Research into the Application of a Liquid State Machine for the purposes of Intrusion Detection

Thomas Hentsch

Bachelor of Science in Computer Science with Year in Industry, Honours
University of Bath
May 2005

Abstract

This is a two part investigation into the application of Liquid State Machines to the field of intrusion detection. The first part concerns the state of current development in the field of Liquid State Machines and Intrusion Detection Systems. The second concerns the development of an interface between packet sniffing and a Liquid State Machine.

Acknowledgments

To Dr Richardson for his faith in my schemes, his encouragement, and above all his patience.
To my Parents for starting me on my path.(And a fair bit of proof reading)
To The Universe for creating someone as screwed up as me.
To Jess for teaching me to fly.

Contents

Research into the Application of a Liquid State Machine for the purposes of Intrusion Detection

Signed.....

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institiuion of learning. Except where specifically acknowledged, it is the work of the author.

Signed.....

1 Introduction

1.1 Area of Investigation

Intrusion Detection, in a network sense involves being able to tell when a network has been infiltrated. There are many approaches and methodologies for doing this, each with their supporters and detractors. The idea of using Neural Networks to aid in the process of detection, has been proved both useful and practical. In the past few years, computer power has reached a point at which Spiking Neural networks are practical. While there was a great deal of interest up until a couple of years ago about their application, that same interest seemed to have waned.

Among several interesting neural networks using spiking neurons, I am most interested in Liquid State Machines, which essentially are unplanned networks with a short term memory. This report is an investigation into the possibility and potential usefulness of these constructs in efforts to detect network intrusions.

1.2 The path that led to this idea

I have currently worked for 2 years in the computer industry. My first experience was a year working in computer support. The second was split into software development and system support. The first company I worked for implemented a closed network, so few viruses plagued our systems. Beyond the occasional boot sector virus there were few security concerns. However, while working in my most recent job at the Alberta Cancer Board, we weathered several worm based virus strikes. Some of which occurred while I was working in development, while the others occurred during my time in system support.

While attempting to contain Sasser, I had several ideas about how to help fight the problem. Not all were implemented or possible during that time frame. One of these ideas involved examining the network traffic to identify the worm in transit and then filter out the packets containing the worm. It is largely from this experience that I envisaged the original concept that led to this project.

The second component of my idea came from the way in which virus writers release their progeny. To gain maximum damage, the infection beings on the Thursday evening or Friday morning. Because of working practices concerning Fridays, the virus spreads undetected throughout network systems. After this stage, it has the entire weekend to propagate unchallenged. This exploit of social conditions enables the virus to be even more devastating than if it were released at another time during the week.

The Sasser outbreak was a good illustration of this problem. This and other several vulnerablities in our network made our job especially difficult. Firstly our systems were not patched against this threat (this was a problem addressed in a changed policy after the incident). There was no way at first to immunize computers which we had cleaned; the traffic from the infected computers was crashing the routers. We also could not block the infection port since this was the port which Windows used to authenticate the domain. Lack of appropriate understanding by users resulted in further problems for our team.

Under normal secure operations our internet firewall would have prevented our network from being vulnerable to the virus. However someone logged in using a vpn account from home, which was not similarly protected and so a conduit for the virus came into existence. This is an easy example of how a simple mistake allowed an exploit of a vulnerability in our defenses.

The lesson to learn here is that short of putting together a closed network of diskless workstations running a proprietary operating system on a proprietary networking protocol across quantum photon links, there can be no guarantees of absolutely network security.

From this I came up with the concept of a new kind of firewall, in which it would be possible to recognise new attacks based on knowing what old attacks looked like. Based on my layman's knowledge at the time I believed that a form of neural network could be used to do the identification of new attacks. From these beingings my initial idea crystallised.

I extended my original idea to conceive a community of cooperating firewalls to enable a internet-wide security system.

1.3 My initial project concept

This is a summary of my original project proposal, the full text can be found in Appendix B.
The scope was "To develop a firewall capable of learning to recognize and deal with threats to it's ward(s)".

The idea was to begin with teaching a learning system based firewall to be able to scan packets and identify good and bad packets, discarding the bad packets and being transparent to the good ones. Also the firewall should be able to learn from its mistakes and anticipate situations where it will be compromised.

Using a virtual community of firewalls, I believed that you could have anti-virus information spread quicker than a worm could spread. The implication would be that all machines protected by such a firewall would be potentially safer than other systems facing the same threats without help from other machines.

To facilitate "Trust" between the computers I envisaged a "friendship" protocol, that two firewalls would use to develop a level of trust so as to be able to share data between each other.

Additionally I was wondering about whether genetic algorithms could be used to develop packet filters in combination with the neural network. The idea was based on my rudamentary knowledge of genetic algorithms however I believed the idea had potential.

Finally I was interested in whether it might be possible to teach the firewall to learn how to defend itself, using methods that may not be strictly legal.

1.4 My Hypothesis

My knowledge of Liquid State Machines suggests that they will prove useful in the field of Intrusion Detection.
I intend to prove that it is possible to interface a packet sniffer interface with a Liquid State Machine.

2 Literature Survey

2.1 A Brief History of the Internet

Looking at history, one is left with the conclusion that technological advances have unforeseen consequences far beyond the imagination of their creator, and the creators of the first computer networks would have been hard pressed to imagine a world where a large and ever growing percentage of the first and second world's population are connected together in a global network. There are few places or people that are not affected by this world spanning device. However it had humble beginnings as a project developed as a project by a US defence agency known as ARPA[1].

2.1.1 Fragile beginnings of the Internet - ARPANET and UUCP

This parent of the internet was known by the name of ARPANET, whose foundation became one of the keystones of the future network we call the Internet. Its protocols and router technology are the direct ancestors of our TCP,UDP/IP protocols and routers that are vital to the every day running of our Internet.

After being developed by ARPA, the prototype was turned over to the control of the DCA[2], as a non-commercial and research based network primarily between educational establishments, however as with most things businesses began making their inroads on the system. In part due to this there has become the great problem with the allocation of internet addresses has limited the flexibility of the system even more than it needed to.

Meanwhile civilian networks such as UUCP[3] were developed, which enabled the start of Usenet communities. These technologies were absorbed into the growing behemoth of the newly public network successor of ARPANET known as the Internet.

As an interesting side-note, there was a military based successor to ARPANET known as MILNET, used for transmitting unclassified information between military installations, however there seems to be little information about the current state of this network, and it is quite possible given the possibility of relatively secure transmission of data over the conventional Internet, that such a network would not be necessary, either way its eventual fate is unknown.

2.1.2 The Internet it/his/her self

Not that blue e icon on the desktop, or even the red, orange and blue one, the Internet is probably one of the most important inventions in human history as important as the printing press. Admittedly I'm a geek, so I may be slightly biased about its importance, but from a social point of view it has far reaching implications and huge potential benefits to all of humanity.

It is difficult to define, given its protruding into many fields. But in terms of networking, it is a global network that has potential to transfer massive quantities of packets both good and bad to anywhere on its network potentially anonymously.

This is primarily the child of the backbone of ARPANET and much of the userbase of UUCP. While this Inter(national)-net has existed for about 20 years, however what was needed was a standardised protocol, which again was formed out of the technology of ARPANET.

2.1.3 IP Packets, bounties, bills, and bombs

The key to the internet is IP packets, which transport higher functions, across a variety of hardware systems. The essence of it is that if you have an IP address of a computer connected to the internet, you can send it

[1] Department of Defence Advanced Research Projects Agency
[2] Defence Communication Agency
[3] Unix to Unix Copy Protocol

an IP packet, even the packets are transmitted via postcards on one of the links. There are theoretically any number of protocols you could transmit encapsulated in IP packets. However the current internet standard of TCP/IP there are 5 protocols, however in the following list, the ones in bold are the commonly used ones.

1. SCTP - Stream Control Transmission Protocol, a new protocol with a hybridization of features from UDP and TCP. The reliability and out of packet sorting of TCP, is combined with the ability to transmit on the same connection between multiple IP addresses at both ends so that you could have several hosts/network cards at each end able to pick up the connection should one receiver/transmitter fail. The original usage envisaged was to use it for transmitting reliable phone connections across an IP network. [9]

2. **TCP** - Transmission Control Protocol, a stream protocol that has error checking, packet optimization (through the nagle algorithm), out of order resequencing and flow control it is far from perfect, but is the protocol used for transmitting most of the traffic on the internet. Many application layer services including HTTP, ssh and e-mail are built on top of it.[6]

3. **UDP** - User Datagram Protocol, is a far simpler protocol than its sibling TCP, much of the features such as flow control and out of order resequencing do not feature, however this means the implementation of UDP is far smaller and so it is the basis of application layer protocols where storage space is at a premium, for example boot proms. Also applications such as bittorrent use UDP to maximise packet transfer speeds, since a highly effective error checking system is built into the application itself.[7]

4. **ICMP** - Internet Control Message Protocol, which usually not seen at a user level, it is used to send error messages between computers, for example sending reasons as to why a service is unavailable. The most common human use for this protocol is the ping command which is one of the most basic tests of a connection between 2 computers.[5]

5. DCCP - Datagram Congestion Protocol, a protocol for which speed of delivery is the priority over accurate/ordered data streams. Potential applications include live streaming media and internet telephony. This protocol is still in development and has yet to be formally standardised. [8]

Most packets will be benign but packets can in their contents hide attacks that are far more sophisticated than the traditional DDoS attack involving dummy packets sent to overwhelm a host.

The IP packet structure is the basis for all modern networks using the umbrella term "TCP/IP" protocol. There are some legacy networks using NetBUI and IPX, but these are slowly being replaced by TCP/IP counterparts.

While the systems of today are as far from their forbears as we are from our distant hominoid ancestors, it is important to understand this field's beginnings so as to understand how developments have occurred, and sometimes why seemingly illogical conventions exist.

2.2 History of Intrusion Detection Systems

2.2.1 Cliff Stoll

One of the parents of Intrusion Detection, was Clifford Stoll an astronomer and System Administrator. He was one of the first civilians to connect irregular network/server log activity with hacking attempts. His investigation into discrepancies in a usage database under his administration, led to one of the earliest arrests for hacking with malicious intent.[25]

2.2.2 The beginnings of research into Intrusion Detection

[11] While this review is not a catalogue of papers researched, this paper deserves special mention, since it represents one of the earliest researches into understanding intrusion detection. There are papers that show earlier security, but their focus is more around the field of locking a system down, as opposed to this which aims to recognise when security has failed.

The paper itself is dated in its technology, imagining a world of servers and dumb terminals, without an internet full of script-kiddie[4] hackers, but it addresses the concerns of the day, which were of the different ways into a computer system. It does not address the physical concerns, such as how someone would gain physical access to a terminal without approval, because that is outside the scope of investigation. Even if this era, remote hacking was possible, but the pool of potential attackers was far smaller.

While other research analysises the security systems to declare their safety, this route was by Cliff Stoll. The change in thinking is that annomolies are evidence of network tampering. You have to accept the the assumption security breach will occur, and such an occurance is not avoidable indefinatly, only detectable. Evidence of a security breach, proves firstly that the system is flawed, and secondly may provide clues as to how the compromise occurred.

The jump between this era, and the modern internet era is fairly significant, in no small part due to the arena changing so much. While larger companies and universities in the past had internet access, it was not the Internet we use today. For a start, the list of home users with access could probably be comfortably fitted on a sheet of A4 paper. Nowdays global internet access is at 13.9%. In the first world[5] Internet Access is between 67 and 36%[13]. This means there are now a pool of approximately 900 million potential hackers. The effect is that world where Cliff Stoll may have been able to track his attackers back to Germany and pursue their arrest has been replaced by a world where most hackers are barely noticed, let alone pursued by overworked security teams.

