Report Number: WUCSE-2007-9

2007

Gigabit Concept Mining: A Sensitivity Analysis, Masters Thesis, December 2006

Andrew Levine

Massive amounts of data are passed over public networks. There is a need for network administrators to analyze this traffic, but it was not previously possible to analyze live network data at high speed. It has been shown that streaming computation and deep packet analysis are possible at very high rates through the use of hardware acceleration. This work provides analysis for a larger project that involves digesting large amounts of network traffic. In this system, we process the traffic using hardware that has constraints. The workings of the system are first discussed. Tradeoffs in the design of hardware... **Read complete abstract on page 2.**

Recommended Citation

Levine, Andrew, "Gigabit Concept Mining: A Sensitivity Analysis, Masters Thesis, December 2006" Report Number: WUCSE-2007-9 (2007). *All Computer Science and Engineering Research.* https://openscholarship.wustl.edu/cse_research/156

Gigabit Concept Mining: A Sensitivity Analysis, Masters Thesis, December 2006

Andrew Levine

Complete Abstract:

Massive amounts of data are passed over public networks. There is a need for network administrators to analyze this traffic, but it was not previously possible to analyze live network data at high speed. It has been shown that streaming computation and deep packet analysis are possible at very high rates through the use of hardware acceleration. This work provides analysis for a larger project that involves digesting large amounts of network traffic. In this system, we process the traffic using hardware that has constraints. The workings of the system are first discussed. Tradeoffs in the design of hardware and software components are also discussed. Next, an experiment to classify topics of newsgroups is described that utilizes the system. The contribution of this thesis is to show that it is possible to change the parameters of the system to minimize the representation of concepts.

2007-9

Gigabit Concept Mining: A Sensitivity Analysis, Masters Thesis, December 2006

Authors: Andrew Levine

Corresponding Author: aal1@cec.wustl.edu

Web Page: http://www.arl.wustl.edu/~aal1/

WASHINGTON UNIVERSITY

THE HENRY EDWIN SEVER GRADUATE SCHOOL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GIGABIT CONCEPT MINING: A SENSITIVITY ANALYSIS

by

Andrew A. Levine

Prepared under the direction of Prof. R. P. Loui and Prof. John W. Lockwood

---

A thesis presented to the Henry Edwin Sever Graduate School of
Washington University in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

December 2006

Saint Louis, Missouri

WASHINGTON UNIVERSITY

THE HENRY EDWIN SEVER GRADUATE SCHOOL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

GIGABIT CONCEPT MINING: A SENSITIVITY ANALYSIS

by

Andrew A. Levine

ADVISOR: Prof. R. P. Loui and Prof. John W. Lockwood

December 2006

Saint Louis, Missouri

Massive amounts of data are passed over public networks. There is a need for network administrators to analyze this traffic, but it was not previously possible to analyze live network data at high speed. It has been shown that streaming computation and deep packet analysis are possible at very high rates through the use of hardware acceleration. This work provides analysis for a larger project that involves digesting large amounts of network traffic. In this system, we process the traffic using hardware that has constraints. The workings of the system are first discussed. Tradeoffs in the design of hardware and software components are also discussed. Next, an experiment to classify topics of newsgroups is described that utilizes the system. The contribution of this thesis is to show that it is possible to change the parameters of the system to minimize the representation of concepts.

To My Mother Pat Levine

Dedication to my mother Pat Levine and my family.

I dedicate this document to my mother Pat Levine and my family. I want to thank them for all the free meals and support. Without them, I would not have been able to complete my degree.

Contents

List of Tables

List of Figures

Acknowledgments

I would like to acknowledge the work of all the people who contributed to this research project. I would like to especially thank Ron Loui, John Lockwood, Alan Ratner, Steve Eick, Doyle Weishar, and Justin Mauger who all helped me accomplish my goals.

This research was sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under Contract Number MDA972- 03-9-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

Andrew A. Levine

Washington University in Saint Louis
December 2006

Chapter 1

Introduction

1.1 Motivation

Massive amounts of information and data are passed on the public networks of the world. To protect their networks, network administrators analyze as much of this information as is possible. Data needs to be processed in real-time to avoid building a backlog of accumulated data. Currently, when administrators try to process large volumes of data, they are overwhelmed by the sheer volume of information.

To enable processing of large volumes of data, we implemented a system – the hardware-accelerated processor for the Automated Front End (AFE). This automated system performs a triage of the data to avoid the classic problem of data falling on the floor.

Data is classified and presented with a score that indicates proximity to a topic of interest. The Analyst is the final consumer of the data derived from the system. How well the triage is filtered is a matter of engineering. Finding too much data overwhelms an analyst and causes the system to be quickly labeled as unusable. Finding too little data is also unsatisfactory because important information is overlooked.

The AFE system is a hardware implementation of a Machine Learning technique. Classification of data is performed at high speed by using a Field Programmable Gate Array (FPGA) on the Field Programmable Port Extender (FPX) [19, 20] platform. The system processes and annotates data at gigabit rates.

While developing the project, many hardware and software design considerations factored into the configuration of the AFE system. The hardware was implemented as a set of modules using multiple FPX platforms that communicate through a stock of boards. Partitioning and verification of each stage of data flow were necessary for the successful integration of the system.

Software systems were used during system development to test, verify, and implement parts of the AFE system. Software simulation used the same data format standards as did the hardware. A tool set was developed to run batches of jobs through the system. Through the use of standard interfaces for the components of the AFE system, multiple developers were able to build upon and integrate components into the system.

1.2 Thesis Outline

This thesis begins with a brief history of classification techniques and the issues that are possible in a hardware implementation of an algorithm. Next, a description of the platform that is used to implement the AFE system, the Field Programmable Port Extender (FPX), is provided. Next, a description of the hardware design considerations and their affect on the algorithms for classification is provided. An experiment is conducted to show well the system can classify newsgroup articles into topics. The experiment investigates how trade offs in the parameters of the system affect accuracy and throughput. An appendix details file formats used for hardware and software are described in detail. Another appendix provides a description of the software developed for simulating the system. Specifically, appendix A contains the information on the XML file formats used in the system, while appendix B describes how the software was used to emulate the functioning of the AFE.

Chapter 2

Semantic Processing

2.1 Background on Classification Techniques with Supervised Learning

The study of supervised learning has evolved over many years. The goal in this field of study is to learn a topic from a training set and to be able to use the knowledge to classify new items. This problem falls into the category of a Machine Learning supervised learning technique. Examples of the techniques include: Latent Semantic Indexing (LSI) [10], Information Theory [27], NGrams [9, 26], and Bayesian Reasoning [21].

2.1.1 Latent Semantic Indexing

Latent Semantic Indexing (LSI) was introduced by Deerwester et. al. in [10]. The paper introduced a new method of doing text classification and Information Retrieval. The technique uses vectors to represent a concept. A concept is determined by a set of documents. In order to retrieve the concept vector, the statistics of words in documents of the concept are arranged into a matrix:

$$A = [a_{ij}]$$

where a_{ij} is the frequency of term i in the document j. From the document matrix, Singular Value Decomposition [10], SVD, is used to generate n-dimensional concept vectors that represent the k orthogonal dimensional dispersion of the data in the matrix. The method is useful in reducing the dimensionality of the problem. By determining the closest angle between the unknown, A, and concept vectors, C_i, we classify the unknown documents.

$$max_i(\cos(\theta_i)) = max_i(\frac{|A \cdot C_i|}{|A||C_i|})$$

2.1.2 Information Theory

Techniques of Information Theory [27] incorporate extensive use of probabilistic analysis. One of the techniques is the use of Information Theoretic Co-clustering [11]. The idea is to determine the probability distribution of words across concepts. With this information, when an unknown document needs to be classified, each word can be scored against the learned probability distribution.

The major benefit of the Co-clustering idea comes from dimensionality reduction. Rather than tracking every possible word that could appear in documents; a reduction in the size of the array is performed. There are many methods possible for reducing the number of words: stemming, dropping stop words, dropping the exceptionally infrequent words, and clustering. Co-clustering clusters the probability distributions for words. If the desired output is to have 500 word groups, then all words in the training set are grouped into 500 groups. The centroid of each group then gives the score for each concept when a word from that group is seen. K-Means [12] can be used as the clustering algorithm. Concepts can be developed from the training set of documents.

2.1.3 *n-grams*

A model of text mining that uses statistical inference is the method introduced by Damashek and Shaner [9, 26] that utilizes *n-grams*. The idea behind the *n-grams* is the use of strings of n characters to match content in text. In the AFE project, an additional circuit uses a tetra-gram method to identify the language and character set of a document [16].

Portions of words can be nearly as indicative of concepts than whole words. Certainly this is true when identifying languages.

The *n-grams* method is to take a sentence like:

Foundations of NLP

and divide the sequence into a quantized pieces. For tetra-grams, the result of tokenizing would be:

'Foun', 'ound', 'unda', 'ndat',
'dati', 'atio', 'tion', 'ions',
'ons ', 'ns o', 's of', ' of ',
'of N', 'f NL', ' NLP'

Each character string is counted and a probability is derived such as:

$$P(C_i|g_j)$$

where C is a set of concepts and g is the set of *n-grams*. For each ngram, a distribution over all concepts is derived:

$$P(g_j) = \sum_{i=1}^{n} P(g_{ij})$$

where there are n concepts. When putting n-grams into practice, a document is parsed into its n-grams and the probabilities are used to determine the categorizations.

The n-grams model is exceptionally accurate at identifying languages with very little training information. Languages are different in word sets and character sets. The most identifiable portions of words are the beginning and end of words. In languages, these n-grams become very powerful identifiers.

2.1.4 Statistical Reasoning on Words and Word Positions

The methods that are possible with statistical inference are numerous. Beyond the n-grams method, statistical reasoning can take on many possible incarnations.

Bag of Words

Counting the occurrence of words is what is called the "Bag of Words" (BOW) [2] technique. All words are counted from a labeled training set to determine the probabilities of each word in relation to the topics being used for training. The sentence "The boat sailed through the water to port." has several words that indicate the sentence relates to marine topics. Because contextual relationships are eliminated, a sentence like "Let's blow by the bridge" is very close in interpretation to "Let's blow up the bridge." The two sentences differ by one word each, "by" and "up." Both statements are slang while only one is threatening. Using the *bag of words* method makes the assumption that concepts are defined by a set of words. When thinking of a concept such as "bridge," it is possible to think of all the words that might get associated with the term. For example, words that might show up in documents that are talking about the concept "bridge" are words like "bridging," "suspension," and "toll."

Contextual Relations

The above example does not take into account the relation of words such as "The Bridge" might refer to a night club or a restaurant. So, extending words around the term "bridge" can cause confusion as well as be completely wrong in certain circumstances. "Bridging the Gap" could just as easily be a usage that is not considered. Therefore, contextual relations can be important. The probability of the word "driving" in relation to the word "bridge" would definitely be of greater occurrence than "painting." The latter term would not be inconceivable but "driving" would occur in greater frequency.

Many methods incorporate these relations. Only considering words that are in the same sentence as a metric is suitable for Spam classifications; however, considering the entire document as a contextual relation would be better suited for the entire concept classification system.

2.1.5 Methods Suited for High Speed Classification - The One Look Paradigm

Classification at Internet backbone link speeds requires significant processing power. The *One Look Paradigm* dictates that data is looked at only essentially once. This is limiting in the context of concept classification from packets as they pass on networks. The *One Look Paradigm* can operate on data passed through a pipeline. A hardware accelerator was developed to analyze, count, and score the words in the documents passing over a network.

Consider the problem of attempting to process the payloads on a gigabit link. Subtracting all the data needed for protocol headers and connection negotiation, the amount of data that can be passed in a single second is approximately half of that measured data on the link. This is the amount of data available to be moved in the payloads of the packets and is the data of interest. Usually, data is passed in EMail, Web pages, and text messaging. Each of these methods uses a protocol to preserve the structure of the data but does not provide meaning in the analysis of payloads. Within HTML, a large portion of the data is used for formatting and could be stripped using a Context Free Language parser [5]. Thus, the amount of data that is interesting to process is only a fraction of actual traffic that moves across a network.

Still, the amount of stored data which needs to be processed quickly adds up. In on-line terms, a complete document would be a complete flow of traffic. Flows can be continuous over time, generating megabytes to gigabytes of data.

The study of how a limited number of relationship could be analyzed is proposed for future work on this project. In addition, clever engineering of the system could maximize the amount of useful data that could be stored.

With a few twists that will be explained in later chapters of this document, the *bag of words* technique is what has been implemented in hardware. This is a good first step in the analysis of concepts at high speeds from on-line deep packet analysis. Also, this limitation in complexity fits in the *One Look Paradigm* because every stage of the method can be put into a computational pipeline. Feature extraction, feature counting, and finally scoring would be the stages for any text classification system. These are the exact stages in the AFE pipeline.

