NC STATE UNIVERSITY

Configurable String Matching Hardware for Speeding up Intrusion Detection

Monther Aldwairi, Thomas Conte, Paul Franzon

Dec 6, 2004

North Carolina State University

{mmaldwai, conte, paulf}@ncsu.edu

www.ece.ncsu.edu/erl

919 513 2015

Outline

- Introduction
  - Motivation
  - IDS Overview
  - Related Work
- Accelerator Architecture
  - Our Approach
  - Aho-Corasick
  - State Tables
  - The Interface
- Performance Analysis
  - Snort Rules Analysis
  - Memory Size
  - Throughput
  - Comparison with Previous Work
- Conclusions

Motivation

- The explosion of recent attacks by Code Red and MSBlast affected the productivity of computer networks all over the world.
- Software based IDSs running on general purpose processors cannot to keep up with increasing network speeds.
- There is a need to accelerate IDS.
- IDSs need to be configurable to detect new attacks.
- This paper focuses on the string matching of the packet payload against hundreds of patterns.
  - We suggest a memory based accelerator that is reconfigurable and have high throughput.

IDS Overview

- Snort is a widely used open-source IDS

```
alert tcp any any -> 10.1.1.0/24 80 (content: "/cgi-bin/phf")
```

- Intrusion detection can be divided into two problems
  - Packet classification based on header fields
  - String matching over the packet payload.
- The second problem of string matching is the most computationally intensive.
  - String matching routines in Snort account for up to 70%* of the total execution time.
  - 87% of the rules contain strings to match against.

*S. Antonatos, K. Anagnostakis, and E. Markatos. Generating realistic workloads for network intrusion detection systems. In ACM Workshop on Software and Performance, 2004.

Related Work

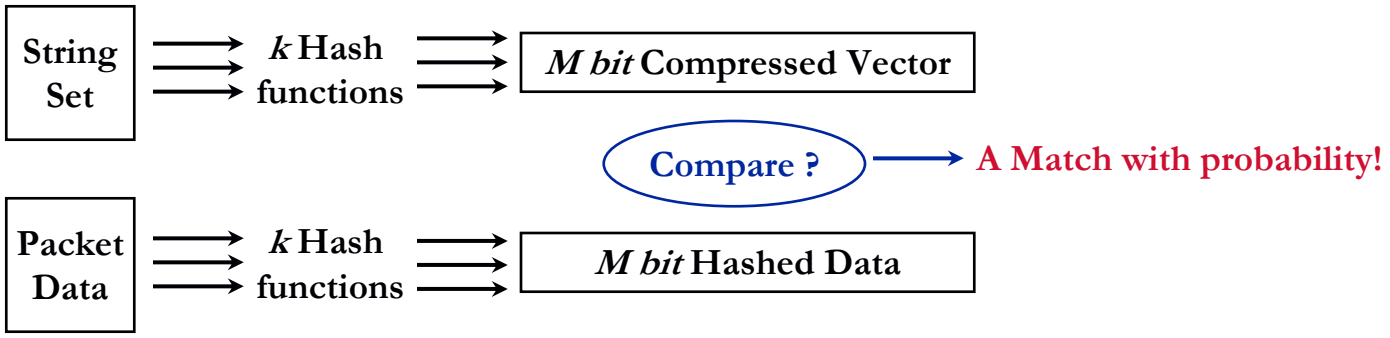
- There have been several attempts to accelerate IDS recently:
  1. Regular expressions
  2. Discrete comparators
  3. CAMs along with comparators
  4. Bloom filters
- Regular expressions are generated for every string in the rule set
- A Nondeterministic/Deterministic Finite Automata (N/DFA) that examines the input one byte at a time is mapped onto an FPGA
- FAs disadvantages
  - Complex and hard to implement
  - Need to be rebuilt every time a string is added
  - Result in designs with a modest throughput.

Related Work

- Discrete comparators were used to exploit parallelism and achieve higher throughput.
- The disadvantage of this approach is the poor scalability in terms of area.

- CAMs and discrete comparators were used to reduce the area.
- The drawback is the high cost and power requirement of CAMs.