2.3 Methods of Detection

Leading on from the early research despite its limitations, has formed starting points for current methods of detection.

While my initial proposal talked about prevention as well as detection, the primary focus of this investigation is into the detection side of security.

2.3.1 Activity Logs

Given that any activity can be recorded, analysis of the system activity logs, may show up illicit endevours. While this is theoretically possible, the advancement of network technology has made this ever more difficult to do. Firstly in storage terms, to record all activity on a network such as the one at the University, would run into hundreds of gigabytes/day[6].

If storage of such quantities were possible, the computational requirements of analysing such quantities of data would require a similarly excessive amount of computer power to do the discerning.

Most activity logs do not record anywhere near this level of information, but a carefully selected subset of this information will contain sufficient information to detect intrusions.

[4] A term to define hackers who do not understand the nuts and bolts of hacking but use existing software to fulfill their goals

[5] Europe, Australia and North America

[6] Assuming access across the gigabit backbone is running at 100th maximum capacity would be 1.25 MB/s → 108 GB/24hr day

Some of the earlier neural network usage in intrusion detection involved picking out similarities between network events.[12]

This methodology is an excellent way of detecting intrusions, with two caveats, initially that the data is actually examined for anomalies, and that this analysis is done within sufficient time to be useful. Knowledge of an breach should be acted on as soon as possible so you need a trained person to interpret any analysis, who will be capable of acting upon finding a breach.

2.3.2 'Viewing' Activity

In spite of the limited conscious processing ability of the human mind[7], we have significant recognition abilities. There have been moves to 'visulise' network activity in a way that enables system administrators to recognise abnormal events without refering to logs or even looking at a computer screen. One fairly useful example of this is Peep[14] which uses background sounds to reveal the level of network activity, changes in this can reveal network problems such as intrusions.

The peep system has been developed in such a way that background sound can offer the information, without being irritating [15].

While this is not the only method for giving sumaries of network data, it is in my opinion one of the most inventive and usable, since it does not require visual display mediums, and can be played unobtrusively in workplace environments.

This is not without its limitations, but its recent addition of a content scanning module [9], has new potential for alerting to more subtle network anomalies.

2.3.3 Firewalls

Firewalls are not thought of as methods of intrusion detection, but it is often a secondary function of software and hardware firewalls, who record traffic directed at ports and internal addresses they shouldn't.

Most internet users who use Windows with a softwarefirewall, will have had some kind of error message quoting on a regular basis that

"43.243.98.53 attempted to ping this machine" or
"82.32.125.69 attempted to Log into this machine"

Users often assume that any report is an attack, which I myself believed for a while, when first signing onto the campus network in the first year. This resulted in a rather impolite message to someone I thought was trying to hack into my computer when it was just a general net broadcast message from his computer.

However this is the essence of Firewall intrusion detection, and with more advanced filters the false alerts can be reduced.

2.3.4 NAT, Proxies and Stateful/NonStateful firewalls

If this were an investigation into intrusion prevention, these 3 topics would be mentioned in detail, but since their application is about preventing rather than detecting intrusions, it would be outside the scope of research.

1. NAT - Network Address Translation - Prevents computers inside the NAT being seen from outside, only the NAT router is at risk
2. Proxies - Use of proxies can add a layer of separation between the at risk client and the dangerous internet, additionally filters applied to the proxy cache can remove potentially dangerous website code/material.

[7] Approximetly 0.1 FLOPS[8]
[9] See below for more details

3. Stateful/Non-Stateful firewalls - This is just classification of firewalls, the first type knows roughly what is going on its local universe, with access to a past, while the Non-Stateful variety are only focused on the current input packet and nothing else.

2.4 Effectiveness of current intrusion detection methods

1. need a paper/research into difference between attacks detected and attacks postulated - done
2. maybe something on cost caused by intrusions over the past 10 years to show scale of the problem

2.4.1 What is the real threat?

The world is very different, we've been told its the case that since 11th September 2001, honestly I don't believe it, since I see nothing to back it up beyond a level of saber rattling unseen since the Reagan era.

However in the electronic world things have changed, the first world is becoming yet more Internet connected, just ahead of the development by some of the more enlightened parts of the second and third world.

Before I began my gap year, it was acceptable for a large scale organisation such as National Grid to be of the Internet, but the idea today any reasonably sized organisation not having some kind of Internet presence.

Cyber terrorism is the threat to our net world order, but the question that should be asked is how much it threatens us, and who is doing it.

For the most part we have seen "Weapons of Mass Annoyance"[16] so far, and the occasional vigilante attack on organisations such as SCO. These problems are still serious, but the intent behind them has been for the most part innocent.

There have already been reports of online protection rackets where hacker gangs will demand money from websites or businessess, a recent example was that of the Ladbrookes website being threatened before the Grand National.

To my knowledge there have been no virus/worms successfully released by 'terrorists', however we are told it is only a matter of time. However as we move more of our infrastructure and society on the Internet, the more at risk we are from an electronic attack.

The implications are clear, the threat is real, and probably more dangerous than the supposed threat in the real-world.

2.5 How much has the current cost been so far?

I should preface this section by saying that economic cost is not an appropriate method of real cost of net attacks, when the risk is to human lives. I admit that sounds melodramatic. However, when I was working in the CCI hospital, all our patient records were contained on our computer systems. Because of the near complete networkfailuree during Sasser there was no access to these systems, and so critically ill cancer patients were put at risk. However without research into the risk of human lives, the economic impact will show the level of the problem.

Surprisingly I could find no research into the cost of past cyber attacks.

All I can therefore refer to is generalities, the fact is that the entire global economy is based not really on gold in vaults, but in electronic numbers in databases held in "secure" locations. The stability of our society is dependent onvulnerablee computers, so the cost could be the end of modern society if a big enough hit happens.

The threat is real, the consequences can be real, we must keep advancing otherwise innocent lives will pay the cost.

2.6 Current intrusion detection systems and advancements

The current solutions areinadequatee, however there has been a great deal of research into evolving current systems which needs to be covered.

Their relevance is important due to these solutions are trying to address certain issues that I wish to show the LSM may be suitable to solve at an alternative.

2.6.1 Flow problems

One of the greatest problems with analysing packets is that you are dealing with hugeamountss of data going along extremely fast connections, and if you slow that flow, you will be causing a problem as bad as a minor intrusion incident.

It has been suggested by research[26] that it is possible to scan packets in parallel, therefore limiting theinterruptionn time to as short as a single scanning time across multiple packets.

This solution is far better for TCP based packets since the parallel scanning is able to scan out of sequence and let thereceiverr sort out the order, but if it is using UDP, this will most likely not be as fast, since the firewall will need to let the packets out in the same order they came in.

2.6.2 Contents of the packet

While there are a near infinite number of potential packet payloads in theory, there is a far smaller number of interpretable payloads. These interpretable packets are understandable because they follow certain patterns. If you know the format of 'good' packets, it is possible to set up regular expressions to filter out everything except 'good' packets.[26]

I did consider this as a way of detecting problems before encountering the research into it, since it came to my intention that certain ports which have specific functions such as the Windows Domain login port. These same unclosable vital ports are often used as the route that viruses use to bypassdefensess. The problem is that the set of expressions describing all 'good' or even all 'bad' packets would be enormous, and ever changing.

2.7 Research into firewalls that use Neural Networks

I am not the first person to attempt a neural network based intrusion detection system. In the past there have been several cases of neural networks trained to recognise past attacks being used to recognise those same attacks.

Using the DARPA database for training, it was shown that 77% of attacks were detected by the firewall, with only 3% false positives[24]. This is a very impressive achievement, while the system is not perfect, it is being highly accurate and focused, picking registering a significant majority of attacks it has been trained to recognise.

While 100% accuracy would be delightful, the truth is that the attacks under different circumstances than trained would look different unless you recognise the nature of the attack.

It is the same as showing a border patrol guard the faces of the FBI 20 most wanted under ideal conditions, informing the guard that at some time in the future one or more of these individuals (in disguise) will be coming through, and he must detect them all.

Under the circumstances making 3 in one hundredcheckss false positives is a testament to the viablitity of this system.

Thanks to the work of InSeon Yoo, and Ulrich Ultes-Nitsche[23] the key concept of my original proposal, the idea of determining new attacks based on old data has been thought of before, however they have yet to produce a proof.

To those of you who are fans of Terry Pratchett, the proposal is not disimillar from Ponder Stibbons efforts to produce "Invisible Writings", by having access to all the books in the library, he theorizes that he can find any book not yet written albeitt slowly).[18] While my idea is not as computationally horrible, I feel a kindred spirit between his ideas and mine.

2.8 Liquid State Machines

2.8.1 The Nature of Liquid State Machines

The Liquid State Machine is a type of partially unplanned neural network that uses spiking neurons. There are differences in thebehaviorr of spiking neurons and traditional artificial neurons, the primary one being the refraction period of a neuron, but what is more important is what this results in is the unique properties of a LSM.

The name, Liquid State Machine, comes from the hidden layer of the network between the input and output, who's internal makeup and connections to the input and output neurons is generated random functions (often using a normal distribution to determine probability of connections). The extraordinary concept that may seem difficult to grasp, is that the liquid layer has a sense of time, and with that sense, it possesses a memory of what has happened in the recent past. In effect it has a short term memory to augment its long term trained memory.

Because of this ability to know the past and potentially how long ago events happened, it represents what is the closest to date example of a human like analysing ability. As discussed in the next subsection, it has applications in the field of identification and classification.

There has been past interest in Liquid State Machines, with several projects were begun to open development into their usage. Amygdala, SNNS, SpikeNET, and C-Sim each had a degree of potential, but due to unknown circumstances, Amygdala abruptly stopped development in 2003, SpikeNET still has yet to release anything more than a demo, SNNS[10] has no method of generating Liquid State Machines, despite now having the capacity to use Spiking neurons. C-Sim is still in development, and was the basis for one of my proofs of concepts. For my usage of these packages see in future sections.

Situation wise, the state of development in general is poor, this I suspect is due to a distinct lack of simple tools for the purposes of using LSMs. I feel that this is an a great shame, without a cause beyond lack of interest. Despite searching I could not find mention of abandoning of development into spiking network packages.

2.8.2 Creating a Liquid State Machine

This is just an simple procedure for creating a Liquid State Machine, which I discovered after reverse engineering a matlab routine in an earlier version of C-Sim[21]

1. Create Layers

   Input Layer

   Hidden Layer (Also known as the liquid layer)

   Output Layer

2. Connect Layers, Based on distance between neurons, and using normal distribution to determine if a connection between a particular pair of neurons exists.

   Input to Hidden Layer

   Hidden to Output Layer

2.8.3 My Knowledge Shortfalls

I started this project with a layman's idea of Neural Networks and an intuition that they could be useful in intrusion detection.

My current knowledge is still extremely limited in my own opinion. I have working knowledge of how to construct neural networks, and have used several packages to create networks. My practical understanding is acceptable, however my understanding of the theory behind Neural Networks is lacking, I do not feel confident in my understanding of this area, however it was my opinion that I would be going outside the focus of my research, since while it is interesting to know why the LSM works, this investigation was concerned primarily with the application of LSMs not their inner functioning.

[10] As of February 2005

However I do possess a working knowledge of why the LSMs work, which has aided me in understanding their nature.