2.2 Hardware Semantic Processing

Although FPGAs can be used to process data at high speeds, certain considerations must be taken into account. For example, floating point computation is possible but it is costly in terms of hardware circuit resource usage. One floating point multiplier is 4 times the size of an integer multiplier. This would not be a major problem if only one multiplier is used. However, in semantic processing, a large number of features are needed to represent a document. Therefore, a large number of multipliers are needed to implement the complete processing system.

2.2.1 Feature Extraction - The Base Word Circuit

Feature extraction is performed by a circuit called the *Base Word Circuit.* Features in semantic processing are words. The extraction process of words is discussed in greater depth in the appendix section of the Base Word Circuit. Basically, the circuit will produce a list of features it has found and it passes them down the pipeline.

2.2.2 Feature Counting - The Count Circuit

The feature counting function is performed by the Count circuit. The Count circuit receives a list of extracted features and increments the appropriate counters for the document. The circuit accumulates statistics of words that appear in the document by tracking the counts for each flow. The hardware uses integers to track the counts.

As one of the design considerations of the AFE system, 2 KBytes is reserved for each vector. This has been translated into a 4000 feature vector with 4 bit counters. So, each feature can count to a saturating value of 15 (0xf). When a document is finished counting, the vector is passed down the pipeline.

2.2.3 Scoring - The Score Circuit

A function that determines how closely a document matches a concept is called the Score circuit. Scores are computed as a dot product computation between a document's representation generated by the count circuit and known concepts. The concepts are represented with the same number of feature dimensions and either 4-bit or 8-bit integer counters. With 4000 dimensions either 2KB or 4KB are used to represent a concept. The Score Circuit computes a dot product of the document against all the loaded concepts. When the computation is complete, the results are passed out of the system.

2.2.4 Hardware Design Configuration

Because the AFE system represents documents with 4000 dimensional vectors, a mechanism was needed to map words to a feature in the range of 4000. The mapping is represented by a Word Mapping Table (WMT). In the Base Word Circuit, words have a minimum word length of 3 characters and are truncated at 16 characters. Each dimension of a document has a 4-bit counter that saturates at 15 (0xf). Concepts can be represented with up to 15 vectors with 8-bit counters or up to 30 vectors with 4-bit counters. A set of concept vectors are referred to as a Score Table (ST) in the AFE lexicon.

These values provide constraints on the algorithm used to develop a WMT and ST. The experiment section, chapter 4, of this document will explore how the precision of the system is affected by altering these design constraints. If documents can be represented with less data, then it may be possible to increase the throughput or concept capacity of the system.

Chapter 3

Processing Platform: Field Programmable Port Extender (FPX)

3.1 The FPX Platform

The Field Programmable Port Extender (FPX) was developed by members of the Reconfigurable Networking Group within the Reconfigurable Network Group at Washington University in St. Louis [17].

The FPX platform was designed with a combination of hardware suitable for processing data in high speed networks. Network connectivity for the device is through the Network Interface Device (NID). A Xilinx XCV600E FPGA is used to route data between network devices and receives commands over the network. One type of command which is sent over the network is used to reprogram the larger Xilinx XCV2000E FPGA device on the FPX platform called the Reprogrammable Application Device (RAD). Circuits can be designed in VHDL or other hardware description languages, compiled for the RAD, then downloaded to the FPX platform. Two banks of Static Random Access Memory (SRAM) are connected to the RAD along with off chip Synchronous Dynamic Random Access Memory (SDRAM). One gigabyte of space is available in the SDRAM.

The FPX can process data in hardware at Gigabit link rates. The FPX platform was designed to be used in high speed networks and therefore it is exceptionally well

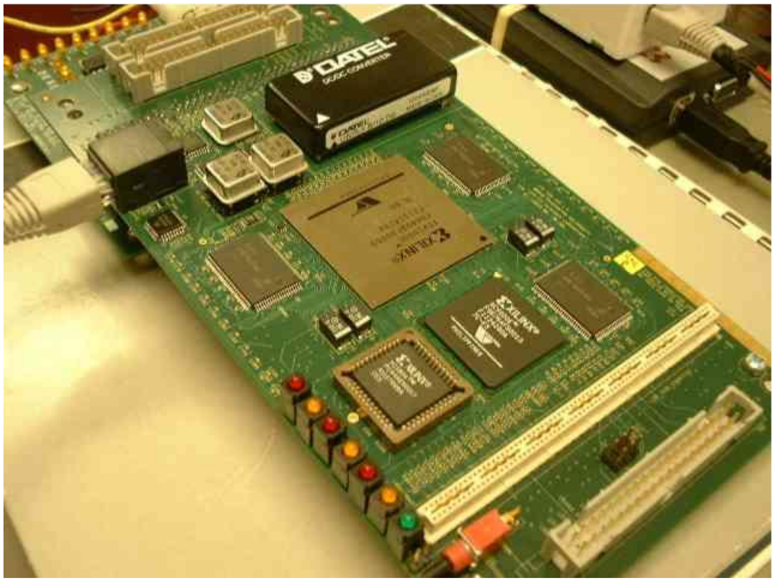

Figure 3.1: The FPX

suited for the task of network deep packet analysis. The FPX platform can process at rates that far surpass conventional computing devices. It can be configured to do many different types of tasks. For the purposes of the system, it is configured to analyze the semantic content in TCP/IP flows.

3.1.1 Configuration of the AFE Hardware

The configuration of the hardware-accelerated AFE system is a stack of 5 FPX cards as shown in figure 3.1. Data is received by the stack from a Line Card (LC). Two types of cards are supported: OC48 and Gigabit Ethernet. OC48 LC is shown in the configuration of figure 3.2. Next, four FPX cards are stacked to process the payloads of packets. Each stage of the circuit is implemented with an individual FPX card. The cards are connected together with their network interfaces directly connected to the FPX above it. A fifth FPX card (called the NID PT) appears on the right-hand side of the stack which reports the output in UDP packets which are sent to a server which catches the results from hardware [13, 18].

The processing stages of the AFE system are broken down into basic functions. First, the TCP Circuit extracts payloads from network traffic. Second, the Base Word Circuit analysis payloads by tokenizing streams into acceptable byte sequences and passing them through a dimensionality reduction system called the Word Mapping Table (WMT). Third, values are passed to the Count Circuit which builds a representation of the the document into an array. Finally, the Score Circuit computes a dot product with preloaded concepts represented by arrays of values. The output of the score circuit is passed to the final NID.

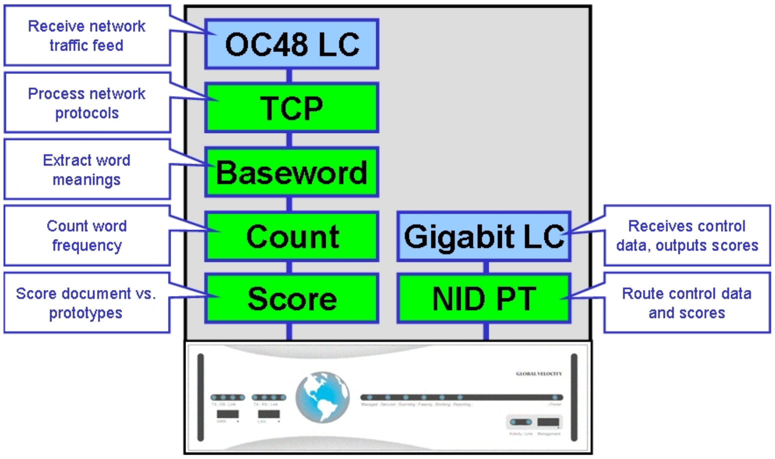
Receive network traffic feed
OC48 LC
Process network protocols
TCP
Extract word meanings
Baseword
Count word frequency
Count
Gigabit LC
Receives control data, outputs scores
Score document vs. prototypes
Score
NID PT
Route control data and scores
GLOBAL VELOCITY

Figure 3.2: Card Stack Configuration of AFE

3.2 Processing Stages

3.2.1 Network Payload Extraction - The TCP Circuit

The first stage of content processing is the extraction of network traffic payloads. This is provided by the TCP Circuit [25]. The TCP Circuit tracks flow information including source and destination IP values, source and destination ports, and sequence number tracking.

Important aspects of the TCP circuit affect the amount of content being processed. Sequence number tracking insures that retransmitted data is not reprocessed. Off-chip memory is used to store the context of a TCP flow, source IP, destination IP, source port, destination port and sequence number. A hash of the addresses and port fields generate a FlowID. The FlowID is a 20 bit value. The TCP circuit can be made to accept only data with sequential tracking numbers. This means that if the circuit sees a data flow that is intended to have a sequence of A B C D, but the circuit sees A C D B, the B packet will be accepted in the system but its payload will be ignored because it is out of sequence. While this can cause some loss of data, it does not cause much. Schuehler [23] has shown that a very small number of packets appear in most network flows out of sequence.

Next, within TCP, a completed flow is signified by a FIN (finish) or RST (reset) packet. When a FIN or RST packet is seen for a flow, all subsequent traffic that matches the stored flow information will be ignored by the circuit until a new SYN is transmitted.

Information about the flow being processed is made available to downstream FPGA circuits in the stack of FPX cards. The source and destination IP addresses and ports are made available along with a segment of 4 bytes. Every clock tick of the system provides a new set of bytes for analysis. This functionality provides a stream of data for the next circuit in the processing pipeline.

3.2.2 Feature Extraction - Word Building and The Base Word Circuit

The Base Word Circuit tracks flow information while building acceptable byte strings. Byte strings are derived from word building engines. Currently, there are two basic word building engines which differ by how they process characters as single-byte and two-byte. The single-byte engines are for processing character sets like ASCII [4], Windows Code Pages [7], and ISO character sets [29]. The two-byte engines are specifically for UTF-16 code pages [15]. To build a word, the character sequence must be a series of acceptable characters. For the single byte engine, the minimum length of a word is 3 bytes. For the two byte engine, 6 bytes are needed. Character sequences without spaces can be longer then 16 characters, but anything past 16 is discarded.

It is possible that words span two packets. For the word “river”, one packet could end with “riv” and the next packet for the flow could start with “er.” The word “river” will be correctly parsed because the Base Word Circuit preserves the context of the flow. In general, the last 16 bytes of the payload are stored in memory by using the FlowID as a memory locator. When a packet from an established flow arrives, the BWC will look at the stored data and continue tokenizing.

When a word is extracted, the byte string is hashed to produce a 20-bit value with a range of one million entries. This value points to a memory location that has a value stored. The stored value then contains a value in the range of 0 to 4000. This dimensionality reduction method is performed by the Word Mapping Table (WMT). The BWC creates lists of the values produced by using the WMT and passes them down the AFE pipeline.

3.2.3 Count Array Construction - The Count Circuit

The Count Circuit tracks flow information while building the count arrays. The FlowID is used as a memory location for the storage of a flow’s count array. Just like the BWC, context of flows are preserved. A document vector is built from a flow that spans over many packets. When a flow is complete, the document vector is passed down the AFE pipeline.

3.2.4 Scoring Concepts - The Score Circuit

The Score Circuit is the only circuit that does not need to keep track of flow context. Data is received by the Score circuit only on completion of a flow. When the count array is received by the circuit, it is scored against all the loaded concept vectors by computing a dot product between the document vector and each of the stored vectors that represent a concept. A score is determined for the count array against all the concept vectors. The scored results are then reported by the system in within a UDP packet.

3.3 Post Processing

The reporting module (NID-PT) outputs UDP data-grams that summarize the content of the flow. The scores are interpreted to determine the correct classification of the flow. At minimum, interpreting the results just requires finding the largest score. Additional post-processing can be performed to determine the most likely topic or topics of the flow. The method of processing the results is dependent on the methods used in training the WMT and ST. Later discussion in this document shows the importance of post processing.

Chapter 4

Parameter Manipulation Experiment

The choice of parameters of the hardware affect how the system processes data. One factor is the minimum and maximum size of acceptable character sequences. Other factors include the size of dimensionality and precision of feature representation of document and concept vectors. Sensitivity analysis of these parameters provides insight to determine the performance characteristics and accuracy for variations of the system. If a system with a smaller configuration can be shown to perform the same task with similar metrics to the original, then it is possible to implement the system by using less hardware resources. Alternatively, if larger numbers of features and topics need to be extracted, more hardware could be required.

4.1 Parameters

4.1.1 Minimum Word Length

In normal operation, word lengths have a minimum of 3 characters and a maximum of 16. In the experiment, all words are still truncated to 16 characters. However, for this analysis, minimum word lengths are varied from 2 to 8 characters.

