Design and Implementation of a Data Collection and Exchange System for SNEWS

by

Ronnie G. Misra

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

May 24, 2002

Copyright © 2002 Ronnie G. Misra. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author ........................................

Department of Electrical Engineering and Computer Science

May 24, 2002

Certified by ........

Kate Scholberg

Assistant Professor, Department of Physics

Thesis Supervisor

Accepted by .......

Arthur C. Smith

Chairman, Department Committee on Graduate Theses
Design and Implementation of a Data Collection and Exchange System for SNEWS

by

Ronnie G. Misra

Submitted to the
Department of
Electrical Engineering and Computer Science

May 24, 2002

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis documents the redesign and implementation of the SuperNova Early Warning System (SNEWS) for the collection and exchange of neutrino emission data. SNEWS is designed to provide automated alarms to the astronomical community based on the coincidence of neutrino burst signals observed at experiments around the world. This thesis addresses the issues of bidirectional communication, anti-coincidence and consistency, network status monitoring, multi-server protocol formalization, and security/robustness in the SNEWS project.

Thesis Supervisor: Kate Scholberg
Title: Assistant Professor, Department of Physics
Dedication

This work is dedicated to the victims of terror on September 11th 2001. May they find peace under the stars.
Acknowledgments

I sincerely thank everyone who made my stay at MIT such a wonderful experience.

I thank Professor Scholberg for her supervision of my research. Without her, I would have no idea what a supernova was. Anne Hunter and Vera Sayzew helped me tremendously in dealing with MIT’s red tape – I’ve lost track of how many times Anne gave me good advice, and how many forms she signed for me!

I owe a lot to my hallmates, THiRD EAst at East Campus, and particularly Robbie Buckingham, Lawrence Chao, Djuna Copley-Woods, Siddhartan Govindasamy, Arif Karim, Sarah Kershaw, Sarah Kolitz, Lindsay Kong, Donald Lee, Joshua Marron, Diane Schnebly, Peter Sun, and Si Wong for their support and friendship. I thank my friends and housemates Kalpak Kothari, Kailas Narendran, Katie Fillion, and James Robertson, for being understanding when my research took longer than I expected. I also thank my friends Nicole Traxler-Wright, Helen Yu, Gloria Jen, and Seung Rhee for reminding me that I'm supposed to be in California!

I am thankful to all of my teachers and mentors from Gallup, who always believed in me. I would especially like to thank Dr. Wilson, Dr. and Mrs. Milligan, and Mr. Hazelton for their guidance.

Most of all, I thank my parents Rachel and Deb and my sister Sheila for their love and emotional support. Without their encouragement, I would not have been able to make it through MIT.
3 Research Topics

3.1 Multithreading
3.2 Anti-coincidence
3.3 Network Status
3.4 Multi-server Protocol
3.5 Security and Robustness

4 Designing a New Architecture

4.1 Server Design
  4.1.1 Alarm Queue
  4.1.2 Anti-coincidence Query
  4.1.3 Inter-server Communication
4.2 Client Design
  4.2.1 Threads and Anti-coincidence Response

5 Implementation

5.1 Coding Practices
  5.1.1 Modularity and Encapsulation
  5.1.2 Portability
  5.1.3 Backward Compatibility
5.2 OpenSSL and Threads
5.3 Server Implementation
  5.3.1 Client Connections
  5.3.2 Alarm Queue
  5.3.3 Anti-coincidence
  5.3.4 Inter-server Communication
  5.3.5 Web-based Summary
5.4 Client Implementation
  5.4.1 Client-side Alarm Queue
  5.4.2 Anti-coincidence Response
6 Conclusion

6.1 Summary of Work ........................................... 37
6.2 Future Work ................................................. 38

A User’s Guide .................................................. 39
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>SN1987A, Before and After</td>
<td>10</td>
</tr>
<tr>
<td>1-2</td>
<td>IMB Neutrino Footprint, SN1987A</td>
<td>13</td>
</tr>
<tr>
<td>2-1</td>
<td>Client/Server Architecture</td>
<td>17</td>
</tr>
<tr>
<td>3-1</td>
<td>Multithreading</td>
<td>23</td>
</tr>
<tr>
<td>5-1</td>
<td>OpenSSL Wrapper</td>
<td>31</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter introduces the physics of supernovae. The SNEWS collaboration aims to create an automated system for detecting supernovae by their expected core collapse neutrino signal. The motivations and goals for such a system are described in this chapter.

1.1 What is a Supernova?

After a star has exhausted its fuel and can no longer maintain the nuclear reactions required to sustain its structure, it collapses upon itself. The star may subsequently explode and release huge amounts of energy and matter into space. This explosion is called a supernova, and is one of the most energetic stellar events. Supernovae are relatively rare events, occurring about once a century in our galaxy [5].