2.8.4 Applications of Liquid State Machines

Despite a fairamountt of interest in the past there still seems to be a lack of current applications. However SpikeNET has used their implementation for the recognition of grey scale facial images. [20]

Any task that requires recognition of patterns in the temporal domain could make use of a LSM. At the moment I theorize that the lack of a viable library for LSM creation and use, has prevented their more wide scale use.

Temporal Recognition could include

1. Sound Recognition

   Voice recognition

   Composer recognition

   Music classification

2. Video Analysis

   Recognizing optimal key frame positions

   Identifying violentbehaviorr in a security camera

This is not any where a total list, and is not intended to be, it is meant to show the broad theoretical capabilities of LSMs if people tried to use them.

2.9 Problems with Python

It is myopinionn that the documentation that comes with Python is lackingseverelyy compared to the documentation available for other languages. The ability to access automatically generated documentation through pydoc is impressive but limited. Compared to the javadocs and tutorials available on the Sun site [2] the Python documentation on the internet is far inferior.

2.10 Conclusions from Literature Survey

I can conclude that many of my concepts are not new, however the application of LSMs for intrusion detection has yet to be done, anywhere.

My research did not uncover the reason(s) for a lack of applications for Liquid State Machines. My only suspicion is that the computational requirements for an LSM is far higher than its earlier counterparts. For further information on this area look in the proof of concept section.

The doomsday attack has yet to come, but given the nature of people it is more likely to be a person showing off than a person with a deliberate agenda, who unleashes said virus.

3 Software Journal

The following section details my investigation into the capabilities of Liquid State Machine packages available. I should note that the intent was to find a package suitable to be able to produce a working model of my intrusion detection system, rather than a comprehensive investigation of the abilities of each piece of software. My focus was primarily on Amygdala, which was a mistake.

3.1 Stuttgart Neural Network Simulator

This was my first and fourth experience with neural network development software, since I first discovered it in the form of the Stuttgart Neural Network Simulator[4]. It represents what is the only visual tool for thepurposess of creating spiking neural networks available.

3.1.1 First experiences

This was a fairly confusing experience in which I attempted to investigate the methods of creating a Neural Network. At the time I was using the JNNS which supposedly would become the successor to the SNNS, but seems to have been run into some troubles since when I came back to it seemed to be not that prominent as before.

At this point my neural network knowledge was non-existant, my later research turned up several simple neural network packages which would have been better for me to learn the basics under, however I did not attempt to use them later on due to their lack of implementation of a Spiking Neuron.

3.1.2 Coming back later

After having left amygdala behind, I was looking for alternatives, which presented itself when I discovered a SpikingNNS patch for the StuttgartNNS, which it turned out already had been updated a while ago with the patch.

So with avengeancee I returned to attempt once again to master this package, While I was partially successful in being able to create a Liquid State Machine in terms of neurons, the facility for creating the synapses, and then training, were exceedingly difficult to use.

While the interface may be capable of doing what I wanted, it seemed to me that Amygdala was a better option. I therefore attempted to make Amygdala work again

In spite of my lack of success with Amygdala, I feel that I made the right choice in moving away from SNNS, while it is a product with a lot of hard development and great potential., it would have taken time I did not possess to master it in such a way as to be able to use it.

3.2 Amygdala

[22] This package is a case study of the problems with Spiking Neuron development.

The first release was in October 2001, which is a release I never used, but I have attempted to use versions 0.2-0.3.4, since there seems to have been lack of current releases. The last project release was in 2003, the release before the 0.4 version containing the OpenGL network debugger, as well as improved samples including a working Liquid State Machine

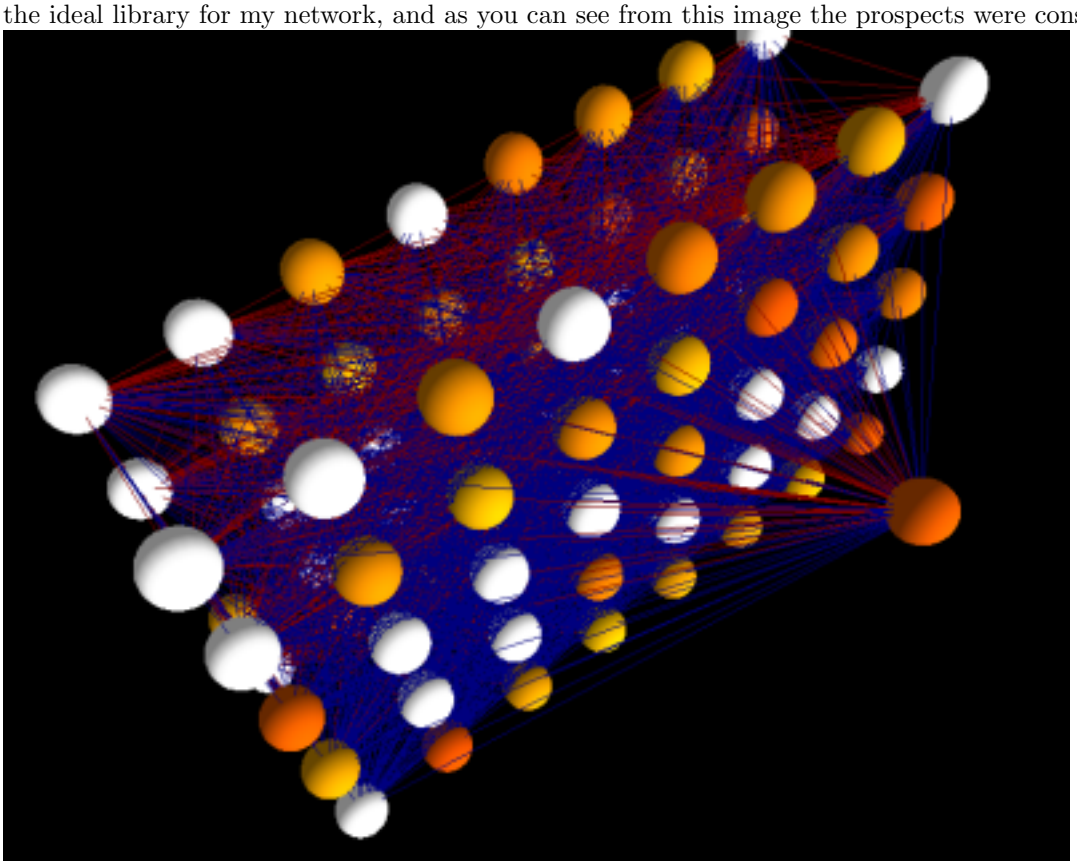
There was an implementation of a Liquid State Machine in version 0.2 but due to its lack of functioning, it was not implemented beyond this version. This in itself was not an insurmountable problem, but it did not help due to the time wasted trying to get the code working.

I encountered a great deal of success with the demo program which included an open gl visuliser that showed in real time the activities of two different demo networks.

On five occasions I attempted to use amygdala, excluding a separate attempt to generate a python binding.

3.2.1 The Demo

Looking at the demo application, I can remember pretty much jumping for joy in the belief that I had found the ideal library for my network, and as you can see from this image the prospects were considerable.

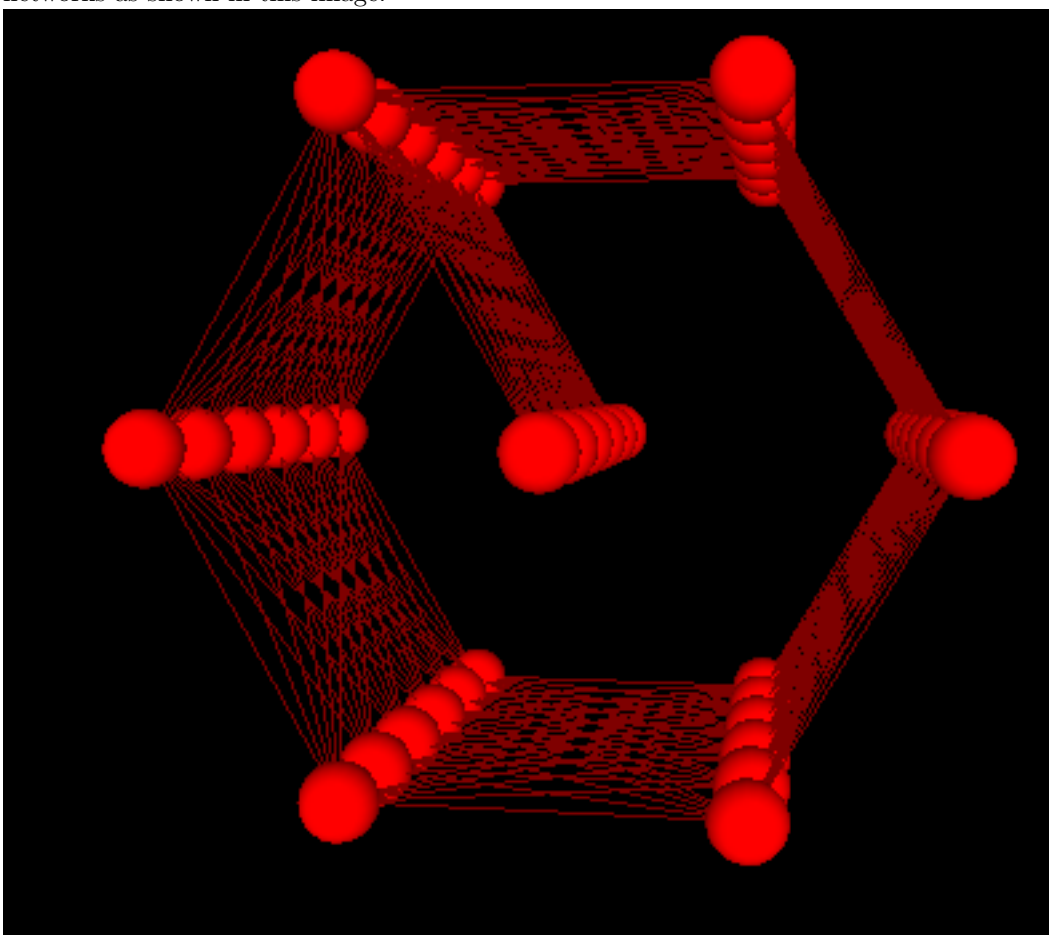


This is an image of a demo simulation of part of a mouse brain listening to audio samples.

It showed a library capable of simulating complicated networks easily, and then visulising them in real time.

3.2.2 Building a SpikeLoop

After experimenting with the demo, I attempted to produce a SpikeLoop, so called the HelloWorld of spiking networks as shown in this image.



However despite follow what sketchy documentation was avaliable, I was unable to produce this network. After this I abandoned amygdala to look for alternatives for a while.

3.2.3 LSM from 0.2 version

After failing to find an alternative I returned to amygdala looking for an automatic method to create an LSM, since there was at least a video file of a working spiking network under amygdala.

3.2.4 Using the Netloader

There was a sample as part of the 0.3.4 deliverable called a network loader that was meant to be able to load an XML file containing a network design, from that producing a network for use.

However again this function was let down by lack of clear or usable documentation

3.2.5 Creating a Python Binding

At this point my python was far superior to my C++ coding abilities, and knowing it was possible to do such an interface I begin researching thepossibilityy in the hope that I could program through python, and

then possibly connect it to my packet interface.

There are several packages on the internet for the creation of Python bindings from C libraries, sadly they are not automatic tools, although swig was the closest I found to such a device.

After reading documentation on several different packages, I realised that in order to import amygdala into python I would have to write a rather complicated header file that would import all its features in.