4.1.2 Count Bit Resolution

Another parameter considered was the bit resolution of the bins in the document count arrays. The value is nominally 4 bits, allowing for a range of 0 to 15 in each bin. The values were allowed to change such that the values were 4 bit (≤ 15), 3 bit (≤ 7), 2 bit (≤ 3), and 1 bit (≤ 1) resolution.

4.1.3 Score Bit Resolution

The next parameter considered was the bit resolution of the score table. The bins in the score table have an option of 8 bits (≤ 255) or 4 bits (≤ 15). The experiment utilized both possible values of resolution.

4.1.4 Dimensionality - Number of Features

The final parameter considered was the number of features, the number of bins in the representation of the documents and concept vectors. In the first inception of the AFE system, 4000 dimensions were available for features. The parameter manipulation experiment uses dimensions of 4000, 3000, 2000, 1000, 500, and 250.

4.1.5 Parameter Variation

The parameters of the AFE system were varied as follows: there were a total of 7 values varied in the size of minimum word length, 4 values varied in count bin resolution, 2 values varied in score table bin resolution, and 6 values varied for feature set number, for a total of ($7 \times 4 \times 2 \times 6$) 336 combinations. In order to get a valid set of results from a specific combination of the parameters, 30 individual runs were constructed. With 30 runs per configuration, the total number of experimental runs is 10,080. Also, the parameter manipulation experiment uses both the Mutual Information (MI) and Heuristic algorithms [13, 18] for developing the WMTs and STs. Therefore, there were 20,160 total runs of the simulated system.

4.2 Corpus Profile

The corpus used in the experiment is from a number of Usenet news groups [14]. The profile of the corpus is described in Table 4.1. The documents are divided in half for all groups except the "chaff" group. The "chaff" group was used as an interference group. The group is meant to represent valid data that is not of interest to the other topics. In this experiment, a subset of articles in the group are used in training so that documents that fall into the classification can be identified. By assumption, when large amounts of data are processed, the ratio of interesting items to uninteresting items will be small. Training on elements that are not desired will provide an attractor class for undesired items, thus, reducing the rate of false positives. Table 4.1 shows the labels and number of documents used for training and testing.

Table 4.1: Corpus

newsgroup	label	training	testing	total
sci.archeology.moderated	archeology	70	70	140
alt.sports.baseball.stl_cardinals	baseball	32	34	66
rec.equestrian	equestrian	41	42	83
misc.consumers.frugal_living	frugal	16	18	34
soc.libraries.talk	libraries	16	18	34
sci.logic	logic	31	31	62
rec.martial_arts.moderated	martial_arts	28	30	58
comp.ai.neural_nets	neural_nets	23	26	49
comp.programming.threads	programming	47	48	95
humanities.music.wagner	wagner	29	32	61
misc.writing.moderated	writing	37	38	75
talk.origins	chaff	368	10402	10770
	Total:	738	10789	11527

An experiment was performed to test the precision of the AFE system while manipulating the parameters of the simulated hardware. When increasing the minimum word length, the number of words derived from the training set of documents decrease. When the minimum word length is too short, many words are rejected. This distribution of word lengths is shown in figure 4.1.

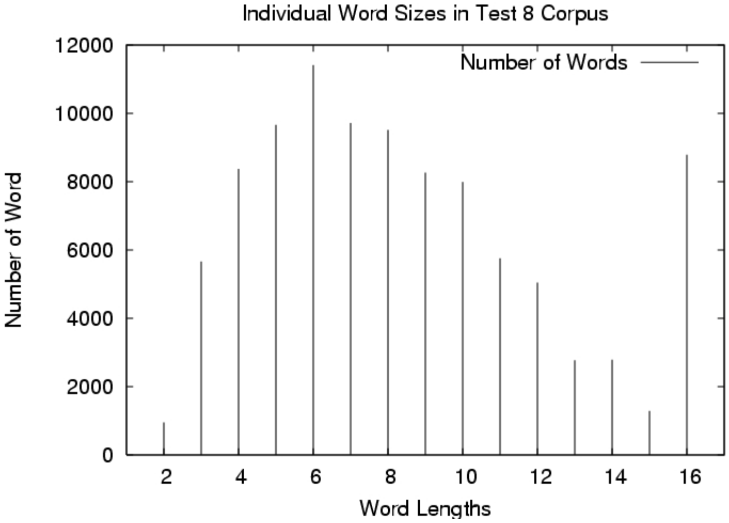

Figure 4.1: Word Sizes for Corpus

Table 4.2 shows the number of words in all documents for each concept. A breakdown of the number of words in each concept by word size based on the training set of documents is shown in the Table 4.3. The values shown are the average values for all 30 sets of randomly selected training documents.

Table 4.2: Number of Words in All Documents per Concept

	Number of Words of Each Length								
Topic	2	3	4	5	6	7	8	≥ 9	Total
archeology	141	366	661	860	927	926	790	1642	6313
baseball	139	286	390	426	441	381	240	433	2736
equestrian	121	227	539	577	537	494	349	516	3360
frugal	65	151	306	317	275	230	151	258	1753
libraries	129	221	359	407	408	400	335	621	2880
logic	105	189	275	332	341	348	264	594	2448
martial_arts	203	553	898	1115	1277	1264	1090	2144	8544
neural_nets	110	206	298	353	355	403	276	673	2674
programming	138	277	386	414	398	405	311	643	2972
wagner	104	242	501	648	682	705	624	1247	4753
writing	104	200	366	450	431	419	321	567	2858
chaff	944	5206	7683	8655	10274	8576	8438	40241	90017

Table 4.3: Average Number Words in Training Documents per Concept

	Average Number of Words of Each Length							
Topic	2	3	4	5	6	7	8	≥ 9
archeology	107.7	262.1	477.4	583.8	605.1	594.6	492.9	121.9
baseball	108.7	216.3	314.2	317.5	322.6	276.1	167.2	37.3
equestrian	82.8	165.8	392.2	387.6	352.4	307.1	208.7	37.0
frugal	47.2	105.3	208.2	198.1	165.1	131.7	88.0	18.3
libraries	91.7	151.6	243.9	256.3	240.4	241.9	194.9	43.1
logic	76.2	141.3	209.5	243.1	242.6	246.5	177.8	49.2
martial_arts	146.6	377.1	660.2	804.7	902.2	878.9	742.9	176.2
neural_nets	77.1	144.5	216.4	237.8	229.7	255.0	168.4	50.9
programming	97.5	196.2	288.1	287.5	273.0	274.8	198.0	49.0
wagner	76.2	169.5	350.4	426.0	429.1	436.5	365.9	87.0
writing	70.9	142.9	265.8	300.0	277.4	262.1	190.7	41.1
chaff	478.6	1083.4	1443.9	1641.7	1859.2	1794.0	1632.6	588.2

4.3 Performance

Performance of the classification system will be defined by the use of the metrics *precision* and *recall*. For a given class, precision is defined as the number of documents correctly retrieved in that class divided by the total number of documents retrieved in the class, whether correctly assigned to the class or not.

$$\begin{aligned} C &= \text{set of documents for a single class} \\ D &= \text{number of documents} \in C \text{ retrieved} \\ F &= \text{number of documents} \notin C \text{ retrieved} \\ D+F &= \text{total number of documents in the classification} \end{aligned}$$

$$\text{Precision of } C = \tfrac{D}{D+F}$$

For a given class, recall is defined as the ratio of the number of documents correctly retrieved in that class, divided by the total number of documents that could have been correctly retrieved in the class.

$$\begin{array}{lcl} C & = & \text{class of documents} \\ D & = & \text{number of documents} \in C \text{ retrieved} \\ E & = & \text{number of documents} \in C \text{ not retrieved} \\ D+E & = & \text{Total number of documents in } C \end{array}$$

$$\text{Recall of } C = \frac{D}{D+E}$$

4.4 Setting Thresholds for Evaluation

The algorithms used in this experiment have different evaluation metrics. Each metric is specific to the chosen method of supervised learning.

4.4.1 $cos(\theta)$ for MI

The MI algorithm is evaluated with a metric of $cos(\theta)$ that is defined as:

$$cos(\theta)_{ij} = \frac{|D_i \cdot C_j|}{|D_i||C_j|}$$

where D_i is a document vector from the set of all documents and C_j is a concept vector in the set of all concepts. When evaluating a document vector against a set of concept vectors, the document is assigned to the concept class that achieves the highest score.

$$class\{D_i\} = max_j\{cos(\theta)_{ij}\}$$

Simply selecting the maximum of the calculation is not sufficient. With this classification metric, all documents in the setting will be classified into a group. This is called “gun to the head” classification, mixing strong reasons for classification with weak. A threshold can be used to control the trade off between high recall with low precision, and low recall with high precision.

4.4.2 Thresholds of $cos(\theta)$

The determination of the thresholds is part of the learning phase of the MI algorithm because they are hard to estimate apriori. The threshold is set by finding the lowest $cos(\theta)$ value of the correct classifications of the current class. That value will serve as the minimum score for allowing a document to be classified in the group. Therefore, even if a document scores best with a certain class, the score must be at or above the threshold for the class, otherwise the document is rejected.

4.5 Order Statistics for Heuristic Algorithm

The Heuristic algorithm determines classification by simply determining the highest score from the dot product of a document vector against a set of concept vectors.

$$class\{D_i\} = max_j |D_i \cdot C_j|$$

Where D_i is a document vector and C_j is a concept vector. The difference between this and the MI method is the lack of normalization with respect to either the concept vector or the document vector. This method forces classification of all documents unless thresholding is used.

4.5.1 Thresholds of Order Statistics

The method to determine a threshold for the Heuristic algorithm is to determine a magnitude of significance between the greatest score and the second greatest score. This value was determined in training. Once again, the correct classification with the least value is used for this metric.

$$threshold_j = min_i\{\tfrac{max_{ij}}{2^{nd}max_{ij}}\}$$

Documents are not classified (rejected) when the classification score does not meet the magnitude significance threshold.

This method of selecting thresholds maximizes recall at the expense of precision. A more thorough analysis would study various points in the precision/recall trade off.

4.6 Results

In the analysis of the experiment, thresholds for the individual runs were determined for each run from the method described above. A confusion matrix was calculated for both the training data and the test data. Table 4.4, for example, shows the confusion matrix for run 4 of the MI algorithm with the parameter setting of: minimum word length = 5, count bit resolution = 2, score table resolution = 8, and dimensionality = 3000. Table 4.5 shows the confusion matrix for the testing of that run.

Table 4.4: MI Confusion Matrix for run 4 training of: mwl=5, count=2, stable=8, dim=3000

	archeology	baseball	equestrian	frugal	libraries	logic	martial_arts	neural_nets	programming	wagner	writing	chaff	reject	recall
	70	0	0	0	0	0	0	0	0	0	0	0	0	100
	0	32	0	0	0	0	0	0	0	0	0	0	0	100
	0	0	41	0	0	0	0	0	0	0	0	0	0	100
	0	0	0	16	0	0	0	0	0	0	0	0	0	100
	0	0	0	0	16	0	0	0	0	0	0	0	0	100
	0	0	0	0	0	31	0	0	0	0	0	0	0	100
	0	0	0	0	0	0	28	0	0	0	0	0	0	100
	0	0	0	0	0	0	0	23	0	0	0	0	0	100
	0	0	0	0	0	0	0	0	47	0	0	0	0	100
	0	0	0	0	0	0	0	0	0	29	0	0	0	100
	0	0	0	0	0	0	0	0	0	0	37	0	0	100
	0	0	0	0	0	0	0	0	0	0	0	368	0	100
precision	100	100	100	100	100	100	100	100	100	100	100	100		

Table 4.5: MI Confusion Matrix for run 4 testing of: mwl=5, count=2, stable=8, dim=3000

	archeology	baseball	equestrian	frugal	libraries	logic	martial_arts	neural_nets	programming	wagner	writing	chaff	reject	recall
	17	0	0	0	0	0	0	0	0	0	0	0	51	25.00
	0	18	0	0	0	0	0	0	0	0	0	0	14	56.25
	0	0	11	0	0	0	0	0	0	0	0	0	29	27.50
	0	0	0	0	0	0	0	0	0	0	0	2	14	0.00
	0	0	0	0	0	0	0	0	0	0	0	5	11	0.00
	0	0	0	0	0	14	0	0	0	0	0	2	13	48.28
	0	0	0	0	0	0	15	0	0	0	0	2	11	53.57
	0	0	0	0	0	0	0	7	0	0	0	3	14	29.17
	0	0	0	0	0	0	0	0	37	0	0	0	9	80.43
	0	0	0	0	0	0	0	0	0	9	0	4	17	30.00
	0	0	0	0	0	0	0	0	0	0	13	2	21	36.11
	1	0	0	0	0	1	2	0	0	0	0	9389	1007	90.28
precision	94	100	100	0	0	93	88	100	100	100	100	99		

