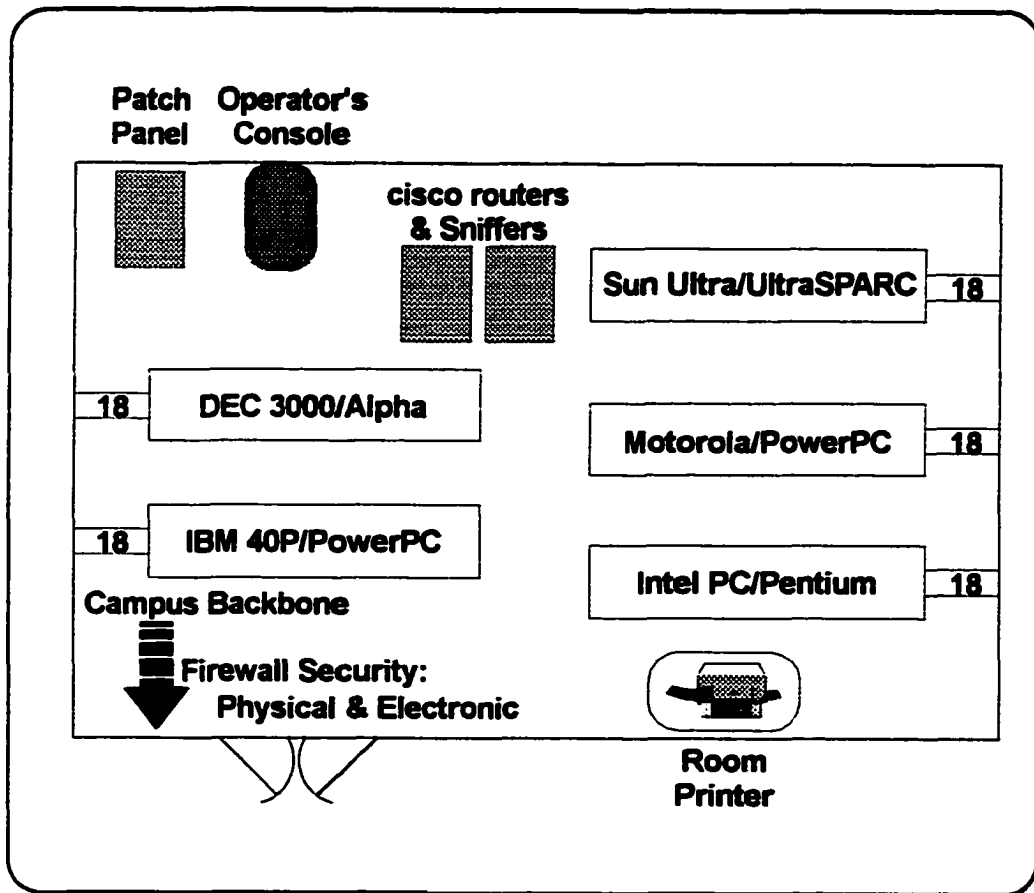


Figure 9. Composition of NeaSEL

6.2 Cluster Computing

6.2.1 Domain

Cluster computing is the use of similar workstations, physically close⁹, attached by a high-speed network, to create the aggregate performance of a supercomputer. NeaSEL offers the following:

- Each island has 6 similar workstations.
- Attached to a switched LAN with virtually zero time of flight.
- Workstations can be configured alike or dissimilar to see the effects of memory, swap space, *etc.*

6.2.2 Possible Studies

1. What problems are appropriate for cluster computing?
2. How to partition the problem to obtain supercomputer performance?
3. Can a tool be developed to automatically partition the problem?
4. What changes must be made to an O/S to enable cluster computing?
5. Where is the best place for swap space? Locally? Over the network?
What effect does network speed have on its location?

⁹ With fiber optics network, physically close can be up to 20 km.

- 6. Develop an algorithm for number of computers vs. speed-up by application type.**
- 7. What effect does network speed, packet size, *etc.* have on the cluster algorithms?**

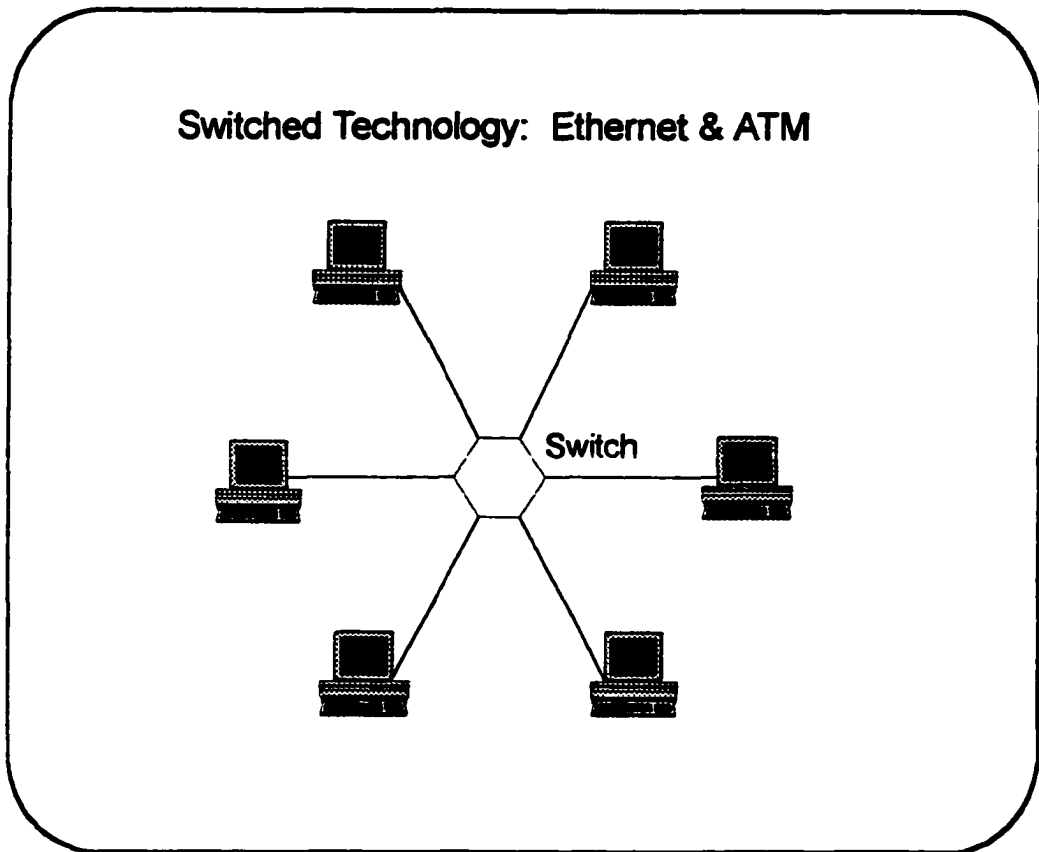


Figure 10. Cluster Computing

NeaSEL specific implementation:

This is the standard configuration so a cluster topology is by default.

6.3 Distributed Computing

6.3.1 Domain

Distributed computing is almost the reverse of cluster computing. In distributed computing, machines are independent and are widely separated so the study of distributed computing algorithms with practical networking becomes very difficult. In NeaSEL, intra-islands are homogeneous in systems and inter-islands are heterogeneous, so different configurations, homogeneous and heterogeneous, can be tested. To repeat part of the ease of use section, delay lines can be inserted anyplace into the network because of the patch panel. This means an experimenter can study real delays all in one room.

This latter point is significant and deserves to be expanded. The time delay between different machines, even those setting next to each other, can be varied from insignificant to satellite delays. This means a researcher does not have to worry about porting code to some remote machine (after first finding one) and then finding some kind soul to help run his tests, all the while hoping there is enough information to precisely characterize the network. NeaSEL is a great labor saver.

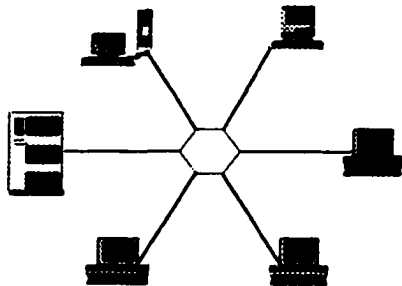
6.3.2 Possible Studies

1. How does one obtain *Sleeping MIPS*.¹⁰
2. What are the issues involved with Sleeping MIPS?
3. How does one obtain Sleeping MIPS with heterogeneous hardware and heterogeneous O/Ss?
4. If spreading a compute job over a distributed network, what happens if the network goes down? Only part of the network goes down?
5. How to divide the problem, particularly considering network delays. For example, what if the network has three sections with totally different delays: section 1 is enterprise; section 2, city-wide; and section 3, continental.
6. What are the issues involved with distributed, discrete simulation across a network?
7. Determine the bounds of simulation: memory, CPU, files, *etc.*
8. Ascertain the loading effects of heartbeats on the network.
9. How fast do I have to conclude when a machine went inoperative?
10. A corollary to above, how often do I have to checkpoint a running job?

¹⁰ A prime example is using idle PCs on everybody's desk at night. To illustrate the point, take a small business with a number of Pentium class machines. How many GFLOPS are setting idle at night?

Studies

Heterogeneous
Systems

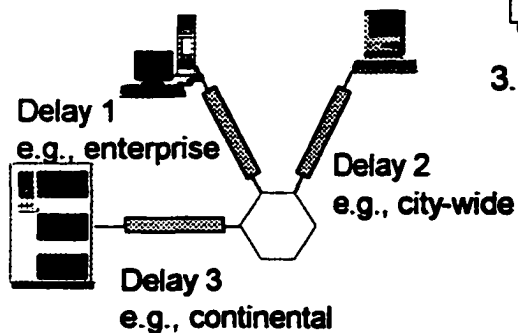


Homogeneous
Systems

1. Comparison of homogeneous and heterogeneous distributed computing.

2. How does one obtain *Sleeping MIPS* with heterogeneous hardware and O/Ss?

Add delay lines to any leg and...



3. Determine how to divide the problem taking into effect network delays.


 = Switch

Figure 11. Distributed Computing with Delays

6.4 Combination: Cluster & Distributed Computing

6.4.1 Domain

Originally, clusters were compute nodes attached to a distributed network of terminals.¹¹ This concept has since migrated to be a cluster of supercomputers on the network where an individual can buy or schedule time. As problems become more complex and companies try to lower their I/S cost by going to network computers, the supercomputer node with MIPS for sale will return. NeaSEL can simulate this scenario exactly:

1. Turn one island into a cluster (segment the Ethernet switch to have an island on its own subnet)
2. Put the other computers on delay lines or not.

6.4.2 Possible Studies

1. Combine all the studies from cluster and distributed computing.
2. How or what do I charge for? MIPS? Time?
3. What are the economic trade-offs between owning system units or buying MIPS?
4. Where should the cluster be located for the best performance?

¹¹ See the unpublished paper, Clusters, by the author, located in NeaSEL.

5. **What classes of problems is this topology suited?**
6. **One super cluster or numerous smaller clusters?**
7. **What is the cost of moving data across a network?**
8. **How is redundancy implemented for numerous clusters?**

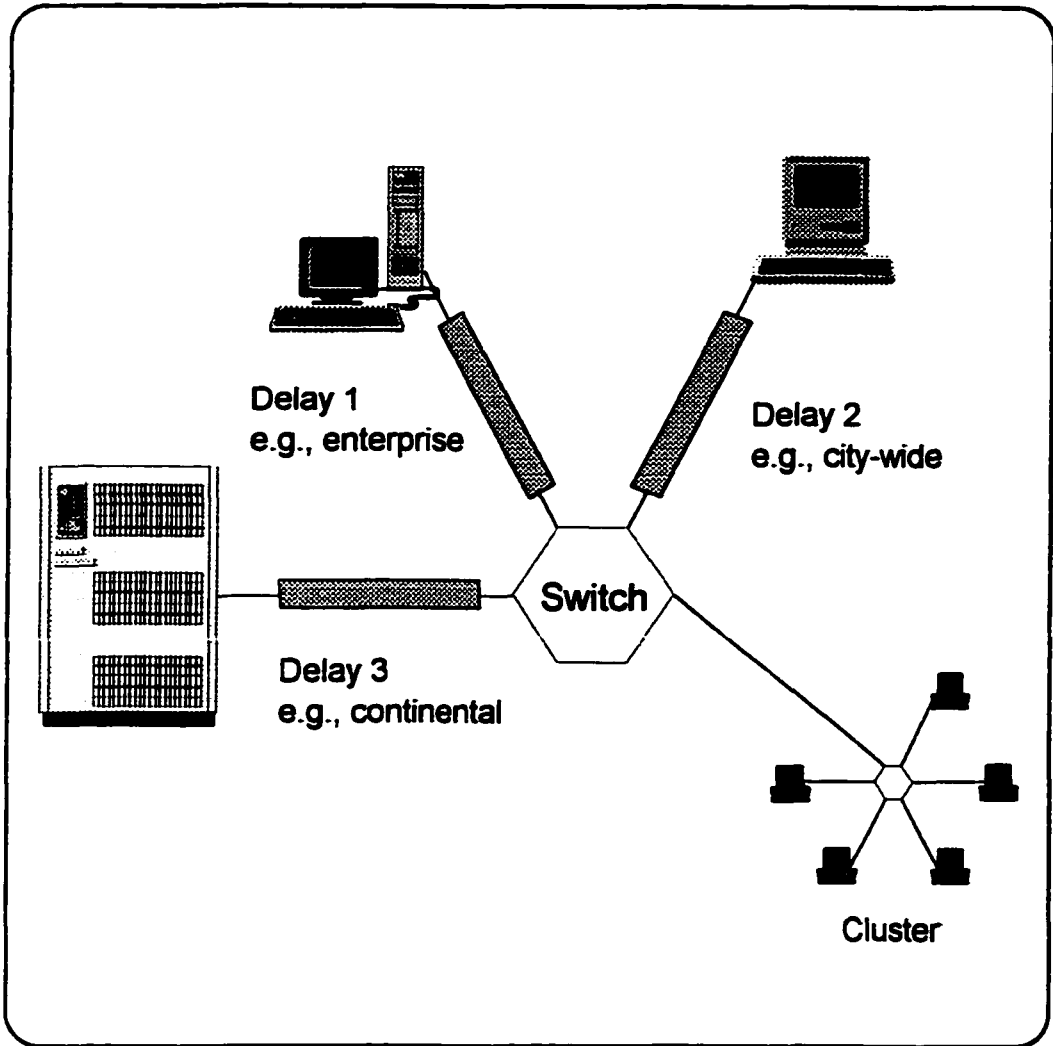


Figure 12. Combination of Cluster & Distributed Computing

NeaSEL specific implementation:

- Switches can be configured to isolate subnets.
- Sniffers & delay lines can be inserted into the patch panel.

6.5 Client/Server Computing

6.5.1 Domain

Client/Server computing is one of the hottest topics in the outside world now, even though there is no precise definition for the subject. For the benefit of this research, let's use a generic definition: client/server architecture is a computer system architecture in which clients request a service and a server provides that service. Writers normally apply the definition to a LAN, not a WAN, though this is not a real restriction.

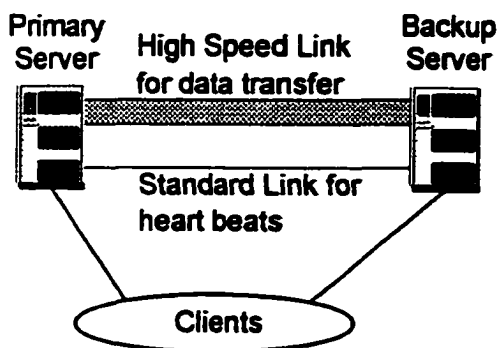
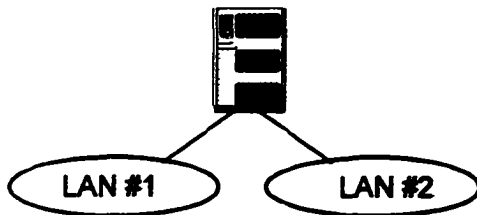
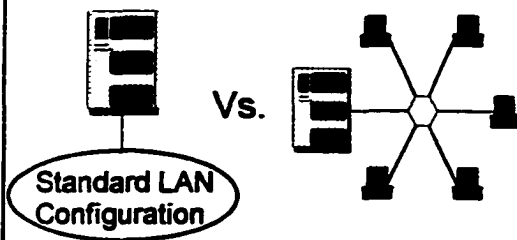
How does NeaSEL act in this role? First, each island has one machine more powerful than the others and this machine is configured as a server. Second, all NeaSEL machines have been pre-configured for multiple NICs on multiple topologies. Third, it is very easy to connect the server to a switch, hub, or a noisy network (with a sniffer). Fourth, using the sniffer in a non-intrusive role, one can see the network traffic for various protocols and data.

6.5.2 Possible Studies

1. Effect of different network topologies on the performance of client/server: switched, hub, standard.
2. Study the change caused by multiple, like NICs inside the server, from the server viewpoint, from the client's viewpoint.

3. Study the change caused by multiple, dissimilar NICs inside the server, from the server viewpoint, from the client's viewpoint.
4. How much traffic does a typical application cause: NIS, NFS, *etc.*?
5. When do I need to back-up a server?
6. What frequency of heartbeats is required?
7. Can I determine an algorithm that tells how many clients I can have per server per application type with what expected performance? Examples are video servers on demand, network computer boot code servers, and the like.
8. If the primary server dies, how does one dynamically change the address of the backup to be that of the primary server?

Studies



1. Comparison of standard client/server network versus a fully switched network.

2. Study of performance improvement caused by splitting clients apart.

3. Study of impact of multiple NICs inside a server.

4. Study algorithms for server redundancy, for example what is the frequency of heartbeats required?

5. How do you dynamically change the address of the backup to be that of the primary server?

Figure 13. Client/Server Computing

6.6 Viruses

6.6.1 Domain

A computer **virus** is a malicious computer program that alters files or system configurations and may spread its destructiveness by copying itself to other machines. Normally, malevolent programs are divided into three types: viruses, worms, and Trojan Horses. *Viruses* require an action of some sort to activate them; therefore, they attach themselves to executable files. A *worm* can run by itself and can spawn a fully working version of itself in other machines (so called a worm because they move across a network without leaving detectable signs). A *Trojan Horse* is a program that is designed to disguise itself as something harmless, waiting for the right moment to do its deed (an excellent, sneaky place to put a Trojan Horse is inside an anti-virus checking program).

Businesses are concerned about viruses because they can be very costly. The foremost goal is to prevent viruses from entering your networks or systems. If this fails, then the goal must be speedy isolation and removal. Obviously, virus experimentation cannot be done on a production network because the virus might escape. This means virus studies can only be done in very specialized locations. Since NeaSEL was designed to be fully self-contained with back-ups and network capabilities, the room can be totally disconnected from the campus network and still be fully functional. Viruses can be studied and then complete O/Ss can be reloaded--all safely and not impacting the campus network.