1.2 Neutrino Bursts

When a star collapses into a supernova, it emits some of its energy in the form of visible light, but the majority (99%) is emitted in the form of neutrinos [3]. (In fact, neutrinos are emitted even if a supernova does not occur after core collapse.) In 1987, the supernova SN1987A was observed in the Large Magellanic Cloud. Two neutrino detectors, Kamiokande II and IMB, reported unambiguous observations of a neutrino
Figure 1-1: SN1987A, Before and After
©Anglo-Australian Observatory
Photograph by David Malin
burst about 2.5 hours before the first observed light, and about 1.5 hours before the arrival of the first observable photons.

1.2.1 The Neutrino Signal

The neutrino burst associated with a supernova is expected to last on the order of tens of seconds, with about 50% of the activity occurring in the first two seconds [1]. Burrows, et. al. describe a detailed model of the neutrino emissions from a supernova, and use this model to predict the response at various neutrino detectors [2].

1.2.2 The Physics of Neutrino Detection

Neutrinos are particles with no charge and very little mass. They react with other matter only through weak nuclear force; because they have no charge, they do not cause ionization. There are a number of different types of neutrino detectors [1]:

- **Scintillation Detectors:**
  A scintillation detector detects the results of the charged-current inverse beta decay reaction:

\[
\bar{\nu}_e + p \rightarrow e^+ + n.
\]

In this reaction, a proton becomes a neutron and a high-energy positron is emitted. The active organic hydrocarbons in the scintillator emit light when excited by the passage of this positron. Some examples of scintillation detectors are LVD, MACRO, Borexino, and KamLAND.

- **Water Čerenkov Detectors:**
  Water Čerenkov detectors can detect the charged-current inverse beta decay reaction described above, but they are also sensitive to neutrino-electron scattering reactions:

\[
\nu_x + e^- \rightarrow \nu_x + e^-.
\]
\( \nu_e \) can be any flavor of neutrino. A Čerenkov detector consists of a large volume of water viewed by a dense array of phototubes. These phototubes capture the Čerenkov radiation released when charged particles travel through water at speeds exceeding the speed of light in water. Two water Čerenkov detectors are Super-K and IMB.

- **Heavy Water Detectors:**
  Heavy water detectors detect the breakup of deuterons due to interaction with neutrinos, by recording emissions of Čerenkov photons or neutrons. SNO is one such detector.

- **Long String Water Čerenkov Detectors:**
  A long string water Čerenkov detector uses long strings of phototubes suspended in clear water or ice to detect Čerenkov light. AMANDA is a detector of this type.

- **High Z/Neutron Detectors:**
  High Z/neutron detectors are sensitive to high energy neutrinos; using a target mass of a high Z material increases the likelihood of neutrino interactions. OMNIS is a high Z/neutron detector.

Although I discuss neutrino detection throughout this paper, it is important to note that there are supernova-sensitive detectors that do not measure neutrino bursts. In particular, gravitational wave detectors have the capability to detect the gravitational wave signal from an asymmetric supernova explosion. In fact, this signal may be more prompt than the neutrino signal. Two gravitational wave detectors are LIGO and VIRGO.

### 1.3 Motivation for Early Alerts

Since the burst of neutrinos associated with a supernova can be observed hours before any electromagnetic radiation, the detection of this burst provides a unique oppor-
Data from neutrino detectors can be used to provide early warnings of the occurrence of a supernova to astronomers. An early alert allows astronomers to observe the initial stages of a supernova explosion, as well as the resulting effects on the star's immediate surroundings. However, individual detectors are subject to false alarms due to noise, detector failure, etc. An individual experiment can issue an alarm only after a human operator verifies the data.

1.4 SuperNova Early Warning System

The SuperNova Early Warning System (SNEWS) collaboration is a team of experimenters representing several neutrino detectors. The primary objective of the SNEWS project is to develop an automated system for alerting astronomers to the occurrence of supernovae, based on neutrino burst detections. As mentioned above, such automation is impossible given data from a single experiment. However, using simultaneous
data from multiple independent experiments allows for the detection of coincidences. Requiring a coincidence between multiple experiments reduces the background noise, so such coincidences can be used to generate a fully automated alert system [1].

1.4.1 SNEWS Requirements

In order to formulate the SNEWS requirements, it is important to understand what the astronomical community wants from such an alert system. Three important properties have been identified [4], the “three P’s”:

- **Prompt:** Alerts must be as prompt as possible to allow the monitoring of the early phases of collapse. The neutrino burst from an exploding supernova can occur several hours before first light, so the SNEWS project has the opportunity to provide warnings hours in advance of the first observable photons.