4.6.1 Averaging Runs

The last column of table 4.4 and of table 4.5 shows the recall for documents in each class. The last row shows the precision. In order to determine how the algorithm performed, the averages of the precision and recall values were computed and listed in the table. An average is computed evenly per class, irrespective of the size of the class. The average recall for the test set in the table 4.5 is 39.72%. The average precision of the test set in the table 4.5 above is 81.32%. For the Heuristic algorithm, this run had an average recall value of 34.74% and an average precision of 79.37%. Once the average of each individual run is calculated, the average over the 30 runs is calculated with a 95% confidence interval. A confidence interval is calculated:

$$95\% \text{ confidence interval} = 1.96 \cdot \sigma_m$$

where

$$\sigma_m = \frac{\sigma}{\sqrt{N}} \text{ is the standard error of the mean.}$$

An example of the plot for a dimensionality of 3000 is shown in graph 4.2 and 4.3

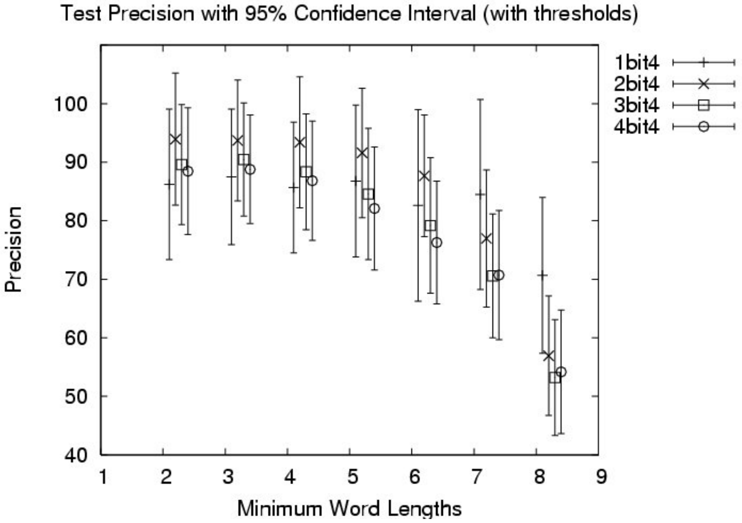

Figure 4.2: MI Algorithm: Precision for Dimensionality of 3000 and Score Table Resolution of 4

All the graphs for the runs are available on-line at: *http://bb3.arl.wustl.edu/∼andrew/test8/*.

4.6.2 Analysis of Results

When viewing all the graphs of the run, it is possible to determine where the dimensionality have a substantial effect on the performance of the algorithms. When combining the results for all dimensions in graphs 4.4 and 4.5, the inverse relationship between precision and recall can be seen most clearly in plots of the Heuristic runs. However, viewing the results from the MI runs does not show such a great separation of results.

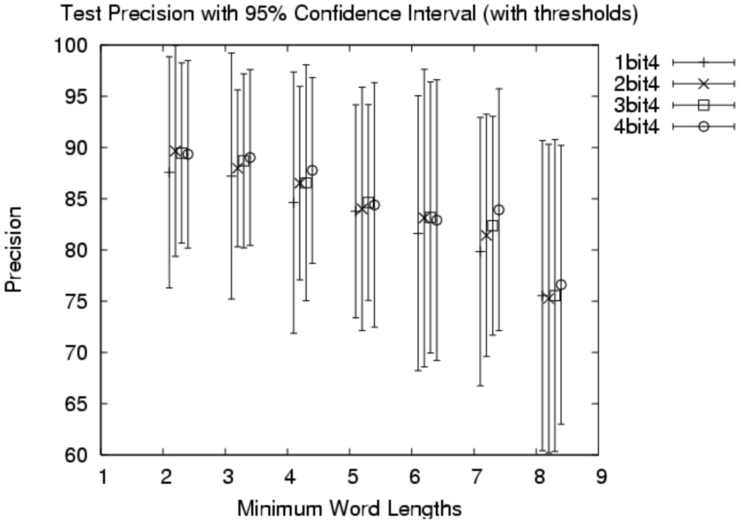

Figure 4.3: Heuristic Algorithm: Precision for Dimensionality of 3000 and Score Table Resolution of 4

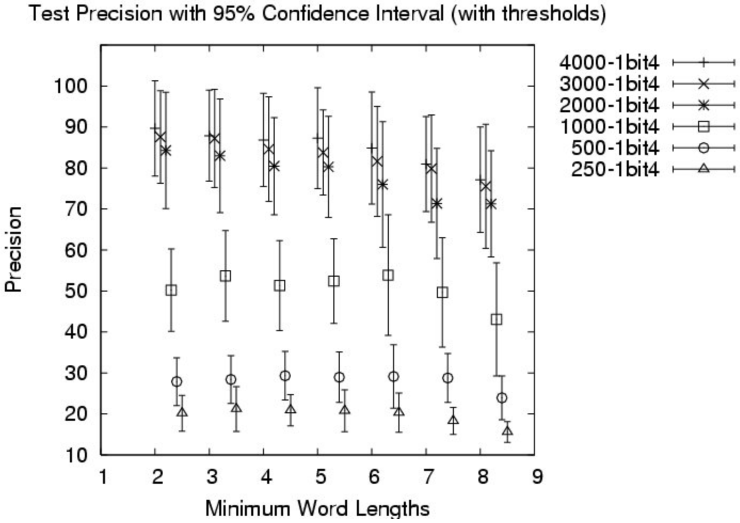

Figure 4.4: Heuristic Test Precision for 1 bit Count and 4 bit Score

Minimum Word Length

Consistently, a trend can be observed in all graphs that shows that as the minimum word length increases, the precision decreases. Conversely, recall goes up as the minimum word length increases. However, the values in the range of 2, 3, 4, and 5

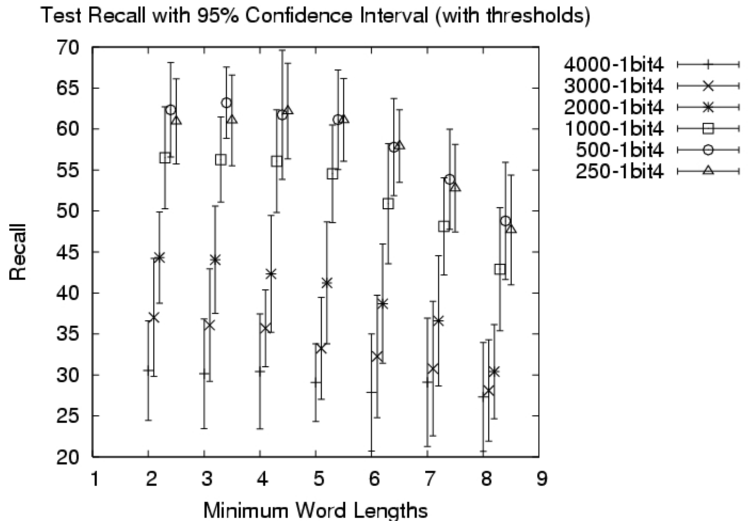

Figure 4.5: Heuristic Test Recall for 1 bit Count and 4 bit Score

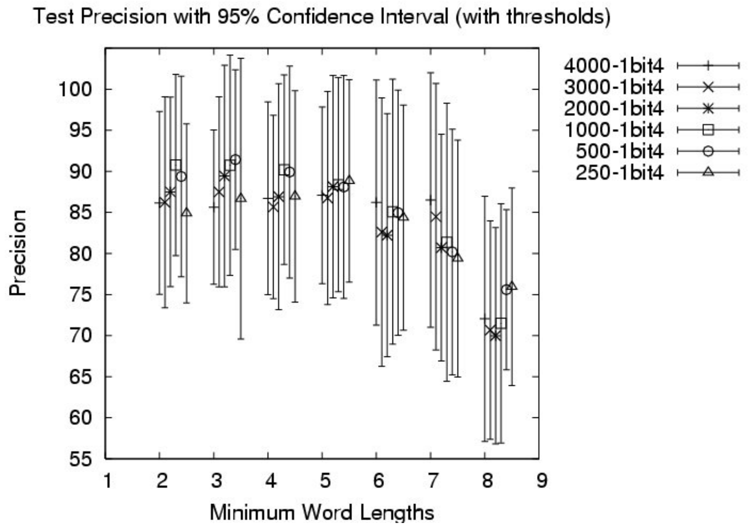

Figure 4.6: MI Test Precision for 1 bit Count and 4 bit Score

seem exceptionally close to each other in all configurations. Both the MI and Heuristic algorithms parallel each each other when considering only a change in the minimum word length. There is an incentive to limit confusing or nonsense words as well as to make the system do less work. The AFE system looks at bytes in chunks of 4 per clock cycle. A worst case scenario with a minimum word length of 2 would have 4

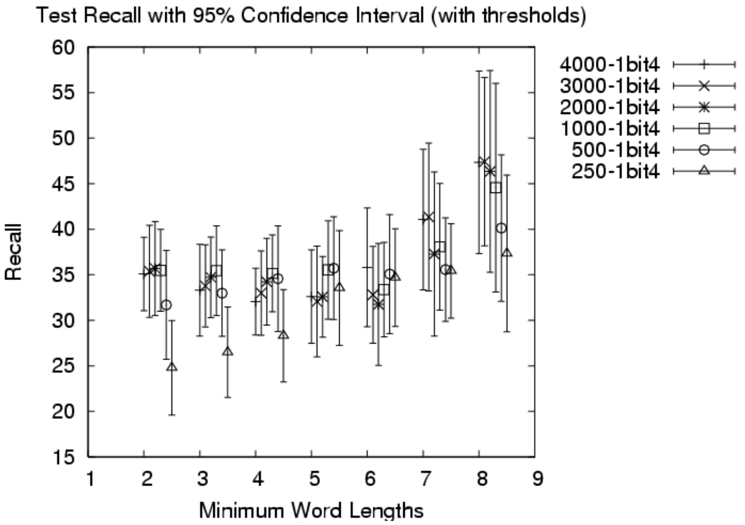

Figure 4.7: MI Test Recall for 1 bit Count and 4 bit Score

words for every 3 clock cycles. Therefore, it is possible to produce more than one word on a clock cycle. This may be possible to handle in a circuit, but if the system does not need to be overtaxed when there is no need for it should to be. With a minimum word length of 3, words could be produced as often as every clock cycle. Although it is currently possible for the count circuit to handle this load, as the system currently does, but with the goal of making the system faster, the creation of base words is a detriment to speed. A small minimum word length forces multiple clock cycles to be used in order to produce a word, timing in the circuit can be more predictable. If 3 is the choice of minimum word length, then the circuit must be designed to handle a new word every clock cycle. If a minimum word length of 4 or 5 is chosen, fewer words need to be counted. It is also the case that most short words are stop words that should be ignored for semantic analysis. Therefore, selecting a minimum word length of 4 has an added advantage.

There are other motivations for using larger minimum word lengths. It is not desirable to retrieve acceptable strings from binary files text. The AFE Base Word Circuit does not use a dictionary to confirm that words fall into a dictionary. So, words like “ZKT” are treated the same as words like “YES” if they happen to hash to the same base word table entry. It happens that the byte values of “ZKT” translate to the hex values “0x5A 0x4B 0x54” which in terms of an RGB mapping of an image is a dark purple color. It is also possible that these nonsense words can produce the same base word as a word like "RIVER" with a hash collision. Reducing noise is one of the benefits of increasing minimum word length.

Number of Features

The separation that is seen when changing dimensions in the Heuristic algorithm remains when count bit and score bit resolutions are increased. The confusion matrices for the Heuristic algorithm show that the precision is low because of a large amount of the chaff which were misclassified. This is especially noticeable when the parameters of the system are set so that there are 500 or 250 bins for features. The Recall for those runs is high. However, if the goal of the system is to present data with high precision, limiting the number of features for the Heuristic algorithm becomes an issue. This is even more evident looking at the precision of the Heuristic algorithm when the number of features drops from 2000 to 1000. The runs with 4000, 3000, and 2000 feature results are close for all other parameter manipulations. But, 2000 features appears to be the lower end of the number of features necessary to maintaining high precision. Using less than 2000 features significantly degrades the Heuristic algorithm.

Results for the MI algorithm show that the change in the number of features is not a as much of an issue for the performance of the algorithm. The results show that the MI is not much affected when the number of features was reduced to 2000. The reason is that for the categories and training data used in these experiments, the MI algorithm rarely uses more than 1500 features.

Count Bit Resolution

Results from running the Heuristic algorithm show only minimal differences in precision occur when the count bit resolution changes. This effect can be observed in Figures 4.8 and 4.9. The MI algorithm does show some difference as a result of changes to the count bit resolution. Figures 4.10 and 4.11 show how changes have a greater affect on the performance of the system. Consistently in the MI algorithm, 2 bit count resolution shows to works a little better on average than the other values.