Bloom Filters



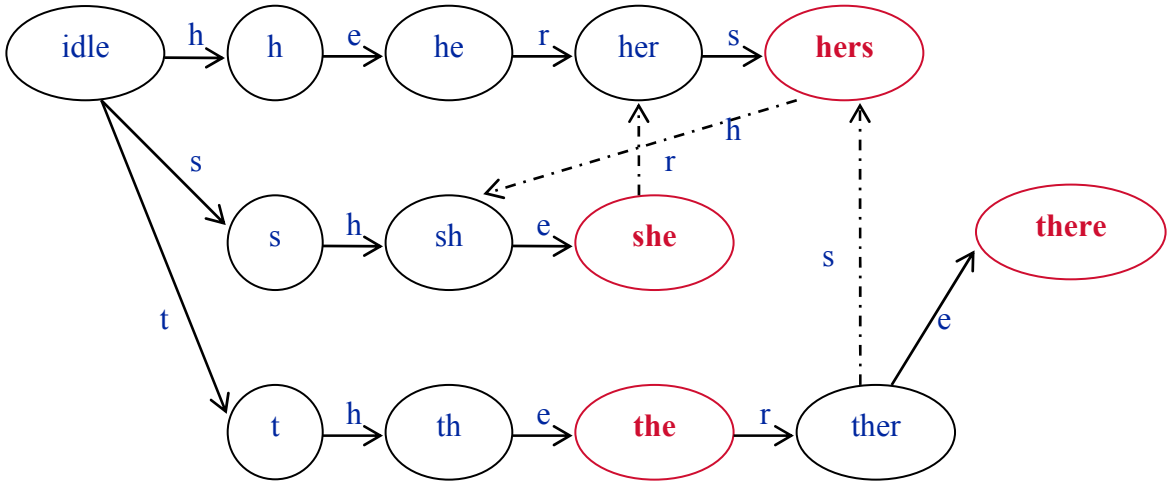
- Bloom filters use less memory and are easy to reprogram.
- The throughput is slightly better than DFAs (about 3 Gbps).
- The number of false positives depends on the number of bits in the compressed vector (*m*) and the number of hash functions (*k*)

Our Approach

- The IDS is composed of two components
  - A software that runs on the VLIW core
  - A hardware string matching accelerator
- The software extracts the strings from the Snort database, creates the FSM tree and generates the state tables
- The hardware implements a Mealy FSM and consists of
  - A RAM to store the state tables
  - A register to hold the current state
  - A control logic to access the RAM and find a match.
- The FSM matches multiple strings at the same time based on the Aho-Corasick string matching algorithm.

Aho-Corasick

- This figure shows a state machine constructed from the following strings {hers, she, the, there}.

