Design Contest Overview: Combined Architecture for Network Stream Categorization and Intrusion Detection (CANSCID)

Michael Pellauer*, Abhinav Agarwal*, Asif Khan*, Man Cheuk Ng*,
Muralidaran Vijayaraghavan*, Forrest Brewer†, Joel Emer*‡

*Massachusetts Institute of Technology
Computation Structures Group
Computer Science and AI Lab
Cambridge, MA

†University of California, Santa Barbara
High Level Design Group
Electrical and Computer Engineering
Santa Barbara, CA

‡Intel Corporation
VSSAD Group
Hudson, MA

1. Introduction

This year's MEMOCODE Design Contest challenged teams to implement the architecture for a unique type of Deep Packet Inspector called CANSCID. This type of architectural challenge represents a new direction for the contest, as previous years had focused on the acceleration of algorithmic specifications such as matrix multiplication. Despite such a challenging problem domain the contest received 8 submissions, 6 using FPGAs and 2 using GP-GPUs. This exceeds the total of all FPGA-based submissions from all previous years of the contest combined.

In this paper we describe this year's unique problem statement, and our motivation for choosing it. This paper is followed by short descriptions prepared by individual teams detailing their particular approach to solving the problem.

2. Motivation

During initial meetings, the organizers identified that ideally the contest challenge should:

- Focus on something that traditional software programs have difficulty with.
- Be accessible enough that it can be done in the contest timeframe.
- Demonstrate the utility of hardware accelerators such as FPGAs and GP-GPUs.

Given these criteria we identified Regular Expression (RE) matching as a potential problem domain. In particular, we were aware that recently a large volume of research that been directed at accelerating Deep Packet Inspection (DPI) using FPGAs, which involves testing one or more incoming streams of packets against a body of regular expression patterns.

There were two potential problems that we identified with using DPI for the contest problem. First, some organizers were concerned that the difficulty of implementing a Deep Packet Inspector would be too high, and would constrain teams to use platforms that had direct control over an Ethernet MAC. In order to address this concern we decided to use a simulated Ethernet approach. Our testbench would create "packets" that were subdivided into 32-bit flits and laid out as an array in memory. In order to add a degree of realism we then identified a line rate of 500 Mb/s that teams would have to maintain in order to simulate accepting input from a real Ethernet.

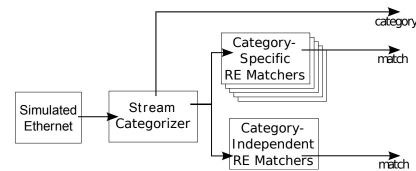

Fig. 1. Overview of CANSCID deep-packet inspector.

Second, some organizers raised the concern that implementing a regular expression matcher on an FPGA might be too close to existing research projects. While we wanted to encourage teams to use concepts from the literature, we wanted to make sure that no existing project could be submitted with minimal adjustments, as this would give teams with access to such a code base an unfair advantage.

In order to solve this we created a novel twist on a Deep Packet Inspection architecture. Commonly, deep-packet inspection has two different main uses: stream categorization (such as L7-filter, `l7-filter.sourceforge.net`), and intrusion detection (such as snort, `www.snort.org`). Our insight was that the contest problem could be to construct a packet inspector which performs both of these tasks. Thus we named the design CANSCID: Combined Architecture for Stream Categorization and Intrusion Detection.

3. CANSCID Overview

A high-level overview of CANSCID is presented in Figure 1. The system consists of 3 main regular expression units, which are described in the following sections. Thus the challenge for the teams is not only to match using an efficient RE engine, but also to coordinate between multiple sets of such matchers.

978-1-4244-7886-6/10/$26.00 ©2010 IEEE

3.1. Stream Categorization

When CANSCID detects a new TCP connection (based on host IP and port, and destination IP and port) it attempts to categorize the stream based on the payloads of the first four packets. This functionality is similar to the software package L7-filter, which also provided the regular expressions for this part of the contest. The regular expressions define patterns which may cross packet boundaries. For instance, the regular expression for detecting an SMTP mail session:

```
220\s* (E?SMTP|[Ss]imple [Mm]ail)
```

With this pattern, the "220" string does not necessarily need to reside in the same packet as the "Simple Mail" string for a match to occur. Therefore the design must support a scheme for storing the state of the regular expression matchers associated with a particular stream, and swapping that state in when a packet of the stream is detected (based on host IP and port, and destination IP and port). For the purposes of the contest we required designs to handle up to 64 simultaneous open connections. This represented a major simplification over a real packet inspector, which would handle thousands of simultaneous connections.

When a stream is successfully categorized, the CANSCID records a histogram of the number of streams seen of each type. Additionally, the category of each stream must also be remembered, so that it can be used for intrusion detection.

3.2. Category-Specific Intrusion Detection

When a stream has been categorized, CANSCID inspects its packets for intrusions or other malicious behavior. This functionality is similar to the software package snort, which also provided the regular expressions for this part of the contest. For instance, here is a snort regular expression to detect an SMTP vrfy decode attempt:

```
vrfy\s+decode
```

Checking non-SMTP streams for this pattern would waste resources and could result in false positives (for instance, if a user loaded a webpage describing the vulnerability). Combining many regular expression parsers can result in large and inefficient automata which can limit a packet sniffer's bandwidth.

Snort avoids this problem by hardcoding which IP addresses and ports represent SMTP servers, and only checking connections to those locations for these patterns. This approach can lead to problems—for instance if new servers are added without the configuration files being updated—or if an attacker manages to convince a computer to open an SMTP server on an unexpected port. Additionally, attack attempts which *originate* in the administered domain can be missed.

CANSCID works around these limitations by using the stream categorizer described above. The categorizer employs regular expressions to identify protocols which are host- and port-independent. Once a stream has been categorized, CANSCID only needs to apply the appropriate category-specific regular expressions against that stream for the remainder of its lifetime.

When a match does occur, CANSCID outputs a message identifying the connection destination and host, category, regular expression matched, and the offending location in the packet stream.

3.3. Category-Independent Intrusion Detection

In addition to the category-specific patterns, CANSCID includes a set of patterns that are run on every stream, regardless of their category. These include patterns which represent things such as *shellcode* — executable code masquerading as ASCII text. Snort disables these patterns by default because of the large performance hit which they can entail, so we were interested if an accelerated implementation could do better.

4. Contest Patterns and Scoring

Category	Number of Patterns	
	Mandatory	Optional
finger	1	4
ftp	1	9
http	1	9
imap	1	9
netbios	1	9
nntp	1	9
pop3	1	9
rlogin	1	4
smtp	1	9
telnet	1	9
all*	5	10
other**	0	25
total	15 + 10 categories = 25	115

* The "all" category refers to category-independent patterns that must be run on all streams.
** The "other" category refers to other protocols which the packet sniffer can identify, but does no further checking on.

TABLE I
OVERVIEW OF PATTERNS OFFICIALLY SUPPORTED BY THE CONTEST

For the contest the organizers identified patterns as shown in Table I. There are 5 mandatory patterns which teams had to run on each stream regardless of category. Each category then requires 1 pattern to identify streams of that protocol, and has 1 associated mandatory pattern which must be run on streams of that type. As there are 10 categories, the minimum requirement for a functioning submission was to implement 25 patterns (10 category-matching patterns, 10 category-specific patterns, and 5 category-independent patterns).

The first goal of this contest for teams to construct a design which can handle these 25 patterns while meeting the target line rate of 500 Mb/s. The organizers determined that if no submission is able to meet the line rate, then the winner will be the team which achieves the fastest overall rate while implementing all mandatory patterns.

Once a team has achieved line rate, there are additional optional patterns that they can implement: 80 category-specific patterns, 10 category-independent patterns, and 25 additional

Team Name	Place	Number of Patterns		Line Rate	Platform	Institution
		Mandatory	Optional	(Mb/s)		
Sasao Lab	1 (tie)	25	115	798	FPGA: Altera Stratix III	Kyushu Institute of Technology, Japan
Limenators	1 (tie)	25	115	500	FPGA: Xilinx V5LX330	IBM Research, USA
SpbSU	3	25	101	500	FPGA: Xilinx ML505	Lanit-Tercom, Russia
Kraaken	4	25	60	734	FPGA: Xilinx XUPV5	AMD, USA
Battery	5	25	10	524	GPU: NVIDIA Tesla T10	Iowa State University, USA
Team IISC	6	25	0	584	FPGA: Xilinx XUPV5	Indian Institute of Science, India
Tosan	7	25	0	524	GPU: NVIDIA GTX 295	IPM, Iran
[Ii][Ss][Uu][0-2]{4}	8	10	32	534	FPGA: Altera Stratix III	Iowa State University, USA

TABLE II
FINAL PERFORMANCE RESULTS

categorization patterns. These optional categorization patterns have no category-specific patterns associated with them. For instance, it may be useful for a packet sniffer to identify a World-of-Warcraft stream in order to perform network-policy enforcement. However such a stream does not need to be checked for intrusion detection (beyond the checks which are applied to all streams). The submission which implements the most patterns while meeting the line rate would win the contest (assuming any submission managed to meet the line rate).

Additionally, teams that were able to implement all 140 patterns were allowed to contact the organizers to request more patterns. This scaling points system was designed to encourage participation in the contest. The organizers hoped that the 25 mandatory patterns would provide a realistic enough goal that teams would be interested in participating, which would then lead to the challenge of implementing as many patterns as possible in parallel. Furthermore, some patterns were much more complex than others, so there was some strategy involved in deciding where teams should invest their effort.

Previous years' contests have employed a "normalization" function to attempt to make comparisons between different platforms. We find it significant that the nature of this year's problem allowed us to do away with normalization and rely only on absolute metrics.

5. Reference Design and Default DFAs

In order to ease the implementation burden on the contestants the organizers created a reference implementation of CANSCID that could either run on a Xilinx Microblaze (for FPGA-based teams) or X86 (for software or GP-GPU based teams). Although the reference design was written in C++ it was purposely structured in a "hardware-like" way, as shown in Figure 2.

The goal of these hardware-like blocks was to ease the groups' transition to hardware by already defining a potential implementation architecture (though not necessarily an optimal one). Similarly, the reference design included a set of Deterministic Finite Automata (DFA) for each pattern, generated using the JLex lexical analyzer. This approach was straightforward to implement, but might not result in the best performance. The goal was to offer teams a progressive implementation strategy whereby a non-optimal design could be brought up and then refined for performance.

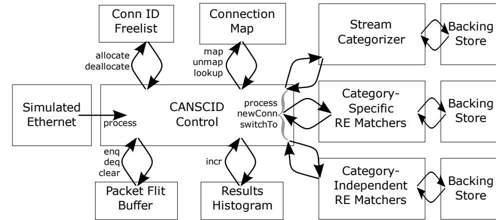

Fig. 2. Overview of the CANSCID reference design implementation.

The DFA generators (taken from a student project done for a class at MIT) actually contained an interesting latent bug. It is well known that a pattern like `vrfy.*` results in a cheaper DFA than `.*vrfy.*`. In a misguided attempt to take advantage of this, the students used the `vrfy.*` style DFA, but replaced all transitions to the "error" state with ones to the "start" state. The result was DFAs that seemed to work, but actually could not handle strings like `vvrfy` or `vrvrfy`, as the second `v` causes a transition to the "start" state rather than the "seen a v" state. When the contest began, the bug was quickly reported by multiple teams. The organizers decided that—rather than unexpectedly changing the complexity of the DFAs in the middle of the contest—we would just ensure that no such prefixes occurred in our testing benchmarks. This would not penalize teams using the bugged DFAs, but also would not hurt teams that decided to fix the bug.

6. Final Results

This year we received 8 submissions for our Deep Packet Inspection problem. 6 submissions used FPGAs, and 2 used GP-GPUs. The organizers find it significant that no team submitted a software-only solution that did not use some kind of hardware accelerator— an indication that software alone could not meet the required line rate.

This year the contest ended in a tie. Congratulations to the joint winners, Team Sasao Lab and Team Limenators, each having implemented 140 patterns while maintaining line rate. Additionally, Team Sasao Lab was the only team to use an NFA approach rather than DFAs for matching the regular expressions. Full results are given in Table II. The performance of the two winners was verified by the organizers using undisclosed test inputs. The performance of the other teams is self-reported.