Hardware-Accelerated Parser for Extraction of Metadata in Semantic Network Content

James Moscola, Young H. Cho, John W. Lockwood
Reconfigurable Network Group
Washington University in St. Louis
1 Brookings Drive, Campus Box 1045
St. Louis, MO 63130
{jmm5, young, lockwood}@arl.wustl.edu
http://www.arl.wustl.edu/arl/projects/fpx/reconfig.htm

Abstract—We have implemented a new network information processing system using reconfigurable hardware that scans volumes of data in real-time. One of the key functions of the system is to extract semantic information.Before we can determine the meaning of text, we must identify its language. In a previous project, we have implemented an N-gram based language identifier that can process up to 1 Gbps throughput. However, a large percentage of computer network traffic, such as email and web data, consists of markup information such as tags and protocol specific options. This additional data interferes with the language identification process causing decreased accuracy. Thus, we developed a hardware architecture for configurable application level processing. Our Application Level Processing System (ALPS) is a custom processor that is automatically generated using syntactic structure of the content. The resulting circuit is mapped on to a reconfigurable device to efficiently extract only the relevant data for the language identifier. To illustrate the effectiveness of the architecture, we have implemented a system that can process electronic mail. Our experiments show that ALPS can improve the accuracy of the hardware language identifier by up to a factor of 200 as compared to a system that does not decode the application-level protocol data. [1] [2]

TABLE OF CONTENTS

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.

[1] 1-4244-0525-4/07/$20.00 ©2007 IEEE

[2] IEEEAC paper #1281, Version 1, Updated January 7, 2007

1. INTRODUCTION

In previous Aerospace papers [1][2], we have developed hardware-accelerated semantic processing system for analyzing computer network traffic. This system uses latent semantic indexing techniques to analyze and classify the topic of streaming documents at multi-gigabit per second data rates [3].

In order to assist the system to efficiently process the data, an additional module was developed that identifies language and character encoding used in network data. This module, called HAIL (Hardware-Accelerated Identification of Languages), uses N-grams discovered from training documents to determine the language and encoding of documents that pass through the system. Experimental results with multilingual datasets showed the accuracy of HAIL to be very high. In many cases, the identification accuracy reached above 99 percent [4].

However, most Internet documents, such as XML, HTML, and email contain a large amount of markup and header data. This markup data can have a negative impact on the language detection algorithms used by HAIL. To improve the accuracy of HAIL, and hence the accuracy of the whole document classification system, there is a need to identify and extract document content from the markup data.

In this paper, we describe the implementation and results of a high-speed hardware-based parser implemented with Field Programmable Gate Array (FPGA) technology. The circuits that implement the hardware-accelerated parser are automatically generated using a custom compiler that we have developed. The compiler accepts a Lex/Yacc style grammar that specifies one or more grammars. Once generated, the parser can be configured to forward and/or remove specific fields of the grammar based on user preference. We show how our hardware-based parser can be configured to remove headers and attachments from email messages, leaving only the message content to be processed by the language identification algorithm and document classifier. By removing markup and header data, we can improve the accuracy of language identification algorithms. Our current hardware-accelerated parser can process documents at over 600 Mbps in in a Xilinx VirtexE 2000 FPGA.

2. Background

The hardware platform and protocol processing that this work builds upon has been described in previous papers. This section includes a short description of that work, including the Field-Programmable Port Extender platform and the TCP protocol processor.

Field-Programmable Port Extender

The Field-Programmable Port Extender (FPX) is a general purpose, reprogrammable platform that performs data processing in FPGA hardware [5]. As data packets pass through the device, they can be processed in the hardware by user-defined, reprogrammable modules. Hardware-accelerated data processing enables the FPX to process data at multi-gigabit per second rates, even when performing deep processing of packet payloads.

The FPX contains two FPGAs. A Xilinx Virtex XCV600E FPGA routes packets into and out of the FPX. It also controls the routing of packets to and from the application FPGA. The application FPGA, which executes the user-defined hardware modules, is a Xilinx Virtex XCV2000E (upgraded from an XCV1000E that was used on the first version of the platform). The FPX also contains two banks of 36-bit wide Zero-Bus-Turnaround Static RAM (ZBT SRAM) and two banks of 64-bit PC-100 Synchronous Dynamic RAM (SDRAM). Fully configured, the FPX can access four parallel memories with a combined capacity of 1 Gigabyte.