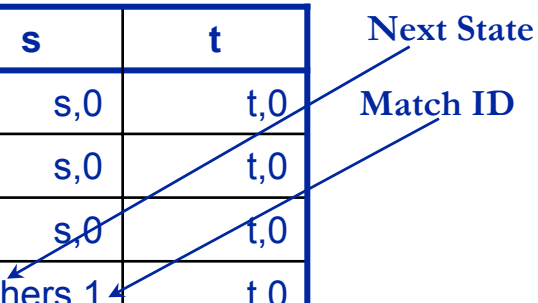


State Table

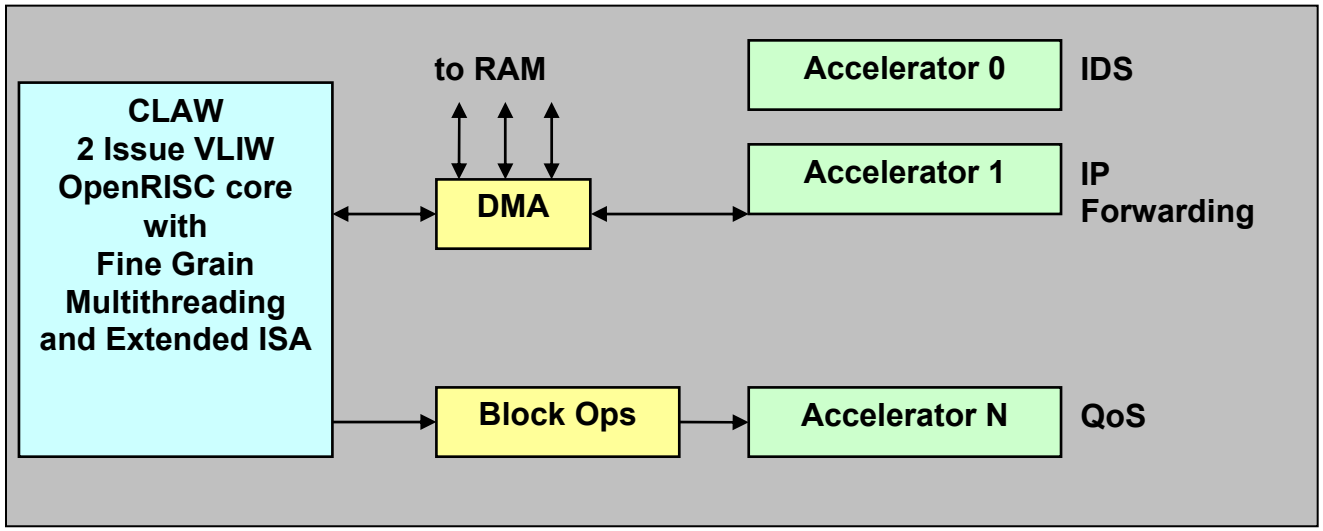
- {hers, she, the, there}

Input Character

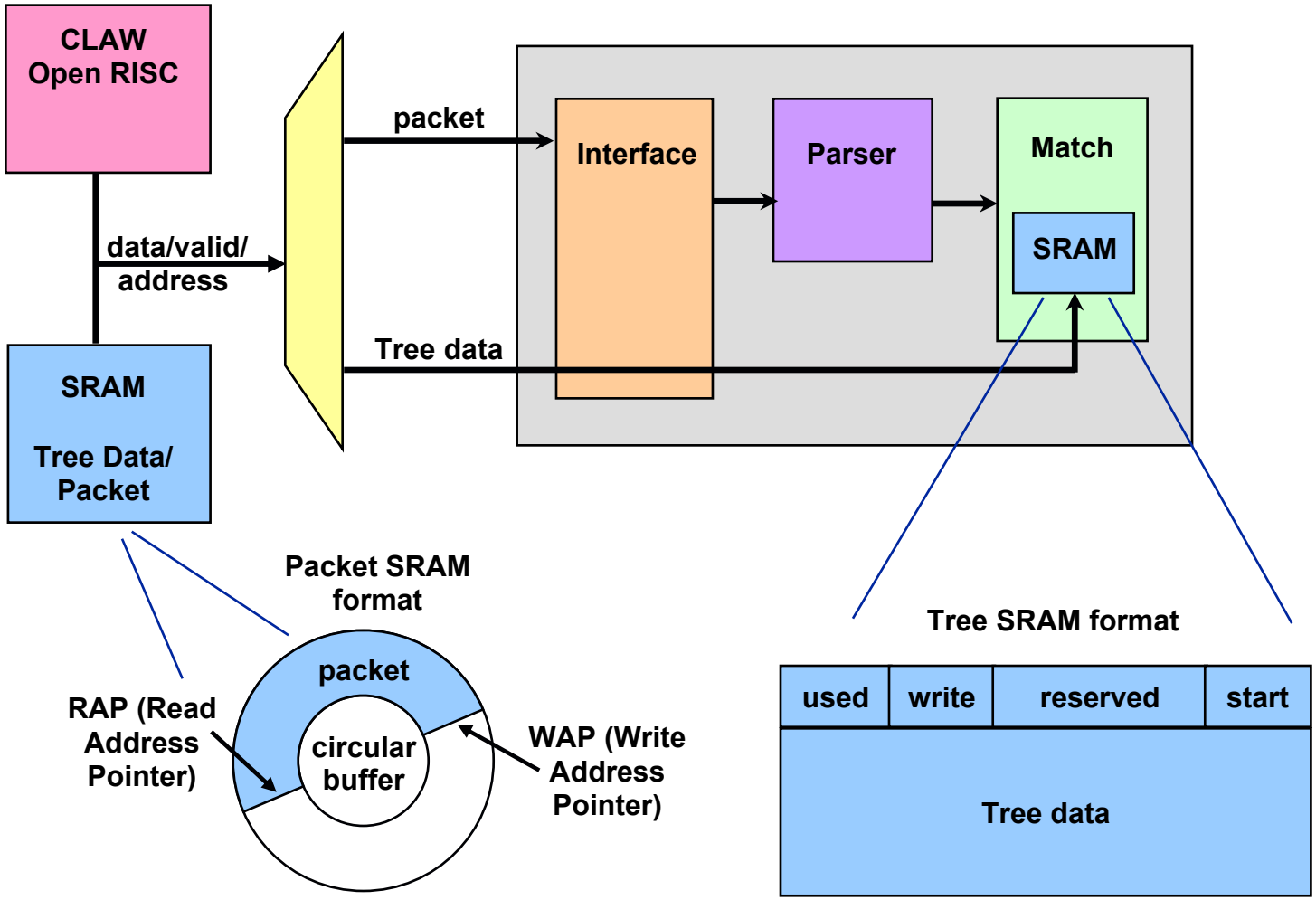
Current State	e	h	r	s	t
- /(idle)	-,0	h,0	-,0	s,0	t,0
h	he,0	h,0	-,0	s,0	t,0
he	-,0	h,0	her,0	s,0	t,0
her	-,0	h,0	-,0	hers,1	t,0
hers	-,0	sh,0	-,0	s,0	t,0
s	-,0	sh,0	-,0	s,0	t,0
sh	she,2	h,0	-,0	s,0	t,0
she	-,0	h,0	her,0	s,0	t,0
t	-,0	th,0	-,0	s,0	t,0
th	the,3	h,0	-,0	s,0	t,0
the	-,0	h,0	ther,0	s,0	t,0
ther	there,4	h,0	-,0	hers,1	t,0
there	-,0	h,0	-,0	s,0	t,0