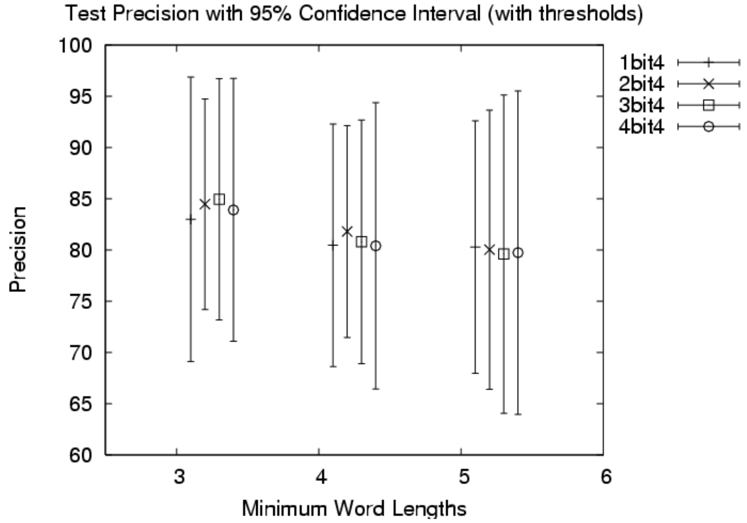

Figure 4.8: Heuristic Test Precision for Dimensionality 2000 and 4 bit Score

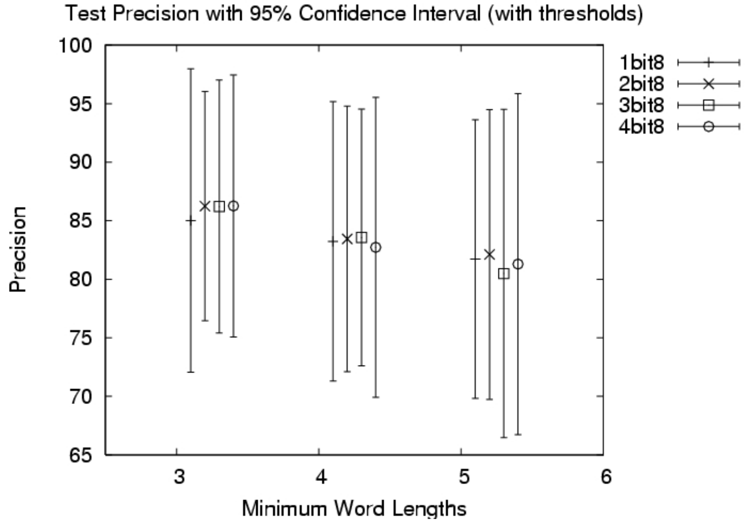

Figure 4.9: Heuristic Test Precision for Dimensionality 2000 and 8 bit Score

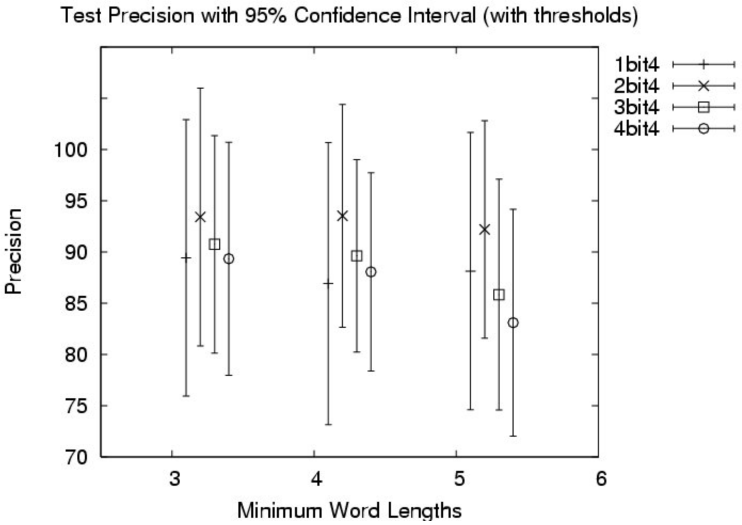

Figure 4.10: MI Test Precision for Dimensionality 2000 and 4 bit Score

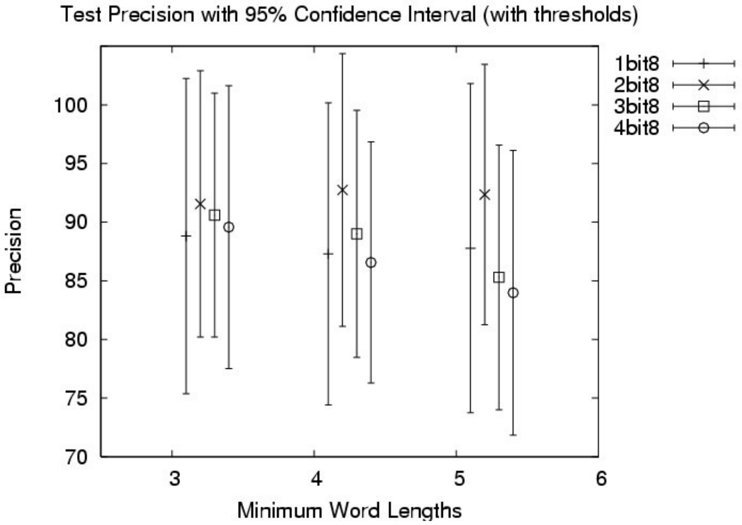

Figure 4.11: MI Test Precision for Dimensionality 2000 and 8 bit Score

Significance of Parameter Adjustment

An experiment was run to find a set of parameters that requires fewer resources while not losing precision. In this experiment with 12 newsgroups, the choice of minimum word length of 4, 2000 features, 2 bit count bit resolution, and 4 bit score bit resolution will not lessen the precision of the system. When using the MI algorithm the average precision of the original configuration is 89.25% with a standard deviation of 29.22. The proposed setting has a precision of 93.53% with 30.80 standard deviation. The proposed setting has a higher average precision with a slightly higher standard deviation. It would appear that the changes did not degrade performance, but rather improve it. However, looking at the results from the Heuristic algorithm, the original setting yielded an average precision of 91.29% with a standard deviation of 28.16. This is in contrast to the proposed setting that had an average precision of 81.80% with a standard deviation of 27.79. For the Heuristic algorithm, the change in settings appears to be detrimental to precision.

To test the significance of the results, a *Paired T test* is used. The results from the two settings provide a distribution of results and the test indicates the amount of difference between the two results. The evaluation takes the form:

$$\text{Paired } T \text{ Test Statistic} = \frac{\bar{D}}{S_d/\sqrt{n}}$$

where $\bar{D}$ is the average of the differences at each data point. S_d is the standard deviation of the differences and n is the number of data points (30). The results from the Heuristic algorithm are compared for the original and proposed settings. The $\bar{D}$ value is 9.50 and the S_d value is 6.11 which results in the value of 8.51. The P value for $t_{0.0005}$ is 3.646 which shows that there is a statistical difference when changing the settings of the AFE. Applying the same T test to the MI results show $\bar{D}$ is -4.28 and S_d is 5.82. This results in a value of -4.03, which indicates a significant difference in the results of the system. For the MI algorithm, the performance increases and for Heuristic it decreases. Doing the T test between the results of the two algorithms shows that there is no difference between the two algorithms with the original settings of the system. The T test at the original settings has a value of 1.90 which does not beat the criteria of 3.646. In comparing the performance of both algorithms with the proposed settings, the T test has a value of 10.84. It is clear that the MI algorithm is more effective at the new settings.

4.7 Conclusion

The conclusions from performing sensitivity analysis for the experiment with 12 newsgroups indicates that the MI algorithm can slightly improve precision by using a minimum word length of 4, 2000 features, 2 bit count resolution, and 4 bit score resolution. With the new setting, the performance of the base word circuit increases because it can process 1.25 words per clock cycle rather than 1 word per clock cycle. The space required in hardware to store a document vector has been cut by 75%. The original settings required 4000 by 4-bits which is 2000 bytes. The proposed settings require 2000 by 2-bits which is 500 bytes. This results in the saving of substantial space in the Count circuit. Because of the shortened data size, the communication between Count and Score will take 75% less data to transfer a document.

Chapter 5

5.1 Conclusion

Through this thesis, a framework was developed to determine the optimal parameters for representation of flows for classification. The representation of a flow affects the area in the FPGA and computational processing time. Finding optimal parameters for the AFE system means that precision in classification is maintained while maximizing the high performance.

The experiment performed within this thesis shows that for 12 newsgroups, the amount of area needed to represent flows can be reduced by 75% without loss of precision. The ideas set forth for the parameter experiment can be utilized to obtain settings for many types of data.

If the system maintains the same area for text representation, it is possible to use the remaining area other ways. Since the AFE system makes use of a pipeline for processing, other types of processing circuits could be inserted into the pipeline. Image annotation, for example, can make use of the remaining area. The extra information could be utilized to store metadata and other information about a flow and could assist in the classification.

5.2 Future Work

The AFE system is highly configurable. Many other algorithms could be used to build word mapping table and a score table. Neural Networks and Support Vector Machine techniques, for example, could be utilized during the training process. The effect reducing the amount of training data could be explored to determine how well the algorithms can learn with less labeled input data. Previous work has shown that the heuristic learning algorithm described in [13] will out perform the MI learning algorithm described in [13] as the amount of training data decreases. More exhaustive study is needed to determine what amount of training data is adequate for the algorithms used in the system.

5.2.1 Sensitivity of Many Concepts

Another sensitivity study could be performed to determine how well the AFE system performs as the number of topics increases. The AFE hardware could be upgraded to use more FPGA resources to handle more concepts. Some configurations may fit within the available space of the FPGA devices while other configurations would benefit from larger devices, like the Xilinx Vertex 4. Analysis of results may depend on the type of data targeted. If language ID is the focus, the number of features needed is less than for identification of concepts.

5.2.2 Contextual Relations for Words

The current AFE system extracts features as a Bag of Words (BOW). Perhaps there is a better way of interpreting the text. Rather than basing the concept of a document only on the words that appear, contextual relations of sentences could be utilized. This type of analysis would require parsing of the text to find the start and end of sentences. One example would be to track of words and their X closest neighbors because there is likely to be a relationship between these nearby words. Consider a sentence like “I am crossing the river on the bridge.” It is possible to process all the relations produced by the word “crossing” in relation to concept C_i in $P(C_i \mid crossing, I)$, $P(C_i \mid crossing, am)$, $P(C_i \mid crossing, the)$, and $P(C_i \mid crossing, river)$, but this method requires knowledge of the probabilities of words in relation to other words. The amount of data that must be stored can be reduced by elimination of stop words such as “the”, “and”, and “for”. Stemming could be utilized so that a plural word like “Trucks” could be shortened to “Truck”. These methods require substantial prior knowledge about the language that appear in the document.

Limiting the set of words does not eliminate the large number of relations that would need to be stored. An average America adult has a lexicon of approximately 5000 words. With stemming and elimination of stop words, the lexicon may be reduced to 2000 important words. If only the relations between pairs of important words is maintained, the amount of storage required to represent all pairs would be about 4,000,000 items (2000 × 2000).

The number of items increases when pronouns are incorporated into the relationships. The use of very specific word pairs or phrases could help to limit the number of terms needed for classification. Phrases like “london bridge” or “car bomb” are important to identify a specific topic whereas other phrases like “jump over” are not significant. The system could be enhanced to take into account significant word pairs and word phrases. If the sequence is determined to be important or is specifically a target, the AFE system could be configured to parse for key phrases.

Appendix A

File Formats for Simulation, Learning, and Verification

Simulation of the AFE functioning system is accomplished by a series of programs written in ANSI C [22]. The implementation allows for testing and verification of the working AFE system. In order to verify that the circuit is working correctly, the input and the output of the circuits can be compared to the input and output of hardware. The format chosen for structuring the data is XML [3]. Beyond the testing and verification of hardware, XML is used to frame data in other portions of the working system. These range from configuring experiments to showing results of new concept discovery.

A.1 Hardware Configuration File Formats

Data files delivered to the hardware are for configuration of the two reconfigurable circuits. The Base Word circuit uses a file called the Word Mapping Table (WMT), and the Score circuit uses a file called the Score Table (ST).

A.1.1 Word Mapping Table

The Word Mapping Table, (WMT) represents the mapping of byte strings hashed to 20-bit values to array positions for a 4000 dimension document vector. The WMT has a legacy name of *Squeeze Table*. The legacy name is present in the tags of the *WMT* XML file. Frequently, the file created for an experiment is called *squeezetable.xml*. Table A.1 shows the format of the *WMT*. Each tag in the *WMT* XML serves a purpose. Table A.2 describes each of the elements of the *WMT* XML file.