TCP Protocol Processor

The TCP-Processor is an FPGA based TCP protocol processor [6]. It was designed and implemented to support the processing of up to 8 million simultaneous TCP flows in high-speed networks. The TCP-Processor provides stateful flow tracking and TCP stream reassembly for network applications which process TCP data streams. Additionally, the TCP-Processor includes encoder and decoder circuits which enable it to serialize a TCP flow along with corresponding flow information. This serialized information can then be transported off-chip to multiple other FPGAs for packet processing.

3. Semantic Classification System

As described in [1][2][3], a hardware-accelerated document classification system has been developed that uses latent semantic indexing techniques to classify streaming documents. The system is designed to ingest and classify large volumes of network data in real time. Classification is accomplished through a series of mathematical transformation algorithms as shown in figure 1. Each step of the transformation is implemented in reconfigurable hardware on a series of stackable FPX devices.

As data packets enter the system, they are first processed by the TCP processor. The TCP processor reconstructs packets into consistent TCP flows. Each flow is augmented with control signals that indicate where the IP header, the TCP header and data segment of each packet starts. Additionally, a flow identification number is assigned to each TCP flow so downstream components can manage per-flow context. Each TCP flow is considered to be a single document.

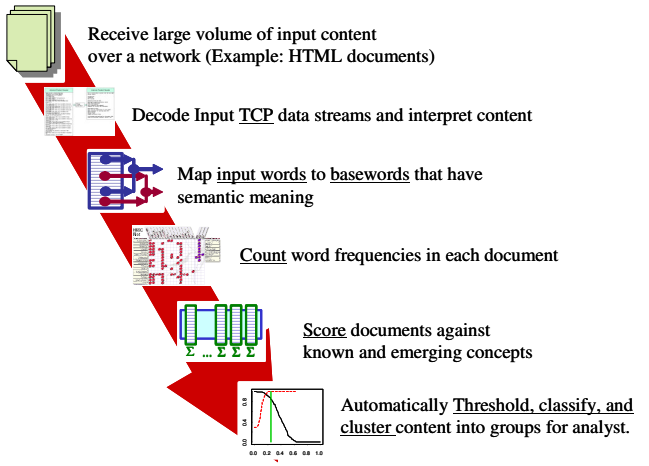

Figure 1. Flow of documents through the document classification system

Reconstructed TCP flows are subsequently processed by a word mapping circuit. The word mapping circuit tokenizes each TCP flow to find words in the flow. For each word that is found, a hash is computed that is used as an address into a word mapping table (WMT). The WMT is a 1 million entry table that maps 1 million input words down to 4000 *basewords*. A *baseword* is a numerical representation of the semantic meaning of the input word. For example, the input words "sleep", "sleeping", "nap", and "siesta" all have similar semantics. As such, the WMT would map all of these words to the same baseword value. For details on the different techniques used to create the WMT, refer to [1].

Once a flow has been tokenized, the list of basewords is sent downstream to a module that maintains a count of the basewords for each active flow. The count is maintained in a 4000-dimension document vector, where each dimension represents one of the possible baseword outputs of the WMT.

At the conclusion of each flow, the 4000-dimension document vector is sent to the scoring module. The scoring module computes a dot product of the document vector against up to 30 previously defined 4000-dimension concept vectors. The flow is subsequently classified according to the highest scoring concept.

4. Language Identification Hardware

In an effort to further enhance the capabilities of the semantic document classification system, the HAIL module was developed. This module integrates into the document classification system and identifies the language and character encoding of each TCP flow prior to classification [4][7].

Identifying the language and character encoding allows the semantic classification system to employ encoding specific tokenizers as well as language specific WMTs. Using an encoding specific tokenizer allows the system to reduce noise by only accepting characters known to be a member of a given character set in a given language. Language specific WMTs allow the system to differentiate between words that occur in multiple different languages but have different meanings in each language. Separating WMTs by languages also has the benefit of breaking one large WMT into many smaller WMTs, thereby reducing the number of entries in each table. Fewer table entries can help to alleviate any hash collisions that may occur during the baseword translation.