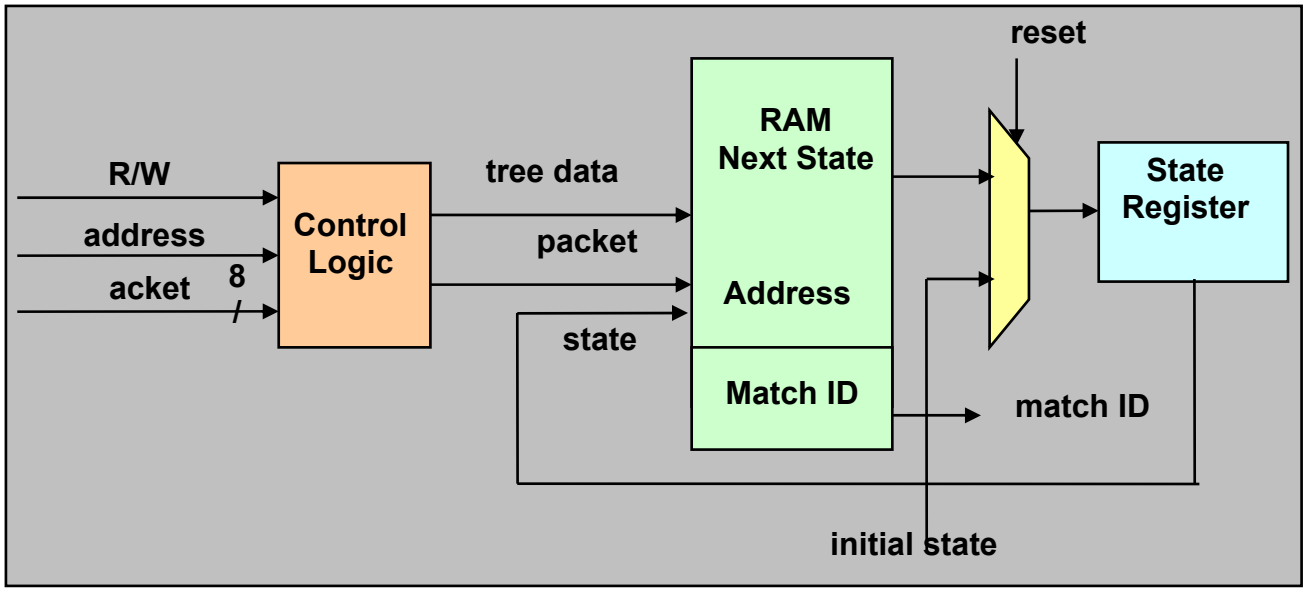
Configurable Processor Architecture



Processor-Accelerator Interface



Match. Block Diagram



The Interface

- CLAW (Clustered Length Adaptive Word) deals with the accelerator as a memory mapped peripheral
  - No complex instructions, just memory loads and stores
- The interface is composed of two RAMs
  1. The packet SRAM
  2. The internal tree SRAM
- The packet SRAM is a circular buffer that stores the packets, and has two pointers (first two words of the SRAM)
  1. RAP (read address pointer) updated by the accelerator and points to the next word to read
  2. WAP (write address pointer) updated by CLAW and points to the last written word
- CLAW writes the packet(s) and asserts the start bit in the tree SRAM.

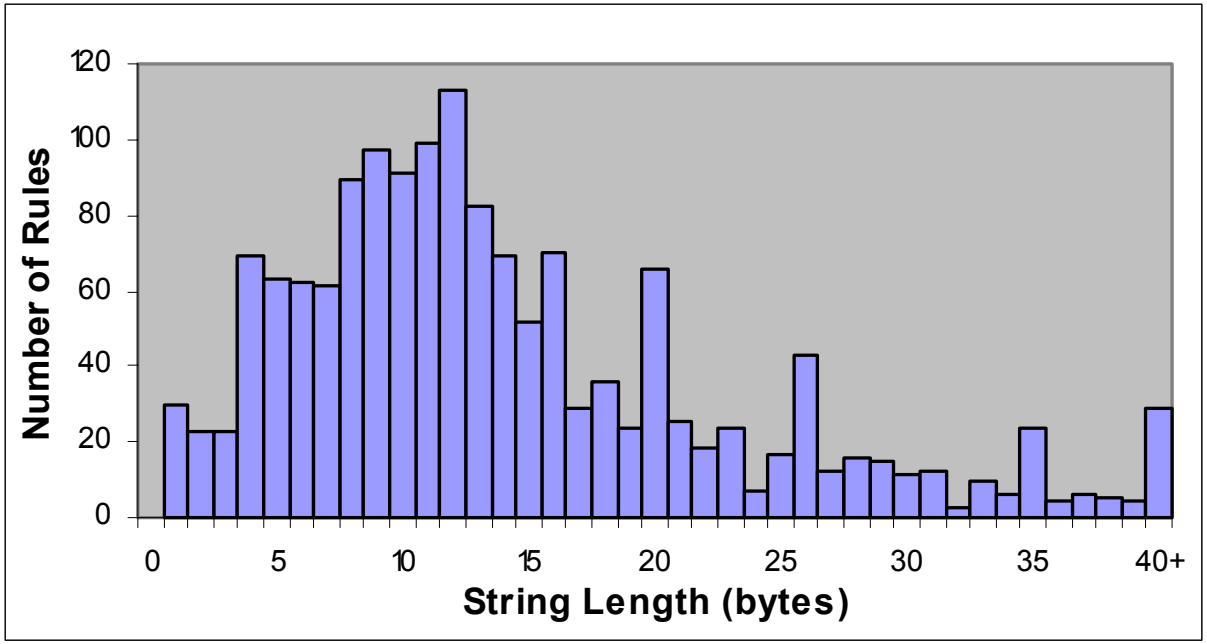
The Tree SRAM

- The first word in tree SRAM is a control word with three bits

- The used bit indicates that the SRAM is being read by the accelerator (CLAW shouldn't update it)
- The write bit indicates that CLAW is writing the tree data (the accelerator shouldn't use it)
- Start bit is set by the processor to instruct the accelerator to start processing the packet
- Once the accelerator finishes processing a packet it resets the start bit and interrupts the processor with the match ID stored in the second word

Snort Rules Analysis

- Our study of the Oct. 2003 Snort rules set showed that 87% of the 1777 rules studied contained strings to match against.