6.6.2 Possible Studies

- 1. How to determine a network virus has infected your network?**
- 2. How to determine a system virus has infected your system?**
- 3. How to prevent a network virus from infecting your network?**
- 4. How do viruses propagate on a network?**
- 5. Since systems can be isolated, virus infection by system type can be studied.**
- 6. Can one virus propagate across different network types? How? Should one, non-trusted, node be allowed to have different network topologies?**

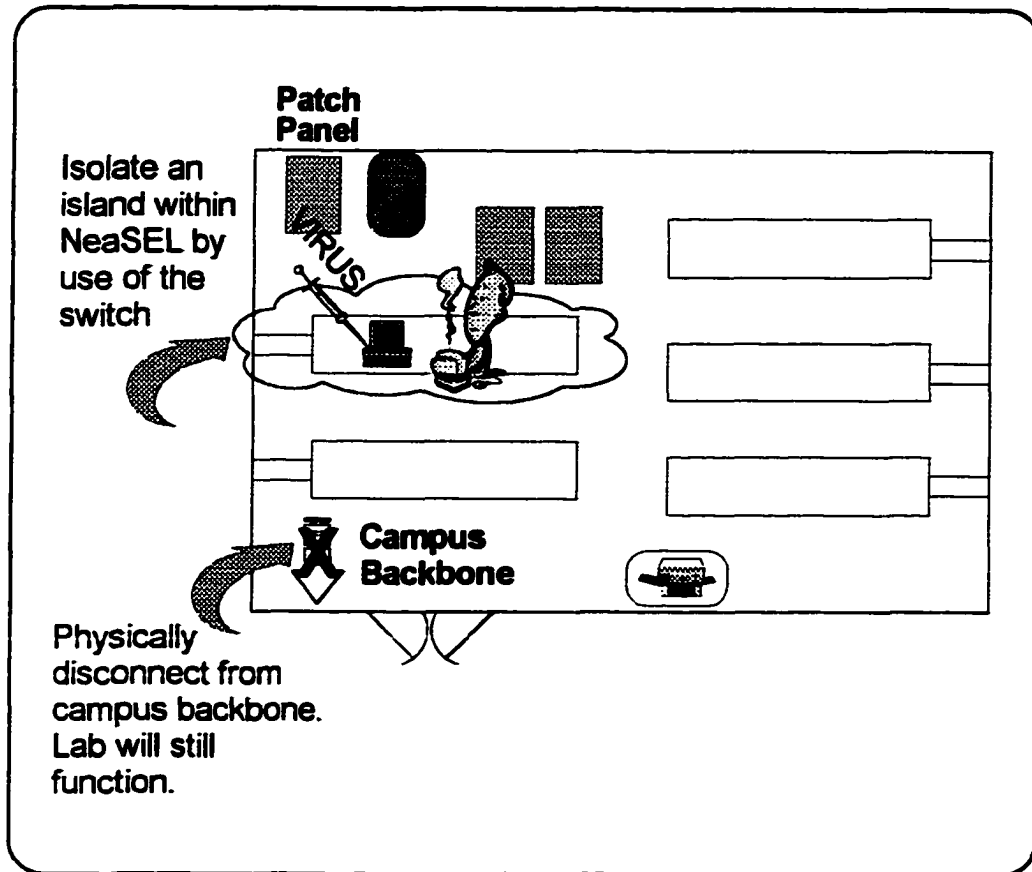


Figure 14. Virus Infection

NeaSEL specific implementation:

An island or islands can be isolated either physically or electrically from each other and from the campus backbone. A malicious program can be deliberately injected into a machine to determine its behavior. With this *forced isolation* other experiments can be run safely in NeaSEL at the same time.

6.7 Network Security Issues

6.7.1 Domain

With the whole world becoming networked, network security has become important to businesses. What does NeaSEL have to offer? NeaSEL has switches that have firewall capabilities. All system units have multiple NIC capability, so each machine can be configured as a router. NeaSEL recently received three racks of cisco routers: hardware vs. software firewall capabilities can be studied.

An explanation of firewalls would stray too far afield from the subject of this dissertation.¹² Rather than leaving a reader in the dark though, a typical proxy services firewall is shown in the figure. Since the systems in NeaSEL are all multi-homed and most of the O/S allow routing between NICs, then a simple proxy services firewall can be studied. This arrangement would be a good place for a student to begin. The student would have to learn the protocols, some services, and routing. The host is already configured to be multi-homed. This means the student can begin with firewalls rather than creating networks.

6.7.2 Possible Studies or Classes:

1. How do firewalls work?

¹² See Firewalls, A Tutorial. by the author. located in NeaSEL.

2. **What is the performance impacts of firewalls?**
3. **Hardware vs. software firewalls: performance, ease of administration, etc.?**
4. **How do trusted hosts, remote logins, etc. work?**
5. **How do hackers penetrate?**
6. **How does one log hacking attempts?**
7. **Physical vs. electronic vs. encrypted security.**
8. **What applications are insecure?**
9. **Can network sniffers determine passwords, credit card numbers, etc.?**
10. **How safe is your credit number on the network?**
11. **Create more secure algorithms for data transmission.**
12. **How can the system administrator determine false traffic?**
13. **How does one configure a firewall?**

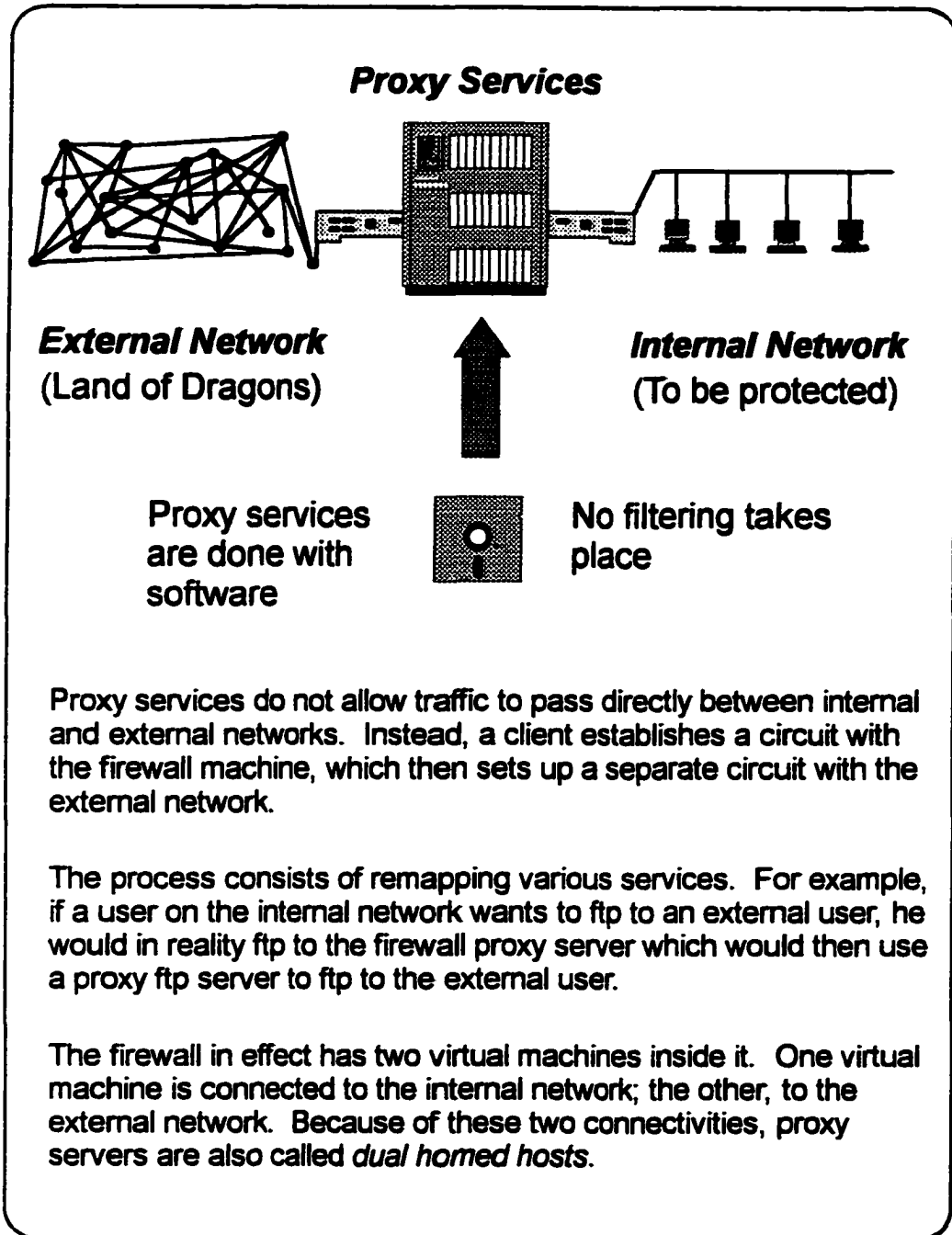


Figure 15. Typical Firewall

6.8 Network Performance

6.8.1 Domain

Network performance is probably the blackest art of the arcane world of networking. The reason being there is no precise definition of network performance. Is it number of packets, bits, end to end flow, what? The complexity grows geometrically when one tries to compare performance between different network topologies which may be running different protocol stacks.

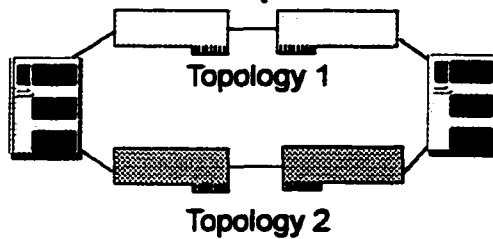
Since there is no fixed definition of network performance or its measurement, performance measurements can run the gambit from ftp performance between different system manufacturers to a throughput comparison between NICs from different vendors. Figure 23 shows the latter case.

6.8.2 Possible Studies or Classes:

1. Teach how to make performance measurements.
2. What does network performance mean?
3. What parameters are important to measure? In general? For a specific topology?
4. Measure file transfers, messages, *etc.* between different systems using different topologies.
5. Repeat item #4 except switched vs. non-switched topologies.

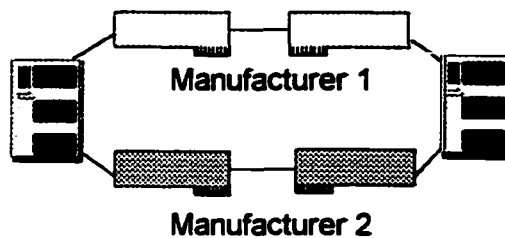
Studies

Different network topologies, i.e., different speeds



1. What is the network performance for each topology? For file transfers, for messages?
2. Redo #1 for homogeneous systems, for heterogeneous systems.

Same configuration but different manufacturers



3. Which manufacturer yields the fastest throughput?

Figure 16. Network Performance

NeaSEL specific implementation:

- All tests can be run simultaneously.
- All systems are fully configured so at most you might have to load a device driver.
- NeaSEL was designed for automated script usage.

6.9 Network Parameters

6.9.1 Domain

Network parameters, such as loading, time of flight, corruption, *etc.*, are very difficult to measure unless an researcher has a dedicated network laboratory, with patch panels, and with sniffers. This perfectly describes NeaSEL.

Delay lines can be installed in-line with the network (or even use systems) to increase the time of flight between systems. This means realistic timings can be injected into any study.

Another facet of this study is how do one monitor the network? This opens up the world of SNMP, RMON, and MIBs. This can be a researchers laboratory, a standard class, or a network administrators class. In the latter case, new monitor programs can be tested in NeaSEL before putting them into production.

6.9.2 Possible Studies or Classes:

1. Sniffers can be put into the network to determine loading effects, noise effects, or the like on any type of study (adds realism to the results).

2. **What effect does time of flight have on any study? Across the country? Within a city? To a satellite (yes, a satellite, one just increases the time of flight).**
3. **What effects do total or partial network outages have on the cluster, distributed or client/server algorithms?**
4. **How soon can a network administrator determine a network outage?**
5. **Are certain topologies better at partitioning a network outage?**
6. **What network factors should one monitor?**

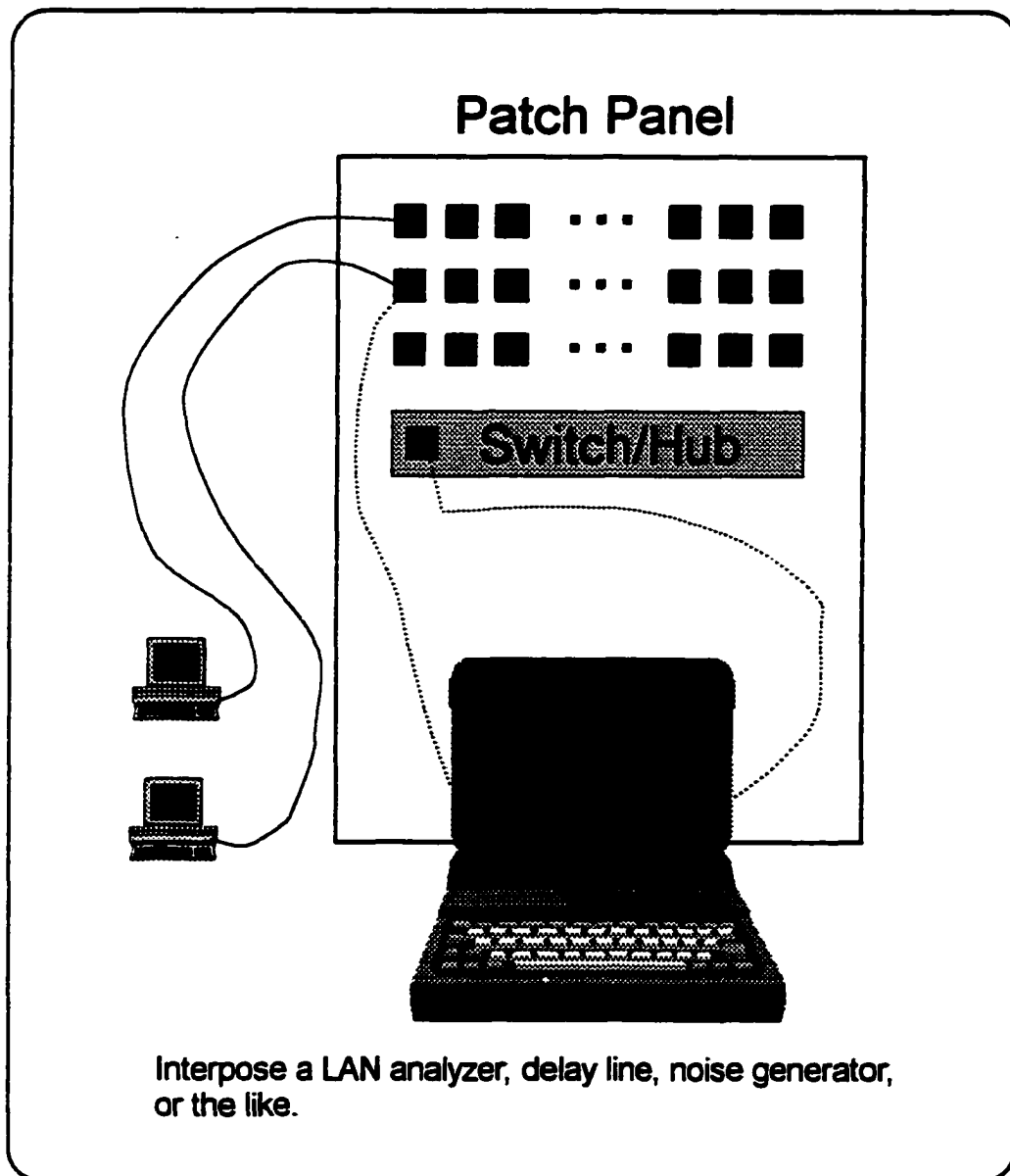


Figure 17. Networks Parameters with a Sniffer

6.10 Ethernet Switch vs. Ethernet Hub

6.10.1 Domain

Given the number of machines connected to Ethernet topologies, Ethernet will be with us forever. The economic questions associated with switches versus hubs will also be with us forever. Switches have more capability, are more difficult to manage, and are more expensive. Hubs are exactly the opposite.

6.10.2 Possible Studies

1. For a large number of system units, determine the algorithm for calculating the number of and the placement of switches and hubs for the best throughput given economic parameters.
2. For small topologies, repeat the exercise.
3. Develop an algorithm to calculate the number of systems per hub for a certain throughput.

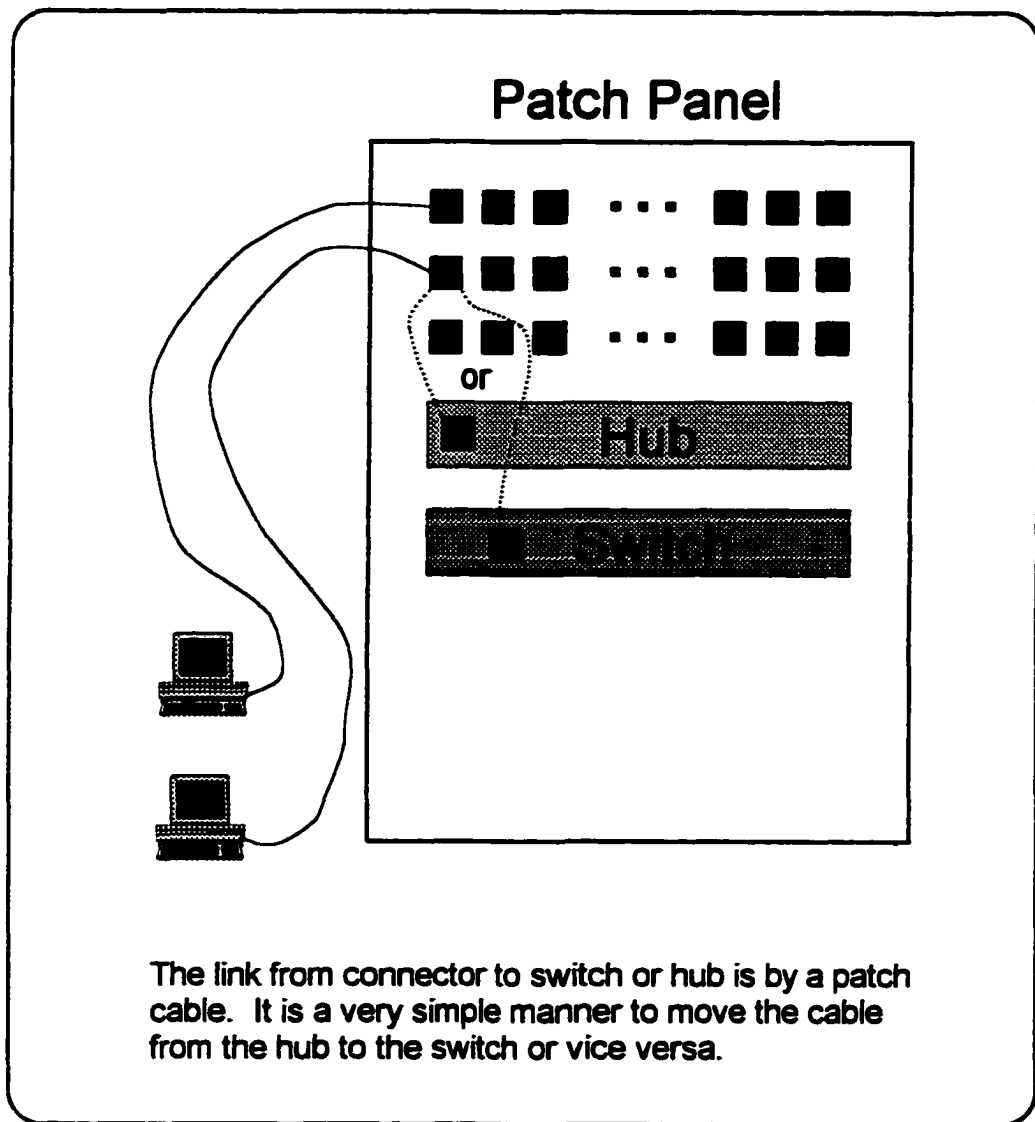


Figure 18. Switch versus Hub

6.11 Data Striping across Networks

6.11.1 Domain

Historically, people have studied data striping within system units, *i.e.*, RAID. Some groups are now studying data striping across networks. To study this, one needs multiple NICs per machine and an addressing structure that facilitates this. NeaSEL fits these requirements perfectly.

The following diagram shows data striping across a network in which multiple NICs are doing the striping. An interesting variation on this study is to let the protocol stack AND the NICs do the striping simultaneously. This added feature means the students must study device drivers and protocol stacks.

6.11.2 Possible Studies

1. How do I data stripe across networks? With NICs? In the protocol stack? Both simultaneously? Benefits of each?
2. What effects do different networks of different speeds have on network data striping?
3. Does it make sense to data stripe across different topologies and different adapter types simultaneously?