HAIL utilizes N-grams to identify both the language and character encoding of a data stream. An N-gram consists of N sequential characters that have been extracted from the data stream. As HAIL processes a data stream, a series of five sequential characters (tetra-grams) are extracted to represent the document. Each unique tetra-gram is associated with a single language/encoding pair through offline training. As tetra-grams are extracted from a document, a counter for the associated language/encoding pair is incremented. At the end of a document, HAIL identifies the language and encoding based on the counter with the highest value.

5. Application Level Processing System

When properly trained, HAIL can accurately identify the language of a streaming document up to 99.95% of the time. However, this result, assumes that HAIL is processing clean documents composed of mostly (if not entirely) text in a single language. This is unlikely to be the case when processing many types of Internet traffic. For example, both HTML and email documents contain headers and tags that are likely to be identified as English whereas the body of the document may actually be Spanish, German, or Arabic. Given a document with enough header information, HAIL may incorrectly identify the language of the document due to the N-grams found in the header.

Consider the sample email shown in figure 2. The email header consists of over 1500 bytes of data, whereas the email body consists of only 65 bytes of data. It is desirable that HAIL identify the language found in the email body and not in the email header. Additionally, it is desirable that only the body of the email message is processed by the document classification system. If the email is processed by both HAIL and the document classification system with the headers intact, it is unlikely that either HAIL or the document classification system will exhibit the desired behavior.

To alleviate the problems described above, we built an additional module, the Application Level Processing System (ALPS), which has been discussed previously in [8][9][10]. ALPS is a hardware-based parsing module that has been implemented on an FPX to serve as a preprocessor for the HAIL module. The ALPS circuits can be automatically generated

Figure 2. Sample email message

via a custom compiler when provided with a grammar in a Lex/Yacc style format. Additionally, ALPS allows specific fields of a grammar to be either forwarded or removed based on a users preference. Referring back to the sample email in figure 2, ALPS can be configured to identify and parse the entire email message and forward only the email body downstream to HAIL and the document classification system.

6. Implementation of Email Parser

Implementing an email parser using ALPS first required defining a grammar for email. We utilize the standard email grammar as defined in RFC 2822 [11], and incorporate grammars defined in RFC 2045 [12] and RFC 2046 [13] to properly parse both MIME header extensions and multipart email messages. A small excerpt representing the date portion from RFC 2822 is shown in Augmented Backus Naur Form (ABNF) in figure 3. Figure 4 shows the same portion of the grammar after it has been converted from ABNF into the format accepted by our hardware generator. The complete grammar for the email parser consists of over 160 tokens and 200 production rules.

```
date-time = [ day-of-week "," ] date FWS time [CFWS]
day-of-week = [FWS] day-name
day-name = "Mon" / "Tue" / "Wed" / "Thu" / "Fri" / "Sat" / "Sun"
date = day month year
year = 4*DIGIT
month = FWS month-name FWS
month-name = "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" /
"Jul" / "Aug" / "Sep" / "Oct" / "Nov" / "Dec"
day = [FWS] 1*2DIGIT
time = time-of-day FWS zone
time-of-day = hour ":" minute [ ":" second ]
minute = 2DIGIT
second = 2DIGIT
hour = 2DIGIT
zone = ( "+" / "-" ) 4DIGIT
```

Figure 3. ABNF for date portion of email grammar

```
date-time:       date-time-opt date FWS time CFWS-opt;
date-time-opt:   day-of-week "," | ;
day-of-week:     FWS-opt day-name;
day-name:        "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun";
date:            day month year;
year:            DIGIT DIGIT DIGIT DIGIT year-opt;
year-opt:        DIGIT year-opt | ;
month:           FWS month-name FWS;
month-name:      "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" |
                 "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec";
day:             FWS-opt DIGIT day-2dig;
day-2dig:        DIGIT | ;
time:            time-of-day FWS zone;
time-of-day:     hour ":" minute time-of-day-opt;
time-of-day-opt: ":" second | ;
minute:          DIGIT DIGIT;
second:          DIGIT DIGIT;
hour:            DIGIT DIGIT;
zone:            zone-a zone-b;
zone-a:          "+" | "-";
zone-b:          DIGIT DIGIT DIGIT DIGIT;
```