- **Positive:** An automated alert system must have a very low false alarm rate, since bogus alerts lead to a waste of resources and hurt the credibility of the system. Requiring a coincidence of neutrino signals from multiple independent detectors allows for a low false alarm rate. While false coincidences can still be expected to occur, the average false alarm rate can be made sufficiently small by adjusting the alarm sensitivities at the individual detectors. Additionally, anti-coincidence routines can be used to query individual detectors to generate a full summary of recorded neutrino activity at the experiments, which can be used to further suppress false coincidences by requiring a coincidence between all experiments which should have been sensitive.

- **Pointing:** Any information indicating the location of a supernova in the sky is extremely valuable. Directional information may be obtained by using knowledge about asymmetries of the specific experiments, or by triangulation of the detector signals. Triangulation requires events from detectors to be tagged with timestamps of millisecond precision, and so is unlikely to be very accurate. However, even an imprecise triangulation could be helpful.
1.4.2 Coincidences and Increased Sensitivity

As mentioned above, requiring a coincidence between multiple experiments provides a reduced false-positive rate for alerts. However, there is another advantage of tracking coincidences: individual experiments can reduce their alarm thresholds. This will increase the background noise for each experiment, but such noise is unlikely to be correlated between widely separated laboratories [1].

1.4.3 Alerting the Astronomical Community

Alerts will be distributed to an electronic mailing list of interested communities. These alerts will provide event times, as well as any directional information that may be available. In actuality, the error box for such information is likely to be several degrees or larger. However, the wide-angle viewing capabilities of satellites, small telescopes, and the community of amateur astronomers should allow for quick pinpointing of optical events [4].
Chapter 2

Background

This chapter provides a detailed description of the existing SNEWS architecture for data exchange between neutrino detector experiments and the central server. It also introduces some of the problems and shortcomings of this original design.

2.1 Existing Implementation

The existing SNEWS architecture consists of individual detectors running client code, and a single central server responsible for detecting coincidences and issuing alerts.

2.1.1 Client/Server Architecture

The prototype SNEWS software was written in standard C by Alec Habig, Kate Scholberg, and Corrinne Mills. The server runs in a loop waiting for data from a client. Clients can send one of three types of packets:

- **Alarm**: An alarm packet is sent when a client detects a neutrino burst. Upon receipt of an alarm packet from a client, the server inserts the alarm into an alarm queue sorted by event time. It then searches for a coincidence within a given time window (10 seconds by default). If two or more different experiments have sent alarms within this time window, the server generates an alert.
alert to astronomical community

SNEWS server

10 second coincidence detection

Figure 2-1: Client/Server Architecture

- **Retraction:** When the server receives a retraction packet, it removes all alarms from the given experiment within the given time window.

- **Ping:** A client sends a ping packet in order to verify its connection to the server. No action is taken by the server upon receipt of a ping packet.

Time stamps are specified with millisecond precision, and all times are in UT (based on the Greenwich Mean Timezone).

To eliminate bias, no experiment should be aware of the results from any other experiment before issuing an alarm. Therefore, the individual experiments generate and deliver neutrino burst alarms automatically without human intervention. This information will not be disseminated to the other experiments. Automation also allows coincidences to be detected more promptly than would be possible otherwise.

### 2.1.2 Security

Security is an important concern for the SNEWS collaborators. The SNEWS prototype uses the OpenSSL library to send packets from individual clients to the central server. This library requires a source of cryptographically secure pseudo-random numbers. Some operating systems provide a "/dev/random" device for such a random
source. On other platforms, a separate entropy-gathering package must be installed to provide random numbers.

Two types of security are desired:

- **Privacy**: Neutrino burst alarms are to be delivered unchecked to the SNEWS server. Making such unchecked data available to the general astronomical and neutrino communities is undesirable. The SNEWS architecture guarantees that the raw alarm stream will not be visible to external parties by encrypting all traffic between the clients and the server.

- **Authentication**: It is also important that alarms cannot be forged by hackers. To this end, the SNEWS architecture makes use of OpenSSL’s packet signing mechanisms. Each client is issued a certificate; the server verifies that the packet streams from each client are signed with a valid certificate.

### 2.1.3 Portability

The original SNEWS prototype was written with portability in mind. It converts all numeric data into “network order” before sending it on the wire to properly support clients and servers with mixed architectures. The code also only makes calls to standard C functions. It has been successfully compiled and tested on AIX, IRIX, Linux, Digital UNIX, Solaris, and OpenVMS [1].

### 2.1.4 Library API

The individual detectors have different procedures for receiving data. Additionally, these experiments perform other tasks besides sending alarms to the SNEWS network. Therefore, the SNEWS routines are provided as a library to link against. This library provides functions for sending alarm, retraction, and check packets. Individual detectors can then run custom software for gathering and processing data, and make appropriate SNEWS library calls to send packets to the SNEWS network. This library must provide support for development in C and FORTRAN.
The library provides a simple interface for sending packets, exporting three functions. The C API for these functions is outlined below. The types u16 and u32 are system-dependent typedefs for unsigned 16-bit and 32-bit integers, respectively. Each time parameter to these functions is specified as a two-integer array: the first integer is a six-digit number of the format “MMDDYY”, and the second integer is a six-digit number of the format “hhmmss”. Each function also takes an experiment parameter, indicating the client’s experiment ID number; the values for these IDs are currently hard-coded.