I decided that despite my limited C++ coding experience, it would be better to try to reverse engineer sample code, get a C++ program working and then create an interface for a much smaller simpler program.

3.2.6 Reverse Engineering

I tried to understand how the sample networks were constructed, which I believed at first I had managed, however when I tried write my own very basic network I was frustrated by a lack of compilation ability that I finally traced back to the way in which amygdala was installed on the system.

I found out that not even the sample networks would compile without their specially created makefiles that had all the correct paths.

On discovering this I decided to stop flogging a dead project and moved on.

3.2.7 Conclusions on Amygdala

I am deeply saddened to say it, but there needs to be a warning on the amygdala site, to prevent future researchers into Liquid State Machines from wasting even a fraction of the time I wasted on this dead end.

To someone with time and skill on their hands, it is possible that amygdala could provide an base from which to extend to produce a workable spiking neural network solution. However it is not in a usable state and Iseverelyy doubt it ever will be.

4 SpikeNet

Spikenet while existing in a demo package todemonstratee its facial recognitionpossibilitiess, was my first experience of using a Spiking Neural network.[20]

However I also considered trying to use it as a replacement for Amygdala, but its sole release as a Demo, made this possibility untenable.

Without access to the API or library, in order to make use of this techology I would have had to reverse engineer the demo executable. This process would still not have given me a workable solution since I would then still need to develop my own system based on the innards of SpikeNET. While SpikeNET makes grand claims about its capabilities, I could find noverificationn of what the creators were saying. Therefore I could see no advantage for me to base a potential system I would design on code decompiled from their executable.

This was already after I had decided that the route of designing a spiking neuron system was not the best application of my time resources.

In spite of this setback there was one positive aspect which came out of this, the SpikeNET team lauded the ability of their networks to be able to detect patterns on images. This enabled me to come up with the prototype interface discussed in the proof of concept below.

While this would have been possible, the fact of the matter would be since I would still have to engineer my own code in the end to make use of the SpikeNet code. It would have been moreefficientt for me to attempt to write my own package. However since I had already decided to not develop my own code for the LSM.ere are a number of similarities of the code used in both despite being written by different authors.

4.1 C-Sim

4.1.1 Introduction to C-Sim and Neural Micro Circuits

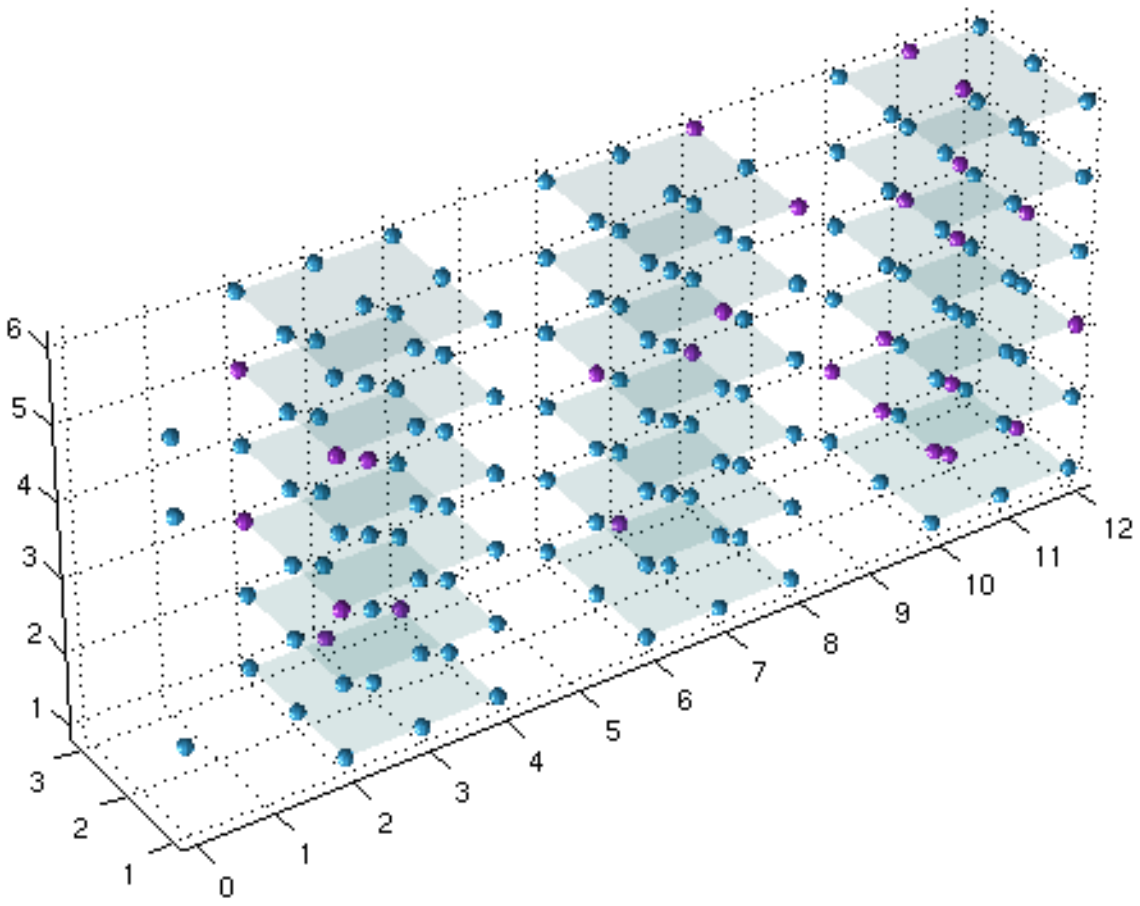
Allow me to introduce the one somewhat bright light in spiking neural network development. This package for the most part is working, and seems to be still being worked upon by its creator.

I encountered one problem where it would not run the sample code, so I found an error checking line that seemed to perform no useful function which caused the crash out and disabled it, after this the program seemed to run fine.

4.1.2 Creating a Liquid State Machine with the Neural Micro Circuits packge

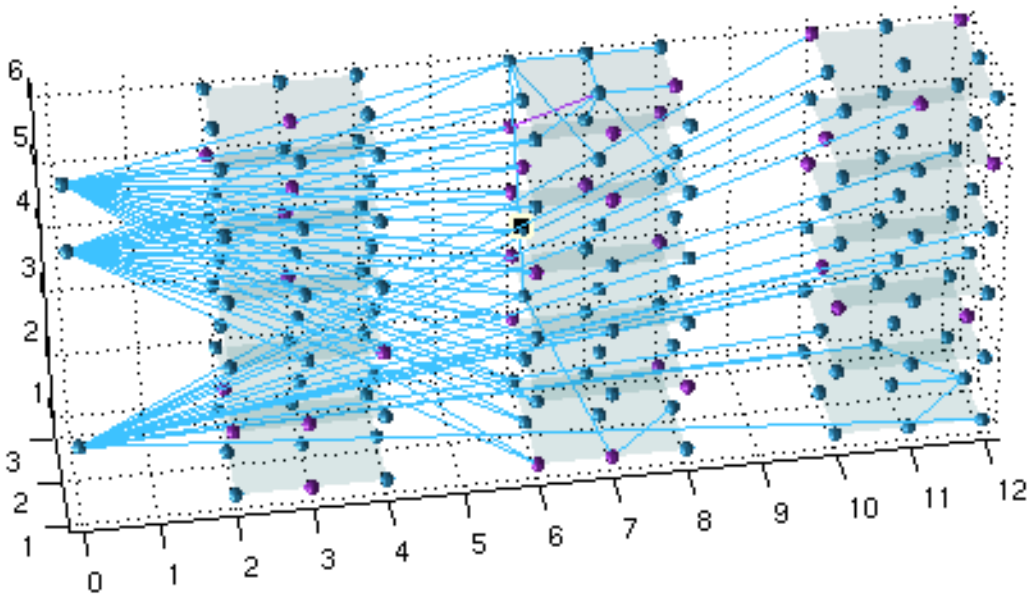
The code to create a Liquid State Machine was rather complicated, however the manual provided some help in this matter.

3 commands created the following



I was able to easily create the grid of neurons, and then with a little bit of tweaking make connections between them.

Followed by a series of commands that produced the linked variety.



However beyond a rudamentary understanding of what each command does, the manual gives no indication as to what the different commands actually are made up of, so beyond a few guesswork changes it would be very difficult with this level of documentation to make good use of what is otherwise a good package.

4.1.3 Shortfalls

There are only 3 criticisms I can make against C-Sim, the first is its lack of a way to access its libraries outside matlab, a python wrapper, or even a C based API would have made it possible to be used in a practical application.

The second is that the methods for viewing the neural network are veryinefficientt, even on my powerful apple the rotating is exceedingly jerky.

Finally the nail in the coffin of a potentiallybrilliantt system is it lack of good documentation, true it does supply a manual but the code I type in to create an LSM means nothing to me, so I cannot even bug test when there is a problem.

4.1.4 Opinion

Of the avaliable packages, I believe that C-Sim would present in theory the best solution for developing a Liquid State Machine, however this is given that the user does not have the capability to develop their own system since there are numerous flaws and bugs still present in C-Sim, not to mention a lack of quality documentation for both C-Sim and the Neural Micro Circuits package I used.

5 Proof of Concept

The code and notes for my proof of concept can be found in Appendix D.

5.1 Network Functionality

5.1.1 Python sockets

My first efforts with trying to do network interaction was to attempt to open sockets on specific ports and get the information out of the network connections.

I based my initial code on some socket based networking code in python, [1], I experimented with both the simple and complicated servers used, as well as the echo client. These can be found in Appendix B.

This use of sockets was based on my at the time limited understanding of networking knowledge, in an effort to get packets going to specific sockets. Although I abandoned this route, I did some development of the code to produce my own multithreaded server, that operated on multiple ports. Admittedly I came across a rather interesting flaw in the python threading code.

The problem was caused by having threads that would never end running at the same time. The way a python thread is stopped is by its finishing. Normally this is the way threads work, however what I needed to do was on user command shut those repeating threads down. I came up with a solution that caused there to be a flag checked each time the cycle loops.

My solution can be found in Appendix B.

5.1.2 Python pcap and pcapc

After doing further reading, I discovered that the library pcap, would capture all packet activity on a network card (incidentally including all packets sent accidentally to the network card if so asked). While the technology has been around for a while, there is a legal concern of reading packets not addressed to you.

Thankfully there are excellent python bindings for pcap and I was able to set up in 5 lines of code a python pcap program.

```
import pcap #import the library
a = pcap.pcapObject() #create a new capture object
a.open_live('eth0',256,0,10) #open up a capture device on eth0
while a.next():  #while there are new packets to capture
    a.next() #show the next packet
```

Under OS X I was forced to use the less developed pcapy, which had less functionality and also seemed less stable. I could not make an unending loop run extracting packets. This is the similar version of the above code for pcapc which was still able to extract some packets before having a fault. Erroneously the developer claimed superior abilities for his wrapper over the pcap one.

```
import pcapy #import the pcapy library
a = pcapy.open_live('en1',256,0,10) # create a pcap object
while a.next():  #while there are more packets
    a.next() #display the contents of the next packet
```

The output from this code in either case was a series of asci characters. The steps taken in the proof of concept will detail the method of interfacing a packet with a neural network.

It should be noted that pcap and pcapc are sniffers, not interception libraries. the next section covers my investigation into intercepting packets.