Figure 4. BNF for date portion of email grammar

As shown in figure 5, the email parser consists of several main components: the lexical analyzer, the parsing structure, and the filtering module. Both the lexical analyzer and the parsing structure are automatically generated from the grammar tokens and the grammar production rules respectively. The filtering module receives information from the parsing structure and can be configured to either keep or discard data given the state of the parser. The following sections discuss the architecture and generation of the lexical analyzer and the parsing structure.

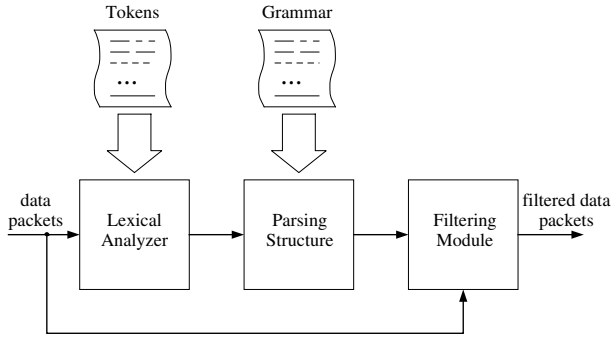

Figure 5. ALPS email parser architecture

Lexical Analyzer

As data enters the email parser it is first processed by the lexical analyzer. The lexical analyzer is a pattern matcher that scans the input data for strings that match tokens in the input grammar. The pattern matcher architecture is an 8-bit pipelined character grid as described by Baker in [14]. A detailed block diagram of the decoded character pipeline is shown in figure 6. The top half of the figure shows the decoded character pipeline, whereas the bottom half shows a sample string detector required to match the `From:` token in the email grammar. The string detectors required for each of the tokens in the grammar are automatically generated by our custom tools.

The pipeline receives one character per clock cycle from the input data stream. Before entering the pipeline registers, characters are passed into an 8-to-256 bit decoder. The 256-bit output represents a single bit line for each of the 256 possible

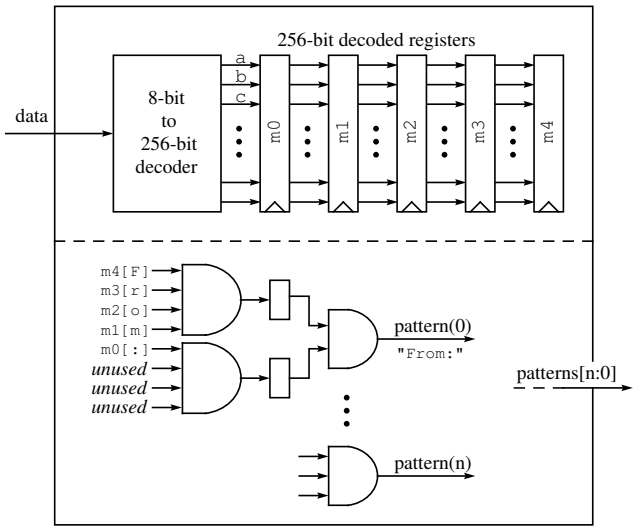

Figure 6. Pattern matcher for lexical analysis

ASCII characters. This decreases the hardware routing resource required for matching the tokens in the grammar. The decoded character lines are passed into the pipeline registers as illustrated in figure 6. The pipeline can detect patterns that are less than or equal to the length of the pipeline. Additionally, the pipeline only needs to be as long as the longest pattern in the grammar

The actual pattern matching is executed by a series of string detectors which are automatically generated. A match is found by ANDing together the appropriate bits from the decoded character pipeline as characters traverse through the pipeline. A single bit line is output from the lexical analyzer to the parsing structure for each of the string detectors. These signals indicate to the parser when a match is found.

Parsing Stucture

The parsing structure gives the email parser the ability to understand input data stream at the application level. It defines the semantics of tokens as they are detected by the lexical analyzer and maintains the contextual state of the data stream. The hardware logic required for the parsing structure is determined from the input grammar. The production list of the grammar defines all of the possible transitions for the grammar. Maintaining the state of the grammar while processing data allows the parser to determine which tokens can occur next.

In the parsing structure, each token is represented using a simple primitive that consists of a single register and a single AND gate. The inputs to each of the AND gates are the output signals from the lexical analyzer. The output of each AND gate represents a transition in the state of the grammar and is routed to the input of other token registers. Transitions are determined from the production list of the grammar using the well known *FIRST* and *FOLLOW* set algorithms [15]. The resulting sets are then used to map the output of token

primitives to the input of each of the token primitives listed in its *FOLLOW* set.

To clearly illustrate how the parsing structure is generated, we use the example grammar defined in figure 7 and generate the corresponding parser hardware.

To generate the parser hardware, we first need to find the *FOLLOW* set for each of the tokens in the grammar. Figure 8 shows a table of the *FOLLOW* sets for all the tokens in the grammar. In addition to the terminal tokens, we need to find all the possible start terminals. Since production E is the starting production, we can use the *FIRST* set of E as the starting symbols. Then, we forward the output of each token primitive to the inputs of the tokens listed in its *FOLLOW* set. When there is more than one connection to the input of a token primitive, an OR gate is used to combine the signals into a single bit input. Figure 9 shows our hardware parser for if-then-else grammar.

No.		Production
1	E →	if C then E else E | go | stop
2	C →	true | false

Figure 7. Production list for if-then-else grammar

Tokens	*FOLLOW* Set
if	$\{true, false\}$
then, else	$\{if, go, stop\}$
go, stop	$\{else, \epsilon\}$
true, false	$\{then\}$

Figure 8. *FOLLOW* set for each of the terminal tokens

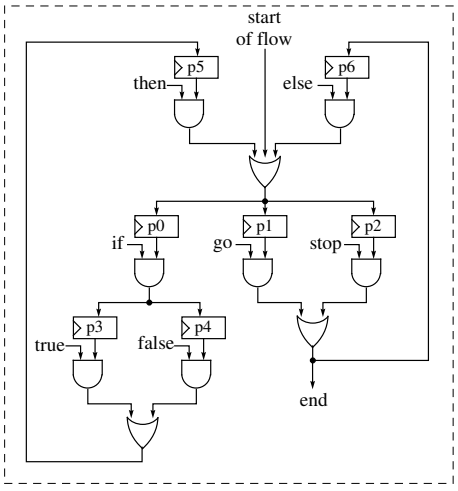

Figure 9. Detailed view of a sample parser core

As the hardware parser processes a data stream the parser receives a signal from the lexical analyzer for each token that is found. These signals allow the parser to traverse the grammar and maintain the contextual state of the data stream. While doing so, the parser also forwards token information along with the state of the parsing structure downstream for further processing.

7. Data Sets & Results

To test the effectiveness of the ALPS email parsing circuit, five different data sets were created. Each data set consisted of 10,816 email messages in 14 different languages. The email headers for each of the emails in all five data sets were similar in both size and content to the email header shown in figure 2. The size of the email body was different for the five different data sets, with the smallest email body being 75 bytes and the largest being 1200 bytes. The source text for the email bodies was the same for all five data sets. Therefore, the first 75 bytes of all 1200 byte emails is identical to the first 75 bytes of the corresponding email in the 75 byte data set (and all other data sets). The table in figure 11 shows the email body sizes for all five different data sets.

Each of the five data sets was first created as 10,816 text files that consisted of the email headers and body. We built a tool to convert these text files into 10,816 TCP flows that could be replayed into the hardware for live testing.

The results for processing the 300 byte emails are shown in figure 10. The *Lang ID* column is a unique number that HAIL uses to represent each language. The *Language* column shows the 14 different languages that were part of our data set. The *TRUTH* column represents the number of documents in the data set that were actually the given language. The *HAIL* column represents the number of documents HAIL reported as the given language when operating alone. The *ALPS+HAIL* column represents the number of documents HAIL reported as the given language when each flow was preprocessed using the ALPS email parser. Note that the total number of documents in the *HAIL* and the *ALPS+HAIL* columns are not the same as the total number of documents in the *TRUTH* column. This is the result of some documents being reported as a language that is not in our data set (i.e. some documents may have been reported with a language ID of 35 which is not shown in the table).

The table in figure 10 indicates that the email headers used in our data set have a strong negative affect on the language identification results for the 300 byte emails. From the results, it appears that the email headers have a significant number of Estonian tetra-grams. When using HAIL alone, 10,280 out of 10,816 documents are identified as Estonian; only one of those documents is actually Estonian. This is because the email headers in the data set are 1500 bytes, whereas the email body is only 300 bytes. This means HAIL extracted five times more tetra-grams from the email headers than it extracted from the email body. This large disparity gave the email headers a greater significance than the email body when counting the number of tetra-grams to identify the language.

Overall, for the 300 byte data set, we can see that HAIL alone only correctly identified the language of 3.98% (431 out of 10,816) of the documents, whereas ALPS+HAIL correctly identified the language of 85.33% (9,229 out of 10,816) of the documents.

Lang ID	Language	TRUTH	HAIL	# Correct	# Incorrect	ALPS+HAIL	# Correct	# Incorrect
1	albanian	1	0	0	0	6	1	5
2	arabic_trans	1447	390	390	0	1448	1447	1
4	czech	29	1	1	0	77	21	56
5	english	2691	0	0	0	2535	2321	214
6	estonian	1	10280	1	10279	28	1	27
7	french	916	0	0	0	875	835	40
8	german	700	0	0	0	604	585	19
13	italian	789	0	0	0	960	739	221
20	norwegian	43	144	39	105	75	40	35
24	portuguese	1634	0	0	0	1146	1103	43
28	spanish	2514	0	0	0	2346	2130	216
29	swedish	20	0	0	0	1	0	1
32	turkish	20	0	0	0	4	3	1
34	uzbek	11	0	0	0	3	3	0
	TOTALS	**10816**	**10815**	**431**	**10384**	**10108**	**9229**	**879**
	% Correct			**3.98%**			**85.33%**	

Figure 10. Language identification results for the 300-byte data set

Body Size	# Correct	% Correct HAIL Alone	# Correct	% Correct ALPS+HAIL
75-bytes	24	0.22%	5006	46.28%
150-bytes	38	0.35%	7532	69.64%
300-bytes	431	3.98%	9229	85.33%
600-bytes	4925	45.53%	9699	89.67%
1200-bytes	7290	67.40%	9837	90.95%

Figure 11. Percentage of correctly classified documents when using HAIL alone and when using ALPS+HAIL

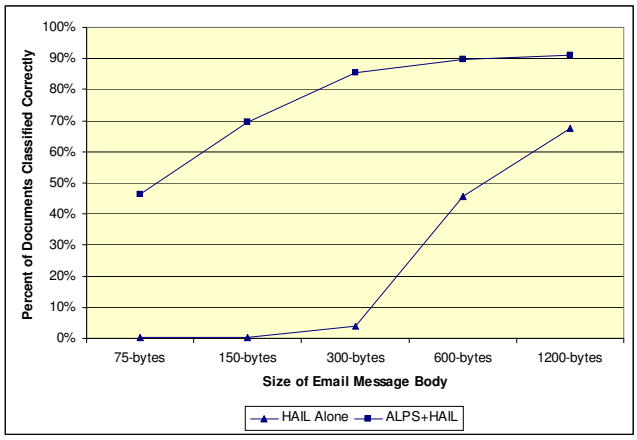

Figure 12. Percentage of documents correctly classified by HAIL for each of the five data sets

Data Set	% Increase
75-bytes	20758.33%
150-bytes	19721.05%
300-bytes	2041.30%
600-bytes	96.93%
1200-bytes	34.94%

Figure 13. The percent increase in accuracy when using ALPS+HAIL as opposed to HAIL alone

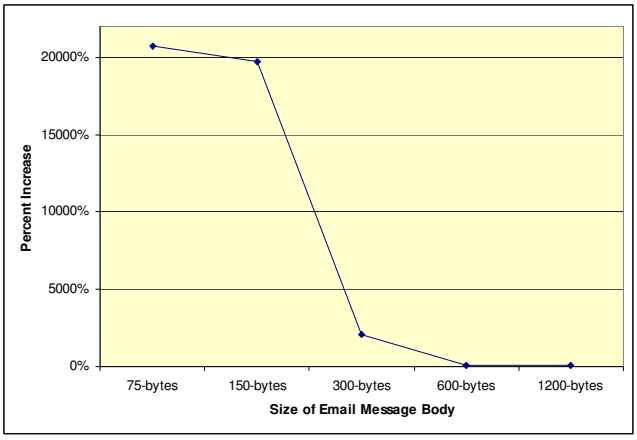

Figure 14. Graph showing the percent increase in accuracy when using ALPS+HAIL

The percentage of correctly identified documents for all five of the data sets is shown in figure 11. The same data is represented graphically in figure 12. The data for HAIL alone has the expected behavior. When the size of the email header is significantly larger than the email body, the output of HAIL alone is skewed towards the language identified in the email header. However, as the size of the email body increases and becomes a larger percentage of the complete email message, the results of HAIL alone become more accurate. As illustrated by the trend in figure 12, if the email body is large enough in comparison to the email header, it will counterbalance the affects of the email header.

The data for ALPS+HAIL in figure 12 illustrates that the accuracy of HAIL increases as the size of a clean document increases. HAIL is more likely to correctly identify the language of a larger documents due to the larger number of available tetra-grams.

Figures 13 and 14 show the percent increase in accuracy when using ALPS+HAIL as opposed to HAIL alone. For large emails where the size of the email body is close to the size of the email headers using ALPS provides a 34.94% improvement on our data set. For smaller emails, using ALPS as a preprocessor to HAIL provides over 200 times improvement.

8. CONCLUSIONS

In this paper we presented our Application Level Processing System and showed how it can be used to preprocess traffic for hardware-accelerated identification of languages and document classification. The custom ALPS processor is automatically generated using the syntactic structure of the content to be processed. In the functional system prototyped on the FPX platform, ALPS extracts data from data streams at over 600 Mbps.

To illustrate the utility of ALPS, we implemented a system that parses email messages and extracts the body of the message for language identification. Our experiments show that ALPS can dramatically improve the accuracy of HAIL. For large (1200 bytes) emails, ALPS increases the accuracy of HAIL 34.95%. For small emails (75 bytes), ALPS increased the accuracy of HAIL up to 200 times.

REFERENCES

[1] S. G. Eick, J. W. Lockwood, J. Moscola, C. Kastner, A. Levine, M. Attig, R. Loui, and D. J. Weishar, "Transformation Algorithms for Data Streams," in *Proceedings of IEEE Aerospace Conference*, (Big Sky, MT, USA), Mar. 2005.

[2] J. W. Lockwood, S. G. Eick, J. Mauger, J. Byrnes, R. Loui, A. Levine, D. J. Weishar, and A. Ratner, "Hardware accelerated algorithms for semantic processing of document streams," in *IEEE Aerospace Conference (Aero'06)*, (Big Sky, MT), p. 10.0802, Mar. 2006.

[3] J. B. Sharkey, D. Weishar, J. W. Lockwood, R. Loui, R. Rohwer, J. Byrnes, K. Pattipati, D. Cousins, M. Nicolletti, and S. Eick, "Information processing at very high-speed data ingestion rates," in *Emergent Information Technologies and Enabling Policies for Counter Terrorism* (R. Popp and J. Yin, eds.), pp. 75–104, IEEE Press/Wiley, 2006. ISBN: 0-471-77615-7.

[4] C. Kastner, A. Covington, A. Levine, and J. Lockwood, "HAIL: A Hardware-Accelerated Algorithm for Language Identification," in *Proceedings of 15th International Conference on Field-Programmable Logic and Applications (FPL)*, (Tampere, Finland), Aug. 2005.

[5] J. W. Lockwood, "An open platform for development of network processing modules in reprogrammable hardware," in *IEC DesignCon'01*, (Santa Clara, CA), pp. WB–19, Jan. 2001.

[6] D. Schuehler and J. Lockwood, "A Modular System for FPGA-based TCP Flow Processing in High-Speed Networks," in *International Conference on Field-Programmable Logic and Applications (FPL)*, (Antwerp, Belgium), pp. 301–310, Aug. 2004.

[7] C. M. Kastner, "HAIL: An Algorithm for the Hardware-Accelerated Identification of Languages," Master's thesis, Washington University, St. Louis, MO, USA, May 2006.

[8] Y. H. Cho, J. Moscola, and J. W. Lockwood, "Context-Free Grammar based Token Tagger in Reconfigurable Devices," in *Proceedings of International Conference of Data Engineering (ICDE/SeNS)*, (Atlanta, GA, USA), Apr. 2005.

[9] J. Moscola, Y. H. Cho, and J. W. Lockwood, "Implementation of Network Application Layer Parser for Multiple TCP/IP Flows in Reconfigurable Devices," in *International Conference on Field Programmable Logic and Applications (FPL)*, (Madrid, Spain), Aug. 2006.

[10] J. Moscola, Y. H. Cho, and J. W. Lockwood, "A Reconfigurable Architecture for Multi-Gigabit Speed Content-Based Routing," in *Proceedings of Hot Interconnects 14 (HotI)*, (Stanford, CA, USA), Aug. 2006.

[11] P. Resnick, "RFC 2822: Internet Message Format," Apr. 2001.

[12] N. Freed and N. Borenstein, "RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," Nov. 1996.

[13] N. Freed and N. Borenstein, "RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types," Nov. 1996.

[14] Z. K. Baker and V. K. Prasanna, "A Methodology for Synthesis of Efficient Intrusion Detection Systems on FPGAs," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, (Napa Valley, CA), IEEE, April 2004.

[15] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles and Techniques and Tools*. Addison-Wesley, 1986.

James Moscola *is currently a graduate student at Washington University in St. Louis doing research for the Reconfigurable Network Group (RNG). His research interests include string and pattern matching, hardware-accelerated parsing, content-based routing, and network intrusion detection and prevention systems. He earned both his BS in Computer Engineering and MS in Computer Science from Washington University in St. Louis. He expects to complete his PhD in Computer Engineering in 2007. He is a member of IEEE.*

Young H. Cho *is a Visiting Assistant Professor at Computer Science and Engineering Department of Washington University in St. Louis. He has earned his BA in Computer Science from UC Berkeley, MSE in Computer Engineering at UT Austin, and PhD in Electrical Engineering at UCLA. He has designed and implemented a number of high performance research and development projects as well as commercial products during his career. His areas of expertise include network security, computer networks, high performance computer architecture, and reconfigurable computers. He is a member of IEEE and ACM.*

John W. Lockwood *designs and implements networking systems in reconfigurable hardware. He leads the Reconfigurable Network Group (RNG) at Washington University in St. Louis. The RNG research group developed the Field programmable Port Extender (FPX) to enable rapid prototype of extensible network modules in Field Programmable Gate Array (FPGA) technology. He is an Associate professor in the Department of Computer Science and Engineering at Washington University in Saint Louis. He has published over 75 full-length papers in journals and major technical conferences that describe technologies for providing extensible network services in wireless LANs and in high-speed networks. Professor Lockwood has served as the principal investigator on grants from the National Science Foundation, Xilinx, Altera, Nortel Networks, Rockwell Collins, and Boeing. He has worked in industry for AT&T Bell Laboratories, IBM, Science Applications International Corporation (SAIC), and the National Center for Supercomputing Applications (NCSA). He served as a co-founder of Global Velocity, a networking startup company focused on high-speed data security, but no longer works for that company. Dr. Lockwood earned his MS, BS, and PhD degrees from the Department of Electrical and Computer Engineering at the University of Illinois. He is a member of IEEE, ACM, Tau Beta Pi, and Eta Kappa Nu.*