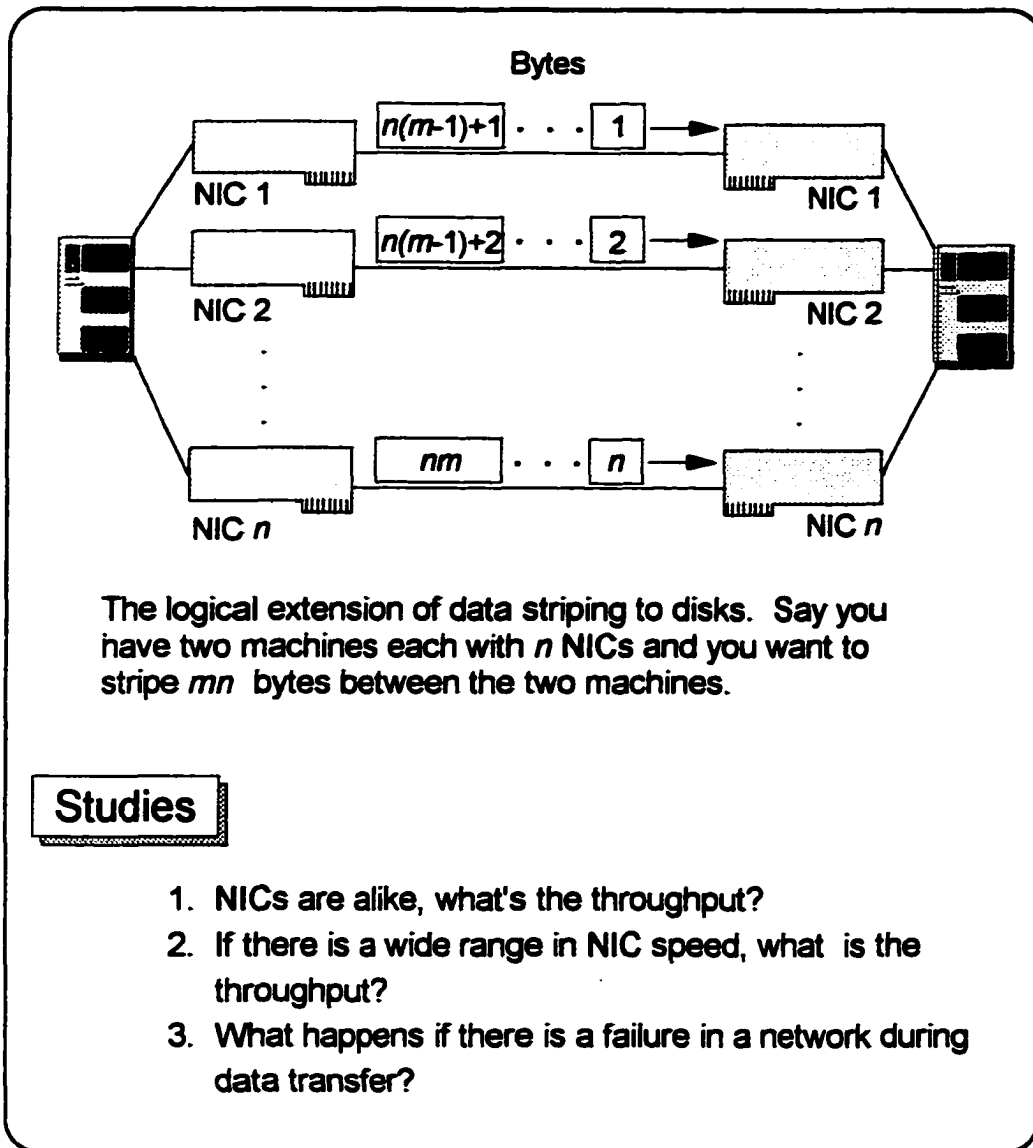


Figure 19. Network Striping

NeaSEL specific implementation:

NeaSEL has multi-homed hosts with a simple naming & addressing convention for each NIC, so sending or receiving to a specific adapter in a specific host is trivial.

6.12 NIC Performance with Downloaded Protocol

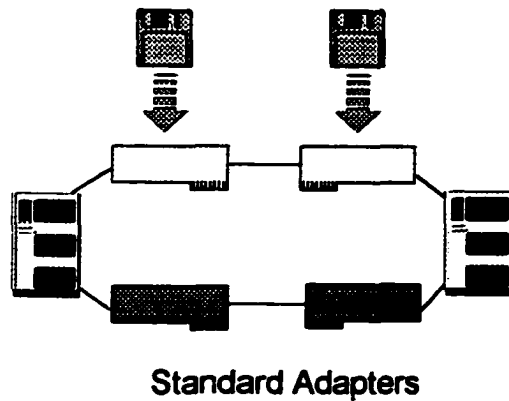
6.12.1 Domain

Some of the newer NICs permit downloading of microcode. A possible consequence of this is it can now be possible to download parts of or the complete communications protocol stack to the NIC. This means communication protocol operations can be run in hardware on the NIC, which is faster running them in software in the O/S. This seemingly straightforward statement is in fact very complex in implementation. The researcher/student must understand protocol stacks, how to put the hooks into the O/S to allow this, and how hardware adapters work.

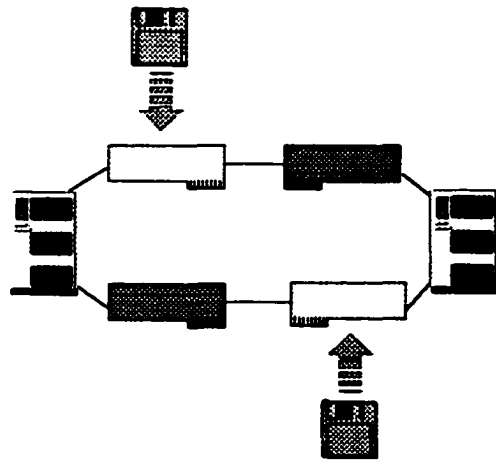
6.12.2 Possible Studies

1. Developing & studying protocols downloaded onto the NICs.
2. When should the protocol be downloaded? Boot time? Time of usage?
3. Does one have to download all the protocol stack or just parts to achieve a speed up? If it is just parts, which parts?
4. Can a fast NIC with downloaded protocols overrun other adapters in the same network? This could slow the network down because data has to be constantly retransmitted.

Studies



1. What is the network performance difference transmitting between two adapters with downloaded protocols and between two standard adapters (ones without downloaded protocols)?



2. What is the network performance of transmitting from an adapter with downloaded protocols to one without downloaded protocols? Are there overrun problems?
3. Repeat the same experiment except receiving instead of transmitting.

Figure 20. Downloading Protocols to Adapters

6.13 Protocol Analysis & Lightweight Protocols

6.13.1 Domain

Most of the network protocols that we use today were developed in an era when networks were unreliable, noisy, and difficult to maintain. For these reasons, the protocols had numerous locations where the data was checked for reliable transmission. For example, TCP/IP has numerous places where a checksum is being performed (the following diagram shows just some of the places). Overhead occurs each time the checksum is done. In the old days, this was necessary to guarantee reliable transmission.

Today, networks are much more reliable and through various coding techniques have error correction built-in at the hardware level. Errors are corrected before the data leaves the adapter for the application. With these capabilities, why should we continue with the old protocols? This statement has led to a tremendous amount of research being done into lightweight protocols.

6.13.2 Possible Studies

1. Can I remove the checksum calculations out of TCP/IP? Where? What are the consequences?
2. Should a user be allowed to do this? Should a NIC be allowed to do this?

- 3. Are there better protocols that will mimic TCP/IP and yet be faster?**
- 4. General topic: developing new or modifying old protocols to be lighter weight.**
- 5. Working with tuning parameters such as window size to determine the best throughput.**

Format of a TCP Packet

Source Port			Destination Port		
Sequence Number					
Acknowledgment Number					
Offset	Reserved	Code		Window	
CHECKSUM			Urgent Pointer		
Options					Padding
Data					
...					

Format of a UDP Packet

Source Port			Destination Port		
Length			UDP CHECKSUM		
Data					
...					

Format of an IP Address

Version	Length	Type of Service	Total Length		
Identification			Flags	Fragment Offset	
Time		Proto	HEADER CHECKSUM		
Source IP Address					
Destination IP Address					
Options					Padding
Data					
...					

Figure 21. Locations of Checksums

6.14 Study of Dynamic Network Load Balancing

6.14.1 Domain

Today, there are numerous network topologies to choose from, from the highest speed (FCS, HiPPI) to medium speed (FDDI, 100 Mbit Ethernet) to low speed (10 Mbit Ethernet, token ring). Most machines can be configured for any one or multiple of these adapters. The question is which pipe should be used to move data?

A quick answer to this question has always been, the larger the amount of data, the faster the pipe used. But, is this really correct? For example, a lightly loaded 10 Mbit Ethernet on a switched topology might move data faster than a heavily loaded switched HiPPI network. Today, network administrators determine which pipe to use for which data and then hard code that into the system configuration (static determination). A research topic would be to determine if large data should always go the fastest pipe. If that's not true then determine/create an algorithm which would dynamically pick the pipe that would yield the fastest throughput based on packet size, application or the like.¹³

6.14.2 Possible Studies

1. Develop an algorithm. What parameters are important?

¹³ As an interesting side note, this was the author's original dissertation topic. The first realization was that I could not do this kind of development without a sophisticated yet very flexible laboratory. This was the genesis of NeaSEL. NeaSEL grew to be such a powerful tool that the methodology and implementation became the author's dissertation topic. The original dissertation topic is still available!

- 2. What range of network speeds does it make sense to apply such an algorithm?**
- 3. What applications are appropriate for this algorithm?**
- 4. Generate a cost trade-off analysis of improvement of data throughput versus a new network topology.**
- 5. What network protocols are appropriate for this algorithm?**
- 6. If there is such an algorithm, could buffered adapters create the same benefit?**

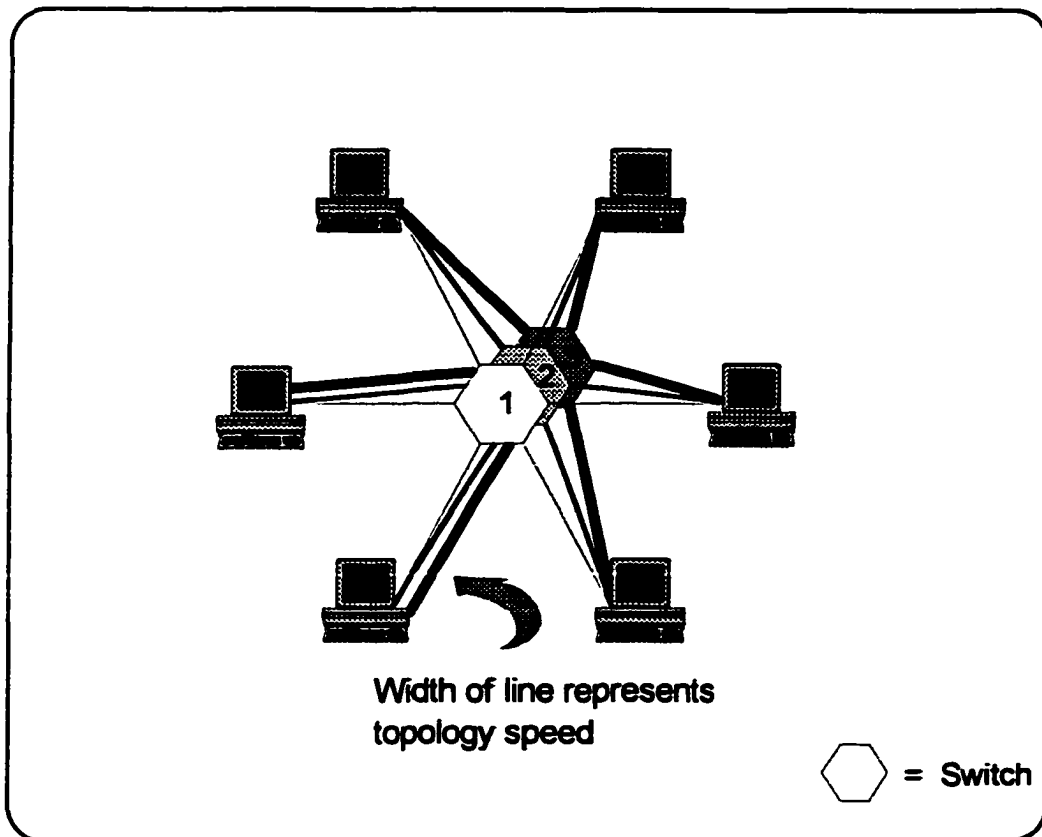


Figure 22. Dynamic Load Balancing

This figure may be somewhat confusing. Each system is multi-homed, with three NICs of different network speeds (e.g., Ethernet, FDDI, and ATM). Here we are assuming all switched topologies.

The goal is for the application to determine **dynamically** which pipe to use for which packet, data load, or the like.

6.15 SPECmark Analysis

6.15.1 Domain

SPEC is a consortium of companies that publish a set of standardized test suites that manufacturers use to determine the performance of their systems. Since SPECmarks equate to bragging rights in advertising literature, some manufacturers go to great lengths to ensure their machines will achieve the highest number. One such example is the adding of special options to compilers just for SPECmark test cases. Since NeaSEL has heterogeneous systems, SPECmark test cases can be run and analyses can be made.

Running of SPECmarks in a non-industrial environment can be a valuable learning experience. For example, a compiler class could have as a laboratory project the duplication of the published results for a certain system.¹⁴ One goal would be to answer the question does SPECmarks mean anything in the real world?

6.15.2 Possible Studies

1. How are SPECmarks measured?
2. How do SPECmarks relate to production work?

¹⁴ This is easier said than done. Getting all the special compiler options tweaked is a learning experience in itself.

3. **How does the equipment required for SPECmark relate to real world equipment, *i.e.*, can any user really afford all the memory, file space, *etc.* necessary to maximize the results?**
4. **How does one performance tune for a specific application? For a group of applications?**
5. **Pick a specific vendor and try to duplicate their results? Are there special compiler options?**

<i>Company</i>	<i>System</i>	<i>Base</i>	<i>CPUs</i>	<i>Result</i>
DEC	AlphaServer 1000A 4/266	5.68	1	6.03
	AlphaStation 200 4/100	2.48	1	2.79
HP	HP 9000 Model C100	6.20	1	6.20
	HP 9000 Model D270	14.0	1	15.0
IBM	RISC System/6000 43P	3.08	1	3.38
	RS/6000 43P-140	5.07	1	5.23
Intel	XXpress Deskside (1000\120)	2.24	1	2.81
	Alder System (150 MHz)	4.76	1	5.42
Motorola	MVME2604-2161	8.09	1	8.92
	RISC PC 604	3.32	1	3.38
Sun	Sun Ultra 1 Creator 140E	7.85	1	8.38
	Ultra Enterprise 4001/1	11.5	1	11.7

Figure 23. Sample of SPECfp95 Results¹⁶

Excerpted from results table shown on their Web page. While the SPEC test code is available, how each vendor specifically ran it is not. Trying to duplicate vendor runs will train one to be a tuning expert very fast.

¹⁵ Copyright (c) 1996, 1997 Standard Performance Evaluation Corporation.
 URL: <http://www.specbench.org/osg/cpu95/results/cfp95.html>

6.16 Compression Techniques

6.16.1 Domain

As noted in the Introduction, the new trend in computing is how do I get information to a stand-alone system so a user can process the information to the user's benefit? That is we are sending information to do the analysis. Sometimes this information may be very large, for example, access to a library or topological map information. This trend has caused a renewed interest in data compression techniques.

One of the major problems is how does one test the effectiveness of the technique? Most authors do a software analysis or a software simulation. Seldom do they test across a network, especially multiple topologies.

The problem with this technique is that the algorithm may generate data that could cause inefficiencies or even cause failures in the transfers. For example, the compression algorithm may accidentally compress network routing information which would cause a network failure. The only way to verify the compression algorithm is to test it across real networks.

6.16.2 Possible Studies

1. Does the compression algorithm work across all topologies?
2. Does the compression algorithm work on all O/Ss?

6.16 Summary

This section did validate the that the derived methodology could create a general purpose, multi-use, combined teaching and research, integrated network and software engineering laboratory at the systems level. Sixteen widely varying topics with numerous subtopics could be done in a laboratory created by the methodology and in many cases simultaneously because of the structure of the methodology. This validates in concept that the methodology will create a powerful facility.

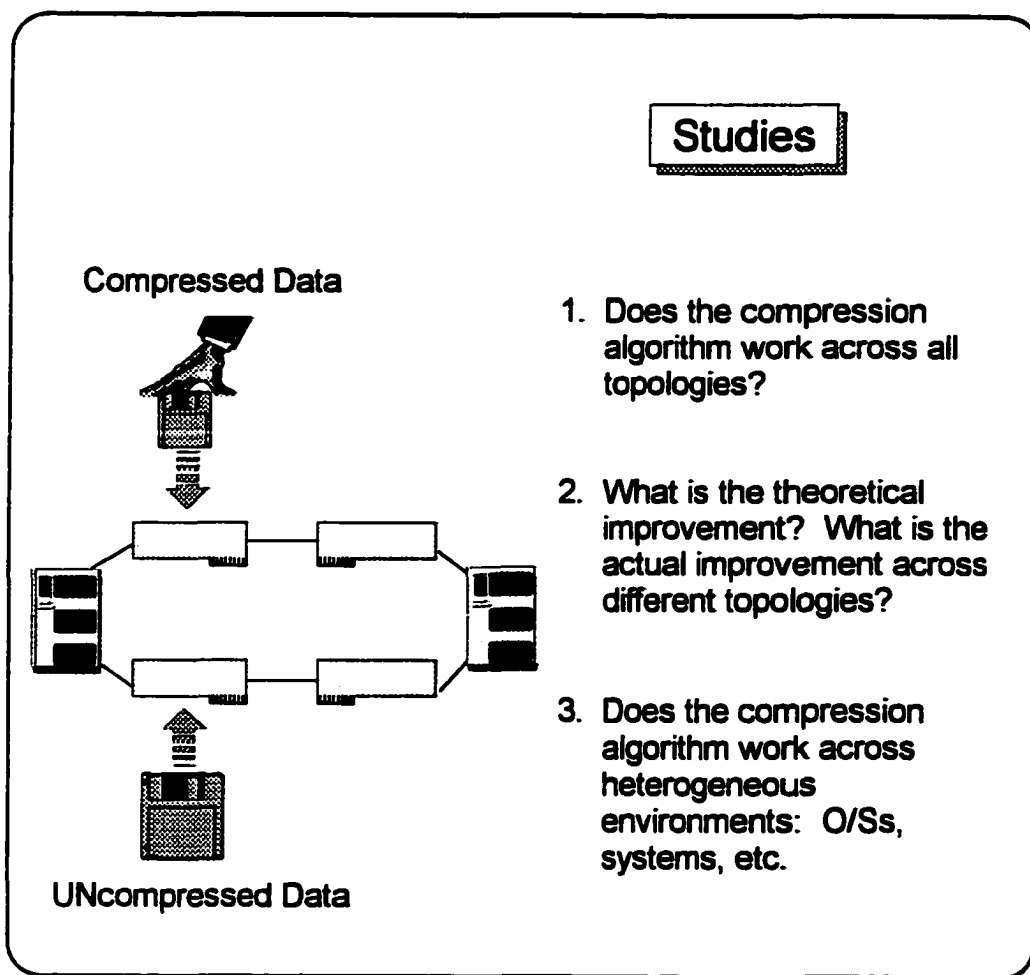


Figure 24. Compression Algorithm Testing

Chapter 7: Validation of the Methodology

The validation process consists of proving an hypothesis: if x then y . Since this dissertation is both a derivation of a methodology and a physical implementation, then there will be a series of hypotheses that must be proved. Consider one specific example, the hypothesis is, "if the methodology is complete, then any third party could build a multi-use, integrated software and network engineering laboratory."

The approach in this chapter will be to present a hypothesis and then prove it. Where do the hypotheses come from? Throughout the methodology creation and physical implementation, hypotheses or goals were either explicitly stated or implied. This chapter collected all these hypotheses and goals in one location and then explicitly answers each.

7.1 Third Party

7.1.1 Hypothesis

If the methodology is complete, then any third party should be able to build a multi-use, fully integrated network and software engineering laboratory.

7.1.2 Validation

The validation is NeaSEL itself, a large scale implementation of the methodology at the University of Austin at Texas (detailed in Chapter 5: Case Study, NeaSEL). Therefore, this is validated.

7.2 Capabilities

7.2.1 Hypothesis

The methodology generates a general purpose, multi-use, combined teaching and research, integrated network and software engineering laboratory at the systems level. A more general way of stating this is if we have a methodology that does x , proving y .

7.2.2 Validation

This hypothesis is the most ambitious one for the methodology because of the wide ranging capabilities. Due to the amount of effort involved, a complete chapter, Chapter 6: Research Topics for NeaSEL, was devoted to proving this hypothesis. Sixteen widely varying topics were presented and for each it was shown how the methodology and its physical implementation, NeaSEL, facilitated the research or teaching.

As a further validation and because of the complexity of this particular hypothesis, the description will be disassembled into components and each will be analyzed separately.

“General purpose, multi-use” means that one facility can do many topics simultaneously. It was shown from historical searches that there were and are many specialized laboratories that can do a few topics extremely well. What was missing was a general purpose, multi-use facility. This methodology allows this capability because many of topics can be done simultaneously in the same facility without impacting each other. Stated another way in practical networking terms, the switches and patch panel allow an experimenter to isolate all or part of the laboratory; and because of the design of the methodology, the components can still function and run experiments. By definition of the word isolation, multiple experiments can be run simultaneously.

“Combined teaching and research” is a powerful statement because it allows the latest research findings to be moved rapidly into the curriculum. From Chapter 6, many of the topics described can be either teaching or research or a combination. A specific example is the theory of firewalls. It can be an excellent research topic or can be a component of network security course.

“Integrated network and software engineering laboratory” means that both networking and software engineering can perform together. Again, Chapter 6 gives examples of where both were fully integrated.

“Teaching” and “research” will be discussed separately in a later section.