Table A.1: XML format for Word Mapping Table

```
<SqueezeFile>
   <MetaData File_Type=”Squeeze_Table” Version=”1.1”/>
   <SqueezeTable>
      <bucket num=”00000”> f3c </bucket>
      <bucket num=”00001”> f3d </bucket>
      ...
      <bucket num=”0003c”> 83b </bucket> <!–1 discontinued –>
      <bucket num=”424f0”> 16c </bucket> <!–3 pro invaded incident –>
      ...
      <bucket num=”fffff”> f87 </bucket>
   </SqueezeTable>
</SqueezeFile>
```

Table A.2: *WMT* Tag Description

SqueezeFile - Designating the file as a *WMT*

MetaData - XML header information

File_Type - The type of XML file.

Version - Version of *WMT*

SqueezeTable - Section for *WMT* data.

bucket - Shows the mapping of hash value to array position (lowercase hex)

num - hash output value (20-bits in lowercase hex)

A.1.2 Score Table

The Score Table, *ST*, is the second file used for both hardware and software configuration. The file represents the number of concept vectors and concept vector formats for the score circuit. The legacy name of the *ST* is a Load Score Document. Table A.3 shows the format of the *ST* document. Each tag in the *ST* XML file is described in Table A.4. For the programs that are written for the AFE system, the important aspects become the **numbits** tag and the number of count arrays specified.

Table A.3: XML format for *ST*

```
<loadscore>
    <MetaData File_Type=”loadscore” numbits=”8” version=”1.1” />
    <countarray num=”0” norm=”312.381177” bias=”0” threshold=”0.039150”>
01 30 12 34 50 ab 43 29 fa .....
a1 32 .....
    </countarray>
    <countarray num=”1”>
01 30 12 34 50 ab 43 29 fa .....
    </countarray>
        .....
    <countarray num=”14”>
01 30 12 34 50 ab 43 29 fa .....
    </countarray>
</loadscore>
```

The **numbits** tag designates if the hardware is to be configured with 15 count arrays that have 8-bit precision for the 4000 elements of the arrays, or 30 count arrays with 4-bit precision. The **countarray** data is represented with 4000 elements in lowercase hexadecimal format where each value is separated by a space or new line. The human readable format of the document will have a limited number of values on a single line. It is necessary to designate all the **countarray** data elements for hardware. The

Table A.4: *ST* Tag Description

loadscore - Opening tag for the *ST* document

MetaData - XML header information

File_Type - The type of XML file.

numbits - Score circuit precision value (8 or 4 are valid)

version - Version of *ST*

countarray - Count array data

num - The count array number

norm - normalization factor

bias - biasing value

threshold - threshold for acceptance

hardware must have all its elements loaded in order to work properly. Software will fill in zero values where **countarray** is not specified.

In Table A.3, the **norm**, **bias**, and **threshold** attributes are stated in the count array *num=0*. These values are not used in hardware but are needed in the post processing evaluation of the scoring of flows.

A.2 Simulated Hardware Output File Formats

AFE has three circuits: Base Word, Count, and Score. Each circuit needs simulated output representation for hardware verification and testing. Each of the circuits is designated in separate XML file formats.

A.2.1 Baseword Circuit Output Representation

The output of the base word circuit contains the flow information of the flow being analyzed and a list of values derived from the hashing of valid byte strings from payload of the flow. These values are the output from the transformation done through the *WMT*. The XML format for the simulated base word circuit output is in Table A.5.

Table A.5: XML format for Base Word Circuit Output

```
<bwout>
    <MetaData File_Type=”BaseWord_Output” version=”1.1” />
    <header new_ff=”1”
        end_ff=”0”
        num_bw=”8”
        flowId=”05592”
        sIp=”128.252.2.5”
        dIp=”55.13.94.224”
        sPort=”128”
        dPort=”55” />
    <basewordList>
a50
538
...
93e
0cd
    </basewordList>
</bwout>
```

Each of the values in the file are described in Table A.6. In simulation, the full flow is reported in an XML file and the list of base words from the flow is listed in the **basewordList** data.

The items in the list can be the hash 20-bit values for each valid byte string the hardware detected. The file format of Table A.5 serves the request with the replacement of the data in the **basewordList** region. The values go from the values derived from a *WMT* to the raw hash values.

Table A.6: Base Word Circuit Output Tag Description

bwout - Designates the file as a base word circuit output file

MetaData - XML header information

File_Type - The type of XML file.

version - Version of the file

header - Header information for the flow is described here

new_ff - New flow designator

end_ff - End of flow designator

num_bw - Number of base words in this message

flowID - Hardware flow identification number

sIp - Source IP address of the flow

dIp - Destination IP address of the flow

sPort - Source port

dPort - Destination port

basewordList - List of base words from the circuit

A.2.2 Count Circuit Output and Representation

The output of the count circuit contains the complete count array for a flow. The output is in Table A.7 and the description of the fields for the XML file is in Table A.8. The values of the count array are in lowercase hexadecimal format. The flow information contained in the **header** field is the same information contained in the Base Word Circuit.

Table A.7: XML format for Count Circuit Output

```
<cout>
    <MetaData File_Type=”Count_Output” version=”1.1”/>
    <header flowId=”05592”
        sIp=”128.252.2.5”
        dIp=”55.13.94.224”
        sPort=”128”
        dPort=”55”
        num_bw=”119” />
    <countarray>
0 1 3 0 1 2 3 4 5 0 a b 4 3 2 9 f .....
9 3 e b 0 .....
    </countarray>
</cout>
```

A.2.3 Score Circuit Output and Representation

The output of the score circuit from a simulation is detailed in Tables A.9 and A.10. The count array for the flow represented by the file is the same count array that appears in the count output file for the same flow. The AFE system has the capability of sending the count array out with the scoring of the flow. For that case, the count arrays are included in the simulated outputs.

Table A.8: Count Circuit Output Tag Description

cout - Designates the file as a count circuit output file

MetaData - XML header information

File_Type - The type of XML file.

version - Version of the file

header - Header information for the flow is described here

new_ff - New flow designator

end_ff - End of flow designator

num_bw - Number of base words in this message

flowID - Hardware flow identification number

sIp - Source IP address of the flow

dIp - Destination IP address of the flow

sPort - Source port

dPort - Destination port

countarray - Data that designates the count array

A.3 Software Learning and Reporting File Formats

Beyond the workings of the AFE hardware, other types of files are used to designate configurations for the learning algorithms that produce the *WMT* and *ST*. These extra files also aid in identifying the performance of the system.

A.3.1 Config.xml

The *config.xml* file defines the configuration for an experiment. The file is best understood by breaking it down into the important sections. First is the header section.

Table A.9: XML format for Base Word Circuit Output

```
<sout>
    <MetaData File_Type=”Score_Output” version=”1.1” />
    <header flowId=”05592”
        sIp=”128.252.2.5”
        dIp=”55.13.94.224”
        sPort=”128”
        dPort=”55”
        num_bw=”119” />
    <scores s0=”10430”
        s1=”40045”
        s2=”18181”
        ...
        s14=”48181”
        ss=”268455”
        sum=”9383” />
    <countarray>
0 1 3 0 1 2 3 4 5 0 a b 4 3 2 9 f .....
a 2 0 .....
    </countarray>
</sout>
```

Table A.10: Count Circuit Output Tag Description

cout - Designates the file as a count circuit output file

MetaData - XML header information

File_Type - The type of XML file.

version - Version of the file

header - Header information for the flow is described here

new_ff - New flow designator

end_ff - End of flow designator

num_bw - Number of base words in this message

flowID - Hardware flow identification number

sIp - Source IP address of the flow

dIp - Destination IP address of the flow

sPort - Source port

dPort - Destination port

scores - Score circuit output values section

s0 .. s14 or s29 - Scores of the count array against the concept vector

ss - Sum of the squares of elements in the count array

sum - Sum of all elements in the count array

countarray - Data that designates the count array

Header

The header section shown in Tables A.11 and A.12 are standard XML format.

Table A.11: Config.xml Header Format

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
<ConfigFile>
<MetaData File_Type=”Config” version=”2.0” />
```

Table A.12: Config.xml Header Description

header - basic XML versing and encoding information

ConfigFile - name of the file signifying a *config.xml*

MetaData - more XML information

File_Type - type of file this is

version - which version of *config.xml*

Runtime Configuration

The information in the **Run** section contains the data for identifying the location of files for this experiment. The Tables A.13 and A.14 show the format. Within this section, the values of the **ScoreTablePrecision** are values from 1 to 8. Any value in the range is acceptable for a software run, however, the hardware will only accept values of 8 or 4. Also, **CountTablePrecision** has a range of 1 to 4, however, the hardware is fixed at values of 4.

Classification Algorithm

The *ClassAlgorithm* section describes the algorithm used in classification of the data. Tables A.15 and A.16 describes the section. The data in the section is dependent on the algorithm used to create the *WMT* and *ST*. These values are important in the preprocessing and post processing procedures.

Table A.13: Config.xml Run section format

```
<Run>
    <Name>May 2005 FI - Test 1 (fast cocl)</Name>
    <CorpusDir>../data/</CorpusDir>
    <BaseDir>base</BaseDir>
    <CountDir>count</CountDir>
    <ScoreDir>score</ScoreDir>
    <ClusterDir>clusters</ClusterDir>
    <ResultDir>results</ResultDir>
    <ScoreTable>support/scoretable.xml</ScoreTable>
    <SqueezeTable>support/squeezetable.xml</SqueezeTable>
    <ScoreTablePrecision>8</ScoreTablePrecision>
    <CountTablePrecision>4</CountTablePrecision>
</Run>
```

Clustering Algorithm

Clustering information is described Tables A.17 and A.18. This section defines how the post process is to proceed. The algorithm and its definitions are to be designated within these tags.

Word Mapping Table Algorithm

The **WordMappingTable** section of the *config.xml* is described in Tables A.19 and A.20. This section is important for the designation of what algoritm is to be used in making the *WMT*. Within the **Params** tag, vendor specific information is designated. Various tags might appear. The **ClassFile** tag that is shown is specific to the MI/HNC/FI COCL algrithm implementation.

Table A.14: Config.xml Run Section Description

Run - Opening tag for the section

Name - This is the name of the experiment.

CorpusDir - The directory of the files for training and validating.

BaseDir - The directory for the output of simulated base circuit.

CountDir - The directory for the output of simulated count circuit.

ScoreDir - The directory for the output of simulated score circuit.

ClusterDir - The directory for clustering output files.

ResultDir - The directory for the postprocessing information.

ScoreTable - Location of the *Score Table* file.

SqueezeTable - Location of the *WMT*.

ScoreTablePrecision - *ST* precision value

CountTablePrecision - Count array precision value

Table A.15: Config.xml ClassAlgorithm section

```
<ClassAlgorithm>
    <Vendor>HNC</Vendor>
    <Alg>Cosine</Alg>
    <Rejection>RATIO</Rejection>
</ClassAlgorithm>
```

Concept Training

The **Concepts** section of the *config.xml* is described in A.21 and A.22. The values designated here specify what concepts will be trained into the *WMT* and *ST*. The concept vectors developed for the classification system are designated within this section.

Table A.16: Config.xml ClassAlgorithm Section Description

Vendor - The vendor's software used for the run. Acceptable values are *HNC* and *WashU*.

Alg - The algorithm used for classification. Acceptable values are *Cosine* and *DotProduct*.

Rejection - The rejection method used in classification. Acceptable values are *RATIO* and *DELTA*.

Confusion Matrix for Post Processing

The *Confus* section of the *config.xml* is shown in A.23 and A.24. This is to designate the structure and labeling of the confusion matrix in the post processing of data.

File Designation

The final three sections of the *config.xml* are in Tables A.25 and A.26. These sections designate the files to be used for training, validation, and testing. The values in the **language** and **encoding** tags are dependent on the files designated. The idea of the three sections is to use a corpus for training and testing the system. Files for training are designated in the **TrainFiles** and **ValidateFiles** sections. The validation files are available for evaluating how the algorithms were trained with all the files available in the training section. The **TestFiles** are for testing the performance of the overall

Table A.17: Config.xml ClusterAlgorithm section

```
<ClusterAlgorithm>
  <Vendor>WashU</Vendor>
  <Alg>K-Means</Alg>
  <Params></Params>
</ClusterAlgorithm>
```

Table A.18: Config.xml ClusterAlgorithm Section Description

ClusterAlgorithm - The tag for the section.

Vendor - The vendor who wrote the code used in clustering.

Alg - The algorithm used for the clustering.

Params - Specific parameter information for the clustering.

system. It is possible to have concepts designated in the **label** that are not specified in the **Concepts** section of the *config.xml*.

Table A.19: Config.xml WordMappingTable Section