- int gccli(int *datetime, u16 level, u32 significance, u16 experiment);

  Sends an alarm packet, indicating the detection of neutrinos at datetime, with indicated level and significance. The server will filter alarms based on level, discarding all alarms that fall below some specified threshold. Significance numbers are not used by the server itself, but will be included in any alerts generated by the server.

- int gcretcli(int *starttime, int *endtime, u16 experiment);

  Sends a retraction packet, instructing the server to remove all alarms received from the given experiment between starttime and endtime.

- int gcpingcli(int *datetime, u16 experiment);

  Sends a ping packet, used to verify the status of the network connection between the client and the server.

### 2.2 Design Shortcomings

Although prototype SNEWS software exists and is in operation today, the current architecture is too simple to be robust and scalable. Instead, it was written as a proof of concept. Additionally, anti-coincidence routines were never implemented in the prototype.
2.2.1 Scalability and Extensibility

The original SNEWS software was not designed with extensibility in mind. The packet protocol cannot easily be modified to support new features such as anti-coincidence queries. Additionally, the code is not very modular, so a redesign of the protocol would require significant changes to many different parts of the software.

2.2.2 Security and Robustness

As mentioned above, security is an important issue for the SNEWS collaboration. While the prototype does make use of OpenSSL features for signing packets, it does insufficient checks of client certificates. In particular, it does not provide a means for invalidating a certificate in the event that it is compromised.

A related issue is software robustness. To minimize downtime, the SNEWS code should be resilient to bugs in the client and server code. Currently, the server does not perform any sort of input verification. It is possible that invalid packets due to buggy clients or malicious attacks could cause the server to crash. Similarly, some error conditions are silently ignored by the SNEWS prototype code. It is difficult to guarantee bug-free software. Therefore, the SNEWS software should have a mechanism for recovering from crashes; this is another feature missing from the SNEWS prototype.

2.2.3 Anti-coincidence

When a coincidence alert is generated, it should contain as much information about each of the individual experiments as can be obtained. This requires an anti-coincidence mechanism – the server should contact each experiment and collect data pertaining to the given coincidence time-window. The original SNEWS prototype does not provide this feature; in fact, the architecture makes it difficult to implement such functionality.
Chapter 3

Research Topics

In this chapter, I describe the goals of my thesis project, outlining the major topics of my research. This project focuses on the continued development of SNEWS, addressing five main areas:

- **Multithreading:** The SNEWS library interface requires that library calls be tightly coupled to the main data-collection loop of a neutrino detector. The lack of multithreading could cause delays in data collection due to network bottlenecks or blocking system calls. Additionally, implementing two-way communication between an experiment and the server is difficult in a single-threaded architecture.

- **Anti-coincidence:** In the event of a coincidence, the server should be able to query each of the individual experiments in an attempt to further reduce the possibility of false alarms. Routines for anti-coincidence detection did not exist in the original system, and as mentioned above, implementation of such routines would be difficult with the original architecture.

- **Network Status:** In the original system, the SNEWS server had no way of monitoring the status of the network connection to the individual experiments; similarly, clients could not track the network status of a server.
• **Multi-server Protocol:** The protocol for data exchange between clients and a server in the original SNEWS prototype was not formalized; also, there was no support in the design for multiple redundant servers, and no support in the protocol for inter-server communication.

• **Security and Robustness:** While security is a stated goal of the SNEWS collaboration, the original system lacked input verification and code robustness. Errors in packet data or malformed packets could cause the server and/or clients to crash. Also, while the code provided packet-level encryption using the Secure Socket Layer, this implementation did not provide a means to retract compromised certificates.

### 3.1 Multithreading

The original SNEWS code makes it difficult to implement two-way communication between experiments and the central server. Two-way communication requires the client software to manage and respond to packets from the server while simultaneously collecting data from a neutrino detector. A single-threaded client would have to poll a socket for network traffic from the server, while simultaneously polling the detector for new neutrino data. The use of multithreading and interprocess communication allows for a simpler interface for client software, because support for two-way exchange of information between the server and the clients can be implemented in the library, transparent to the caller.

In this multithreaded architecture, one thread is responsible for data collection from the neutrino detector, while another thread manages all data exchanges with the server. Thus, when a server requests anti-coincidence information from a client (described below), neutrino data collection will not be interrupted.
Figure 3-1: Multithreading
3.2 Anti-coincidence

While a coincidence between multiple experiments in the SNEWS network decreases the likelihood of false alarms, there still exists the possibility of bogus alarms. To further reduce the false alarm rate, anti-coincidence information is used. On detection of a coincidence between multiple experiments, a SNEWS server queries each experiment to determine whether it was up and sensitive to a signal, and if so, whether it received a signal.