With this disassembly into components, this statement has been validated as a whole and as individual components.

7.3 Automated Tests

7.3.1 Hypothesis

The methodology allows all levels of students to run automated tests with the use of scripts or under program control.

7.3.2 Validation

The automated test criterion has been realized through the precise host naming convention used in NeaSEL. As noted in a previous chapter, a simple change in one part of the host name can change topology, adapter, or system unit. This capability was used in the test scripts and greatly simplified the running of test cases.

7.4 Ease of Use

7.4.1 Hypothesis

An important design point for the methodology was ease of use. The hypothesis is that the methodology helps to create a facility that is easy to use and therefore saves experimenters' time.

7.4.2 Validation

The methodology and hence its physical implementation, NeaSEL, was constructed to remove the drudgery from network and software experimentation. To validate this, a standard performance measurement was used to validate ease of use. The test, a comparison of file transfers between systems over different network topologies, is a typical experiment in a network class. Specifically, the test was the comparison of file transfers between the topologies of 10 Mbit Ethernet and 155 Mbit ATM. Compare the process before NeaSEL to that after NeaSEL.

Before NeaSEL, a student had to:

1. Find machines not connected to the school backbone.
2. Make sure the machines would work with ATM & Ethernet.
3. Find how to determine how the machines are configured.
4. Determine how the machines are configured.
5. Configure the machines to be multi-homed, *i.e.*, Ethernet & ATM.
6. Find and assign multiple addresses to each machine (or be forced use the names and addresses already assigned to them).
7. Find an ATM switch with unused ports.
8. Pull cable.
9. Debug all of the above.
10. Write very specific scripts to run tests.
11. Run the tests.
12. Take the set-up apart.

Eleven of the 12 steps were drudgery work that has nothing to do with the tests, a very laborious process to run what should be a straightforward test. In many ways, the test had been lost in the details of the laboratory set-up.

Compare this to NeaSEL. Five steps have already been done:

1. *Done:* NeaSEL was designed to be removed easily from the campus backbone for testing.
2. *Done:* machines are already configured to be multi-homed.
3. *Done:* cables and switches already installed and debugged.
4. *Done:* manuals clearly show how to determine what's inside.
5. *Done:* Host naming conventions and host addressing conventions are already defined so the experimenter can easily run automated test scripts and perform automatic logging of data.

The drudgery work has already been done. The student might have to:

1. Install an adapter in a machine (if not already there).
2. Physically hook up wires to a patch panel that are clearly defined and color coded, *i.e.*, just match numbers and color code.
3. Run the test.
4. Remove the adapter and patch cables, if required.

A reduction of twelve steps to at most four. The test becomes the primary work effort, with the remaining non-test work minor.

An important corollary to this process is other tests can be run immediately before or immediately after this test without a schedule impact because NeaSEL has been preconfigured for this capability. In fact, by using the isolation properties of the switches, different tests can be run simultaneously.

7.5 Research Facility

7.5.1 Hypothesis

If this is a research facility, then the methodology and facility should have facilitated the awarding of degrees or the publication of papers. Another way of stating this is there should have been features inherent in the methodology that was in the critical path of the awarding of degrees and publication of papers.¹⁶

7.5.2 Validation

NeaSEL has been operation since March 1996. Since then one Ph.D. [62] and two Master's [63, 64] have been awarded on research done in the laboratory. There have been articles published, senior laboratory projects done, and a major new digital simulation effort started.

Rather than doing a litany of work, let's focus on one thesis and the digital simulation effort. The thesis by D. Kaprzak [64] was about using COBRA for video on demand. The effort required two network topologies, a traffic free network for performance measurements, and a way to run numerous automated test scripts. At the University of Texas, the only laboratory meeting these requirements is NeaSEL.

Could the work have been in one or more facilities. Yes, but this movement would have caused extensive delays because only NeaSEL has all the

¹⁶ Obviously, this must ignore degrees awarded on the methodology itself.

capability. Also, because NeaSEL can be easily isolated, the measurements were precise and were repeatable and reproducible.

The digital simulation effort is proposing using Java for digital simulation. In published articles [67, 68], specific mention is made of NeaSEL and its value add for this effort. This effort required a controllable heterogeneous environment for experimentation. At the University of Texas, only NeaSEL is a fully controllable heterogeneous environment.

From these brief descriptions it is obvious that NeaSEL is a powerful addition for research.; therefore, the research requirement is validated.

7.6 Teaching Facility

7.6.1 Hypothesis

The methodology, when implemented in a laboratory such as NeaSEL, can facilitate teaching and course development.

7.6.2 Validation

In the Spring term of 1998, a course entitled Network Engineering, Unique Number 14630, will be taught by Dr. Bard. This class will be exclusively taught in NeaSEL using much of the methodology developed in this dissertation. The class was so popular at registration that it had to be immediately expanded into two sections and even then students were turned away.

This class could only be taught in a facility like NeaSEL that was developed from the methodology. Key requirements that could only be met by the methodology were (1) both isolation or connection to the campus network, (2) precise naming convention to be able to track data, (3) a facility properly wired that would allow the insertion of test equipment into the network, and (4) island concept to be able to isolate test cases. Even though this course has become very popular, a specialized facility solely for this course could not be cost justified. It had to be taught in a general purpose, multi-use facility and the only facility in the world that meets that requirement is NeaSEL which was designed from this methodology.

7.7 Typical Experiments

7.7.1 Hypothesis

Can typical network experiments be run in NeaSEL and what is it about the methodology that facilitates or is a requirement for the experiments?

7.7.2 File Transfers

Typical experiments in a protocol class or network class are to measure the file transfer rate between different system types, with different O/Ss, and different network topologies using standard protocols such as ftp or specialized code such as writing to sockets.

These tests was chosen as a validation test for NeaSEL for several reasons:

1. Validates the infrastructure of NeaSEL.
2. Validates ease of use.
3. Validates automated testing.
4. Validates any level of student can do the testing.

Three of these points are usability statements and a fourth is a mechanical statement. Let's pursue the usability statements first. To be a valid laboratory, any level of student should be able to perform an experiment. To prove this, different levels of students were asked to perform the experiments at various

times. They ranged from a junior level EE student to a second year Master's student.¹⁷ The results, within experimental error, were the same.

The results shown here (see Figures 25-27) are from the second year Master's student. His were chosen because he is a software engineer and therefore not intimately familiar with networks and because I added the extra complexity of forcing the code to be totally portable, *i.e.*, it would compile on all systems and all O/Ss. This is appropriate for a Master's level student who is about to enter the work force.¹⁸

These tests were easy to perform because the students could run on a perfect network (switched topology so it was point to point), because there was no outside packets (NeaSEL can be isolated from the campus backbone), and because experiments could easily be rerun in cases of student mistakes (there is no laboratory breakdown required afterwards). These facts allowed an in depth analysis of the Ethernet protocol.

¹⁷ Having the author perform the tests proves nothing. It's my methodology and I should know how it works.

¹⁸ The program `ftptest`, shown in Appendix A was used to perform the performance test by measuring the data transfer time between hosts on a network using the ftp protocol.

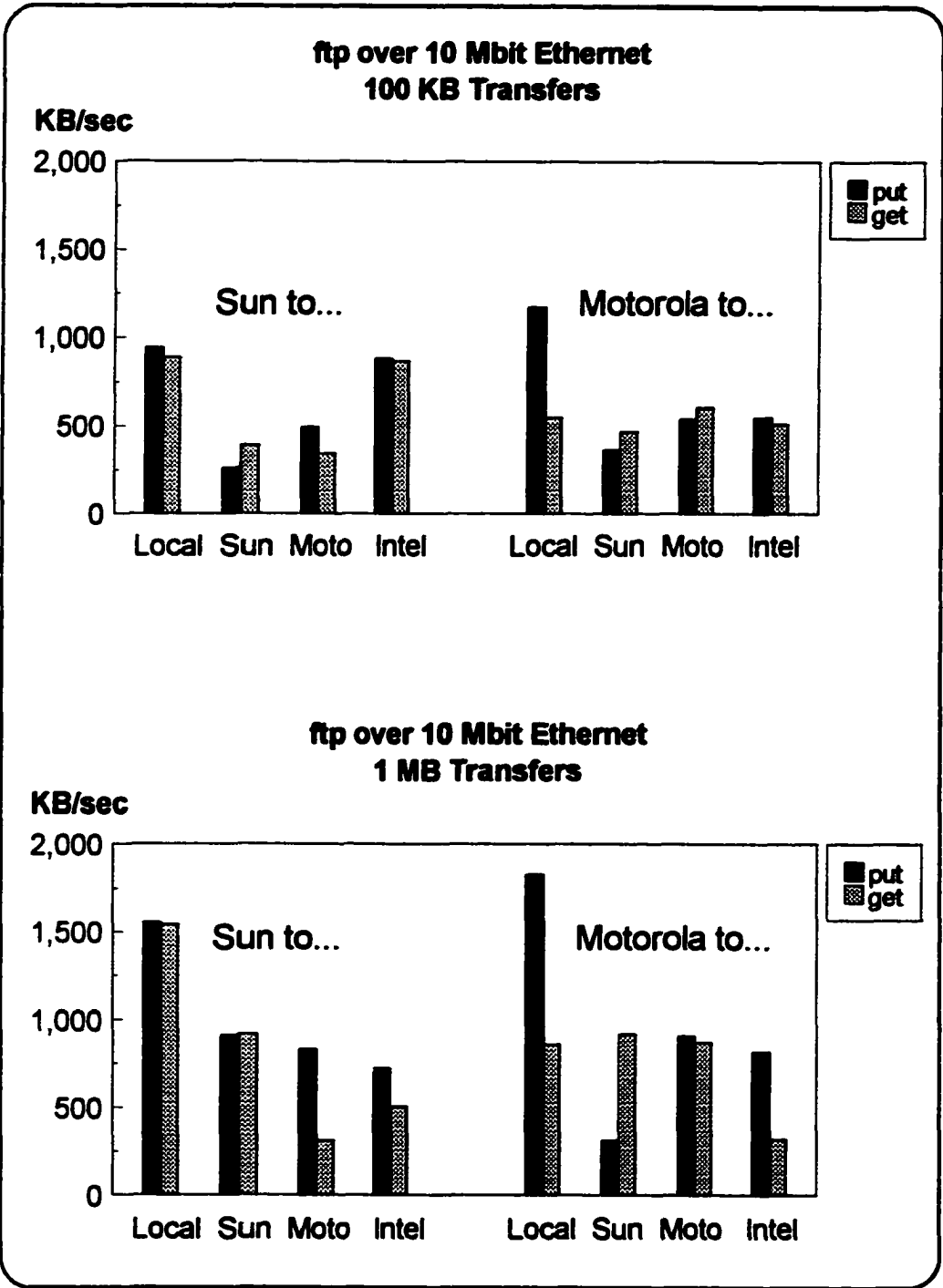


Figure 25. ftp Transfers over 10 Mbit Ethernet

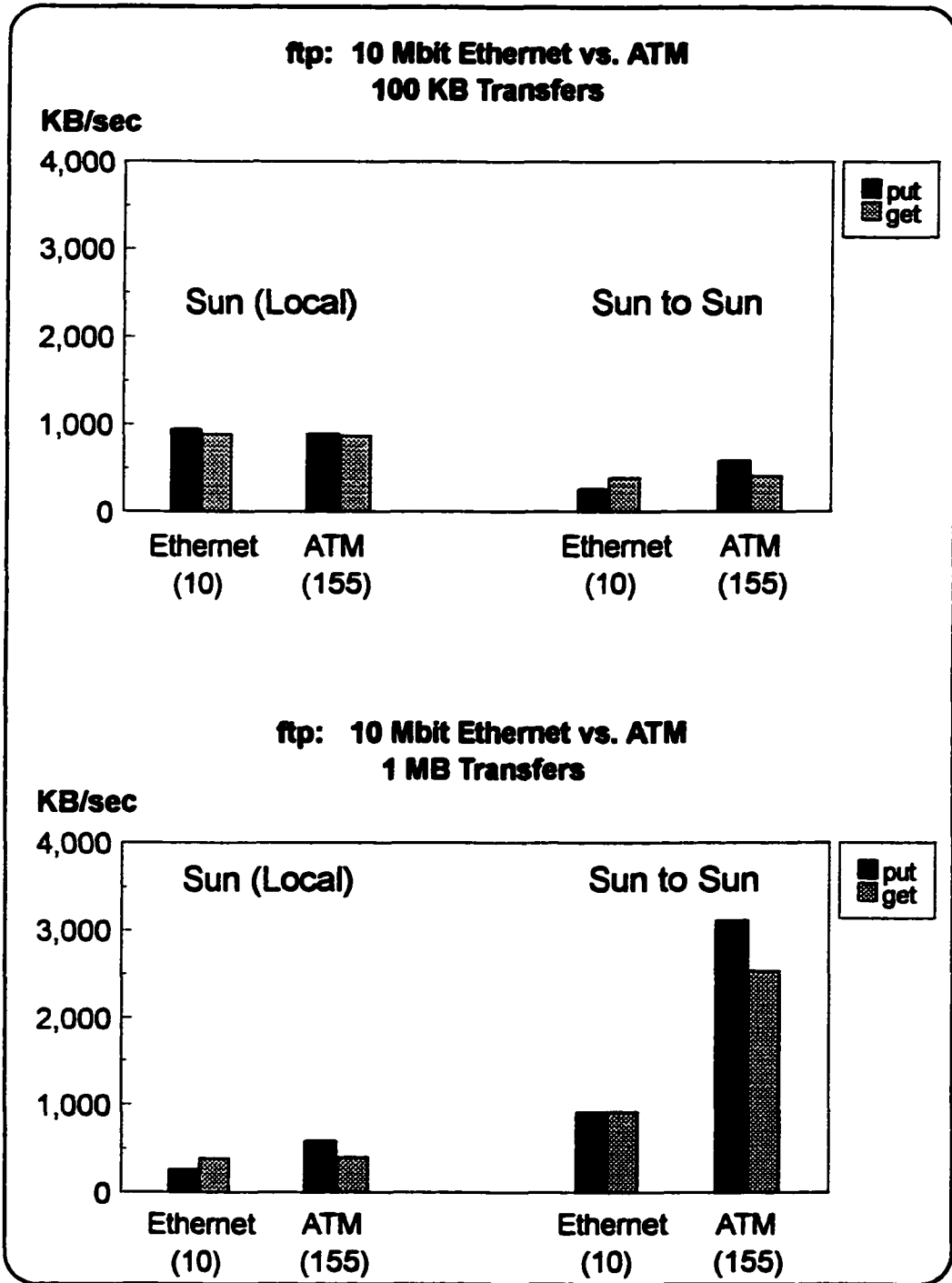


Figure 26. ftp Transfers, Ethernet vs ATM

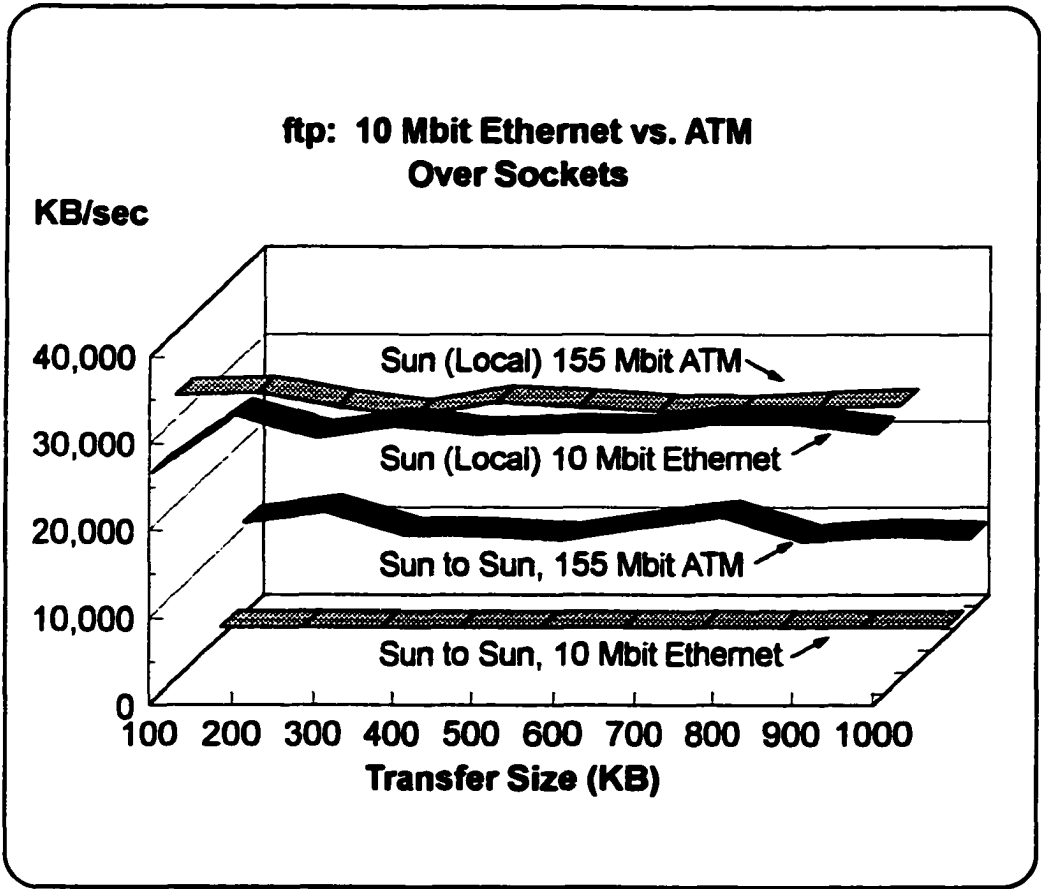


Figure 27. ftp Transfers over Sockets

7.7.3 Cluster

A different experiment which is software engineering oriented exclusively is cluster computing. Research into cluster computing has been stimulated anew because inexpensive, high-powered workstations, attached to high-speed LANs, are now plentiful. It is very easy to set-up a GFLOPS level compute cluster. In fact, many vendors during new product announcements will display their equipment running a cluster compute problem. Since cluster computation has become a new paradigm in the compute world, NeaSEL should enable this type of research.

One problem that readily lends itself to cluster computing is the calculation of fractals and is a standard problem that the IBM RISCSystem/6000 Division uses for demonstrations. Since NeaSEL has IBM equipment, this problem proved to be an ideal candidate for validation.¹⁹

For validation, the two questions were (1) can NeaSEL's architecture perform as a cluster and (2) can it be done with minimum modifications? The default island architecture is cluster; therefore, both conditions are validated.

¹⁹ There are two programs involved in the fractal demonstration and are shown in Appendix A. The first runs compute modules on various connected machines. The second is the compute module. The compute module sends the output as X traffic back to the system being used as a display head (note, a system can do both: display and compute).

7.7.4 Network Striping

Striping is a procedure in which a packet of information is split into n groups and sent over n different paths. Originally, this term was applied just to RAID units in which the information was stored on n disks. This term has recently been expanded to include striping across networks. While there are numerous ways to stripe across a network, for validation of the methodology and of NeaSEL, I concentrated on striping across multiple NICs.

The same program employed in the file transfer test was used for this test. The only addition was that multiple processes were spun off. NeaSEL's naming convention allows multiple runs with only minor changes, further validating the naming convention and automatic test script capability.

The figure shows the laboratory configuration in the experiment. With NeaSEL, this was an extremely simple experiment to perform.

1. All the host names and addresses had been assigned during the laboratory creation.
2. All the subnet masks had already been properly defined. The user was not required to learn subnetting just for this experiment.
3. The only necessary work was to install the adapters. Since the ISA Ethernet adapter was considered an unique installation, its installation was fully documented in the *Unique Installation Guide*.

4. **Attach the cables. Color coded cables, jacks, and the patch panel made the connections a trivial process.**
5. **With the naming and address convention, it was very easy to write a script to perform the test.**

With this, the experiment became a clone of the first validation experiment, ftp transfers.