Memory Requirement

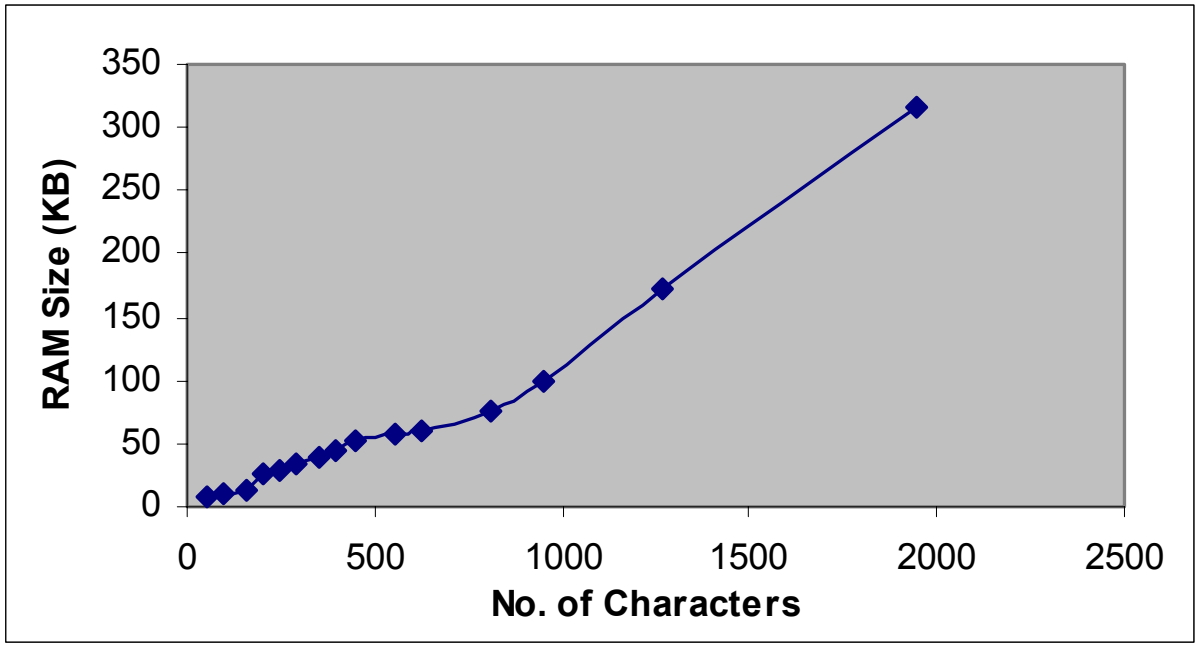
- The RAM dictates the size and limits the throughput of the accelerator.
- The equation below shows the memory requirement in bits

$$RAM = \left(\lceil \log_2 s \rceil + \lceil \log_2 n \rceil\right) * s * c$$

  - where *s* is the number of states, *n* is the number of strings and *c* is the number of characters per string.
- The number of states depends on the number of strings and the number of characters per string.

Memory Requirement

- The empirical memory requirement increases linearly with the number of states (number of characters per string set)



Memory Requirement

> The size of the state tables for all of Snort 2003 rule set is around 3MB which can be fitted on chip