This required significant changes to the design of the SNEWS architecture. As mentioned above, the original SNEWS code did not easily support two-way data interchange between a client and a server. With the migration to a multithreaded SNEWS library, the following architecture for anti-coincidence detection is possible:

- A “data-collection” thread acquires data from the neutrino detector.
- A “network” thread manages packets from the server.

When the data-collection thread detects a supernova-like neutrino pattern, it issues an alarm to be sent to the server. When the network thread receives an anti-coincidence message from the server, it generates a response based on data from the data-collection thread. All communications between these threads are performed using standard sockets, and as such, should be fairly portable to various operating systems. It is critical for the implementation of this anti-coincidence function to hide the threading details under a simple programming interface.

3.3 Network Status

With the migration to a multithreaded architecture comes the ability for sophisticated network status tracking, if desired. A persistent connection will be maintained between a client and the server. A heartbeat can be employed to monitor the status of this connection, or to measure the “ping time”. A client can then attempt to reconnect to the server (or another one); a server could collect statistics about the network
connections to the clients, and use this data in its coincidence calculations. Since the neutrino data itself is sensitive, any such tracking would be limited to network status.

### 3.4 Multi-server Protocol

As new features are added to the SNEWS code, the protocol has evolved to support them. The current incarnation of the SNEWS protocol is thus the result of piecemeal additions rather than planned design. In redesigning the SNEWS code, some rearchitecting of the protocol was necessary. For example, in order to support an extensible network of clients and multiple servers, a formal design of server interactions was required. Packet version numbers make it easier to support significant modifications to the protocol in the future.

### 3.5 Security and Robustness

In attempting to minimize the chance of false alerts due to spoofed client alarms, it is important to use secure channels for data transmission between the clients and servers. The Secure Sockets Layer (the OpenSSL toolkit) can be utilized to maintain data integrity and to prevent unauthorized access to neutrino data. Additionally, the issue of data authentication can be addressed by using digital signatures to verify the source of data.

Software robustness is also important for SNEWS. To minimize downtime, care must be taken that illegal memory accesses do not cause either the client or server code to crash. Currently, however, no input verification is performed. Similarly, error codes from library functions are often silently ignored. A redesign of the SNEWS software will have to take such issues into consideration.
Chapter 4

Designing a New Architecture

This chapter describes the design for my new SNEWS software. Some of the details of this architecture changed during implementation – these changes are also discussed in this chapter.

4.1 Server Design

The basic design of the server alarm queue remains mostly unchanged from the prototype SNEWS implementation. However, the routines for anti-coincidence and inter-server communication were designed from scratch.

4.1.1 Alarm Queue

The SNEWS server maintains a queue of neutrino burst alarms that it has received. When a new alarm packet is received, the alarm is inserted into the queue. Upon receipt of a retraction packet, the server searches for matching alarms (alarms in the given time window, from the specified client) and removes them from the queue. Every time a new alarm is added to the queue, the server searches for a coincidence.

When the alarm queue is modified, the entire queue is stored to disk. Upon startup, the server reads the previously stored queue from disk. If the server crashes and restarts, it will recover the state of the queue before the crash.
4.1.2 Anti-coincidence Query

Upon detection of a coincidence as described above, a SNEWS server must be able to query the individual clients to retrieve anti-coincidence information. The server does this by sending a coincidence packet to each of the connected clients.

Each client should respond with all its observations in the time window surrounding the discovered coincidence. The server will wait for a given amount of time (20 seconds by default) to receive anti-coincidence responses. This allows the server to verify that all experiments that should have detected a neutrino burst actually did, reducing the probability of false alarms. The server will then generate an alert including the anti-coincidence information it has collected from all experiments.

4.1.3 Inter-server Communication

The SNEWS architecture should support the existence of multiple different servers. The prototype software provided preliminary support for this by having the client attempt to connect to a list of servers provided in a configuration file. However, in the event of a server crash, data could be lost because individual servers did not communicate with each other. This lack of communication also meant that multiple servers could issue an alert for the same coincidence.

The newly designed SNEWS servers interact with each other to avoid these issues. Servers synchronize their alarm queues so that if one server goes down, the alarms it received are not lost. One server is marked as the primary server; all the rest are backup servers. The primary server is responsible for receiving data from clients, and propagating this data to the backup servers. A backup server will not accept connections from client. If the primary server crashes, however, one of the backup servers will be elected as the new primary server.
4.2 Client Design

The client library required significant redesign to support the multithreaded architecture proposed in the previous chapter. These changes were necessary to provide support for anti-coincidence communications.

4.2.1 Threads and Anti-coincidence Response

The client design consists of two threads of operation, a data-collection thread responsible for acquiring and managing data from the neutrino detector, and a network thread that manages the connection to the central server.

I originally designed and implemented a complex system of shared memory, signals, and semaphores to facilitate the interaction between these threads. After reexamining my client design, I realized that much of the complexity of the client library could be eliminated if I implemented a client-side alarm queue and generated anti-coincidence responses based on this queue.

When the data-collection thread generates an alarm, it sends it to the network thread via a socket. The network thread maintains an alarm queue similar to the queue described above for the server, and also forwards alarms to the central server. When the network thread received a coincidence packet, it can examine the data stored in its client-side queue, and respond to the query appropriately.
Chapter 5

Implementation

This chapter documents the implementation of the SNEWS design described in the previous chapter.

5.1 Coding Practices

A requirement of the SNEWS software is that it should be maintainable. In this section, I discuss the general practices I employed toward this goal.

5.1.1 Modularity and Encapsulation

The SNEWS architecture proposed in Chapter 4 is complex, composed of several mostly independent pieces. In implementing my design, I attempted to make use of this independence. Abstraction is a recurring concept in software-engineering – complex systems often benefit from a separation of interface and implementation.

The C programming language provides several features to aid such abstraction; these features can collectively be described as encapsulation. In C, independent pieces of code can be divided into separate source files or modules. A function can be defined as static, preventing it from being called from outside its module.

Similarly, internal structures for the SNEWS library can be shielded from users of the library. An examination of the library API presented in Chapter 2 shows that
external callers do not need definitions of internal structures (such as network packets, or the alarm queue). Therefore, these structures are defined in internal C header files used in compiling the library; these header files are unnecessary to normal users of the SNEWS library.

5.1.2 Portability

The SNEWS software must be usable on several different platforms. As mentioned in Chapter 2, the prototype was compiled and tested on many different UNIX platforms. In order to maintain the same level of portability for my new SNEWS implementation, I decided to make use of only standard C library functions and functions provided by the OpenSSL library.

The design described in Chapter 4 mentions the use of threads. A number of different threading packages exist; many of the targeted operating systems provide their own threading functionality, and at least two separate projects, GNU pth and MIT pthreads, exist to provide (mostly) portable threading across several operating systems. UNIX also provides a fork function that can be used for simple multithreading.

I decided to use the fork function as the threading mechanism for my SNEWS implementation. This is sometimes called “heavyweight” threading, because the operating system makes a copy of the entire process. The pth and pthreads packages both provide lightweight threads that are more efficient, but as my design required only a few threads, high-performance threading is not a concern.

The GNU autoconf system provides a means to build software packages in a system-independent fashion. I used autoconf to further enhance the portability of my SNEWS software. In particular, autoconf has routines to discover system-dependent information, such as the location of particular system header files, and the size of various C types.

Another form of system-dependence is “endianness”. Computers with different architectures can store multi-byte numeric data in different formats. Therefore, all numeric data sent over the network between machines must be converted to a con-
sistent format. The C library defines routines htons, ntohs, htonl, and ntohl to convert 16-bit and 32-bit numeric data between host order and network order (network order is big-endian).

5.1.3 Backward Compatibility

The new SNEWS library supports existing software designed to use the original SNEWS code, requiring minimal or no modification to the existing source code for such legacy applications. As mentioned in Chapter 2, the SNEWS routines are provided as a library of functions. I implemented all of the new client functionality of my design while keeping source compatibility with code linked against the old library. This way, applications can be recompiled against my new SNEWS library, and they will automatically take advantage of new features (such as anti-coincidence support).

5.2 OpenSSL and Threads

In my initial tests with the OpenSSL library, I found that sharing OpenSSL connections between multiple processes was problematic. I discovered that the OpenSSL library is not thread-safe. It makes use of per-process structures; since these structures are not shared between different threads, when two different threads attempt to communicate on the same OpenSSL socket, the encryption stream becomes corrupted. To get around this issue, I implemented a wrapper around the OpenSSL library to provide thread-safe OpenSSL socket sharing.

- int sserver(int port);

  Returns an SSL server socket listening on the specified port.
• int saccept(int socket, struct sockaddr *sa,
             unsigned int *salen, char **name);

Accepts an SSL connection on the specified socket and fills in sa, salen, and name with information about the client; returns a socket for the new connection.

• int sconnect(char *hostname, int port, char **name);

Connects to an SSL server at the specified hostname and port and fills in name with the server’s certificate name; returns a socket for the new connection.

Upon creation of a new OpenSSL socket, a thread is spawned to manage all traffic over that socket. This thread is linked to the original calling process by a “socketpair” – the C library provides a function socketpair that creates two unnamed sockets connected to each other. The new thread is wholly responsible for making all OpenSSL calls, and for verifying the peer’s certificate initially.

Besides providing thread-safe access to OpenSSL sockets, this wrapper also provides another feature – the caller obtains a standard socket, and can use the normal C functions for interacting with sockets, such as read, write, and accept.

However, using this wrapper means that the caller no longer has access to normal OpenSSL routines. In particular, the caller cannot directly discover information about the name of the certificate being used. In order to provide access to this information, the OpenSSL wrapper reads this information and asynchronously makes it available to the caller through the name parameter.

5.3 Server Implementation

A SNEWS server has several major functions. The server maintains state for each connected client, manages a queue of alarms and detects coincidences, collects anticoincidence information upon detection of a coincidence, and interacts with other servers. Another non-required but useful feature of my SNEWS server implementation is an embedded web-based summary.
5.3.1 Client Connections

The server spends most of its time waiting for new connections from clients or waiting for data from connected clients. This functionality is implemented with a select loop. The UNIX select function allows a process to wait for a network event. The SNEWS server selects on a server socket for new connections, as well as on all of the sockets for clients that are already connected.

Sockets for connected clients are maintained in an array; when a new connection is received, the new socket is stored in that array, and when a client disconnects, the array entry is cleared. When a new connection is accepted, the server verifies the client's certificate; this is automatically handled by the OpenSSL wrapper as mentioned above.

5.3.2 Alarm Queue

The SNEWS server stores, in memory and on disk, a queue of alarms it has received from clients. An alarm is composed of an experiment id, the alarm's level, the alarm's significance, and the date and time of the alarm. In memory, the alarm queue is represented as a doubly-linked list. On disk, this queue is stored as a text file; each line represents a different alarm.

When the server receives an alarm packet, it inserts the alarm into its queue in chronological order, sorted by the event time specified in the alarm rather than the time the packet was received. If the new alarm conflicts with a previously received alarm, the old alarm is removed and replaced with the new one. Therefore, the alarm queue should always be sorted chronologically. Additionally, every time an alarm or retraction packet is received, the queue is "cleaned". All alarms that exceed the \texttt{MAX\_ALARM\_AGE} (24 hours by default) are deleted from the queue.

When an alarm packet is received, the server searches its queue for a new coincidence. For each alarm $a$ in the queue, the server examines the window of alarms following $a$ with timestamp no later than \texttt{COINC\_WINDOW} seconds after $a$ (30 seconds by default). The server counts the number of distinct experiments appearing within
this window. This process is repeated for each such window in the queue. The window containing alarms from the largest number of experiments is chosen as the best coincidence.

To prevent the server from processing the same coincidence twice, each alarm is annotated with an additional field, reported; when the server finds a coincidence, it sets the reported field to true for all alarms in the coincidence window.

5.3.3 Anti-coincidence

The implementation of anti-coincidence query routines in the SNEWS server is straightforward. As mentioned in Chapter 4, when a coincidence is detected, the server sends a coincidence packet to each connected client. The server also stores the time that it detected this coincidence. It waits WAIT_FOR_AC seconds (20 seconds by default) to receive anti-coincidence data from the clients. After this amount of time, it generates an alert if a configurable number (3 by default) of the connected clients which should have been sensitive detected a neutrino burst within the given time window. Otherwise, the server records a summary of the alarm data for the coincidence window, but does not generate an alert.

5.3.4 Inter-server Communication

Support for multiple servers is required so that if one server crashes, coincidence detection will automatically be migrated to a new server. To provide this function, I implemented a backup server mode. Upon startup, each server will attempt to contact all other servers. If it is able to connect to another server, it enters this backup mode.

In backup mode, a server maintains a copy of the alarm queue stored by the primary server; when the primary server updates its queue, it informs all backup servers. Backup servers will not accept connections from clients. However, if the primary server ever crashes or becomes unreachable, the backup servers elect a new primary server (the currently running server with the lowest ID number). This server

34
switches from backup mode to primary mode, and all other servers connect to it in backup mode.

5.3.5 Web-based Summary

The server can provide up-to-date summaries via an embedded web server. Originally, this was implemented in order to aid in debugging. However, this functionality could be useful in a production SNEWS environment, and as it is not an expensive operation, I did not remove this feature.

When the server receives a packet from a client that is formatted as an HTTP request (a packet starting with the string “GET / HTTP/1.0”), it automatically responds with an HTML-formatted summary of its alarm queue and the state of its connected clients. Since only OpenSSL connections are accepted, the client must be an HTTPS-enabled browser. Also, the browser must have a valid certificate installed. This ensures that the alarm queue will not be visible to unauthorized users.

5.4 Client Implementation

The SNEWS client library implements the multithreaded client architecture described in Chapter 4 such that this threading is transparent to the caller. It provides the three functions defined in Chapter 2 (gccli, gcretcli, and gcpingcli), and one additional function described below.

5.4.1 Client-side Alarm Queue

The client-side queue reuses the code described above for the server queue. Because of this, it automatically provides support for disk-based backup and recovery of alarm information. The SNEWS client library spawns a new thread upon initialization. The original thread can then continue to collect data from the detector, while the new network thread maintains an alarm queue. The two threads are linked together by a socketpair. When the client calls one of the SNEWS library functions, a packet is
sent to the network thread. This thread updates its queue as necessary, and forwards the packet to the server.

5.4.2 Anti-coincidence Response

When the server sends an anti-coincidence query, this packet is received by the network thread, which generates an anti-coincidence response. This response should enable the server to determine whether the detector was sensitive to a signal during the given coincidence window. By default, the client library assumes that when the client is running, the detector is sensitive. This is accomplished by storing a special “begin_sensitivity” record in the client-side queue upon library initialization. When the client exits, an “end_sensitivity” record is stored in the queue.

However, an experiment may want to continue running its SNEWS client even during periods of signal insensitivity. Therefore, one additional function is provided to manually store sensitivity records in the client-side queue:

```c
void gcsetstatus(int status, int *datetime);
```

The `gcsetstatus` function will store a sensitivity record for the specified date and time. If the `status` parameter is 0, this will be an end_sensitivity record; otherwise, it will be a begin_sensitivity record.
Chapter 6

Conclusion

The goal of the SNEWS collaboration is to develop a system for detecting the co-incidence of signals at multiple neutrino detectors, in order to accurately alert astronomers of the occurrence of supernovae. My thesis project concerned the continuing development of this system.

6.1 Summary of Work

The main motivation for my research was to add support for anti-coincidence routines to the SNEWS architecture. My software accomplishes this, and also adds enhanced security and robustness, and support for multiple servers. It is implemented in a modular and portable fashion, so future development and maintenance should be straightforward.

In the course of my research, however, I designed a general, secure architecture supporting two-way communication and data exchange between a collection of clients and servers. This project allowed me to apply my theoretical software engineering education to a practical system. In particular, I was exposed to the iterative nature of software design when I made substantial changes to the client design in the process of implementing it.
6.2 Future Work

My SNEWS implementation is ready to provide automated alerts of coincidences of neutrino bursts. However, continued development could focus on making parts of the code more configurable. For example, while the code can read parameters from a configuration file, the name of this file is hard-coded. Also, some options could possibly be set by environment variables or command-line switches.

Although I have extensively tested my software, I have not been able to run a stress-test involving many clients and many servers and high packet rates. Such tests should be carried out to verify the operation of my system under heavy load.
Appendix A

User’s Guide

Random Numbers

On systems that do not have a /dev/random device, the RANDEGD variable must be specified. This variable should point to an EGD-style socket. For instructions on obtaining PRNGD (an EGD replacement), see:


By default, prngd will attempt to find its configuration file in /etc. You can point it at another configuration file. For example, I use the contributed linux configuration file included in the distribution:

    ./prngd $HOME/egd -n -c contrib/Linux-2/prngd.conf.linux-2
    setenv RANDEGD $HOME/egd

Seeding the pseudorandom number generator can take a while. For quick debugging, set the RANDEGD variable to “none” to skip seeding.

**WARNING:** DO NOT do this on production machines, as it will compromise OpenSSL’s encryption!!!
Certificate Administration

There should be one SNEWS certificate authority responsible for creating/signing all client and server certificates. To create a new certificate authority, use the “CA” script that comes with OpenSSL (this is installed in /usr/share/ssl/misc/CA on RedHat).

```
/usr/share/ssl/misc/CA -newca
```

Make sure to remember the passphrase you enter, because you will need it to create client and server certificates.

1. Sign new certificates using:

```
openssl ca -in gccli.cert.req -out gccli.cert.cert -notext
```

(Replace gccli_cert with gccliserv_cert if necessary.)

2. Email the created certificate file back, along with a copy of your server’s certificate (found in “demoCA/cacert.pem”).

New Client Instructions

1. Create a client certificate request using:

```
make gccli.cert.req
```

2. Email “gccli.cert.req” to the SNEWS certificate administrator, and wait for a response.

3. Copy the received file into this directory as “gccli.cert.cert”.

4. Unlock the certificate using: make gccli.cert.pem

5. Install the CA cert file as “cacert.pem”.

40
New Server Instructions

Follow the new client instructions, substituting gcserv_cert for gccli_cert.

Other Notes

gcserver has a built-in web interface for monitoring its status. To use it, you will have to import a certificate into your web client certificate database.

Netscape will only import PKCS#12 format certificates. To generate a PKCS#12 certificate from your pem file:

    make gccli_cert.p12 or make gcserv_cert.p12

You can then import the certificate into your Netscape certificate database by clicking on the lock icon in the lower left corner of the browser. Select Certificates/Yours, and click on the “Import” button. You can now point your browser at https://hostname:13001/.
Bibliography