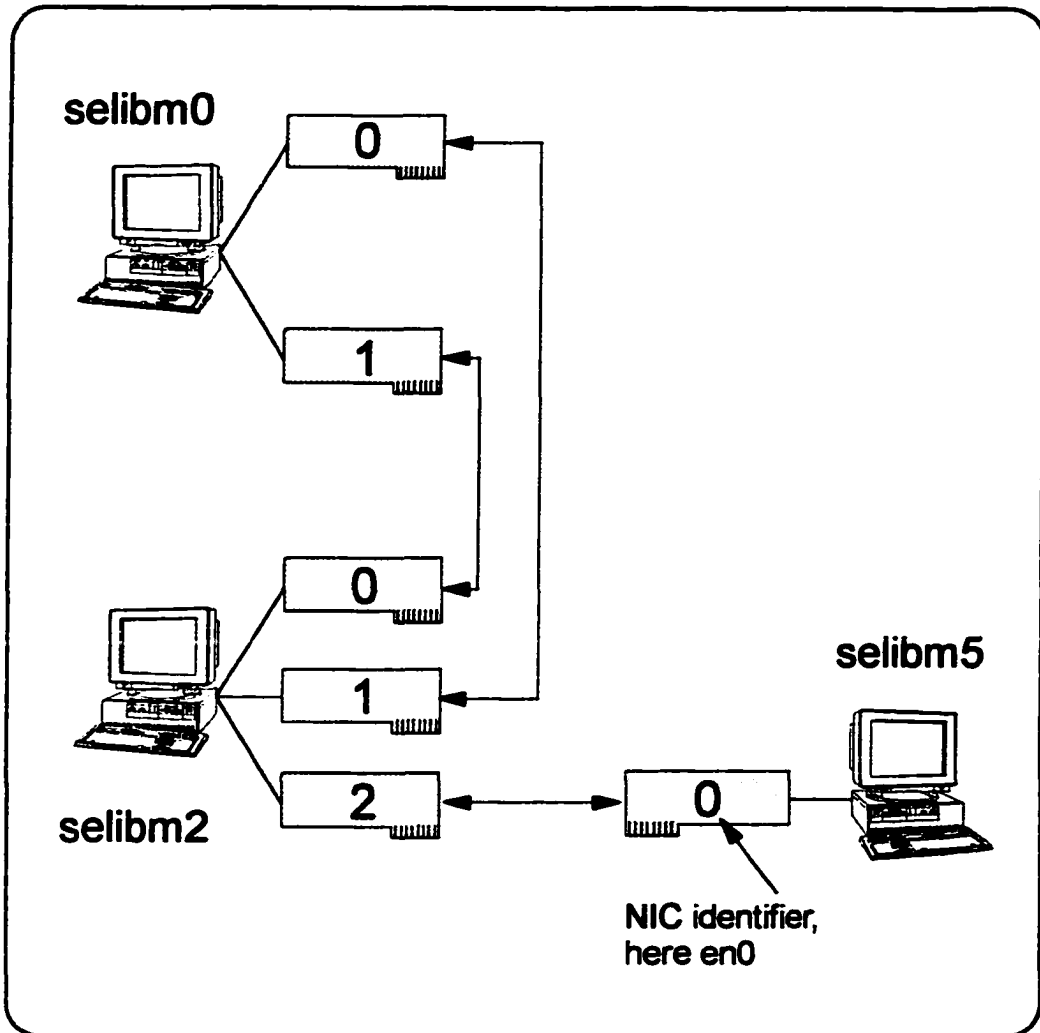


Figure 28. Validation of Network Striping

The switch has been omitted for clarity.

selibmx is the hostname.

The connections shown are those used in the experiment. They are not just to make the illustration "pretty".

7.8 Conclusion

This section has reported how the methodology and the implementation, NeaSEL was validated. The procedures covered global issues, such as the island concept, to the mechanics of automated testing. Given the range of the validation effort and the number of procedures used, the methodology has been validated as being able to generate a general purpose, multi-use, combined teaching and research, integrated network and software engineering laboratory at the systems level.

Chapter 8: Components of the Methodology

My methodology in this dissertation leads to a fully integrated systems laboratory, by definition a very powerful tool but with a significant initial implementation cost. Some organizations may not need the full capability or they may not have the financial resources for the complete implementation. This section will address that issue by listing what components of the methodology can be independently employed and what benefits can be derived from the components.

Before pursuing this topic, it must be clearly noted that my methodology is a prime example of the whole being greater than the sum of the parts. Applying x of y components will definitely give less than x/y percentage of benefits. Also, the full range of research topics will not be available.

8.1 Component Boundaries

The easiest way to use the components is to split the methodology along fundamental boundaries: naming convention, host numbering convention, compute islands, isolation procedures, and physical vs. electronic indication procedures.

8.2 Applying Individual Components

8.2.1 Naming Convention

This is the easiest component to apply outside the full methodology. The naming convention is recommended even if one does not run automated tests. It should be applied whenever there are more than a few machines co-located and always when systems have more than one NIC. The power of the naming convention outside the methodology is that it allows a system administrator to achieve the full benefits of addressability plus ease of administering networks remotely.

8.2.2 Host Numbering Convention

Applying this as a separate entity makes sense only if a number of machines are involved. If that is the case, then the convention as already noted should be followed and the most powerful machine made to be machine 0.

8.2.3 Compute Islands

The power of compute islands is that they allow an experimenter to see quickly and simply which compute paradigm is being studied: client/server, cluster, or distributed. As already noted, the algorithms are complex. Anything that aids the student to recognize the paradigms will be of great benefit. This

means if one expects to do cluster work, the workstations should be physically close together, *i.e.*, clustered. If they are for distributed computing, then physically separate them.

8.2.4 Isolation Procedures

This component is the easiest to separate from the integrated methodology. Even so this procedure should only be used when the laboratory absolutely needs this capability because changing resolution orders can be a significant impact on system administration costs (time and effort).

8.2.5 Physical vs. Electronic Indication Procedures

This is so fundamental that it should be used in all laboratories. It makes maintenance much easier, and by enforcing tag out (see Appendix B), controllability is greatly enhanced.

Chapter 9: Future Research

As previously shown, the systematic approach and its implementation fosters research and teaching of many topics, but what about the methodology itself? Is there research left? Are there extensions? The answer is yes.

This dissertation is the first to define a systematic approach for designing an integrated systems laboratory. The need for such a facility was clearly shown in the introduction. One expects that many such integrated systems laboratories will be built by various organizations. This general usage will lead to the first research extension.

The methodology was defined by deductive reasoning and only after many false starts. The extension should be to formalize the methodology by the use of predicate language. This is particularly important for the compute island concept and the naming convention. The compute island concept is so simple in appearance that is very easy to forget it really represents the compute paradigms that can be studied. Formalizing the islands will allow the concept to be extended to future compute paradigms, whatever they may be.

The naming convention again appears to be so simple that formalism is not required. This is obviously not true since the complete methodology hinges on the naming convention. Formalism should be applied. This will allow for future extensions, such as multiple NICs with the same address, NICs with dynamically changing addresses, and the like.

A second path for research would be a logical analysis and definition of the methodology, similar to the work already done for computer architecture. This will allow students to study the architecture in depth and to enhance it.

Other future approaches might be of a more practical nature. My particular case study was of IP protocols over several network topologies, the specific protocol was IP Version 4 (IPv4). The latest version, IPv6, is on the horizon. While there is nothing in the methodology, systematic approach, or explicit implementation that would prevent working with IPv6, this should be formally proved and documented.

There are other protocols, such as NetBios, where this methodology should work unchanged. Again, this should be formally proved and documented.

I am sure that future students and researchers will find many avenues of investigation to follow.

Chapter 10: Conclusion

Until this methodology, there were deficiencies in applicable knowledge in the research and educational fields of network engineering and software engineering. These growing deficiencies were due to the fact that there was seldom any cross-development or cross pollination between the two disciplines.

The dissertation directly addresses these issues by defining how an organization can create a general purpose, multi-use, combined teaching and research, integrated network and software engineering laboratory at the systems level. These principles were then validated during the design and construction of the large scale implementation, called NeaSEL, at the University of Texas at Austin.

The power of this methodology is twofold. First, it serves as a sound guideline for the creation of the facility and helps to eliminate the frustration of unworkable ideas. Second, the application of this methodology creates a single laboratory in which a wide range of teaching and research activities can be carried on, thus eliminating many specialized laboratories.

Appendix A: Validation Code Samples

A.1 FTP Test

A.1.1 Program Description

The program `ftptest` performs a network performance test by measuring the data transfer time between hosts on a network using the ftp protocol. `ftptest` generates scripts for the ftp client, and measures times of put and get operations for a specified file size, to a particular host. Results are saved in an output file.

A.1.2 How to run

Source code of `ftptest` consists of three files: `ftp.h`, `ftptst.c`, `start.c`. They may be compiled using `Makefile` which is included in the same directory. The program `ftptest` is run by the script `exscript`, which is included in the same directory. Input variables are:

- destination host address
- login name on a destination host
- password on a destination host
- source directory
- destination directory

The programs `exscript`, `ftptest`, and `measure` must reside in the same directory when a test is performed.

A.1.3 Source code

exscript

```
#!/bin/sh
#
# author: Dominik Kacprzak
# NeaSEL: 2/16/97
#
# this is an example script
# don't forget that names of the directories must be ended with '/'
#
ftptest 146.6.148.223 root syf /root/ /root/ sun10.log 10 100000
```

Makefile

```
CC = g++
all:      ftptest measure
ftptest:  ftptst.c
          $(CC) -o ftptest ftptst.c
measure:  start.cc
          $(CC) -o measure start.cc
```

ftp.h

```
#ifndef _ftp_h_
#define _ftp_h_ 1

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
#include <sys/times.h>
```

```

#include <sys/param.h>
#include <limits.h>
#include <netdb.h>
#include <ctype.h>
#include <malloc.h>

#define TIMEFILE "t"

typedef struct
{
    char *machine;
    char *username;
    char *password;
    char *dest_dir;
    char *filename;
} connection;

unsigned int iterations;
unsigned int max_machines;
char      *source_dir;
connection ftpsites;
int m_index = 0;
int i;

void create_transfer_file(long l)
{
    int i;
    FILE *f;
    char temp[100];
    char ch;

    sprintf( temp, "%ssndfile", source_dir );

    f = fopen(temp,"w");
    for ( i = 0; i < l; i ++ )
    {
        ch = (char)i;
        fputc(ch,f);
    }
    fclose(f);
}

```

```

void create_command_files(void)
{
    FILE *f;
    char filename[100];
    char temp[10];
    int i;

    sprintf( filename, "%sftpcmds", source_dir);

    f = fopen(filename,"w");

    fprintf(f, "user %s \"%s\" \n", ftpsites.username,
    ftpsites.password );

    fprintf(f, "binary\n");

    fprintf(f, "!%smeasure >> %s%s\n", source_dir, source_dir,
    TIMEFILE );

    fprintf(f, "put %ssndfile %stmpfile\n", source_dir,
    ftpsites.dest_dir );

    fprintf(f, "!%smeasure >> %s%s\n", source_dir, source_dir,
    TIMEFILE );

    fprintf(f, "get %stmpfile %srcvfile\n", ftpsites.dest_dir,
    source_dir );

    fprintf(f, "!%smeasure >> %s%s\n", source_dir, source_dir,
    TIMEFILE );

    fprintf(f, "quit\n");

    fclose(f);
}

void process_data_file_1( void )
{
    strcpy( ftpsites.machine, "146.6.148.223" );
    strcpy( ftpsites.username, "root" );
    strcpy( ftpsites.password, "syf" );
}

```

```

    strcpy( ftpsites.dest_dir, "/root/" );
    strcpy( ftpsites.filename, "sun1.dat" );
    strcpy( source_dir, "/root/" );
    iterations = 20;
}

#endif /* _ftp_h_ */

```

ftptst.c

```

#include "ftp.h"

int main(int argc, char *argv[])
{
    int i,j,k;
    long length;
    char *temp1;
    char *temp2;
    char temp3[20] = "";
    timeval start;
    timeval middle;
    timeval end;
    time_t t;
    FILE *f;

    if (argc < 9)
    {
        printf("\nSyntax is : ftptest <dest IP address> <user>
<password> <source dir> <dest dir>" );
        printf(" <log file> <number of iterations> <size of test
file>\n" );
        exit(1);
    }
    else
    {
        ftpsites.machine = argv[1];
        ftpsites.username = argv[2];
        ftpsites.password = argv[3];
        source_dir = argv[4];
        ftpsites.dest_dir = argv[5];
        ftpsites.filename = argv[6];

```



```

    sscanf( argv[7], "%i", &iterations );
    sscanf( argv[8], "%i",&length );
}
gethostname(temp3,20);
printf("Hostname is: %s\n",temp3);

t = time(NULL);
printf("Start time: %s", ctime(&t) );
printf("Size is: %i\n", length);

temp1 = (char *)malloc(100 * sizeof(char));
temp2 = (char *)malloc(10 * sizeof(char));

create_transfer_file(length);
create_command_files();

printf("Machine: %s\n", ftpsites.machine );
printf("Username: %s\n", ftpsites.username );
printf("Password: %s\n", ftpsites.password );
printf("Dest Dir: %s\n", ftpsites.dest_dir );
printf("Filename: %s\n\n", ftpsites.filename );

/* create an output file */
sprintf(temp1,"%s%s", source_dir, ftpsites.filename);
f = fopen(temp1,"w");
fprintf(f,"Hostname is: %s\n",temp3);
fprintf(f,"Destination host is: %s\n", ftpsites.machine );
fprintf(f,"Start time: %s",ctime(&t));
fprintf(f,"Size of the test file is: %d bytes.\n", length );
fclose(f);

for (i = 0; i < iterations; i ++)
{
    printf("\nIteration no : %d\n", i);

    sprintf(temp1, "%s%s", source_dir, TIMEFILE);

    if (f = fopen(temp1,"r"))
    {
        fclose(f);
        unlink(TIMEFILE);
    }
}

```

```

fclose(f);

printf("Machine : %s, Source dir: %s\n", ftpsites.machine,
source_dir);

sprintf( temp1, "ftp -n %s < %sftpcmds",ftpsites.machine,
source_dir);
printf("%s\n", temp1);

system(temp1);

t = time(NULL);

/* reading results from time file */
sprintf(temp1, "%s%s", source_dir, TIMEFILE );
f = fopen(temp1,"r");
fscanf(f,"%ld\n",&(start.tv_sec) );
fscanf(f,"%ld\n",&(start.tv_usec) );
fscanf(f,"%ld\n",&(middle.tv_sec) );
fscanf(f,"%ld\n",&(middle.tv_usec) );
fscanf(f,"%ld\n",&(end.tv_sec) );
fscanf(f,"%ld\n",&(end.tv_usec) );
fclose(f);

/*
 * creating result lines for output file
 */
if(i<9)
    sprintf(temp1," %d: Transfer put time: ", i+1);
else if(i<99)
    sprintf(temp1," %d: Transfer put time: ", i+1);
else if(i<999)
    sprintf(temp1,"%d: Transfer put time: ", i+1);

    sprintf(temp2,"%0.3f",(double) (( (middle.tv_sec-start.tv_sec)
*1000000 + middle.tv_usec ) - start.tv_usec )/1000000 );

printf("Transfer put time: %s sec.\n",temp2);
strcat(temp1,temp2);
strcat(temp1," Transfer get time: ");

```

```

/*   sprintf(temp2,"%0.3f",(double)(end-
middle)/(double)CLK_TCK);*/
    sprintf(temp2,"%0.3f",(double) (( (end.tv_sec-middle.tv_sec)
*1000000 + end.tv_usec ) - middle.tv_usec )/1000000 );

    printf("Transfer get time: %0s sec.\n",temp2);
    strcat(temp1,temp2);
    strcat(temp1," Date: ");
    strcat(temp1,ctime(&t));
    strcpy(temp2,source_dir);
    strcat(temp2,ftpsites.filename);

    /*
    * save results in output file
    */
    f = fopen(temp2,"a");
    fputs(temp1,f);
    fclose(f);
}

free(temp1);
free(temp2);
return 0;

}

```

start.cc

```

#include <iostream.h>
#include <sys/time.h>

int main(void)
{
    timeval tm, tm2;
    gettimeofday( &tm, NULL );

    cout << tm.tv_sec << endl;
    cout << tm.tv_usec << endl;
    return 0;
}

```

A.2 Network Performance Test

A.2.1 Program Description

The program `nettest`, which is based on a set of network object classes, was used to perform a network performance test using sockets over TCP/IP network. For maximum flexibility `nettest` was an object oriented design, therefore, it can be easily modified to embrace different type of networks. For example the base class, `Access`, defines a standard interface that is used by inheriting classes. In its existing implementation, `nettest` uses the `accessTCP` class that implements a network transfer object using socket implementation. In future, an implementation of `accessATM` class using native ATM protocol is possible.

A.2.2 How to Run

Porting `Nettest` to a new platform should be straightforward thanks to `Makefile` that defines dependency between different source files.

In order to run a test, the user has to run the programs `netsnd` on the source host and `netrcv` on the destination host. `Netsnd` must be started first. A recommended method of execution is to open two terminal windows in X-windows, start `netsnd` in one of windows and then `netrcv` in another. Input variables are:

- transfer block size
- number of loops
- socket port number.

Running either program without parameters will place the user in a help mode which will explain the variables.

A.2.3 Source code

Makefile

```
CC = g++
INCL =
FILES = $(DIR)
FLAGS = -fhandle-exceptions
all: netrcv netsnd
netrcv: netrcv.cc
      $(CC) $(INCL) $(FLAGS) netrcv.cc -lsocket -lnsl -o netrcv
netsnd: netsnd.cc
      $(CC) $(INCL) $(FLAGS) netsnd.cc -lsocket -lnsl -o netsnd
```

access.h

```
#if !(_Access_h_)
#define _Access_h_ 1

#include <iostream.h>
#include <unistd.h>
#include <fcntl.h>

/*
 * An abstract class.
 * Will be used for an implementation of I/O classes.
 *
 * Goal: TCP/IP(sockets) and ATM network modules.
 * ATM module should additionally ensure a QoS.
 */
```

```

*/

class Access
{
public:
    int virtual readFrom(char *, int) = 0; // reads from whatever is
    controlled by modul
    int virtual writeTo(char *, int) = 0; // writes .....
    virtual ~Access() {}
};

#endif /* _Access_h_ */

```

accessTCP.h

```

#if !(_accessTCP_h_)
#define _accessTCP_h_ 1

#include <iostream.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <strings.h>
#include "access.h"

class accessTCP : public Access
{
private:
    int sockfd;
    char *pname;
public:
    // server constructor
    accessTCP(int TCP_PORT);

    // client constructor
    accessTCP(char* HOST_ADDR, int TCP_PORT);

    int readFrom(char *msg, int size);
    int writeTo(char *msg, int size);
    ~accessTCP();

```

```
private:
    // write exactly nbytes to fd
    int writen(int fd, char *ptr, int nbytes);
    // read exactly nbytes from fd
    int readn(int fd, char * ptr, int nbytes);
};

#endif /* _accessTCP_h_ */
```

accessTCP.hcc

```
#include <iostream.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <strings.h>
#include "accessTCP.h"

accessTCP::accessTCP(int TCP_PORT)
{
    struct sockaddr_in serv_addr, cli_addr;
    int clien;

    /*
     * open a tcp socket
     */
    if ( (sockfd = socket(AF_INET,SOCK_STREAM, 0)) < 0)
    {
        cout<<"server: can't open stream socket"<<endl;
        exit(-1);
    }

    /*
     * binding a local address
     */
    // bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(TCP_PORT);

    if ( bind(sockfd, (struct sockaddr *) &serv_addr,
    sizeof(serv_addr))<0 )
    {
        cout<<"server: can't bind local address"<<endl;
        exit(-1);
    }

    if (listen(sockfd, 5))
```



```

    {
        cout<<"server: listen error"<<endl;
        exit(-1);
    }

    cliilen = sizeof(cli_addr);
    int tsockfd = sockfd;
    sockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &cliilen);
    close(tsockfd);

    if(sockfd<0)
    {
        cout<<"server: accept error"<<endl;
        exit(-1);
    }
}

accessTCP::accessTCP(char *HOST_ADDR, int TCP_PORT)
{
    struct sockaddr_in serv_addr;

    /*
     * Fill in the structure "serv_addr" with the address of the
     * server that you want to connect with.
     */
    // bzero((char *) &serv_addr, sizeof(serv_addr));
    // bzero( (void *) &serv_addr, (size_t)sizeof(serv_addr) );
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(HOST_ADDR);
    serv_addr.sin_port = htons(TCP_PORT);

    /*
     * Open a TCP socket (an Internet stream socket).
     */
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        cout<<"client: can't connect to server"<<endl;
        exit(-1);
    }

    /*
     * Connect to the server.