```
<WordMappingTable>
<Vendor>HNC</Vendor>
<Alg>COCL</Alg>
<Params><ClassFile>classes.txt</ClassFile></Params>
</WordMappingTable>
```

Table A.20: Config.xml WordMappingTable Section Description

Vendor - The vendor who provided the code. Acceptable values are *HNC* and *WashU*.

Alg - The algorithm used. Acceptable values are *COCL* and *Order*.

Params - This is specific information for the software. As in the above example, the class file for COCL is described.

Table A.21: Config.xml Concepts Section

```
<Concepts>
    <concept num=”0”><label>arabic-marriage</label></concept>
    <concept num=”1”><label>english-marriage</label></concept>
    <concept num=”2”><label>spanish-marriage</label></concept>
    <concept num=”3”><label>turkish-marriage</label></concept>
    ...
</Concepts>
```

Table A.22: Config.xml Concepts Section Description

Concepts - the tag for the section

concept - designation of a new concept

num - the concept vector number to be created

label - the name of the concept

A.3.2 Base Word Table

It is often beneficial to track the words that formulate the mappings in the *WMT*. This formulation is specified in an XML file called the Base Word Table (*BWT*). The format and description of the *BWT* is shown in A.27 and A.28. The data shows the mapping of words to the count array positions of the *ST* and the data flows processed by the AFE system. It is possible to have a very large number of valid

Table A.23: Config.xml Confus Section

```
<Confus>
    <Columns>
    <column><label>astrology</label></column>
    <column><label>cars</label></column>
    ...
    </Columns>
    <Rows>
    <row><label>astrology</label></row>
    <row><label>cars</label></row>
    ...
    </Rows>
</Confus>
```

Table A.24: Config.xml Confus Section Description

Confus - tag for the overall Confus section

Columns - tag for the Columns section of Confus

column - designates a column item

label - the label of the column or row

Rows - tag for the Rows section

row - designates a row item

strings in each **word** designation. This file is useful for learning, conceptualization, and visualization. The values in the **word** values can become complex when multiple character set encodings are designated for training purposes. The **encoding** field may not satisfy all the character sets used in the creation of the mappings.

Table A.25: Config.xml Test, Train, and Validate Section

```
<TrainFiles>
   <file><name>filename1</name><label>conceptX</label>
      <language>english</language><encoding>utf-16BE</encoding>
      <type>Signal</type></file>
   <file><name>filename2</name><label>conceptY</label>
      <language>english</language><encoding>utf-16BE</encoding>
      <type>Signal</type></file>
   ...
</TrainFiles>

<ValidateFiles>
   <file><name>filename10</name><label>conceptX</label>
      <language>english</language><encoding>utf-16BE</encoding>
      <type>Signal</type></file>
   <file><name>filename11</name><label>conceptY</label>
      <language>english</language><encoding>utf-16BE</encoding>
      <type>Signal</type></file>
   ...
</ValidateFiles>

<TestFiles>
   <file><name>filename20</name><label>conceptX</label>
      <language>english</language><encoding>utf-16BE</encoding>
      <type>Signal</type></file>
   <file><name>filename21</name><label>conceptY</label>
      <language>english</language><encoding>utf-16BE</encoding>
      <type>Signal</type></file>
   ...
</TestFiles>
```

A.3.3 Cluster Output

Concept discovery is done as a post process by clustering the output of the system. The format of the clustering output is shown in Tables A.29 and A.30. The format has room for highly specific information for any algorithm used. It is allowed that specific algorithm information can be put into the file. Usually, before the **cluster** section,

Table A.26: Config.xml Test, Train, and Validate Section Description

TrainFiles - tag for training files section

ValidateFiles - tag for validating files section

TestFiles - tag for test file section

file - For a single file, its attributes are described within.

name - The name of the file.

label - The concept that the file is associated.

language - The language of the file.

encoding - The encoding of the file.

type - The type of file for the experiment. Acceptable values are *Signal* and *Interference*.

information is added to specify run time values for the algorithm used to create the output. For instance, <GlobalStatistics>, <MutualInformation>, <ElapsedTime> are all acceptable. The **parameters** section of the file will have information determined from the program that created the clustering and is specific to the individual cluster. Tags such as <level>, <numChildren>, and <isLeaf> are all acceptable. The **file** designations are associated with a corpora used. The files used in clustering have be processed by the AFE system. Next, the **countarray** of the individual cluster section is represented in hexadecimal values of 8-bit precision. The idea of making the **countarray** available is so that the AFE system can be populated with the output shown in the Cluster Output file. The file is meant for a general description of clustering and attempts to handle both hierarchical and flat partitioning. The **subcluster** section is used to start a new **cluster** section.

Table A.27: Base Word Table XML format

```
<?xml version=”1.0” encoding=”utf-8”?>
<BaseWordFile>
    <MetaData File_Type=”BaseWord_Table” Version=”1.2” />
    <BasewordTable>
        <bucket>
            <num>000</num>
            <wordlist>
                <word>’55</word>
                <word>22RUBIA</word>
                <word>236</word>
                ...
                <word>the</word>
            </wordlist>
        </bucket>
        ...
    </BasewordTable>
</BaseWordFile>
```

Table A.28: Base Word Table Description

header - basic XML versing and encoding information

BaseWordFile - Designates the file as a base word table file

MetaData - more XML header information

File_Type - The type of XML file.

version - Version of the file

BaseWordTable - Tag for the data section

bucket - Shows the mapping of the array value to words

num - designates the array value

wordlist - designates the list of words

word - a word in the bucket

Table A.29: Cluster XML format

```
<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>
<ClusterOutput>
    <MetaData FileType=”Cluster_Output” Version=”1.1” />
    <cluster>
        <filenames>
            <file>talk.origins-113327_015</file>
            <file>talk.origins-138768_015</file>
        </filenames>
        <parameters>
            <type>fixed</type>
            <countarray>
                18 17 00 00 00 1c ...
            </countarray>
        </parameters>
        <subcluster>
            <cluster>
            ...
        </subcluster>
    </cluster>
</ClusterOutput>
```

Table A.30: Cluster XML Description

header - basic XML versing and encoding information

ClusterOutput - Designates the file as a cluster file

MetaData - more XML header information

File_Type - The type of XML file.

version - Version of the file

cluster - a single cluster's data

filenames - section for listing files in the cluster

file - a single file name

parameters - cluster's specific information

countarray - representation of the count array

Appendix B

Software for System Simulation for Learning Algorithms

The functioning of the AFE system is simulated in a series of programs. Each program produces files in the XML formats described in Appendix A. Without the simulation, the process of testing the circuits would be exceedingly more difficult. The Base Word Circuit is simulated in the program *simBaseOutput.c*. The Count Circuit is simulated in the program *simCountOutput.c*. And finally, the Score Circuit is simulated in the program *simScoreOutput.c*. The programs are executed in a sequence that follows the path of the hardware.

B.1 Programming Language Choice

The C Programming Language [22] is the choice for simulating the AFE system. Any language would have worked. However, C provides an inherent ability to get to the actual memory locations of data very easily. In simplest terms, C is the management of memory with some calculations. It is very fast and integral in the linux operating system. These features of C are very important in the simulation process. The FPGA is a system of manipulating bytes and bits in memory locations and gate arrays. Simulating this system requires the ability to do the hardware data manipulations in software. The bit operators of C, (>>, <<, &, |, ˆ), allow for the same type of bit manipulations that occur in hardware.

B.2 Simulation of the Base Word Circuit (*simBaseOutput*)

The first program in the sequence of simulation is the program to simulate the Base Word Circuit and it is called *simBaseOutput*. The program can work in a number of ways. First, it can read a directory full of files and produce the appropriate outputs. It can also be called to process a single text file. The program, to work like hardware, requires a WMT. However, the program can be called to output the raw 20-bit values instead of the values of the WMT. All instances of input will produce output in the XML Base Word output format.

B.2.1 Word Length and Word Parsing Engines

The Base Word Circuit of the AFE system identifies words by identifying acceptable byte sequences. Essentially, a word is a series of bytes that are identified to be acceptable in a given sequence. This is a very important distinction. Acceptance of byte sequences is slightly different than tokenization. The tokenization of words is acheived by knowing the tokens and segmenting sequences of bytes at those known tokens. Building words from acceptable byte sequences is done by allowing sequences to build as long as the bytes adhere to a specific acceptable value and sequence. In the AFE system, sequences of 3 to 16 characters constitute an acceptable word. Sequences that extend past 16 characters are truncated to 16. For example:

$$\text{antidisestablishmentarianism} \Rightarrow \text{antidisestablish}$$

Depending on the language, a character can be a single byte or two bytes. The AFE system reads different languages. Multiple encodings of character sets and multi byte encodings are supported.

Single Byte Word Parsing Engine

When building words with single byte characters [4, 7], the method requires making sure each byte fits an acceptable criteria. AFE tries to work with encodings used commonly on the Internet by configuring the tokenizing engines in ASCII with some Extended ASCII, and a combination of Windows Code Pages. Like hardware, many engines can be building words concurrently. As seen in the Figure mutual exclusion does not totally exist. Also, in Figure multiple encodings are accommodated by allowing overlap in the acceptable bytes. In practice, these acceptable ranges are configurable within the programming of the hardware and software. The values shown are tailored for a configuration suitable for testing purposes.

Multibyte UTF16 Engine

The multibyte encoding of UTF16 [15] requires at least 2 bytes and can extend to 4 bytes. However, with the focus of AFE, only 2 byte characters are needed. UTF16 2 byte encoding is defined as:

$$\begin{aligned}\text{byte 1} &\equiv \text{code page designator}\\ \text{byte 2} &\equiv \text{character on code page}\end{aligned}$$

Individual multibyte tokenizing engines run on the hardware concurrently with all other tokenizing engines. Each engine is looking for sequences of characters from individual code pages. On the Internet, Big Endian encoding is used. For example, an engine looking for sequences in code page 06, Arabic, will need to see every other byte equal 06. Further, the second byte must adhere to a designated range acceptable for Arabic characters.

B.2.2 Hash Function in Simulation

When sequences are found by tokenizing, software incorporates a hash function to obtain a 20-bit number. The software hash is functionally the same as hardware. The functioning of the hash is quite simple. The hash is described in Table B.1

Table B.1: Hash Function Description

1 Initialize 20 bits to zeros
2 Read 4 characters into buffer (assuming 2 bytes per character)
3 For the low byte of each character, take the low 5 bits and
pack them into a 20 bit variable and XOR with the output 20 bits
4 Rotate the output 20 bit value by 1 bit to the right.
Continue until the input has been hashed.

Hashing Example

For instance, it will be easier to understand the hash function with the following example for the word *TUNNELING*.

First the sequence of characters must be prepared for hashing. So, single byte sequences are turned into multibyte sequences. The high byte will be 0x0. Therefor:

‘T’ ‘U’ ‘N’ ‘N’ ‘E’ ‘L’ ‘I’ ‘N’ ‘G’ ⇒ ‘0x0’ ‘T’ ‘0x0’ ‘U’ ...

In hexadecimal: “0054 0055 004E 004E 0045 004C 0049 004E 0047”

Now the sequence is ready to be hashed. Now the output value is initialized to zero:
out ← 0 in bits: 0000 0000 0000 0000 0000

The first 4 characters are considered now: “0054 0055 004E 004E”

The low 5 bits of the characters are taken to build a 20 bit value:
“0054 0055 004E 004E” is in bits:
00000000 01010100 00000000 01010101 00000000 01001110 00000000 01001110

Taking the low 5 bits to form a 20 bit number would result in:
10100 10101 01110 01110

Now we XOR the output value with the value we just derived and get:
out ← 1010 0101 0101 1100 1110

There are more bytes to consider so we do a 1 byte right rotate and now we have:
out ← 0101 0010 1010 1110 0111

Now we take the next 4 characters:
0045 004C 0049 004E is in bits:
00000000 01000101 00000000 01001100 00000000 01001001 00000000 01001110

Taking the low 5 bits:
00101 01100 01001 01110

The new low 5 bit value now gets XORed with the out value:
out ← 0101 0010 1010 1110 0111 $\otimes$ 0010 1011 0001 0010 1110

out is now: 0111 1001 1011 1100 1001

There is still a single character left to process so we need to do a 1 bit right rotate on the out value:
out is now: 1011 1100 1101 1110 0100

Continuing on, we need to build a 20 bit value from the remaining characters. Since there are less then 4 characters left, zeros will be filled in for the absent values:
0047 0000 0000 0000 is in bits:
00000000 01000111 00000000 00000000 00000000 00000000 00000000 00000000

The resulting 20 bit number from the low 5 bits is:
00111 00000 00000 00000

We now XOR this value with the current out value:
out ← 1011 1100 1101 1110 0100 $\otimes$ 0011 1000 0000 0000 0000

out is now: 1000 0100 1101 1110 0100

One last right rotate of out: 0100 0010 0110 1111 0010
We now have the output of the hash: (in hex) 0x426F2

B.2.3 Output of Simulated Base Word Circuit (BWC)

In order to simulate the BWC correctly, the workings of the TCP circuit [25] must be considered. The design of the BWC incorporates the signals made available by the TCP processor. The inner workings of the BWC cannot be separated from the TCP circuit. This fact makes it impossible to fabricate the input to the BWC. All data must be passed into the BWC via the TCP processor. Therefore, when simulating the BWC, aspects of the TCP processor must be considered.

All processing of data in the TCP circuit is done on a per flow basis. TCP communication, such as the retrieval of a web page, is actually considered as two TCP flows and may occur over many packets. A TCP flow has a source IP, destination IP, a source port, and a destination port. These values are used to calculate a 19-bit Hash value called the FlowID. This value points to space for 524288 individual flow representations. Flows are represented as source IP, destination IP, source port, destination port, and sequence number. The 19bit value is used to point to a memory location for the flow information. If there exists no previous data in the memory location, the TCP sets its NEWFLOW_OUT_APPL signal and stores the information. Also, since the NEWFLOW_OUT_APPL signal is set, the BWC processes the payload of the packet. The BWC processes data on a per packet basis with some context tracking. The BWC keeps track of partially completed words by storing in memory the context of the word tokenizing engines.

Next, if the packet being processed from the TCP circuit is part of a flow, the sequence numbers are analyzed. If the sequence number coming in is earlier that what is stored the packet is passed to the BWC for processing. If the sequence number is not correct, but it is increasing, the packet will be analyzed by the BWC. The BWC will reinstate the tokenizing engines information from stored memory and continue processing data.

The BWC processing of data will continue as though the packets were in the correct order.

Hash collisions for TCP flows are possible and do happen. If a flow has been hashed to a memory location, a new flow could "crush" old flow information by hashing to the same memory location. This causes the NEWFLOW_OUT_APPL signal to be set and makes the BWC ignore old information. The BWC will not send any information about the previous flow to the count circuit. Therefore, it is possible to have multiple outputs of the BWC for a flow that may never have an end of flow. The other possibility is for "thrashing" where two flows continually crush each other until one, or both end. Notice that the XML file format accommodates the actions of new flows by use of the "new_ff" attribute and the end of flow signal is accommodated by the "end_ff" attribute.

The BWC passes information on to the Count Circuit via cells. The information contained in the cells is the same information that gets written to a file in the BaseWord XML format A.5. Files are produced in the order that they are processed. The data in the files is a list of base word values. The base word values are determined by use of a WMT.

B.2.4 Usage of *simBaseOutput* Version 1.0

The basic usage of the *simBaseOutput* program requires a WMT (option "-s"), a source directory containing a corpus (option "-r"), and a output directory (option "-o"). A basic usage is:

```
simBaseOutput -s wmt.xml -r /corpus/dir/ -o /output/dir/
```

All the options for the program are shown in Table B.2.

The *simBaseOutput* program has special features for aiding in the testing of hardware. The MI/HNC/FI coclustering algorithm requires that the raw hash values (20-bit) values be made available for each file instead of the output of a WMT. MI/HNC uses the raw values to train its algorithm to produce a WMT. The 20-bit values are put

Table B.2: Usage Options for *simBaseOutput*

```
Usage: simBaseOutput -<flag> <param> -<flag><param> ...
          -r <directory to read from>
          -o <directory to write to>
          -s <squeeze file to use>
          -f <use only this one file>
          -w “will not use a squeeze table”
          -len <length of words: ≥ are ok
          -v “will produce vebose output”
          -a “will use only ascii and utf16-00”
```

in the XML output instead of the values produced from a WMT. The use of the “-w” flag will make the program ignore the need for a WMT.

Another special feature is the use of the “-f” flag. This condition will process only a single file. A WMT can be used with the option or the raw hash output can be achieved with the “-w” flag.

Since the programs written for the project are for testing purposes as well as verification, the minimum size of words can be adjusted. The “-len” flag will accept values in the range of 2 to 15. The default of the *simBaseOutput* program, and of the hardware, is to process sequences in the range of 3 to 16 characters. Chapter 4 in this thesis is the manipulation of the minimum word length parameter of the system.

B.3 Simulation of the Count Circuit (*simCountOutput*)

The program for simulating the count circuit is called *simCountOutput*. This program represents the processing of the second circuit in the AFE system. The BWC output is the input to the Count Circuit. In the simulation, the output of the BWC is a series of files representing the processing of individual packets. XML files from the output of *simBaseOutput* are processed into XML count output files. The list of base words in the base output XML files are counted into the 4000 dimension document array. The options for the *simCountOutput* program are shown in Table B.4.

Table B.3: Usage of *simCountOutput*

```
Usage: simCountOutput -<flag><param> -<flag><param>...
            -r <directory to read from>
            -o <directory to write to>
            -f <use only this one file>
            -sat <value to saturate counts: 15, 7, 3, or 1
            -v "will produce vebose output"
```

The basic usage of the program requires a source directory (option "-r") and an output directory (option "-o"). For example:

```
simCountOutput -r /baseoutput/xml/dir -o /output/dir
```

A single file can be processed with the "-f" option like above. The *simCountOutput* program has the ability to alter the precision of values in the count arrays. The "-sat" option can change the number of bits available for each bucket. This is important in evaluating the performance of the hardware and software by limiting the precision of representation of buckets.

B.4 Simulation of Score Circuit (*simScoreOutput*)

The final stage of the hardware pipeline is the Score Circuit. The simulation program for the Score Circuit is *simScoreOutput*. The program requires a *scoretable.xml* file (option "-s"). Like the other above programs, a directory of count XML files needs to be specified (option "-r") along with a directory for the output (option "-o"). For example:

```
simScoreOutput -s scoretable.xml -r /countoutput/xml/dir -o /output/dir
```

The options for *simScoreOutput* is shown in Table B.4. A single file can be processed instead of a complete directory fill of files by using the “-f” option. However, when a single file is processed, a score table needs to be specified.

Table B.4: Usage of *simScoreOutput*

```
Usage: simScoreOutput -<flag><param>-<flag><param>...
        -r <directory to read from>
        -o <directory to write to>
        -s <score file to use>
        -f <use only this one file>
        -v “will produce vebose output”
```

References

[1] The libpcap project. http://sourceforge.net/projects/libpcap/, 2005.

[2] Ron Bekkerman and James Allan. Using Bigrams in Text Categorization. Technical report, University of Massachusetts, Amherst, Massachusetts, December 2003.

[3] J. Boyer. Canonical xml version 1.0. http://www.faqs.org/rfcs/rfc3076.html, 2001.

[4] Vint Cerf. Ascii format for network interchange. http://www.faqs.org/rfcs/rfc20.html, 1969.

[5] Young H. Cho, James Moscola, and John W. Lockwood. Context-Free Grammar based Token Tagger in Reconfigurable Devices. In *Proceedings of International Conference of Data Engineering (ICDE/SeNS)*, Atlanta, GA, USA, April 2005.

[6] 20 newsgroups. http://people.csail.mit.edu/jrennie/20Newsgroups/, 2005.

[7] Microsoft Corporation. Code pages supported by windows. http://www.microsoft.com/globaldev/reference/wincp.mspx, 2006.

[8] J. Heinanen D. Grossman. Multiprotocol encapsulation over atm adaptation layer 5. http://www.faqs.org/rfcs/rfc2684.html, 1999.

[9] Marc Damashek. Method of Retrieving Documents that Concern the Same Topic. U.S. Patent 5,418,951, 1994.

[10] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[11] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-Theoretic Co-clustering. In *Proceedings of SIGKDD '03*, Washington, DC, USA, August 2003.

[12] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, March 1973.

[13] Stephen G. Eick, John W. Lockwood, James Moscola, Chip Kastner, Andrew Levine, Mike Attig, Ron Loui, and Doyle J. Weishar. Transformation Algorithms for Data Streams. In *Proceedings of IEEE Aerospace Conference*, Big Sky, MT, USA, March 2005.

[14] Google Inc. Google groups. http://groups.google.com/, 2005.

[15] Unicode Inc. Unicode. http://www.unicode.org/, 1991-2006.

[16] Chip Kastner, Adam Covington, Andrew Levine, and John Lockwood. HAIL: A Hardware-Accelerated Algorithm for Language Identification. In *Field-Programmable Logic and Applications (FPL)*, Tampere, Finland, August 2005.

[17] John W. Lockwood. Evolvable Internet Hardware Platforms. In *NASA/DoD Workshop on Evolvable Hardware (EHW'01)*, pages 271–279, Long Beach, CA, 2001.

[18] John W. Lockwood, Stephen G. Eick, Justin Mauger, John Byrnes, Ron Loui, Andrew Levine, Doyle J. Weishar, and Alan Ratner. Hardware accelerated algorithms for semantic processing of documentstreams. In *Proceedings of IEEE Aerospace Conference*, Big Sky, MT, USA, March 2006.

[19] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor. Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX). In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, pages 87–93, Monterey, CA, USA, February 2001.

[20] John W. Lockwood, Jon S. Turner, and David E. Taylor. Field Programmable Port Extender (FPX) for Distributed Routing and Queuing. In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, pages 137–144, Monterey, CA, USA, February 2000.

[21] Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing.* The MIT Press, 1999.

[22] D. M. Ritchie, B. W. Kernighan, and M. E. Lesk. The C Programming Language. Comp. Sci. Tech. Rep. No. 31, Bell Laboratories, Murray Hill, New Jersey, October 1975. 31 Superseded by B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, Englewood Cliffs, N.J., 1988.

[23] David Schuehler. Techniques for Processing TCP/IP Flow Content in Network Switches at Gigabit Line Rates. Technical report, Washington University, St. Louis, Missouri, November 2004.

[24] David V. Schuehler and John Lockwood. TCP-Splitter: A TCP/IP Flow Monitor in Reconfigurable Hardware. In *Proceedings of Hot Interconnects 10 (HotI-10)*, Stanford, CA, USA, August 2002.

[25] David V. Schuehler, James Moscola, and John Lockwood. Architecture for a Hardware Based, TCP/IP Content Scanning System. In *Proceeding of Hot Interconnects 11 (HotI-11)*, Stanford, CA, USA, August 2003.

[26] Richard Shaner. Method of Identifying Data Type and Locating in a File. U.S. Patent 5,991,714, 1998.

[27] Claude Shannon. A Mathmatical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[28] J. Brian Sharkey, Doyle Weishar, John Lockwood, and Ron Loui et. al. *Chapter 4: Information Processing at Very High Speed Ingestion Rates*. Wiley, 2006.

[29] K. Simonsen. Character mnemonics and character sets. http://www.ietf.org/rfc/rfc1345.txt?number=1345, 1992.

Vita

Andrew A. Levine

Date of Birth March, 1968

Place of Birth St. Louis, Missouri

Degrees M.S. Computer Science, Washington Univ., December 2006
B.S. Physics, Washington Univ., May 1990

Publications

Sensitivity Analysis of Gigabit Concept Mining System, by Andrew Levine, Ron Loui, John W. Lockwood, Young H. Cho, textitIEEE Aerospace Conference, Big Sky, MT; March 3-10, 2007.

Streaming Hierarchical Clustering for Concept Mining, by Moshe Looks, Andrew Levine, G. Adam Covington, Ronald P. Loui, John W. Lockwood, Young H. Cho, *IEEE Aerospace Conference*, Big Sky, MT; March 3-10, 2007.

High Speed Document Clustering in Reconfigurable Hardware, by G. Adam Covington, Charles L.G. Comstock, Andrew A. Levine, John W. Lockwood, *16th Annual Conference on Field Programmable Logic and Applications (FPL)*, Madrid, Spain; August 28-30, 2006.

Hardware Accelerated Algorithms for Semantic Processing of Document Streams by, Steve Eick, John Lockwood, Ron Loui, Andrew Levine, Justin Mauger, Doyal Weisher, Alan Ratner, John Byrnes, *IEEE Aerospace Congerence*, Big Sky, MT March 2006.

HAIL: A Hardware-Accelerated Algorithm for Language Identification, by Charles M. Kastner, G. Adam Covington, Andrew A. Levine, John W. Lockwood, *15th Annual Conference* *on Field Programmable Logic and Applications (FPL)*, Tampere, Finland; August 24-26, 2005.

Transformation Algorithms for Data Streams, by Stephen G. Eick, John W. Lockwood, James Moscola, Chip Kastner, Andrew Levine, Mike Attig, Ron Loui, Doyle J. Weishar, *IEEE Aerospace Conference*, Big Sky, MT March 5-12, 2005.

December 2006

Short Title: AFE Sensitivity Levine, M.S. 2006