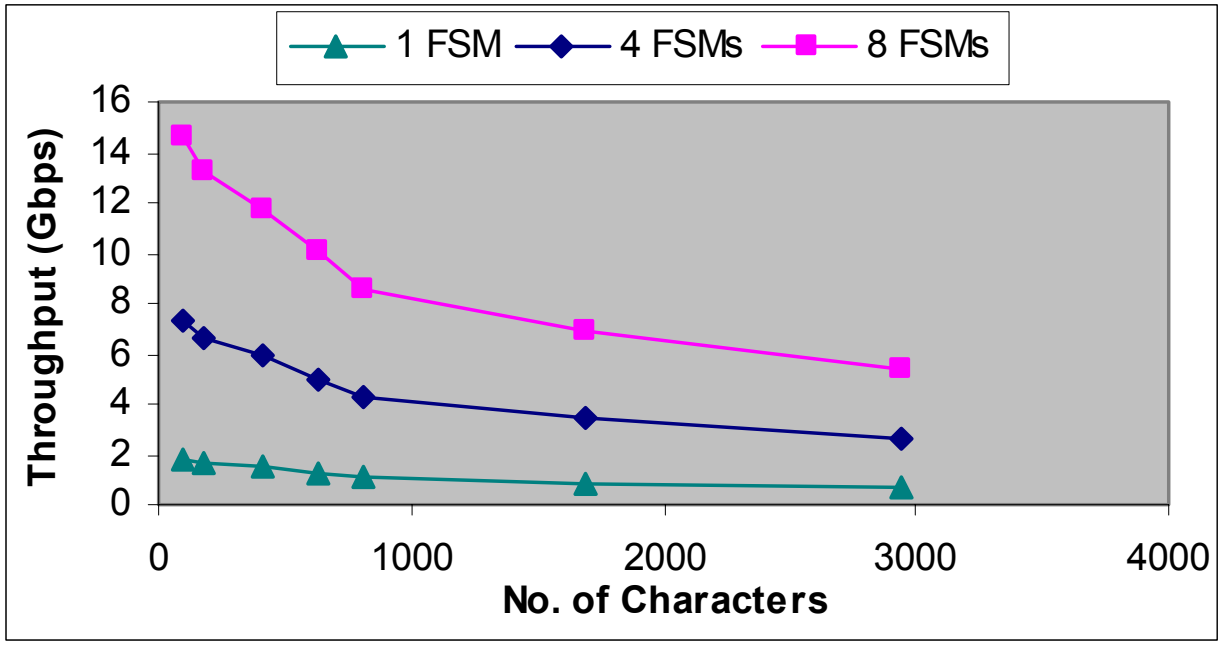
Rule class	Rules	Strings	States	RAM (bytes)
FTP	50	49	268	43,997
SMTP	18	24	362	56,840
ICMP	22	11	138	17,501
RPC	124	58	720	132,623
Oracle	25	25	265	40,366
Web-CGI	311	311	3133	747,939
Web-Misc	275	275	3242	768,975
Web-IIS	108	108	1514	314,652
Web-PHP	58	58	914	172,132
Web-Coldfusion	35	35	572	98,081
Web-Frontpage	34	34	367	59,926
Other classes	717	554	-	709,963
Total	1777	1542	-	3,118,996

Throughput

- CACTI version 3.2 was used to model the on-chip RAM
- Higher throughput is achieved by using more FSMs in parallel.
  - By using 8 FSMs a throughput of around 14Gbps is achieved as opposed to 7 and 2Gbps for 4 and 1 FSM(s), respectively.
- The throughput decreases as the number of characters increase
  - due to the fact that as the number of characters increases, the number of states increases and the state table size increases as well.
  - The throughput for 8 parallel FSMs decreases to about 5Gbps for 3000 characters.

Throughput

- 8 FSMs with 500 chars per class > 10Gbps



Comparison with Previous Work

Description	Input Bits	Device	Through-put(Gbps)	Logic Cells/Char
Aldwairi *et al.* State tables in a RAM *(This work)*	64	Altera EP20k400E	10.1	15
	32	Altera EP20k400E	5.0	
Sourdis *et al.* Pre-decoded CAMs	32	Virtex2 6000	9.7	3.56
Gokhale *et al.* CAMs/Comparators	32	VirtexE-1000	2.2	15.2
Cho *et al.* Discrete Comparators	32	Altera EP20K	2.9	10.6
Sidhu *et al.* NFAs/Regular Expression	8	Virtex 100	0.75	~31

***The data for the other designs was obtained from I. Sourdis and D. Pnevmatikatos. Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching. In Proceedings of 12th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM04), April 2004**

Conclusions

- 87% of the rules contain strings to match against.
- We have presented a configurable string matching accelerator based on a memory implementation of the AC FSM.
- This results in a small memory requirement of 3 MB that is likely to fit in on-chip SRAM.
- We have shown that the accelerator can achieve more than 10 Gbps throughput by using 8 parallel FSMs.

Questions?