```

```

    */
    if (connect(sockfd, (struct sockaddr *) &serv_addr,
        sizeof(serv_addr))<0)
    {
        cout<<"client: can't connect to server"<<endl;
        exit(-1);
    }
}

int accessTCP::readFrom(char *msg, int size)
{
    return readn(sockfd, msg, size);
}
int accessTCP::writeTo(char *msg, int size)
{
    return writen(sockfd, msg, size);
}

accessTCP::~accessTCP() { close(sockfd); }

int accessTCP::readn(int fd, char *ptr, int nbytes)
{
    int nleft(nbytes), nread;
    // nleft = nbytes;

    while( nleft >0)
    {
        nread = read(fd, ptr, nleft);
        if (nread<0)
            return nread; // error, return < 0
        else if(nread == 0)
            break;

        nleft -= nread;
        ptr += nread;
    }
    return (nbytes-nleft);
}

int accessTCP::writen(int fd, char * ptr, int nbytes)
{
    int nleft(nbytes), nwritten;

```

```

//nleft = nbytes;
while (nleft>0)
{
    nwritten = write(fd,ptr,nleft);

    //cout<<"write: "<<nwritten<<endl;

    if(nwritten<=0)
        return nwritten;

    nleft -= nwritten;
    ptr += nwritten;
}
return nbytes-nleft;
}

```

netsnd.cc

```

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#include "accessTCP.hcc"

int main(int argv, char **argc)
{
    accessTCP *serv_sock;
    int SIZEOFBUFF;
    int SKEY;
    int LOOP;
    int PORT;

    if(argv<4)
    {
        cout<<"usage: netsnd block_size[KB] loop_number port"<<endl;
        exit(-1);
    }
}

```

```

    }
else
{
    sscanf( argc[1], "%i", &SIZEOFBUFF );
    sscanf( argc[2], "%i", &LOOP );
    sscanf( argc[3], "%i", &PORT );

    // cout<<SIZEOFBUFF<<" "<<SKEY<<" "<<LOOP<<"
    "<<PORT<<endl;
}
SIZEOFBUFF = SIZEOFBUFF*1024;

char *msg = new char[SIZEOFBUFF];

// network initialization
serv_sock = new accessTCP(PORT);
for( int i=0; i<LOOP; i++ )
{
    serv_sock->writeTo(msg, SIZEOFBUFF);
}

delete[] msg;
delete serv_sock;

return 0;
}

```

netrcv.cc

```

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#include "accessTCP.hcc"
#include "timer.hcc"

```

```

int main(int argv, char **argc)
{
    Timer test;
    timeval result = {0,0}, tmp;
    int SIZEOFBUFF;
    int RKEY;
    int LOOP;
    int PORT;
    char *IP;
    accessTCP *cli_sock;
    if(argv<5)
    {
        cout<<"usage: netrcv block-size[KB] loop-number IP port"<<endl;
        exit(-1);
    }
    else
    {
        sscanf( argc[1], "%i", &SIZEOFBUFF );
        sscanf( argc[2], "%i", &LOOP );
        sscanf( argc[4], "%i", &PORT );
        IP = argc[3];
        // cout<<SIZEOFBUFF<<" "<<RKEY<<" "<<LOOP<<" "<<IP<<"
        <<<PORT<<endl;
    }
    SIZEOFBUFF = SIZEOFBUFF*1024;

    char *msg = new char[SIZEOFBUFF];

    // network initialization
    cli_sock = new accessTCP(IP, PORT);
    for( int i=0; i<LOOP; i++)
    {
        test.start();
        cli_sock->readFrom(msg, SIZEOFBUFF);
        test.stop();
        if(i!=0)
        {
            tmp = test.result();
            result.tv_sec += tmp.tv_sec;
            result.tv_usec += tmp.tv_usec;
        }
    }
}

```

```

    cout << "Final result is:"<<endl;
    result.tv_usec += result.tv_sec*1000000;
    cout << "average time is: "<< result.tv_usec/(LOOP-1)<< " [usec] "
<< endl;
    cout << "total time is " << result.tv_usec << " [usec]" << endl;

    delete msg;
    delete cli_sock;
    return 0;
}

```

timer.h

```

#include <iostream.h>
#include <sys/times.h>

```

```

class Timer
{
private:
    timeval tm1, tm2;
public:
    Timer(){}
    ~Timer(){}
    void start();
    void stop();
    void show();
    timeval result();
};

```

timer.hcc

```

#include "timer.h"

void Timer::start()
{
    gettimeofday( &tm1, NULL );
}
void Timer::stop()
{

```

```

gettimeofday( &tm2, NULL );
if( tm2.tv_usec<tm1.tv_usec )
{
    tm2.tv_sec--;
    tm2.tv_usec += 1000000;
//    cout<<"ext"<<endl;
}
}
void Timer::show()
{
    cout<<"secs: "<< tm2.tv_sec-tm1.tv_sec <<endl;
    cout<<"usec: "<< tm2.tv_usec-tm1.tv_usec <<endl;
}

timeval Timer::result()
{
    timeval tmp;
    tmp.tv_sec = tm2.tv_sec-tm1.tv_sec;
    tmp.tv_usec = tm2.tv_usec-tm1.tv_usec;

    return tmp;
}

```

A.3 Xfract

A.3.1 Description

Xfract is a program that computes complex fractals on a cluster of workstations and visualizes the result on a server screen. It is an AIX program that was and is being used by the IBM RISC System/6000 Division of IBM as a standard demonstration. It is available from many IBM Web sites, royalty free, as compiled code.

A.3.2 How to run

Instructions are available in a README file in the package one receives when the code is downloaded. In general, to run this program, the user has to add a new service for the fractal computation. This is done by modifying the following system files:

- To invoke fracttcp, add an entry in /etc/inetd.conf file as follows:
xfract stream tcp nowaitroot /your directory/fracttcp
- Also update the /etc/services file as follows:
xfract 3737/tcp #Xfract port number

After updating the two system files, run `refrest -s inetd` to update the internet daemon.

The machines composing the cluster, with their percentage of workload, are defined in the `xfract.servers` file. For example, the following lines of code would define three machines, each doing 33% of the work:

```
selibm2      0.33
selibm3      0.33
```

The server computes the remaining workload, here 33%.

Appendix B: Case Study Implementation Details

The purpose of this dissertation is to provide a methodology and architecture which can lead a group to a physical implementation of a laboratory such as the case study, NeaSEL. For such an implementation, the actual builders may not be as interested in the theory behind the discovery process as they are in practical execution details. To that end, I present this appendix. While this appendix is not a complete set of construction plans, there is enough detail presented to start a group well into an implementation. Also, these points are a result of a post analysis of NeaSEL. They include lessons learned and I will point out where NeaSEL was lacking.

In the following, the order of details presented are solely for convenience and do not imply any priority.

B.1 Layout

B.1.1 Tables Same Shape as Laboratory

The shape of the room and the shape of the tables should be the same. There are two reasons for this statement. The first is the theoretical. It's to reinforce the compute island concept. Having the tables the same shape as the room is pleasing to the eye and helps to propagate the island concept. The second is a practical one. Using tables in the same shape as the room gives the maximum amount of usable space in the minimum area.

Note, one can ignore this rule in the design of a laboratory in a physically big room since "small tables" can be placed in many different ways.

B.1.2 30"x48" Tables

From experience, the ideal table size seems to be 30" x 48" (*i.e.* slightly smaller than standard size). This is big enough to hold a system unit/monitor plus test equipment and to have a place to work & hold manuals. Bigger, they take up room unnecessarily. Smaller, they hamper productivity.

The holding of manuals and test equipment is an extremely important concept. We want to encourage students to use test equipment and search the technical manuals. One way to discourage this is to have too small a place to hold test equipment and manuals.

B.1.3 Tables into Compute Islands

In NeaSEL, tables are grouped in a 3x2 cluster with the long side (3 tables) bolted together. Advantage: tables are very difficult to move so students cannot mess the laboratory up and leave it in a disarray. Disadvantage: they are virtually impossible to move, especially with equipment on them. Bolting them together makes sense; it keeps the laboratory neater.

B.2 Wiring

All wiring in the room, including patch cables, is CAT-5. All wiring is enclosed in raceways (G4000 to be specific) attached to the walls.

There are seven points on the raceways for attachment by systems to the network (Figure 29):

- one at the end of each of the five islands,
- one on the front wall for use by mobile equipment and printers
- and one on the back wall for system administration use.

Each connection point on the raceway provides 18 RJ-45 connectors, six each for three different network topologies. The three network topologies currently under study are 10 Mbit Ethernet, 100 Mbit Ethernet or ISDN, and 155 Mbit ATM. The jacks are color coded (see next section).

Numbered systems are generally attached to jacks as shown in Figure 30. Deviations may occur when multiple NICs are used in the same system unit.

The jacks, the patch panel, and the cables are numbered. Connections are made simply by matching numbers and colors.

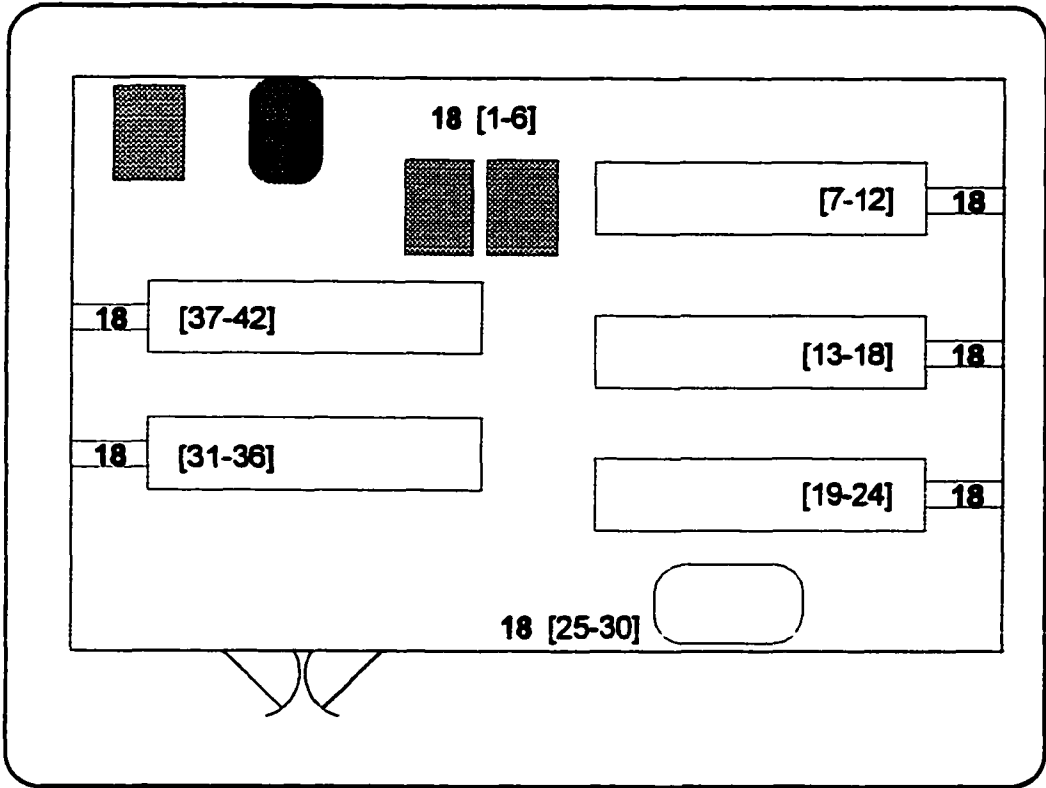


Figure 29. Raceway Attachments

The number 18 shows the locations of the connection points.
 The numbers in [] are the numbers assigned to the connectors.

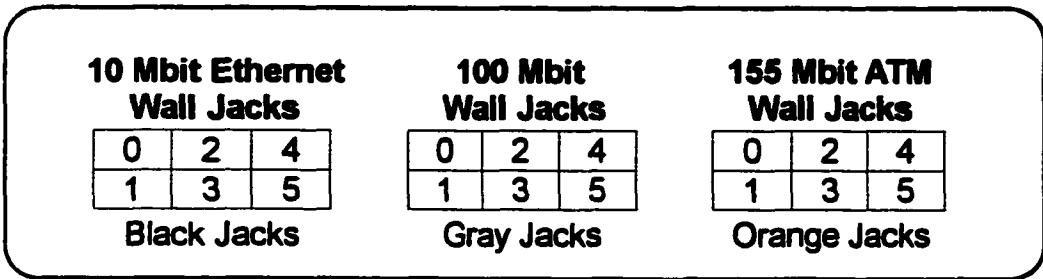


Figure 30. System Attachment to Wall Jacks

B.2.1 Raceways (G4000) for Wiring

NeaSEL used raceways, G4000, for electrical and LAN wiring. They are aesthetically not pretty and they are somewhat expensive (though far cheaper than running wires through the walls), but their benefits greatly outweigh these two disadvantages. The first benefit is that they make all the wiring very neat. Second, even though the cables are enclosed inside the G4000, the raceways are a visual indication of how the network is wired. Third, they protect the wiring. Fourth, wiring changes can be made much easier than fishing cables through the wall. This is particularly important if the laboratory owner wants to go to optical cables in the future.

B.2.2 LAN Terminals at the End of Tables

In our physical implementation, we used RJ-45 LAN connectors at the ends of the compute islands (tables) next to the wall raceways. NeaSEL used a face plate style that allowed 6 RJ-45 connectors in the space of one electrical outlet box. This arrangement is extremely neat and flexible. Everything is connected with patch cables in a very orderly fashion so it is very easy to maintain.

B.2.3 Cable Raceways under Tables

Install raceways under the table to hold all the cables. This makes the laboratory look considerably neater and prevents students from accidentally breaking cables with their feet. This was probably the greatest implementation failure in NeaSEL. We made the decision to use nylon cable ties instead of raceways for cost. While nylon ties appear to be a neat and inexpensive solution, they fix the cables permanently, which makes it virtually impossible to move equipment. After the first iteration of equipment moving, all the ties were cut and the cables were left dangling.

B.2.4 Leave a Gap between Tables and Wall

Leave a gap of four to five inches between each row of 3 of the 6 table cluster. This allows cables to easily fall down between tables and not lay on the table in an unsightly tangle. The same gap should occur between the edge of the tables and the wall for the same reason. There is no need for a gap, side to side, between tables.²⁰

²⁰ In practical measurements, four to five inches is one fist width which is easier for students to use.

B.3 Electrical Power

B.3.1 Electrical Outlets for Tables

Computer equipment is plug intensive. Each table of the 3x2 cluster should have a 4-6 way outlet box. This gives capacity to add equipment or to use test equipment; otherwise, one is always forced to use extension cords (bad for safety and neatness) or stretching power cords to their limit to plug into the nearest outlet.

There should also be a set of electrical plugs on top of the tables. This allows for the powering of test equipment without crawling under the tables. Note, plugs should be on both the top of the tables and under the tables. The top plugs are for test equipment, laptops, calculators, *etc.* Do not plug semi-permanent equipment like systems and monitors into the top plugs because the cords takes up valuable space and make the laboratory look very tangled.

B.3.2 Numerous Electrical Outlets on Walls

This is an extension of the previous item. The more outlets available on the wall, the easier it is to plug in mobile computers, overhead projectors, and the like. The user also has the freedom to plug them in where he wants them and not where the outlet is. NeaSEL did not have wall outlets which made the use of portable test equipment more difficult.

B.4 Color Coding

All connectors, patch cables, patch panel and jacks were color coded.

The colors chosen were:

- **Black** 10 Mbit Ethernet
- **Light Gray** 100 Mbit Ethernet
- **Orange** 155 Mbit ATM.

Connection is very simple: just match the colors.

For maintainability and controllability, the proctors strictly enforce the color coding. For example, they never allow the use of a black patch cable to connect an ATM device.

B.5 Patch Cables

B.5.1 Cat-5 Patch Cables

All patch cables in NeaSEL are CAT-5 and therefore are guaranteed to meet the bandwidth specifications of NeaSEL network topologies. Users are not allowed to use other patch cables of unknown specifications or from unknown sources because they might cause serious failures of the network.

All patch cables are stored in containers separated by color and lengths. The containers are conveniently located and are clearly marked by color and length of cable.

B.5.2 Lengths

All patch cables, no matter their colors, come in three lengths: 8 ft, 12 ft, & 15ft-17ft which match the distances of the systems on a compute island to the nearest LAN attachment point. Using the proper length patch cable means that all systems can be reached without excessive cable lying around, but with enough slack so that slight movements of system units can be tolerated without stretching the patch cable.

B.5.3 Special Use Cables

Some patch cables are special use cables and should only be used under specialized circumstances and never used for making general connections. An example of a special use cable is a null modem cable. Since the wires in a null modem cable are crisscrossed, it can only be used for applications requiring that configuration.

All special use cables have labels attached that clearly define their functions and are of a different color than the standard black, gray, and orange.

B.6 Workstations

B.6.1 Hardware Mix

The hardware mix of the workstations must be selected to provide the broadest possible representation of flexible platforms. Most should host more than one operating system, providing researchers and students with even more exposure to wide-ranging development and delivery host environments.

B.6.2 Number

Obviously, to investigate distributed systems or cluster computing, many machines are needed. Interesting problems start at 4-5 machines and go on to infinity. Using the base numbers, it would appear there should be 4-5 machines homogeneous in nature with 5+ heterogeneous groupings of machines. This will give a lot of flexibility and allow interesting problems.

B.6.3 Full Featured

This may not be obvious but full featured machines are needed. This means large memories, caches, large files, *etc.* It is easier to remove parts if they are physically available rather than to order parts just to run tests. But, there is an auxiliary to this. Machines must be very easy to reconfigure. If they are not, then too much time could be wasted in reconfiguration instead of running tests.

B.6.4 Slot Restrictions & NICs

Wide-ranging network research, from dynamic load balancing to analysis of complex network topologies can be performed only if each workstation allows the installation and configuration of multiple NICs. Therefore, all machines should have a large number of industry standard bus slots with no limitations on adapter placement. The workstation must accept at least three or more NICs with any combination of network topologies. The restriction to industry standard bus slots allows adapters to be moved between machines so tests can be done with fewer adapters.

The workstations must allow their multiple NICs to operate simultaneously.

B.6.5 Robust Expandable O/Ss

Machines should support robust, expandable O/Ss. As an example of non-robustness, a complete O/S should not be recompiled just to add a new communication adapter. The preference is that the adapter should be an extension to the O/S.

B.6.6 Multiple O/Ss Simultaneously

Preference is for machines to handle multiple O/Ss simultaneously. For testing, this saves the reloading of an O/S to run a test.

B.6.7 Cost Effective

Because of performance improvements, machines must be upgraded every two years to stay current. The cost of upgrading hardware must not impact the methodology.

B.7 Label Equipment

Every system unit was clearly labeled with its system name (host name), TCP/IP address, hardware contained inside (memory, hard files, processor, *etc.*) and operating system version with fix level. First, this is absolutely critical information to run experiments. Second, this saves every student from asking the same question and getting different answers from different people. Third, it teaches the student the proper way to run experiments (determine your base system before running the experiment). Fourth, it makes the student aware that such information is important even if he is not running an experiment.

Our methodology in NeaSEL is that all machines have placards that clearly show these key characteristics. Placards are always placed in highly visible locations. For systems with displays, the placards are in the middle top of the display. Figure 31 shows a typical placard

While it is a NeaSEL policy to keep the placards up-to-date, the students are told to always assume they are incorrect for formal tests. This is done to teach students and researchers the correct way to document their tests. Since the *NeaSEL User's Manual*, required reading for all users, gives precise and simple instructions on how to determine the configuration of a machine, this policy has an insignificant impact to any test being run..

↻ Workstation number in the island

1	selibm1	↻ Host name
e10: 146.6.148.201		↻ Default 10 Mbit Ethernet address
e100:146.6.149.201		↻ 100 Mbit TCP/IP address
a155:146.6.150.201		↻ ATM TCP/IP address
IBM RISC System/6000 40P		↻ Make and model
RAM : 64 MB		↻ Memory installed on the system
Disks: 2.2 GB, 1.1 GB		↻ Hard disks present
O/S : AIX 4.1.2		↻ Level of O/S and update level

Figure 31. Typical System Placard

B.8 Patch Panel inside the Laboratory

To achieve the flexibility of NeaSEL and to maintain firewall security, all wiring was done through a patch panel that can be physically and electrically isolated from the rest of the campus. This patch panel is color coded and is fully marked. The patch panel is inside a locked cabinet at the back of NeaSEL.

There is a chart showing what system units are attached to what connectors taped on the outside of the patch panel cabinet. There is also a temporary Change List so an experimenter can keep track of quick changes made for a particular test. We encourage conscientious and deliberate use of the change list because it will save the experimenter time and effort of tracing wiring when returning the network back to its normal state.

We strongly recommend the racks with patch panels & hubs be inside the actual laboratory itself. This makes wiring changes much simpler and it gives a visual indentation of how things are wired together. Also, EE students should know LAN wiring. Having a rack inside the laboratory at least gets them curious and hopefully excited.

B.9 Documentation

B.9.1 Manuals

Appendix C lists the manuals that were written for the NeaSEL implementation. This section will describe in detail several features of the *NeaSEL User's Manual* which pertain to the implementation.