5.1.3 Additional python network technology

Through research that is nicely summed up in a presentation by the name of "Packet Mastering"[10], I came to the conclusion that it would be possible to use a combination of libdnet and libnids with libpcap to be able to intercept and remove the problem packets. However it was also clear that this would be an additional level of complexity which would not be possible to develop within the timeframe of the project.

Therefore after failing to find any current python implementations of a packet interception package, I decided to focus on the neural network and the link between this and the captured packets.

5.1.4 Intrusion Detection using Python

This is a route of development that I did not follow primarily due to lack of time for a project of this size. My focusing was on producing a proof of concept over a fully working system.

My decision was based on arealizationn that I could easily overstretch my time constraints and produce a rushed inferior investigation, due to my desire to do a showy implementation.

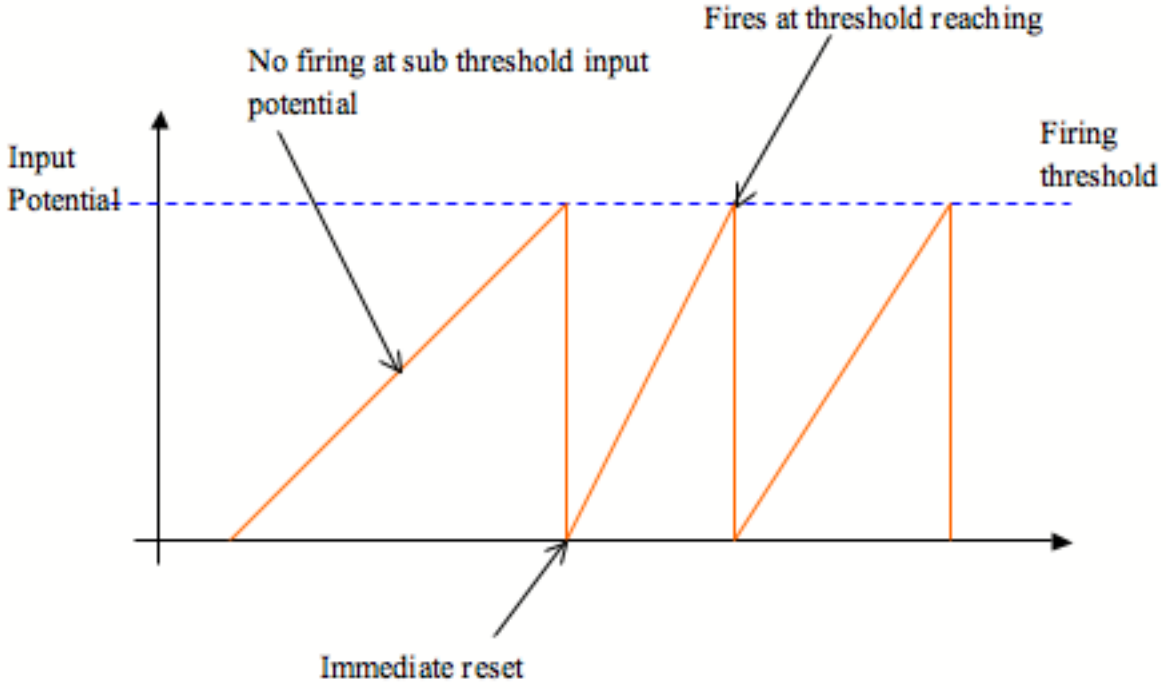
The networking capabilities of python are significant and for the most part fairly mature.

5.2 Modelling Neurons

5.2.1 Classical Neuron Models

Classical neurons models are not created directly from the neurons found in natural neural networks. This is mostly due to the lack of implementing the spiking nature of natural neurons. In the past it has been the case that computational power has been unable to simulate sufficient. quantities of spiking neurons.

This is due to the nature of the "spike" property. In a classical model the neuron will fire once threshold is reached, and then be ready to fire immediately afterwards.



As you can see time is of no concern to this neuron, if it has been fired and then can fire instantaneously afterwards or 10 minutes later. The upshot is that it is very efficient to be able to run large numbers of neuron models of this type.

While there are many useful applications for non-spiking networks, they are not intrinsically sensitive to time in the way that spiking neurons are.

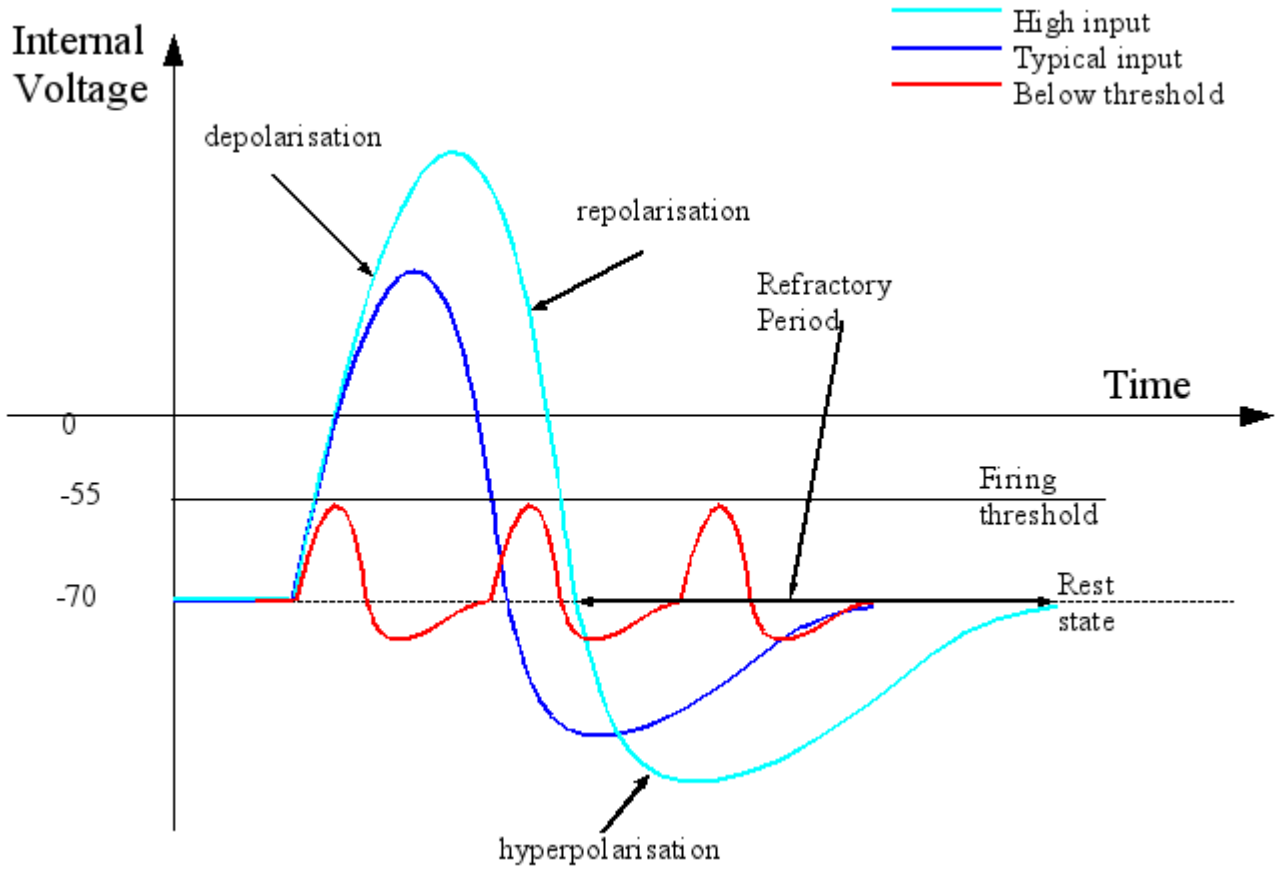
The trouble of classical neural networks comes from the fact that a reaction happens instantaneously, there are no delays, no pauses, and no change in the weighting of the networks once training is not happening.

What is unavoidable is that 2 events happening within microseconds of each will have no influence on each other while being relavant to the state of the system in which the network is working.

Because of this, the neural network is rarely the centre of the system, it is a tool that is accessed by the system as an packet classification library.

5.3 Spiking Neural Models

However the neural network will still be lacking in certain capacities therefore spiking neurons have been developed to the point at which a computer of current technology can without much effort run a simulation of 200,000 spiking neurons. This figure comes from the package SpikeNet detailed later in this document.



The key difference is the cycle of firing for a spiking neuron. It has a rest period that requires that the neuron return to the rest state

The additional problem with the equation is calculating the refraction period for variables above and below the variables, since

5.4 Simulation

Given the nature of a Spiking Neuron it would be relatively simple to simulate the sequence. Here are a couple of possible ways to do the simulation.

1. A global time function that goes through at every cycle each neuron,checkss its state, and does housekeeping (e.g. reduce polarisation by a time function)

2. Each neuron runs in a thread, it does its own housekeeping in as close to real time as is possible.

Personally I think that the threaded model may be more efficient in 3 major ways.

Firstly taking into account that compilers and runtime environments are exceptionally advanced, giving them threads to play with, will give them a greater flexibility when optimising the code.

Secondly there would be a significant overhead for running the global time function since two copies of the network would need to be run. The first would be the real neural network, and the second would be a snapshot of the neural network at the beginning of the cycle. This is because otherwise the network would become temporally out of sync with itself.

Thirdly if the code is multithreaded to begin with, it should be much easier to adapt to multiprocessor environments and so run on these systems more efficiently.

My interface layer did not implement the threaded code, due to the fact that partly it was meant to be a proof of concept rather than a truly accurate implementation, but also the number of neurons in question[11], would be insignificant for my computer to deal with in real time.

While it would have been potentially possible to develop my own neural network library, at the time of decision I believed that it would be better to use an off the shelf piece of software, this in my opinion was a mistake, however I was able to take some of my ideas into developing my interface layer that would sit between a LSM and a network device.

5.5 Artificial Spiking Neuron

The effort I wish to show does not represent an exact model of a spiking neuron, rather a good approximation. The complexity of neuron cells would prevent any current computer from simulating a neural network in real time at little or no accuracy benefit.

The aim was to replicate the depolorisation - repolorisation - hyperpolorisation - refraction period cycle of normal neurons in as computationally simple way as possible. Therefore I attempted to find an equation that could describe thebehaviorr.

The closest I discovered with experimentation was $f(t) = -sin(x^2)$ which described a single spike followed by a refraction period, but this wassymmetricc and thebehaviorr of a spiking neuron is notsymmetricc around the rest value.

The other problem was designing how to recognise when the rest state had been reached since there would be two points at which the neuron would reach the rest value, and the second one would be the true rest state. I planned to differentiate the equation to get the first differential. This would enable me to tell when the rest state had been reached, because at the rest state, the gradient would be zero, and the voltage would be at the resting value.

After investigating the nature of spiking neurons, it became apparent the nature of the nature of the output spike was unrelated to the internal voltage after firing. Due to this I realised that it would be possible to simulate the refraction cycle by measuring the time between the last spike and the current time.

This simple model enabled me to simulate spiking neurons in a way that is very computationally simple.

It should be noted that a natural spiking neuron will need to wait the refraction period whether the threshold is reached by a spike. The problem with this scenario is that at least in my design for the interface, the packet is interpreted into a grid of photons, it is likely that a great degree of the receptor neurons will be spiked if not triggered. The result will be that if a large number of packets come through in a short time period, the neurons may not recover in time, causing packets to be unseen. Therefore I have changed the rules for these simulations and they now only require a pause if the threshold has been reached.

At the same time, I have also stopped the adjustment of the refraction period, which a natural neuron would do. The higher the degree ofdepolorisationn, the longer the refractory period. However I made a decision based on computational concerns, I decided that I would choose the simpler version for now.

[11] 1024

In a future version this might change, however the results below do show promise for this interpretation.

5.6 Interpreting Packets and turning them into Photons

The trouble with the pcap interface for python is that it returns a tuple, containing a unicode string of an undetermined encoding. Lack of documentation for py-pcap made impossible to translate into ascii text.

However the ascii convention means nothing to a computer or a neural network unless it is interpreted. Therefore I decided to convert the values into hexadecimal values so that I could easily translate the packet contents into "photon" data.

I knew from my investigations, that the Liquid State Machines, had been applied for the identification of greyscale images. However I could not reverse engineer SpikeNET to find out how it was done. There are 256 possible values for a packet[12], but a threshold is a binary switch, with one switch I could not get any degree recognition of the character being transmitted. I did not attempt for an 8 neuron grid, instead I came up with a potentially problematic idea of interpreting each character as a random colour.

I first generated a lookup table of values from 0 to 256 (full range of values), which contained a photon of differing R G Bquantitiess. The process was not entirely random, but it wasarbitraryy. I decided against random, because when I was testing I wanted to be able to see repeating patterns of recognition. The RGB values were set from 0 to 255.

Each character would become a hexadecimal value based on its character value, I then could convert this value into an index in my lookup table, the effect of which enabled swift translation to the photon values. At the same time Ilaidd out the photons into a 16x16 grid (Up to the maximum size above set for the packet contents).

5.7 Cone cells and the Eye

To interpret the photons I created cone structures whichseparatelyy interpret the RGB values of a photon, before feeding the combined spikes together to try and spike the inner neuron.

The eye structure is the container for the cone cells and maintains inputs and analysis out from the eye that would be passed to the neuron.

The array of photons is passed in, and a array of true or false values is returned. These output values report which neurons spiked.

5.8 Results

The network generates spikes based on the inputed packet, this spiking pattern is the same for the same packet. The mapping is many to 1, because of the nature of having one output per character.

By implementing this program, I have managed to prove that an interface between a Spiking Neural Network and a packet capture interface is possible.

The interface is possible to simulate in real time with low processor burden,admittedlyy this is on a 2 GHz AMD system with a Gig of memory and a custom compiled Linux kernel. Therefore this interface could be used with little modification in future development.

[12] ascii range

6 Conclusions

6.1 Summary of Conclusions

This proof of concept represents something that has not been done before, it is a rudamentary interface between a packet interface and Liquid State Machine.

It is still needing further development, but it is the first step on developing a practical interface for real world applications.

In doing this project I have challenged and pushed myself, developing my programming skills with a new language, and also learning to combine my creative and analytical talents to produce an output superior than either one of my talents could provide alone.

In answer to the hypothesis, while I have not done the precise link to the Liquid State Machine, I have managed to create from network data, spikes that can be fed directly into a LSM. This interface is a partial proof of my hypothesis, awaiting only the completion of a LSM.

6.2 Project Conclusions

Intrusion detection is a vital field of development in the modern networked world. This world is constantly changing, it is difficult to adapt to new threats, let alone keep ahead of the hackers.

The potential risk is too great to ignore, it is both probable and filled with potential for disaster, but the next time it might not be the computers in your office or home, it might be the drug dispensing computers in hospitals, the wireless entertainment centre that is part of your new car, or the computer containing the only copy of your life's work in the world.

The application of Liquid State Machines has been lacking severely, and specifically in Intrusion Detection a field that I feel would be a natural and obvious area of development, there has been not even a whisper of research.

Based on my research, it is possible that Liquid State Machines may aid in the detection of intrustions. Their moreprivativee forebears have already proved useful, the time based nature of the LSM, could also aid in the identification of dangers that stretch over two or more packets.

Through my proof of concept I have demonstrated that it is possible network data to be interpreted as spikes. I do not state that this is a perfect or even mature solution to the interface problem, but it is my opinion that a future interface could be developed from my initial concept.

As well as showing this new application, my research into the field shows a dismal state of LSM development in the researchcommunityy. There has been research, projects, ideas, concepts, but most are forgotten, only in existence because websites such as SourceForge do not delete inactive projects.

My project has not proved any certainties, but it has opened the door on newpossibilitiess.

6.3 My self assessment

My favorite bit of any piece of work is to criticise myself, which I am generally good at, some have suggested that is a flaw in my self confidence, but I think it keeps me grounded.

You may wonder why I am saying this, but it does go into the nature of the work I did during my project. My initial grand project proposal you can find in the Appendix, refers to a concept that I imagine if I developed successfully and then marketed well, I could probably retire as the creator of a new era of internet security.

As this reveals, I occasionally need a certain degree of grounding, which I have had ample doses of during my project. The first package I looked at was SpikeNET, which failed to run under Cygwin back when I

used Windows. As a result of that encounter back in Semester 1, I retreated from further development until the end of Semester 1, while procrastinating revising, I took my first look at the Stuttgart Neural Network Simulator, which again resulted in another retreat.

This project has taught me many things, the first being that of knowing that I am not a Nobel Prize winning Computer Scientist, however I forgot the mirror lesson until nearer the end, that while I'm not an off the scale genius, I am still very capable if I put my mind to it.

The fluctuation of performance levels has been one of my greatest challenges in this project, I have had days where I have worked for 12 hours nearly solid apart from food breaks, and I have had days and nights of running up brick walls getting nowhere.

As a part of understanding my capabilities, I have been able to meld my imagination with my knowledge, and from that produce not just ideas, but ideas backed up with knowledge and occasionally a form of logic that others understand.

From my initial ideas, and their evolution I have produced several new ideas for future research, each potentially an interest for either myself or others who wish to take them further.

6.3.1 Changes I would have made in my project development

My first focused criticism, was my lack of faith in my coding ability that prevented me from developing my own Liquid State Machine library. I wasted a great deal of time and effort trying to develop my project out of pieces of forgotten projects, and had I realised what was possible if I put my mind to it, I would have been able to produce a proof of concept that featured a working LSM.

In spite of this being a flaw that has caused me grief in the past, I failed to correct my time management. This was further compounded with me ignoring my vow never to do a coursework module again. This rang true with having to do project finalization with handing in HCI coursework on the same day. This final period would be far less stressful without having this coursework.

My final failing was not getting a good grasp of Neural Network theory earlier on, my knowledge is spotty mainly due to that the information has been collated by my brain as I have come across the information in my research. I have attempted to fill in the blanks as I have gone along, but without a clear plan of knowledge I know there are still gaps in my knowledge that might have enabled me to avoid some of the problems I encountered while using C-SIM and SNNS.

6.3.2 What have I learned from this?

Firstly I have delved in a limited fashion into both C++ and matlab, not languages I have had more a brush with in the past. I would not say that I wasproficientt in them at this point, but I am confident that should I need to develop with them in the future, my rudementary knowledge will rise to the challenge.

Secondly, my appreciation of python has increased somewhat, I now understand it, and have reached some ability to program with it. Its debuging and designing methods are a little unorthodox, due to the constant compiling and testing I did using small sections of my code to develop the different objects.

If I am professionally proficient in any programming language, it would be Java, which by my assessment, I believe I am. Based on professional experience I am probably second to only one person in our year with Java coding abilities. However I have learned from experience that I only am able to learn a new language when I have a task to do, so I used this project to take my very limited python knowledge and turn it into what I hope is a solid foundation for future use.

Despite my problems I feel that I have made some progress towards a better degree of time management and personal motivation.

6.4 Future development

6.4.1 Liquid State Machine Library

This is something that this project should have produced in a prototype form, had I at an earlier stage believed it was possible, what is now my proof of concept of interpreting my data, could have been extended into a potentially working model for bad packet detection.

What I envisage is a library designed to be used by those who wish to use the capabilities of a LSM, but do not have the time or knowledge to design and implement one.

The foundation would be based around a c or c++ based library that could be compiled on nix[13] systems, and accessible through wrappers be accessed in high level languages such as Java or Python. These object using[14] languages could then create a LSM object with a simple line of code.

It would be very easy to here define a narrow set of requirements for the library and wrapper, but speaking from experience as one who could have made use of this library in advancing a functioning piece of software rather than a proof of concept, I know there is at least a small need for this development, and depending on my time situation after graduation I might work on developing one myself as a personal project.

6.4.2 IP packet interception Library

The packet is the basis for the entire TCP/IP protocol and it is very easy to packet sniff with applications based on or similar to TCP Dump and its library pcap. However to actually intercept the packet once it has been flagged as hostile is very difficult. There are methods based on the libdnet family of networking libraries. Sadly these are rather complicated and require creating a kludge of code that is unique to the system being used on.

What I propose is the development of a combined library that enables the direct access to packets that libpcap currently allows only on a read-only basis.

This is not a fully developedpracticalityy idea, more an idea tofulfilll a need to aid in future development of anti-intrusion defense systems as opposed to intrusion detection systems. As an incentive, detection systems using pcap style libraries could be adapted to use this library to be able to filter theanomolouss packets as well as detect them.

6.4.3 Alternative Interfaces between the LSM and the packet layer

My prototype interface I know is far from perfect, it is a modification of an idea that I had no way to verify externally.

Based on its binary output it is doubtful that it will displaysufficientt resolution of information to the theoretical neural network.

It is a starting point, it is not a finished product although if a LSM were available it would be useful to see what one made of the packets and see if what it could identify. While the resolution might not be able to identify individual packets, it will have greater success in detecting familiar streams, since the sequence will have a more recognisable set of input spikes spread over time that the LSM can identify as opposed to identifying a single packet.

This is due to the fact that the packet will be interpreted as a matrix of spikes that has only 2^256 combinations, when the packet contents could have 2^2048 combinations. However the size of the packet options is less since there is a subset of all potential packets that are corrent IP packet contents. But there is still a large loss of information that needs to be corrected.

A way ofimplementingg it would be for the retina to have 8 output neurons based on the components of the input character. This is so as have a resolution of output data equal to that of the input.

[13] Unix,Linux,FreeBSD,OS X

[14] I'm not certain if python is officially classified Object Oriented, given it is primarily a scripting language with object based capabilities

Also the pattern for displaying the packet to the neural network may need to be tweaked, I considered a spiral form, but then given that the network would not recognise it as such, I decided that doing it as a left to right top to bottom input image would be the best solution for the initial attempt.

6.4.4 Friendship Protocol

My most ambitious concept from my original proposal, while left alone in the research I did, still has the potential for a large degree of fascinating development. The key problem with developing a community of firewalls trading security information is that of a rogue or compromised firewall who gives false or misleading information in order to compromise other firewalls in the community.

I envisaged a system of building trust, but such a system would need to be flexible to take into account honest mistakes (even a correctly functioning firewall can make mistakes).

Signing the messages with a key that is generated by the integrity of the firewall, something similar to using the checksum as a random key might help but asufficientlyy clever hacker could circumvent that.

Looking at the solution Byzantine Generals Problem with signed messages provides some interesting insight. This is because a solution based on methods worked out in the time of mercenary armies and officers, becomes very applicable in forming estimations of trust.

While it might be nice if 1 firewall comes up with the identification of an attack, its not entirely trustworthy, but if 2 or more come up with the same identificationindependently its more likely that both are telling the truth.

Following the solution the with signed messages, 2 firewalls can be sure the other has sent the message but not if the other has been compromised[17].

In the unsigned message variant, if less than a third of the firewalls have been compromised it is possible to solve.

The signed messages section is not the concern, it is the matter of determining if you can trust the other generals. Provided 2/3rds or more are uncompromised you can trust a majority decision as far as security updates are concerned.

As an alternative area of reference, I would like to see if there is a way of developing a "first law test" that one firewall could apply to another, by asking "questions" to determine its integrity. This is based on the first law test applied to the robot Daneel, in the book Caves of Steel[19]

6.4.5 What is the risk in making a mistake?

There are two major problems in guessing about packets, one is that you make a false negative and a dangerous packet slips through, the other is that of a false positive. The consequences of the first are clear, network security has failed, potentially you've just let in a destructive worm, which is something that should be avoided at all costs. However the packet you have blocked could potentially be be a vital message, which from a security standpoint may beacceptablee but from an overall organisational perspective equally catastrophic.

It is my opinion that given my understanding of the nature of LSMs, that it is probable that dangerous packet diagnosis will be dependent on the environment at the specific time. Given that secure methods of transmission such as TCP or even TFTP[15] do not allow a packet to be dropped and so will attempt to transmit it. It is my hypothesis that the probability of a non-harmful packet being blocked a second time will be low provided that it's mis-classification is based on circumstance as opposed to the contents of the packet. Therefore the retransmitted non-harmful packet will be observed as safe a second time.

I therefore suggest that testing be done to see if this is true, since if this is the case a system based around a LSM can afford to be cautious, since it is less likely to cause irretrievable problems in a datastream.

Also as a part of this overall study it should be determined how much of a risk is there from a solitary packet getting through the firewall. This information will help determine how lenient the LSM's filtering should be.

With this information it would be possible to get a much better picture of how to train the LSM, and also how to react to its reports.

[15]Trivial File Transfer Protocol

6.4.6 Reacting to the unknown

Looking at the patterns of attack by the recent spate of internet worm viruses, there are a great deal of similarities in their attack patterns, there are signs in network activity that can indicate an epidemic. However the problem with waiting for these signs is that by the time you have these signs appearing, the system is already close to failing. It is rare that someone will notice network errors before they are causing major problems, due to the fact that the Internet in general is rarely 100% reliable.

It is my theory that knowing the characteristics of packets containing dangerous attacks from previous attacks, it maybe possible for a LSM to be able to flag potential new virtual contagions before they reach epidemic proportions. I do not believe that this would enable the detection of truly innovative attacks, but it may detect derivative attacks, which often follow the original attacks.

Through training the LSM on the current database of attacks, attacks using similar methods may be detectable. As an example one can look at the similarities between the Sasser and Blaster worms. It has been pointed out that their code shares some common properties. Since both these worms have had devastating consequences, future virus writers may reuse their code as a basis for their work. So by knowing the of Blaster and Sasser, it might be possible to recognise these as yet unseen threats.

6.4.7 Partnership with corporate network security

One of the greatest difficulties in doing research into this project has been the restriction of information available about IDS systems used in corporate or government situations. This is due to a policy of keeping security arrangements private so hostile groups do not know what they will be facing during an attack.

Modern counterparts to the DARPA internet attack database almost certainly exist, but those are kept private from the general public. Through partnership with corporate entities such as the HP security labs in Bristol [3], it may be possible get access to these databases for training purposes. Additionally the security systems they are using could be enhanced through the use of a working version of my LSM firewall. Also working with them could potentially give access to their internet logs which could give new insights into the patterns and nature of attacks.

7 Appendix A - Bibliography

References

[1] David Mertz, Ph.D.
Programming IP sockets on Linux, Part One.
http://gnosis.cx/publish/programming/sockets.html

[2] Sun Microsystems
Java^{TM} Platform Standard Edition 5.0 API Specification
http://java.sun.com/j2se/1.5.0/docs/api/

[3] Hewlett Packard
HP Labs Bristol
http://www.hpl.hp.com/bristol/

[4] University of Stuttgart, University of Tübingen
Stuttgart Neural Network Simulator
http://www-ra.informatik.uni-tuebingen.de/SNNS/

[5] W. Richard Stevens
TCP/IP Illustrated, Volume 1
Chapter 6. ICMP: Internet Control Message Protocol

[6] W. Richard Stevens
TCP/IP Illustrated, Volume 1
Chapter 17. TCP: Transmission Control Protocol

[7] W. Richard Stevens
TCP/IP Illustrated, Volume 1
Chapter 11. UDP: User Datagram Protocol

[8] From Wikipedia, the free encyclopedia
Datagram Congestion Control Protocol
http://en.wikipedia.org/wiki/DCCP

[9] From Wikipedia, the free encyclopedia
Stream Control Transmission Protocol
http://en.wikipedia.org/wiki/SCTP

[10] Jose Nazario, Ann Arbor, MI
Packet Mastering, Presented at Hack in the Box, October 6, 2004
http://www.monkey.org/ jose/presentations/hitb04-tools.d/

[11] James P. Anderson Co.
Computer Security Threat Monitoring and Surveillance
http://csrc.nist.gov/publications/history/ande80.pdf

[12] L. Girardin, D Brodbeck
A Visual Approach for Monitoring Logs
http://www.usenix.org/events/lisa98/full_papers/giradin/giradin.pdf

[13] Internet World Stats
Internet Usage Statistics - The Big Picture
http://www.internetworldstats.com/stats.htm

[14] Michael Gilfix & Prof. Alva Couch
Peep (The Network Auralizer)
http://peep.sourceforge.net/intro.html

[15] Michael Gilfix & Prof. Alva Couch
Peep Higher load average demo
http://peep.sourceforge.net/demo/demo2.mp3

[16] JA Lewis
Assessing the risks of cyber terrorism, cyber war and other cyber threats
http://www.mafhoum.com/press4/128T41.pdf

[17] L. Lamport, R. Shostak, and M. Pease
The Byzantine Generals Problem
http://www.cs.wisc.edu/ sschang/OS-Qual/reliability/byzantine.htm

[18] Terry Pratchett *The Last Continent*

[19] Issac Asmiov *Caves of Steel*

[20] Arnaud Delorme, Simon Thorpe
SpikeNET
http://www.sccn.ucsd.edu/ arno/spikenet/

[21] Group Based at http://www.lsm.tugraz.at/
CSIM: A neural Circuit SIMulator
http://www.lsm.tugraz.at/csim/

[22] Hartmut Prochaska,Kit Mitzel,Matt Grover,Rudiger Koch
Amygdala
http://amygdala.sourceforge.net/

[23] InSeon Yoo, Ulrich Ultes-Nitsche
An Intelligent Firewall to Detect Novel Attacks **http://ssrnet.snu.ac.kr/seminar/lab/p20030122-1.pdf**

[24] Jean-Philippe Planquart
Application of Neural Networks to Intrusion Detection
http://cjlab.altervista.org/security/IDS/neural_networks.pdf

[25] Cliff Stoll *Cuckoo's Egg*

[26] J Moscola, J Lockwood, RP Loui, M Pachos
Implementation of a ContentScanning Module for an Internet Firewall
http://ieeexplore.ieee.org/iel5/8700/27544/01227239.pdf

8 Appendix B - Attached Information

8.1 Original Project Proposal

Adaptive Learning Firewall based on a Neural Network

Scope: To develop a firewall capable of learning to recognize and deal with threats to its ward(s).

The aim of the project is to develop a software (or possibly hardware based depending on processing requirements) firewall capable of firstly recognizing attacks and then learning how to defend itself from this attack (and similar attacks in the future).

An example of how a firewall might react would be to imagine a sasser worm attack. The precise ports might be wrong from my memory.

The primary route of the sasser virus attack is exploiting an exploit discovered in the lsass part of the operating system. The first part of the attack involves attacking a specific port of theIPp address to initiate a remote download of the virus onto the system, now a conventional firewall cannot block that port because it is used for windows domain logins. The firewall should be able to recognize the safe logins but reject the error logins even if they come from safe network addresses. Also the firewall should be capable of learning to know when it is about to beaten (which may vary under different circumstances but for example learning when it failed) and in that situation it should be capable of making the decision to shut down the network connection protecting its charges from attack even if it means separating them from the outside world.

The next step would be a firewall community which would consist of all the firewalls that could communicate with each other. Some kind of friendship protocol might need to be created in order that the system would be able to recognise appropriately compatible firewalls. The idea I was thinking about was countersigning to recognise a friendship i.e.
Firewall 1 Hey how would you react to this packet X
Firewall 2 I would see it as a threat and block it, how would you react to this packet Y
Firewall 1 I would recognise it as safe and let it through, how would you react to this packet Z.....

This could then continue until some kind of trust, had been developed between the firewalls, then if a firewall defends against an attack successfully, it can tell the world about it and its friends can listen. In this model it is assumed that it is possible that a firewall may trust another firewall when it is not trusted back again but that is another level of complexity.

In developing defenses genetic algorithms might be used to generate filters, the algorithms being controlled by the neural network, however this is based on an informed layman's knowledge of genetic algorithms so I do not know if it is possible. The order might be
1. Generate new filter to take into account this packet
2. Apply filter
3. Does packet appear on other side of the filter? If so filter does not work, go back to beginning

Finally the firewall might develop (and given Internet regulations this might be difficult) the ability to stop attacks at the source in some way, even if it is an e-mail to the administrator of the offending host or if the host type can be detected, sending some kind of remote net shutdown command. I hesitate to say the alternative of some kind of counter attack because that is probably going over the line of legality.

8.2 Threaded Socket Server

```
#"USAGE: NetworkServerLayer2.py <port>"
from wxPython.wx import *
from socket import *
import sys, os
from threading import Thread


class RootServer(Thread):
sock = socket(AF_INET, SOCK_STREAM)
running = 1
def __init__(self,PoRt):

  Thread.__init__(self)
  self.sock.bind(('',PoRt))
  self.sock.listen(5)
def run(self):


  while self.running == 1:
   newsock, client_addr = self.sock.accept()
   print "Client connected:", client_addr
   self.handleClient(newsock)
def handleClient(self,InputSocket):
  data = InputSocket.recv(32)
  print data
  while data:
   InputSocket.sendall(data)
   data = InputSocket.recv(32)
  InputSocket.close()
def KillItself(self):
  self.sock.close()
  self.running = 0

A = RootServer(1389)
A.start()



msgshow = wxFrame(None,-1,"Zulu")
dialogitself = wxMessageDialog(msgshow, " Press OK to quit the server","Ending", wxOK)
dialogitself.ShowModal() # Shows it

A.KillItself()
```

9 Appendix C - External Code used in project development

9.1 Python Networking code from Programming Python Sockets on Linux, Part One

This code is copied wholesale from the website "Programming Python Sockets on Linux, Part One" [1]

9.1.1 Python Echo Server, Using SocketServer

This is the most basic version of the echo server, which uses the python SocketServer to pick up the incoming connections.

```
#!/usr/bin/env python
"USAGE: echoserver.py <port>"
from SocketServer import BaseRequestHandler, TCPServer
import sys, socket
class EchoHandler(BaseRequestHandler):
    def handle(self):
        print "Client connected:", self.client_address
        self.request.sendall(self.request.recv(2**16))
        self.request.close()

if len(sys.argv) != 2:
    print __doc__
else:
    TCPServer(('',int(sys.argv[1])), EchoHandler).serve_forever()
```

9.1.2 Python Echo Server, Using Using Raw Sockets

This shows the Echo server that allows finer tuning of the reactions to the program to network activity.

```
#!/usr/bin/env python
"USAGE: echoserver.py <port>"
from socket import *     # import *, but we'll avoid name conflict
import sys

def handleClient(sock):
    data = sock.recv(32)
    while data:
        sock.sendall(data)
        data = sock.recv(32)
    sock.close()

if len(sys.argv) != 2:
    print __doc__
else:
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(('',int(sys.argv[1])))
    sock.listen(5)
    while 1:     # Run until cancelled
        newsock, client_addr = sock.accept()
        print "Client connected:", client_addr
        handleClient(newsock)
```

9.1.3 Python Echo Client

This program will run on the client to send messages to the echo server.

```
#!/usr/bin/env python
"USAGE: echoclient.py <server> <word> <port>"
from socket import *     # import *, but we'll avoid name conflict
import sys
if len(sys.argv) != 4:
    print __doc__
    sys.exit(0)
sock = socket(AF_INET, SOCK_STREAM)
sock.connect((sys.argv[1], int(sys.argv[3])))
message = sys.argv[2]
messlen, received = sock.send(message), 0
if messlen != len(message):
    print "Failed to send complete message"
print "Received:  ",
while received < messlen:
    data = sock.recv(32)
    sys.stdout.write(data)
    received += len(data)
print
sock.close()
```

10 Appendix D - Additional Information for Proof of Concept

10.1 Source Code for Proof of Concept

```
import pcap #import the library
from threading import Thread
import time
class photon:   #class describing an 24 bit photon (8 bits per colour channel)
r = 0
g = 0
b = 0
def __init__(self,rval,gval,bval):
  self.r = rval
  self.g = gval
  self.b = bval



class NetworkLayer:#(Thread): This deals with getting the packets and passing them upwards into the interface layer
KeepRunning = 1
PacketInput = pcap.pcapObject()
FrameWork = "NothingYet"
def __init__(self,InterfaceName,Skeleton):
  self.FrameWork = Skeleton
  self.PacketInput.open_live(InterfaceName,256,0,10)
  #Thread.__init__(self)
  while True:
   self.SendUpWards(self.PacketInput.next())
#def run(self):
# while self.KeepRunning == 1:
#   self.SendUpWards(self.PacketInput.next())
#   print 'PacketRecieved'
def SendUpWards(self,NewPacket):   #just sends the packet upwards
  self.FrameWork.InterpretPacket(NewPacket)
  #self.KeepRunning == 0
class SystemStructure:   #This structure links the eye structure with the network layer (and prepares the matrix of photons
InterpretationTable = list()
TheLowerLayer = 'UnConfigured'
TheEyeLayer = 'AlsoYetToBeTitled'
InterfaceName = 'eth0'
XGridSize = 16
YGridSize = 16
InRef = 0.1 #The MasterSettings for the neurons
InThresh = 128
OutRef = 0.1
OutThresh = 192
def __init__(self):
  print 'Starting Up'

  self.TheEyeLayer=Eye(self.InRef,self.InThresh,self.OutRef,self.OutThresh,self.XGridSize,self.YGridSize)
  print 'eye Started'
  print 'Start the Interpretation layer'
  self.CreateInterpretationLayer()
```

```
  print 'starting looking'
  self.TheLowerLayer = NetworkLayer(self.InterfaceName,self)
  print 'All packets in the universe gone :P'
def InterpretPacket(self,NewPacket):#take the packet and convert it
  print 'Starting interpretation'
  ThePacket = NewPacket[1]
  HexVals = map(lambda x:  '%.2x'% x, map(ord, NewPacket[1])) #converts the  packets into a
sequence of hexadecimal values
  print HexVals
  PhotonMatrix = list()
  for i in xrange(0,15):
   xdim = list()
   for j in xrange(1,16):
    if(((i*16)+j)<len(HexVals)):
     xdim.append(self.InterpretationTable[int('0x'+HexVals[(i*16)+j],16)]) #translate the hex
values into photons
    else:
     xdim.append(photon(0,0,0))
   PhotonMatrix.append(list(xdim))
  print 'finished chopping up, now to show the eye'
  #print PhotonMatrix[1][1].r,PhotonMatrix[1][1].g,PhotonMatrix[1][1].b
  self.ShowTheEye(PhotonMatrix)
def ShowTheEye(self,PhotonMatrix):#Just a method to submit the translated data into the eye structure
  OutValues = self.TheEyeLayer.PhotonsHitTheEye(PhotonMatrix)
  print 'eye looking done'
  self.OutputFunction(OutValues)
def CreateInterpretationLayer(self):  # This function is desgined to create a table that assigns a com-
pletly arbritrary colour value to the hexadecimal input value
  for a in xrange(0,256):
   r = (64*a)%255
   g = (32*a)%255
   b = (16*a)%255
   self.InterpretationTable.append(photon(r,g,b))
   #print 'The length of the table is'
   #print len(InterpretationTable)
def OutputFunction(self,OutValues):  #this is essentially a stub to be replaced with a more useful out-
put function
  print OutValues
class Neuron:  #A very basic model for a spiking neuron
TimeAtLastSpike = 0.0 #last spike at begining of time (or just the epoch if you want to be fussy)
RefractPeriod = 0.01 #a default refraction period, nothing too special
Threshold = 128#again a default threshold

def __init__(self,Refract,FireLevel):
  self.Threshold = FireLevel #set the threshold
  self.RefractPeriod = Refract #set the pause period (how long before it can spike again

def Spike(self,InputSpike):  #apply a spike to the neuron
  CurrentTime = time.time()
  if((CurrentTime-self.TimeAtLastSpike)>self.RefractPeriod):
  #if True:
  # if True:
   if(InputSpike.SpikeSize>self.Threshold):
    self.TimeAtLastSpike = time.time() #admitedly it shouldn't make a difference but 2 calculations
```

```
have just gone through with several more in parallel, I don't want to chance it if the threshold is too low
    return Spike(128)

class Spike:  #just a container for a value of spiking, BUT could be extended further
SpikeSize = 0 # It aint of a size unless its valued
def __init__(self,Jolt):
  self.SpikeSize = Jolt
class Cone:  #a class that links 4 neurons to decide whether the whole should spike or not
RedNeuron = "Nothing"
BlueNeuron  = "Nothing"
GreenNeuron = "Nothing"
OutNeuron = "Nothing"
#SpikeState = False
def __init__(self,inRef,inThresh,outRef,outThresh):
  self.RedNeuron = Neuron(inRef,inThresh)
  self.BlueNeuron = Neuron(inRef,inThresh)
  self.GreenNeuron = Neuron(inRef,inThresh)
  self.OutNeuron = Neuron(outRef,outThresh)
def PhotonHitsCone(self,InputPhoton):  #When a photon is applied to the Cone
  RSpike = self.RedNeuron.Spike(Spike(InputPhoton.r)) #check whether each
  GSpike = self.GreenNeuron.Spike(Spike(InputPhoton.g))#of the colours
  BSpike = self.BlueNeuron.Spike(Spike(InputPhoton.b))#spikes

  SpikeValue = 0
  if RSpike:  #check to see if the colours spiked, if so add that output to the spikevalue of the output
neuron
   SpikeValue = SpikeValue+RSpike.SpikeSize

  if GSpike:
   SpikeValue = SpikeValue+GSpike.SpikeSize

  if BSpike:
   SpikeValue = SpikeValue+BSpike.SpikeSize

  #return SpikeValue
  OutSpike = self.OutNeuron.Spike(Spike(SpikeValue))#spike the output
  if OutSpike:#if it spiked, return a true value
   #self.SpikeState = True
   return True
  else:
   #self.SpikeState = False
   return False
class Eye:  #a container for cones
Cones = list()
Xsize = 0
Ysize = 0
def __init__(self,inRef,inThresh,outRef,outThresh,xSize,ySize):
  self.Xsize = xSize
  self.Ysize = ySize
  for i in xrange (0,ySize):
   row = list()
   for j in xrange (0,xSize):
    row.append(Cone(inRef,inThresh,outRef,outThresh))
   self.Cones.append(list(row))
```

```
def PhotonsHitTheEye(self,PhotonMatrix):   #when the photon matrix is applied
  Spikes = list()
  #print len(PhotonMatrix),len(PhotonMatrix[0])
  #print len(self.Cones),len(self.Cones[0])
  #print self.Ysize,self.Xsize
  for i in xrange (0,self.Ysize-1):
   row = list()
   for j in xrange (0,self.Xsize-1):
    row.append(self.Cones[i][j].PhotonHitsCone(PhotonMatrix[i][j]))
   Spikes.append(list(row))
  return Spikes #return a pattern
SystemStructure() #run the program
```

10.2 Example Result from proof of concept

10.2.1 Hexadecimal Values

['ff', 'ff', 'ff', 'ff', 'ff', 'ff', '00', '0f', '3d', '0c', 'bd', 'b1', '08', '00', '45', '00', '00', 'e8', '1d', '38', '00', '00', '80', '11', '96', '76', 'c0', 'a8', '02', '07', 'c0', 'a8', '02', 'ff', '00', '8a', '00', '8a', '00', 'd4', '39', '3b', '11', '02', '80', '7a', 'c0', 'a8', '02', '07', '00', '8a', '00', 'be', '00', '00', '20', '45', '44', '45', '42', '45', '4d', '43', '41', '43', '41', '43', '41', '43', '41', '43', '41', '43', '41', '43', '41', '43', '41', '43', '41', '41', '41', '00', '20', '41', '42', '41', '43', '46', '50', '46', '50', '45', '4e', '46', '44', '45', '43', '46', '43', '45', '50', '46', '48', '46', '44', '45', '46', '46', '50', '46', '50', '41', '43', '41', '42', '00', 'ff', '53', '4d', '42', '25', '00', '11', '00', '00', '24', '00', '00', '00', '00', '00', '00', '00', '00', '00', '00', 'e8', '03', '00', '00', '00', '00', '00', '00', '00', '00', '00', '24', '00', '56', '00', '03', '00', '01', '00', '01', '00', '02', '00', '35', '00', '5c', '4d', '41', '49', '4c', '53', '4c', '4f', '54', '5c', '42', '52', '4f', '57', '53', '45', '00', '0c', '00', 'a0', 'bb', '0d', '00', '55', '4e', '49', '00', '02', '00', '00', '00', '00', '00', 'f0', 'd8', '18', '00', '00', 'd0', '03', '0a', '00', '10', '00', '80', '00', 'd0', 'f8', '7f', '43', '41', '4c', '00']

10.2.2 Spike Results

F, F, F, F, F, F, T, T, T, T, F, F, F, F, F,
F, T, F, F, F, F, F, T, T, F, F, F, T, F, F,
F, F, T, F, T, F, F, F, T, F, F, F, T, F, F,
T, F, T, F, T, F, F, F, F, F, F, F, F, T, F,
F, F, F, F, F, F, F, F, F, F, F, F, F, F, F,
F, F, F, F, F, F, F, F, F, F, F, F, F, F, T,
T, F, F, T, T, F, F, F, T, F, F, F, T, F, T,
F, T, T, F, T, F, F, F, F, F, F, F, F, T, F,
F, F, F, F, F, F, F, F, F, F, F, F, F, F, F,
F, F, F, F, F, F, F, F, F, F, F, F, F, F, F,
F, F, F, F, F, F, F, F, F, F, F, F, F, F, F,
F, F, F, F, T, F, F, F, F, F, F, F, F, F, F,
T, T, F, F, T, F, T, T, F, T, F, F, T, T, F,
F, T, F, F, T, T, F, F, T, F, F, F, F, F, F,
F, F, F, F, F, F, F, F, T, F, F, F, F, F, F