The user's manual is version controlled and the faculty deems it important enough that it is mandatory reading for users of NeaSEL. This manual contains two appendices that give detailed information about a specific host and how that information was obtained. This information consists of how to determine what processor is inside the machine, hard file size, memory & cache size, and the like.

The first appendix gives a table of information per specific machine (see Figure 32). The information ranges from serial numbers to cache size, from memory slot usage to number of free bays. The goal was to have a single information source that could cover many aspects of experimenting in a laboratory. The information is current as of the date of the *NeaSEL User's Manual*.

While every attempt is made to keep the information in the user's manual current, there are times when it might be out-of-date, for example in the middle of an experiment when numerous reconfigurations are being done.

selibm5

<p>e10 : 146.6.148.205 e100 : 146.6.149.205 atm : 146.6.150.205</p> <p>IBM RISC System/6000 40P S/N: 23K6270 UT: 602638</p> <p>//P: PowerPC 601 66 MHz</p> <p>Memory: 64 MB Mem Slots Total: 6 Free: 1 Max/Slot: 32 MB, 70ns, 168 pin</p> <p>Cache L1: 32 KB I/D L2: 256 KB L3: 0 MB</p> <p style="text-align: center;">Media Bays</p> <p>Max 4: Free 2 File 1: 540 MB SCSI-2 2: 2.1 GB</p> <p>Diskette Drive: Yes CD-ROM: No</p>	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Bus Slots:</th> <th style="text-align: left;">Type</th> <th style="text-align: left;">Total</th> <th style="text-align: left;">Used</th> </tr> </thead> <tbody> <tr> <td>PCI</td> <td></td> <td style="text-align: center;">2</td> <td style="text-align: center;">1</td> </tr> <tr> <td>MicroChannel</td> <td></td> <td style="text-align: center;">0</td> <td></td> </tr> <tr> <td>EISA</td> <td></td> <td style="text-align: center;">0</td> <td></td> </tr> <tr> <td>ISA</td> <td></td> <td style="text-align: center;">3</td> <td style="text-align: center;">1</td> </tr> <tr> <td>TURBOchannel</td> <td></td> <td style="text-align: center;">0</td> <td></td> </tr> <tr> <td>Other</td> <td></td> <td style="text-align: center;">0</td> <td></td> </tr> </tbody> </table> <p>SCSI-II Int: 3 Ext: 2</p> <p style="text-align: center;">Monitor</p> <p>IBM 17V S/N: 232193X UT: None</p> <p style="text-align: center;">Network</p> <p>10BaseT 10 00 5A BD 34 B0 ISDN: No ATM:</p> <p>Audio: Yes</p> <p>O/S: AIX 4.1.2</p>	Bus Slots:	Type	Total	Used	PCI		2	1	MicroChannel		0		EISA		0		ISA		3	1	TURBOchannel		0		Other		0	
Bus Slots:	Type	Total	Used																										
PCI		2	1																										
MicroChannel		0																											
EISA		0																											
ISA		3	1																										
TURBOchannel		0																											
Other		0																											

Figure 32. Typical Informational Table

For this reason, the faculty stress host information must be verified for each experiment run. Since each manufacturer has his own way of determining the information, a separate appendix gives a brief set of instructions per make & model that allow a user to easily determine the configuration of the machine.

While most user's manuals are oriented to students and researchers, there are sections of the manual that help lab proctors and lab technicians. The tabular information section of the manual notes items such as serial numbers, brass tag numbers, proper names of the equipment, and the like. This is an ideal single source for inventory.

B.9.2 Lockable File Cabinet inside the Laboratory

The laboratory must have a lockable file cabinet inside the laboratory where software license agreements, purchase orders, maintenance history of equipment, and other key documents can be placed and secured. These key documents should always remain with the equipment and not put in a central repository where they can be misplaced or lost. Most organizations use the central repository method. Labs such as NeaSEL are far too complex to administer by a central authority. Keys to this file cabinet should be given only to trusted individuals.

B.9.3 Bookcase inside the Laboratory

All manuals for all equipment inside the room should remain with the equipment and not be put in a central location. Manuals with equipment makes

maintenance much easier, particularly in a heterogeneous laboratory. Time is not wasted searching the building for manuals.

Administrators always worry about this arrangement because manuals might be stolen. The solution to this is to keep one set locked up in the previously noted file cabinet. Label all the other manuals as property of the laboratory. Hopefully, if the students see the manuals, they will read them; and isn't it the function of a university to make information available, not to keep it locked up?

B.10 Use of Tag Outs

Industry and the military use a device called a tag out when safety is of the utmost importance. When somebody is working on a piece of equipment, that individual or team lead must tag the equipment at the power source stating that it is being worked on. This is a warning that personnel are working on a piece of equipment and under no circumstances is the equipment to have power reapplied. The individual and the only the individual who has tagged out the equipment has the authority to bring the equipment on-line and to remove the tag. Nobody else may remove the tag or reconnect the equipment.

This is obviously good safety practice but how would this be applied in a university laboratory? First, if somebody is running a lengthy experiment, that individual is expected to tag out the appropriate equipment so that other people will not accidentally use the same equipment and possibly ruin data. A reason, but not the most important.

The most important reason is what this laboratory was designed to do: allow researchers to experiment in network topics, either directly on the hardware or indirectly through software. Great efforts were provided to ensure there is a physical means of disconnecting from the campus backbone to prevent injecting bad network information and therefore, possibly bringing down the campus backbone. Even with these safety measures, what is to prevent one student from accidentally reconnecting the disconnected laboratory to the campus backbone when another is running dangerous network experiments, such as studying network viruses? Tag outs.

Tag outs are strictly enforced. A person disconnecting from the campus backbone must tag the patch cable and only he and he alone can remove the tag out and reconnect the laboratory to the campus backbone.

B.11 Configuration Sameness

Every attempt was made to make machines by the same vendor with the same hardware configuration to be configured in software identically. This makes maintenance much simpler.

What does configuration sameness mean in practice? Let's look at an example of a facility with six machines by a certain vendor and each one has two hard files of 0.5 GB and 2.1 GB. If the operating system of one machine is on its 0.5 GB drive, then the operating systems of all other similar machines should be on their 0.5 GB drives. This may seem extreme and extra work, but consider the following scenario. A researcher is using a machine that has a hard drive crash,

the one with its operating system. A quick fix is to remove the equivalent drive from a working machine, replace the crashed drive with a working drive, and continue with the experimentation. If the machines were not configured identically, one would have to wait for a purchasing organization to buy a replacement drive or completely rebuild an operating system on the remaining working drive. Time wasted and taken away from the experiment.

B.12 Extensive Back-up Capability

The purpose of this laboratory is to experiment with network configurations, operating system modifications, and protocol changes. This is a very polite way of stating things will get broken and data will be lost.

This type of work can only be done with impunity if there is a method to restore the equipment to its previous state. To achieve this, NeaSEL has a 20 GB RAID unit and a tape jukebox. Equipment is already in place to do back-ups. Laboratory proctors have been trained to help students in back-up procedures. Also, the *NeaSEL User's Manual* has typical back-up procedures for each O/S.

B.13 Cordless Phone

This laboratory is powerful but complex to implement. A simple a thing as a cordless phone is a necessity if one is trying to configure equipment and talk to vendors at the same time.

B.14 Miscellaneous Equipment

The first is lots of trash cans. It helps keep the laboratory clean. The second is to have a bulletin board and a white board. The bulletin board can be used to post important messages (for example, laboratory will be down at 11 p.m. for maintenance) or even for student use. The white board: students like to discuss and draw pictures.

Third post the rules of the laboratory. It teaches students responsibility and clearly defines what is expected of whom.

Appendix C: Manuals

The case study of the methodology, NeaSEL, is totally heterogeneous in system types, networks, and O/Ss. A tremendous level of flexibility that allows multiple, simultaneous experiments. The problem with this flexibility is that it leads to overall complexity. To maintain the level of flexibility that gives NeaSEL its level of functionality and to reduce the perceived complexity, I authored a series of manuals that detail the rules, conventions, and guidelines for NeaSEL use.

These manuals were a requirement of the specification to ensure that students and researchers had an accurate and precisely defined source of information. A further requirement was that the manuals be under version control. All these manuals are. While there was no requirement for ISO9000 compliance, many aspects of these manuals obey ISO9000 standards.

This section will briefly describe each of the manuals in the NeaSEL family.

C.1 NeaSEL User's Manual

The first manual that any student, researcher, or professor sees is the *NeaSEL User's Manual*. [59] As the name implies, it is a user's manual. It is not a theoretical document but a nuts and bolts working paper. It consists of two parts. The first describes the architecture and the methodology behind NeaSEL.

The second part details the specific hardware structure, naming conventions, and rules of use of NeaSEL.

It lists everything of importance needed to run the laboratory, configure networks, and rebuild it in case of crashes. For example, one of the problems with this number of heterogeneous systems is how to determine what's inside a system: memory, CPU, cache, hard file size, *etc.*? These parameters are critical for measurements but each vendor uses a different method and even reports the data differently. It would be too time consuming for every experimenter to read every vendors' manuals just to determine how to ascertain a configuration. The *NeaSEL User's Manual* solves this problem by giving the specific commands, with examples, used by each vendor to learn the system configuration.²¹

The *NeaSEL User's Manual* is very detailed and contains considerable information. For these reasons, the laboratory proctors are trained to stress that the user's manual is to be the first source of information when problems arise. To this end, this manual is mandatory reading, with a required sign-off, for all users of NeaSEL.

C.2 NeaSEL Unique Installation Guide

Some systems present unique or unusual problems when installing adapters, memory, software, or the like. Two examples currently in NeaSEL are

²¹ In a previous chapter, it was noted that each system unit had a placard that listed key configuration information. Even with these placards, it was emphasized that experimenters should not trust the placards. This was stated because equipment may change and because we wanted to force students to be very precise in their experimentation methodology.

installing an ISA Ethernet adapter inside an IBM 40P or installing TCP/IP under Windows 3.1.

Given the turnover in students, this experience base rapidly disappears and has to be relearned unless there is a formal documentation process. The *NeaSEL Unique Installation Guide* [60] provides this process. It is a living document that is constantly updated when new problems are found. We have also trained the proctors to have students refer to this manual before doing any installations.

C.3 Research Topics for NeaSEL

The Network and Software Engineering Laboratory can be quite imposing when first entered. There is a profusion of switches, routers, hubs, racks, system units and cables all situated in an orderly but striking fashion. Obviously, the *NeaSEL User's Manual* is a good starting point; but except for a brief section, it does not show what can be done in the laboratory. For students looking for research topics this is a disadvantage.

The *Research Topics for NeaSEL* [61] manual helps in this search by listing research areas that the architecture of NeaSEL enables to be pursued in one room, either by a class or by independent researchers. The manual lists topics of network engineering, software engineering, and combined software engineering with network engineering. It has a large breadth on purpose to open the doors of the mind and to stimulate exploration.

C.4 *NeaSEL Research Presentations*

Every research proposal that uses NeaSEL is kept in a binder for reference by other students or researchers. This is an extension to the *Research Topics for NeaSEL* manual and serves the same purpose.

C.5 *Lessons Learned from NeaSEL Buildup*

This is a list of recommendations that should be followed when building up new computer labs. The goal of any laboratory is to design the utmost in flexibility but keep it under tight control so that the laboratory can be maintained. These recommendations should achieve that goal.

This is a more detailed version of what is shown in Appendix B.

Glossary

802.3 IEEE 802.3 CSMA/CD LAN standard. Virtually identical to Ethernet but does differ in enough minor details so that 802.3 and Ethernet cannot co-exist simultaneously on the same physical media.

A

Alpha DEC's RISC based central processing unit.

American National Standards Institute (ANSI) An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

ANSI American National Standards Institute.

ASYNC Asynchronous.

asynchronous Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. When used in the context of low-speed data communications, especially for PCs, it means there is no predefined timing between the characters sent, *i.e.*, they are not synchronous.

Asynchronous Transfer Mode (ATM) A very high-speed, connection-oriented, fixed-length (48 bytes of data and 5 bytes of overhead), cell-switching scheme that is suitable for data as well as digitized voice and video. It is asynchronous because each cell can be independently addressed to go over different data channels.

ATM Asynchronous Transfer Mode.

B

Berkeley Internet Name Domain (bind) The most common implementation of DNS.

bind Berkeley Internet Name Domain.

C

CASE Computer-aided software engineering.

CAT-5 Category-5, unshielded twisted pair cable. CAT-5 is considered datagrade UTP. It's characteristics are specified out to 100 MHz.

central processing unit (CPU) The part of a computer that includes the circuits that control the interpretation and execution of instructions.

checksum A process used in error detection for data communications in which a mathematical function is applied to some or all of the transmitted data bits. If the written or carried value does not agree with the calculated value, then an error is assumed to have occurred.

client/server A computer system architecture in which clients request a service from a server which provides that service.

cluster computing The use of similar workstations, physically close, attached by a high-speed network, to create the aggregate performance of a supercomputer.

computer-aided software engineering (CASE) A set of computer-based software development tools used to automate certain portions of methodologies, particularly in the development and maintenance cycles.

CPU Central processing unit.

D

DEC 3000 Model 300LX As of the date of this dissertation, a 125 MHz Alpha 21064, 48MB of memory, 2 bus slots, DEC OFS/1 3.2 workstation. Desktop form factor.

distributed computing A paradigm of computing in which independent systems, geographically apart, connected by one or multiple networks, are combined into solving a problem. The systems may be homogeneous or heterogeneous.

DNS Domain Name Service.

Domain Name Service (DNS) The hierarchical system used on the Internet for resolving host names into IP addresses.

E

EISA Extended Industry Standard Architecture.

Ethernet A 10 Mbit baseband LAN on a bus topology. It uses carrier sense multiple access with collision detection (CSMA/CD). Recently, Ethernet has been extended to 100 Mbit and 1 Gbit.

Extended Industry Standard Architecture (EISA) A bus used in PCs that is a 32 bit extension to the standard ISA bus. Runs at 8.33 MHz. Both EISA cards and ISA cards can be plugged into the bus.

F

FCS Fibre Channel Standard.

FDDI Fiber Distributed Data Interface.

Fibre Channel Standard (FCS) A very high-speed, initially defined to 1 Gbps, now up to 4 Gbps, point-to-point connection or connectionless channel based on a switch topology. Both single mode and multi-mode optical fibers have been defined as a media. In its most prevalent form, 1 Gbps, the payload is guaranteed to be 100 MB.

Fiber Distributed Data Interface (FDDI) An American National Standards Institute (ANSI) standard for an 100 Mbit LAN using optical fiber cables. It is a token based ring topology.

file transfer protocol (ftp) In TCP/IP, an application protocol used for transferring files to and from host computers. ftp requires TCP.

firewall A security device that restricts the types of traffic that is allowed to flow between an enterprise's internal network and the Internet. The device can be hardware, software, or a combination. The goal is to protect internal systems from external users.

ftp file transfer protocol.

G

GB Gigabyte.

GFLOPS Billions of floating operations per second. Since there is no standard definition of a floating point operation, this is more of a marketing term than a precise measurement factor.

H

heartbeat At certain time intervals, machines send out a packet stating their status, this is termed a heartbeat. Depending on the frequency and number of machines, this can become a significant overhead on the network.

High-Performance Parallel Interface (HiPPI) A point-to-point, high-speed network technology. Currently defined at 800 Mbits/second or 1.6 Gbits/second. HiPPI uses a 50 pair, shielded twisted pair, cable with a 100 pin connector. Many users refer to this cable as a fire hose.

HiPPI High-Performance Parallel Interface.

I

IBM RISC System/6000 Model 40P As of the date of this dissertation, a 66 MHz PowerPC 601, 64MB of memory, 6 bus slots, AIX 4.1.2 entry workstation. Desktop form factor.

IEEE Institute of Electrical and Electronics Engineers.

Industry Standard Architecture (ISA) A 16 bit bus used in standard IBM-compatible PCs.

InfoExplorer The Hypertext program used by IBM in their RISC System/6000 machines to display help information.

Integrated services digital network (ISDN) A data communication service provided by telephone companies, normally for WAN use. ISDN provides access to both circuit-switched telephone networks and packet-switched services. Basic rate ISDN provides two channels of 64 Kbit each for data and one channel of 16 Kbit for signaling.

Intel Systems As of the date of this dissertation, a 90 MHz Pentium, 64MB of memory, 7 bus slots, workstation. Desktop form factor.

International Organization for Standardization (ISO) An organization of national standards bodies from various countries established to promote the development of standards to facilitate international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity.

International Organization for Standardization 9000 Certification (ISO9000) ISO certification is the endorsement of the management process of quality tracking and reporting of manufacturers and of service providers. It is only a certification of the process; it is not a warranty of the quality of the product. One part of the certification process includes requirements on documentation, such as all pages are numbered, complete version control, update procedures, *etc.*

Internet Protocol (IP) The network protocol that provides connections delivery on the Internet.

IP Internet Protocol.

ISA Industry Standard Architecture.

ISDN Integrated services digital network.

ISO International Organization for Standardization.

ISO9000 International Organization for Standardization 9000 Certification.

K

KB Kilobyte.

L

LAN local area network.

local area network (LAN) A general term for a computer network located on a user's premises within a limited geographical area, hence the name local.

M

MAN Metropolitan Area Network.

Management Information Base (MIB) The variables defining system specific information, such as vendor, model number, and the like, stored by an SNMP agent. This information when acted on by an appropriate program allows a network administrator to determine what is attached to the network.

maximum transfer unit (MTU) The maximum number of bytes that an Internet Protocol (IP) datagram can contain.

MB Megabyte.

Mbit Megabit.

metropolitan area network (MAN) A general term for a high-speed network that covers more territory than a LAN but less than a WAN. It normally applies to an area the size of a city, hence the name metropolitan and is normally considered an area in terms of a few kilometers.

MFLOPS Millions of floating operations per second. Since there is no standard definition of a floating point operation, this is more of a marketing term than a precise measurement factor.

MIB Management Information Base.

MicroChannel The 32 bit bus used by IBM in PS/2's and RISC System/6000 machines. Never gained wide acceptance and has been virtually replaced by the current industry standard PCI bus.

MIPS Millions of instructions per second. A measure of processing performance equal to one million instructions per second, where a specific machine, the DEC VAX-11/70, is used as a basis. This VAX is considered a one MIP machine.

Motorola PowerWorks PCTMT604-100 As of the date of this dissertation, a 100 MHz PowerPC 604, 96MB of memory, 6 bus slots, AIX 4.1.2 workstation. Mini-tower form factor.

MTU Maximum transfer unit.

N

NetWare A network protocol developed by Novell, Inc. which allows file services across networks. Almost exclusively found on PC networks.

network interface card (NIC) The adapter card inside system units that connects or interfaces to the network.

NIC Network interface card.

O

Object Oriented Programming (OOP) A method for structuring programs as hierarchically organized classes which are self contained data and operations. The goal is users can build up complete programs by just copying objects that have already been developed and debugged.

OOP Object oriented programming.

Open Systems Interconnection reference model (OSI model) A model of seven layers that represents a network as a hierarchical structure of functions. Each layer is independent of the others and provides a set of functions that can be accessed from the layer above. The seven layers are (top to bottom): application, presentation, session, transport, network, data link, and physical.

OSI model Open Systems Interconnection reference model.

P

PC personal computer.

PCI Peripheral Component Interconnect.

Peripheral Component Interconnect (PCI) Intel's local bus standard for PCs. Has become the industry standard in the late 1990's for adapter cards.

PowerPC The RISC based processor created by the Apple, IBM, and Motorola alliance.

protocol A set of rules that determines how functional units can access and how they behave when achieving communications. The term functional unit was deliberately chosen to be vague since the units can be either hardware or software, from full systems to NICs, and from O/Ss to microcode.

proxy services Usually used in the context of firewalls. A user behind a firewall cannot contact a user outside the firewall by directly using a specific service. The user behind the firewall must use an equivalent service on the firewall system and that firewall service communicates with the user outside the firewall. This indirectness when combined with more secure services on the firewall provides security for the internal network.

R

RAID Redundant Array of Inexpensive Disks.

RBOC Regional Bell Operating Company.

redundant array of inexpensive disks (RAID) A disk subsystem of more than one disk drive to provide improved reliability, response time and/or storage capacity. The application sees what appears to be a single fast, super reliable disk drive. Data can be spread across multiple drives, termed striping, for reliability or throughput.

Regional Bell Operating Company (RBOC) One of the seven (now five) U.S. holding companies that were formed from the divestiture of AT&T to provide local telephone service in a certain geographical area.

Remote Network Monitoring MIB (RMON) An agreed to procedure for the monitoring of the performance and loading of remote LAN segments.

repeatable When pertaining to measurements practices, all experiments can be rerun numerous times obtaining the same answers except for small measurement errors.

reproducible When pertaining to measurements practices, the test set-up can be broken down and rebuilt from scratch and the same measurement results will be obtained except for a small measurement error.

Request For Comment (RFC) The process for defining new TCP/IP standards in the Internet. The process consists of proposing a standard, making the document available on the Internet, and then requesting comments (hence the name). Each RFC is given a number. The process ends when there are no more comments. At this point, the particular RFC is consisted a standard even though it does not bear that name.

RFC Request for Comment.

RMON Remote Network Monitoring MIB.

S

SCSI Small computer system interface.

Simple Network Management Protocol (SNMP) A protocol, normally over TCP/IP, that is used to examine and change configuration parameters of network attached devices. Uses TCP/IP UDP connectionless transport.

sleeping MIPS Unused processing cycles on idle or lightly loaded machines.

small computer system interface (SCSI) A parallel bus used in PCs and workstations for the attachment of disk drives and similar peripherals.

sniffer A specialized communication computer that mimics network effects and then measures the results. An example of an application is a sniffer can be used to generate a broadcast storm and then measure how well a network handles this event. Typically today, a sniffer is a standard laptop computer running a highly specialized but flexible set of communication programs. While the company Data General was the first and owns the name Sniffer, the term is commonly applied to all such devices, equivalent to how the name Xerox is used.

SNMP Simple Network Management Protocol.

socket The abstraction provided in UNIX that serves as the endpoint for communication between processes or applications. While UNIX specific, many other O/Ss use the term socket when working with applications that communicate on the Internet.

SPEC Standard Performance Evaluation Corporation.

SPECfp92 SPEC benchmark floating point 1992. A floating point benchmark test, introduced in 1992, consisting of fourteen floating point math intensive programs. The results are reported as the geometric mean of the time to run the programs.

SPECint92 SPEC benchmark integer 1992. An integer benchmark test, introduced in 1992, consisting of six integer math intensive programs. The results are the geometric mean of the time to run the programs.

Standard Performance Evaluation Corporation (SPEC) A nonprofit organization, supported by industry, that was formed to "establish, maintain, and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers." SPEC supplies source code of the benchmarks to any system manufacturer so the manufacturer can compile and run the tests on their products.

subnet In TCP/IP, a part of a network that is identified by a portion of the Internet address.

subnet address An extension of the Internet addressing scheme by which a single Internet address can be used for multiple physical networks.

subnet address mask In TCP/IP, a bit mask used by a local system to determine whether a destination is on the same network as the source or if the destination can be reached directly through one of the local network interfaces.

Sun Ultra 1 Model 140E and 170E As of the date of this dissertation, a 167 MHz UltraSPARC, 128MB of memory, 5 bus slots, Solaris 2.5 workstation. Desktop form factor.

T

TCP Transmission Control Protocol.

Transmission Control Protocol (TCP) The connection oriented protocol used on the Internet. It guarantees an error free connection between two devices.

TURBOchannel A proprietary bus used by DEC in many of the company's workstations.

U

Unshielded Twisted Pair cable (UTP) Wiring cable used in telephone and LANs that is unshielded and twisted. The twists are precisely wound to minimize cross-talk; therefore, there are specifications on how long such cable can be.

UTP Unshielded twisted pair cable.

W

WAN Wide area network.

wide area network (WAN) A data communication network that spans a large distance, such as a state or country. Normally a public carrier provides this service, not an individual company.

World Wide Web The network of servers on the Internet, each of which has one or more home pages, which provide information and hypertext links to other documents on that and other servers.

Bibliography

- [1] R. Orfali, D. Harkey, and J. Edwards, *The Essential Distributed Objects Survival Guide*. John Wiley & Sons, Inc., New York, 1996.
- [2] T. Reenskaug, *Working with Objects*. Manning Publications, Greenwich, CT, 1996.
- [3] O. Sims, *Business Objects*. McGraw-Hill, New York, 1994.
- [4] "IBM RISC System/6000 Product Introduction", IBM Presentation, Austin, TX, June 1996.
- [5] K. Pieper and K. Leonard, "FDDI...The Backbone of Choice", *Network World*, vol 9, no 39, pp 11-12, Sep 28, 1992.
- [6] L. Reiss, *Local Area Networks with Microcomputer Experiments*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
- [7] J. Loins, "Computer Networks for Students", *Australian Comput. Sci. Assoc. Newslett.*, vol 3, no 1, pp 37-48, Jul 1990.
- [8] C. McDonald, "A Network Specification Language and Execution Environment for Undergraduate Teaching", *SIGCSE Bull.*, vol 23, no 1, pp 25-34, Mar 1991.
- [9] P. Tyman, "VNET: A tool for teaching computer networking to undergraduates", *SIGCSE Bulletin*, vol 23, no 1, pp 21-24, Mar 1991.
- [10] A. Umar, "Balancing Theory and Practice in a Course on Networks and Distributed Systems," *ISECON '87 Proceedings of Sixth Annual Information Systems Education Conference* San Francisco, CA, Oct 31 - Nov 1, 1987, pp 225-230.
- [11] D. Finkel, and S. Chandra, "NetCP-A Project Environment for an Undergraduate Computer Networks Course," *Twenty-fifth SIGCSE Technical Symposium on Computer Science Education, SICCSSE Bulletin*, Phoenix, AZ, Mar 10-11, 1994, pp 174-177.

- [12] W. Dewar, and S. Sethi, "A Laboratory for Teaching Computer Networks," *IEEE Transactions on Education*, vol 38, no 2, pp 145-149, Feb 1995.
- [13] T. Saadawl and A. Abdelmonem, "Ed-Net; A Multi-Protocol Educational Local Area Network," *IEEE Communications*, vol 25, no 10, pp 34-40, Oct 1987.
- [14] F. Vallejo, M.G. Harbour, and J.A. Gregorio, "A Laboratory for Microprocessor Teaching at Different Levels," *IEEE Transactions on Education*, vol 35, no 3, pp 199-203, Mar 1992.
- [15] M. Sherman and A. Mark, "Using Low-Cost Workstations to Investigate Computer Networks and Distributed Systems," *IEEE Computer*, vol 10, no 6, pp 32-40, June 1983.
- [16] M. Sherman, Engr, TriCount, Jan 5, 1995, telephone conversation with the author, Austin, TX.
- [17] MIT University Catalog [Web page] May 1997; <http://web.mit.edu> [Accessed June 18, 1997].
- [18] CMU Catalog [Web page] Mar 1997; <http://www.cmu.edu> [Accessed June 18, 1997].
- [19] S. Mengel and C. Bowling, "Supporting Networking Courses with a Hands-on Laboratory," *Frontiers in Education Conference, Engineering Education for the 21st Century, 1995*, Atlanta, GA, Nov 22-26, 1995, pp 4C2.20-4C2.24.
- [20] G. Finley, *et. al.*, "Computer Networks and Data Communications: A Laboratory Focus," *Proceedings of the 1997 28th SIGCSE Technical Symposium on Computer Science Education*, San Jose, CA, Feb 27-Mar 1, 1997, pp 1-30.
- [21] M. Levin, "Prototype for a Data Communication Laboratory," *Proceedings of the 1997 28th SIGCSE Technical Symposium on Computer Science Education*, San Jose, CA, Feb 27-Mar 1, 1997, pp 179-183.

- [22] "The Switching Advantage". Kalpana, The EtherSwitch Company, Sunnyvale, CA, Spring 1994.
- [23] P. Mackapetris, *Domain Names—Implementation and Specification*, RFC 1035, <http://www.cis.ohio-state.edu/rfc/rfc1035.txt>, 1987.
- [24] Y. Rekhter, *et. al.*, *Address Allocation for Private Internets*, RFC 1597, <http://www.cis.ohio-state.edu/rfc/rfc1597.txt>, 1994.
- [25] G. F. Pfister, October 19, 1992, letter to author.
- [26] G.F. Pfister, *Clusters, In Search of a New Paradigm*. Prentice-Hall, Englewood Cliffs, NJ, 1995.
- [27] K.W. Plessman, "Basics on a System of a Pool of Processors", *Proceedings of IFCON '84*, Tokyo, Japan, October 1984, pp 384-394.
- [28] K.W. Plessmann, "A distributed computer system on the basis of the pool processor concept," *Distributed Computer Control Systems 1983. Proceedings of the Fifth IFAC Workshop*. Sabi-Sabi, South Africa, May 18-20, 1983, pp 143-155.
- [29] J.M. Bacon, I.M. Leslie, and R.M. Neddham, "Distributed Computing with a Processor Bank", *Progress in Distributed Operating Systems and Distributed Systems Management. European Workshop Proceedings 1989*, Berlin, April 18-19, 1989, pp 147-161.
- [30] S. J. Mullender and A.S. Tanenbaum, "The design of a capability based distributed operating system," *Computer Journal*, vol 29, no 4, pp 289-300, April 1986.
- [31] R. Orfail, *et al.*, *Essential Client/Server Survival Guide*. Van Nostrand Reinhold, New York, 1994.
- [32] G.F. Coulouris and J. Dollimore, *Distributed Systems, Concepts, and Design*. Addison Wesley, New York, 1989.
- [33] D.A. Nichols, "Using idle workstations in a shared computing environment", *Operating Systems Review*, pp 5-12, Nov 1987.

- [34] M.J. Litzkow, M. Livny, and M.W. Mutka, "Condor--A hunter of idle workstations," *Proceedings 1988 Conference on Distributed Computing Systems*, pp 104-111.
- [35] T. Tannenbaum and M. Litzkow, "The Condor Distributed Processing System." *Dr. Dobbs Journal*, vol 20, no 2 pp 40, 42-44, Feb 1995.
- [36] L. Kleinrock and W. Korfhage, "Collecting unused processing capacity: An analysis of transient distributed systems", *Proceedings 1989 Conference on Distributed Computing Systems*, pp 482-489.
- [37] N.M. Theimer and K.A. Lantz, "Finding idle machines in a workstation based distributed system", *Proceedings 1988 Conference on Distributed Computing Systems*, 1988, pp 112-122.
- [38] M.W. Mutka, and M. Livny, "Profiling Workstations Available Capacity for Remote Execution", *Performance '87, Proceedings of the 12th IFIP WG73 Symposium on Computer Performance*, Brussels, Belgium, December 7-9, 1987, pp 310-325.
- [39] M.W. Mutka and M. Livny, "Scheduling Remote Processing Capacity in a Workstation Processor Bank Computing System," *Proceedings of the 7th International Conference of Distributed Computing Systems*, Berlin, West Germany, September 21-25, 1987, pp 2-9.
- [40] H. Mehrpour and S.S. Sethi, "Performance of TCP/IP for Real-time Traffic on an FDDI Backbone Interconnection," *Communication '92, Communications Technology, Services and Systems*, Sydney, 20-22 Oct 1992, pp 213-218.
- [41] D. O'Shea, "Interoperability: Is It All Talk?" *Telephony*, pp 10-11, Sep 25, 1995.
- [42] M.J. Turner, "ATM on the Edge". *Network World*, vol 11, no 36, pp 51-53, Sep 5, 1994.
- [43] M. Arnett, *et. al.*, *Inside TCP/IP*. New Riders Publishing, Indianapolis, IN, 1995.
- [44] J. Ashworth, *The Naming of Hosts, RFC 2100*, <http://www.cis.ohio-state.edu/rfc/rfc2100.txt>, 1997.

- [45] D. Libes, "Choosing a Name for Your Computer", *Communications of the ACM*, vol 32, no. 11, pg 1289, November 1989.
- [46] M. Lotter, *et. al.*, "Domain Name Survey, Jan 1997", namedroppers@internic.net, 1997.
- [47] M. Wong, *et. al.*, "Cool Hostnames" [Web page], <http://www.seas.upenn.edu/~mengwong/coolhosts.html> [Accessed July 1997].
- [48] B. Carpeter, J. Crowcroft, and Y. Rekhter, *IPv4 Address Behavior Today, RFC 2101*, <http://www.cis.ohio-state.edu/rfc/rfc2101.txt>, 1997.
- [49] E. Gerich, *Guidelines for Management of IP Address Space, RFC 1466*, <http://www.cis.ohio-state.edu/rfc/rfc1466.txt>, 1993.
- [50] B. Fraser, editor, *Site Security Handbook, RFC2196*, <http://www.cis.ohio-state.edu/rfc/rfc1466.txt>, 1997.
- [51] C. Braun, "Making Things Real in Electronics Laboratories," *Frontiers in Education Conference, Engineering Education for the 21st Century*, 1995, Atlanta, GA, Nov 22-26, 1995, pp 4C2.10-4C2.13.
- [52] J. Hatfield, "A Freshman Electrical Engineering Course and Laboratory for All Engineering Majors," *Frontiers in Education Conference, Engineering Education for the 21st Century*, 1995, Atlanta, GA, Nov 22-26, 1995, pp 4C2.1-4C2.4.
- [53] V. Hatfield and D. Meinhert, "Characteristics of Information Systems Training Programs," *Proceedings of the 1996 27th Annual Meeting of the Decision Sciences Institute*, Orlando, FL, Nov 24-26, 1996, pp 784-786.
- [54] S. Wunnava, "Model LAN Scheme and Compressed Data Transfers," *Proceedings of the 1997 IEEE Southeastcon*, Blacksburg, VA, Apr 12-14, 1997, pp 188-190.
- [55] S.K. Singhal, *et. al.*, "InVerse: Designing an Interactive Universe Architecture for Scalability and Extensibility", *Proceedings of the 1997 6th IEEE International Symposium on High Performance Distributed Computing*, Portland, OR, Aug 5-8, 1997, pp 61-70.

- [56] S. Dempsey, *et. al*, "Predicting FDDI Computer Network Performance Using a Calibrated Software Simulation Model," *1997 IEEE International Performance, Computing, and Communications Conference*, Phoenix, AZ, Feb 5-7, 1997, pp 1-9.
- [57] M.W. Dixon, T. McGill, and J. Karlsson, "Using a Network Simulation Package to Teach the Client/Server Model", *SIGCSE Bull.*, vol 29, no 3, pp 71-73, Sep 1997.
- [58] A. Schill and T. Hutschenreuther, "Architectural Support for QoS Management", *Computer Communications*, vol 20, no 6, pp 411-419, July 25, 1997 .
- [59] D.D. Daniel, *NeaSEL User's Manual*. University of Texas at Austin, Austin, TX, 1997.
- [60] D.D. Daniel, *Unique Installation Guide for NeaSEL*. University of Texas at Austin, Austin, TX, 1996.
- [61] D.D. Daniel, *Research Topics for NeaSEL*. University of Texas at Austin, Austin, TX, 1997.
- [62] S.A. Shaikh, *Hardware Acceleration of Concurrent Fault and Design Error Simulation*. Ph.D. diss., University of Texas at Austin, 1996.
- [63] S. Goldenberg, *Concurrent Fault and Design Error Simulation in a Distributed Computing Environment*. Master's thesis., University of Texas at Austin, 1996.
- [64] D. Kacprzak, *The Audio/Video Stream Service or CORBA Framework—A Tool for Distributed Video on Demand System Development*. Master's thesis., University of Texas at Austin, 1996.
- [65] M. Shnier, *Dictionary of PC Hardware and Data Communications Terms*. O'Reilly & Associates, Inc., Sebastopol, CA, 1996.
- [66] G. McDaniel, *IBM Dictionary of Computing*. McGraw-Hill, Inc., New York, NY, 1993.

Vita

Dwight David Daniel was born at Fort George G. Meade, Maryland, on May 4, 1948, the son of Aubrey and Alaska Daniel. He attended The John Hopkins University from the fall of 1966 through the spring of 1970. He was awarded a Bachelor of Engineering Science in May 1970 and at the same time was commissioned a Second Lieutenant in the United States Army.

Dwight immediately entered the University of Pennsylvania under a Ford Foundation Fellowship, where he obtained a Master's of Science in Engineering in Electrical Engineering in May of 1972.

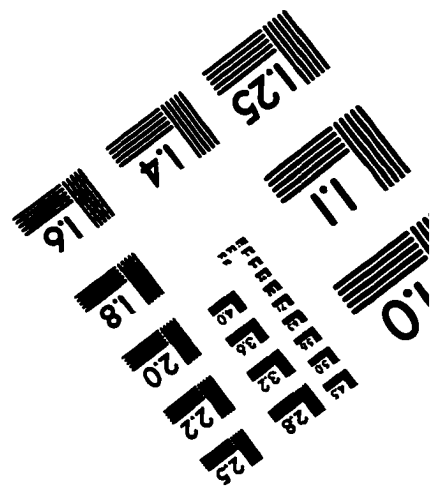
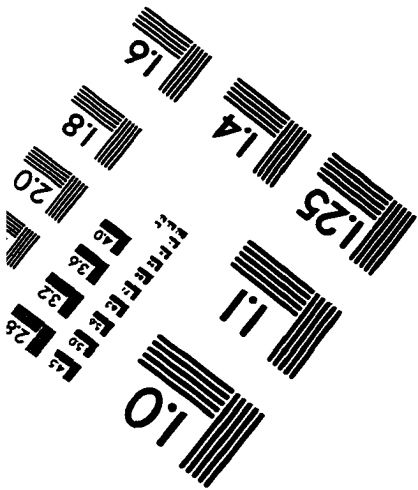
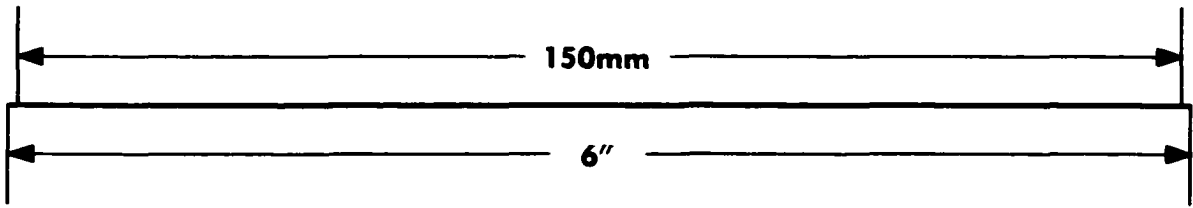
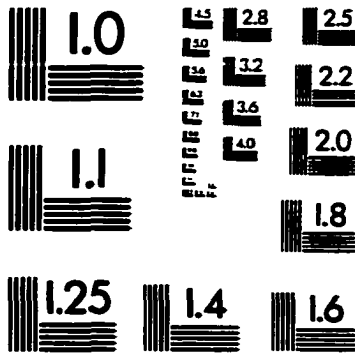
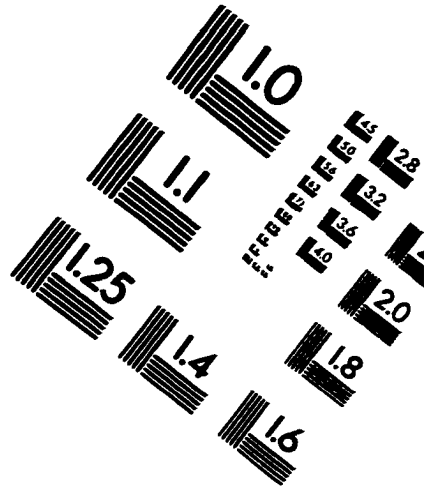
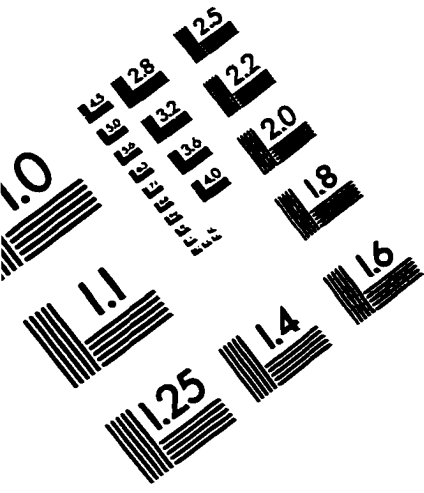
Dwight joined IBM in 1974 where he has been continuously employed either in IBM facilities at Endicott, New York or Austin, Texas. While in New York, Dwight earned a Master's of Science from the State University of New York at Binghamton in May 1978. In the Spring of 1992, he joined the Graduate School of Engineering at the University of Texas at Austin to pursue doctoral studies. He was awarded a Ph.D. in December 1997.

Dwight is still extremely active in the Army Reserves, obtaining the rank of Lieutenant Colonel and has been awarded the Army Commendation Medal.

**Permanent address: PO Box 82263
Austin, Texas 78708-2263**

This dissertation was typed by the author.

IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved