A Signature Match Processor Architecture for Network Intrusion Detection

Janardhan Singaraju Long Bu John A. Chandy

University of Connecticut
Storrs, CT 06269-1157

Abstract

In this paper, we introduce a novel architecture for a hardware based network intrusion detection system (NIDS). NIDSs are becoming critical components of the network infrastructure as they serve as a key line of defense in network protection. However, current methods are much too compute intensive and can not begin to meet the bandwidth requirements of a moderate sized corporate network. Thus, hardware techniques are desired to speed up network processing. This paper introduces a FPGA based signature match processor that can serve as the core of a hardware based NIDS. The signature match processor's key feature is a CAM-based cellular processor architecture that can match strings in an area efficient manner. Using a unique binary tree structure, we are also able to generate priority encoded addresses corresponding to multiple signature matches.

1. Introduction

With the rapid increase in malicious attacks on corporate and government networks, there has been an increased awareness amongst network administrators to deploy tools that protect them from external attacks. Network intrusion detection systems (NIDS) are one of the primary tools available to help in creating a secure network infrastructure. Network intrusion detection is the process of identifying and analyzing packets that may signify an impending threat to organization's network. NIDS can be deployed in a variety of configurations including passive and host-based systems. Passive NIDS entail using a secondary network node to analyze all packets on the network. A host-based NIDS monitors a single machine and is useful in gateways, switches, or routers as a means to block all suspicious traffic.

Inspecting incoming packets for tell-tale signatures can be time consuming especially if the number of possible signatures is large. The Snort open-source intrusion detection software suite has well over a thousand rules [18]. Current high-performance systems can barely process that many rules on a 100 Mbps moderately loaded network [19]. To handle fully loaded gigabit networks, a NIDS must either drop some of the rules or drop some of the packets it analyzes. Neither solution is desirable since they both compromise security. The alternative is to deploy multiple NIDS on the network to inspect packets in parallel. Doing so increases the complexity of the system as traffic splitting techniques are needed to direct packets to the appropriate node in order to retain TCP state.

String matching is the most computational intensive part of the intrusion detection process as each analyzed packet must be inspected against several signatures. More than 30% of the computation time can be spent in string matching [8]. For the most part, previous work in this area has focused on improving software algorithms for string pattern matching [7, 1, 8]. Others have recognized the potential of hardware implementations and used reconfigurable hardware to attack the problem.

Previous approaches to string matching in FPGAs have included finite automata methods that translate regexp signatures into hardware implementations [20, 10, 14, 6]. These methods require representing the regular expression as a finite automata graph and then translating that directly to FPGA circuitry. The complexity of this translation can lead to large amounts of logic circuitry and thus more area on an FPGA.

Another approach to FPGA-based NIDS is the use of content addressable memories (CAMs) or discrete comparators [9, 5, 21] Content addressable memories have long been used for fast string matching against multiple keywords [17, 15, 13]. Common uses include caches and lookup tables, and in the networking area, the use of CAMs as an IP address lookup table in routers is well known [12]. Both caches and IP address lookup tables are ideal candidates for CAMs since the keys are of a fixed-length - address tag for caches or IP address for lookup tables.

For the purposes of string matching in a NIDS, one could conceive of an architecture where the CAM may contain a set of signatures with a fixed key size of k bits [9, 11].

As packets arrive from the network, each k bits could be matched against the CAM to see if there was a match. [1] If a match is found, we know that the signature is present in the packet and we can flag the packet for further analysis. Since the signature set is stored in a writable CAM, CAM-based NIDs systems do not need to reprogram the FPGA every time there is a change in the rule set. However, unlike finite automata, CAM-based designs can not easily handle regular expressions, though the use of TCAMs (ternary CAMs), does allow limited wildcard matching

The primary problem with such an architecture is that NIDS signatures are not of a fixed size. For example, the Snort ruleset has rules that match on content strings that can be anywhere from 1 to 100 characters long. One solution could be to select the value of k to accommodate the largest possible signature, though that will lead to unused space within the CAM. The Granidt system uses such an approach with a 160-bit wide CAM, thereby limiting signatures to 20 characters length. Using a fixed value of k can also cause misses in the incoming stream because of overlaps. For example, consider the case where the CAM contains two signatures FOO and BAR, and the input stream is AFOOBARCD. The CAM will be presented with $k = 3$ characters at once - AFO, OBA, and RCD. Because of overlap, none of these will cause a match, even though the input stream obviously contains the two signatures.

Recent CAM-based designs have recognized this problem and taken a sliding window approach that use single character comparators with shift registers to propagate matches across clock cycles [21, 5]. Optimizations to these approaches include processing characters in parallel, prefix sharing, and pattern partitioning.

In this paper, we present a novel CAM-based signature matching processor that uses an array of cellular automata to process character matches. The cellular automata structure is compatible with the addition of the above mentioned optimizations including parallelism . The compactness of each cellular automata element leads to a highly efficient design in terms of both area and speed.

The paper is organized as follows. We start with a description of the signature matching processor which does string matching and then follow with an overview of how this design would fit in an overall NIDS architecture. We then describe FPGA implementation details and then finally conclude with a note on future directions and applications.

2. Signature Match Processor

An architectural overview of the SMP is shown in Figure 1. The SMP consists of a control unit, a CAM character match array, a signature match array, and address output logic. The SMP receives packets from the network in a stream and outputs the address of any signature that is found in the packet. A managing network processor or CPU can use this information to raise a network alert or attempt to terminate the offending connection.

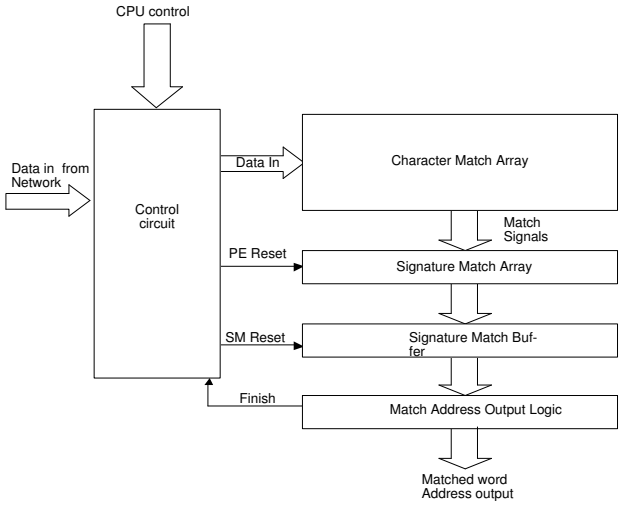

Figure 1. Signature Match Processor Architecture.

2.1. Character Match Array

The core of the SMP is in the character match array and signature match array. In an earlier version of the architecture, the character match array was implemented as an array of CAM cells into which the desired signatures were loaded [4]. The advantage of the structure was that it was easy to update signatures simply by writing into the CAM. The disadvantage, however, was that the CAM array was very large in area. Moreover, the use of a CAM array prohibited the use of optimizations such as parallelism and prefix sharing. Because of these limitations, we decided to abandon the use of the CAM array and instead employ discrete comparators as was used in [21, 5]. Sacrificing the ability to update the signatures without reconfiguring the FPGA is not a serious loss since signatures in a NIDS context change relatively infrequently.

The character match array is comprised of a series of comparators, each of which matches on one of the possible incoming bytes. This, of course, implies that there are 256 comparators to match all possible 8-bit characters. In order to improve performance, it is desirable to be able to match several characters within one clock cycle. Therefore, we use p rows of comparators, where p denotes the degree of parallelism. As shown in Figure 2, there are p match signals per comparator column. Also, for every clock cycle, ex-

1 In such an application, there would be no "value" field in the CAM since presence in the CAM is all that we are concerned with.

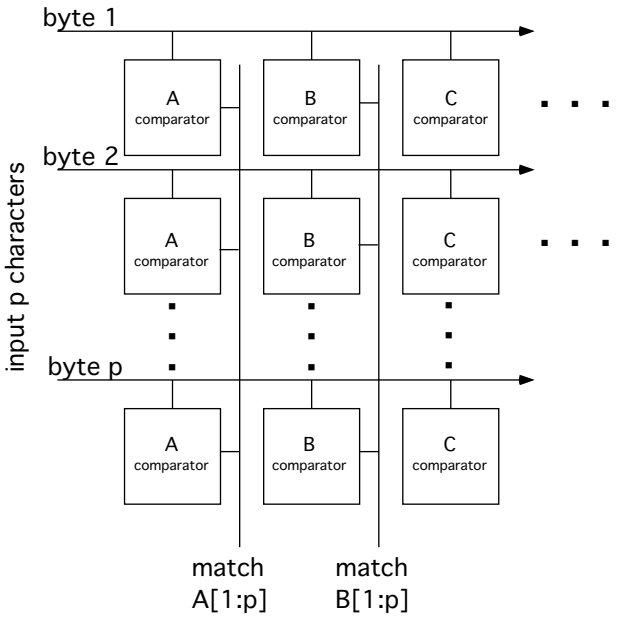

Figure 2. Character Match Array

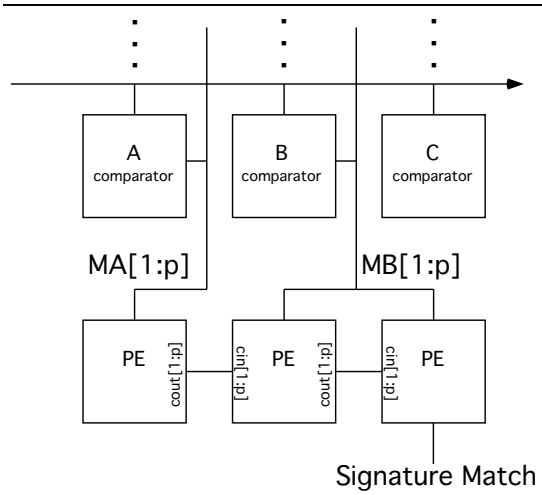

Figure 3. Signature Match Array.

actly p of the $256p$ match signals from the character match array will be asserted.

The signature match array is the distinguishing feature of the design. The signature match array is similar in concept to systolic processing arrays that have been proposed for approximate word search matching in CAM-based dictionaries [15, 16]. The array is comprised of an n x 1 array of processing elements (PE) where n is the number of characters in the signature set to be matched against. Thus, each PE represents one of the characters in the signature set, and likewise, the PE is connected to the character match array column that corresponds to its character. The diagram in Figure 3 shows how the array would be connected for the string ABB.

In general, each PE has $2p$ inputs and $p + 1$ outputs. p of the inputs, represented by $MX[1 : p]$, correspond to the p match signals from the character match array. X refers to the character for which the PE is responsible. The remaining p inputs are carry signals that forward match information from the previous PE, and likewise, p of the outputs are carry signals to a subsequent character in a signature. The final output is a signal that indicates a signature match was found. The signature output match signal is only valid on PEs that signify the end of a signature. In addition to knowing whether the PE is at the end of a signature, the PE also knows if it is at the beginning of a signature.

A summary of the algorithm executed by each PE is shown below for $p = 4$ in pseudo-VHDL.

```
cout1 <= MX1 and (cin4 or sig_beg);
cout2 <= MX2 and (cin1 or sig_beg);
cout3 <= MX3 and (cin2 or sig_beg);
cout4_temp <= MX4 and (cin3 or sig_beg);

sig_match <= sig_end and
(cout1 or cout2 or
 cout3 or cout4_temp);

if ( clk'event and clk='1') then
   cout4 = cout4_temp;
end if
```

The first part of the pseudo-VHDL generates the carry signals to propagate to the next PE. A carry signal indicates

that there is a signature match up to that character. The basic idea is to check each of the MX signals for a match and then see if the previous PE also found a signature match up to the previous byte. You must also check if the PE is at the beginning of a signature, i.e. sig_beg is '1'. If the previous PE has forwarded a signature match or we are at the beginning of a signature, the PE can forward the signature match to the next PE. For example, if $cin1$ is '1', it indicates that the previous PE has determined that the signature has matched up to itself, and the last matched character of the signature is in byte 1. Therefore, if the current PE sees a match on byte 2, i.e. $MX2$ is '1', then you propagate a signature match on $cout2$. If the PE is at the end of a signature, sig_end is '1' and we are going to forward a character match, then we know that we have matched on an entire signature. This will allow us to flag the sig_match signal.

From the pseudo-VHDL it can be seen that the pth carry out is registered in each PE. This is because the only way a match can occur on MX1 is if it is at the beginning of the signature or the last character on the previous clock cycle matched also. The last character on the previous clock cycle corresponds to a registered version of the pth carry out. The implications of this are that only one register is required per character in the signature set and moreover, the number of registers do not increase as we increase parallelism.. This is a significant savings compared to other comparator based techniques which are $O(L^2)$ relative to the length of the signature [21].

As an example of the PE algorithm, consider the following rule from Snort.

```
alert udp $EXTERNAL_NET
 any -> $HOME_NET 31335
  (msg:"DDOS Trin00 Daemon to Master
    message detected"; content:"l44";
    reference:arachnids,186;
    classtype:attempted-dos;
    sid:231; rev:3;)
```

The signature match arrays are shown in Figure 4 and the PEs are configured to detect the signatures `l44` and `adsl`. The input string `fl44` is presented to the SMP in two clock cycles. In the first clock cycle, the `l` matches in the second row and that sets the register in the first PE corresponding to `l`. On the second clock cycle, the `4` matches in both rows. The first '4' PE will have $cin2$ set from the 'l' PE. That allows it to forward a match to the second '4' PE on $cout1$. The second '4' PE will see $cin1$ set as well as $MX2$ and it can then determine a signature match because it has sig_end set as well.

2.2. Address Output Logic

The signature match array outputs a word match signal that indicates that a match was found. However, for the SMP

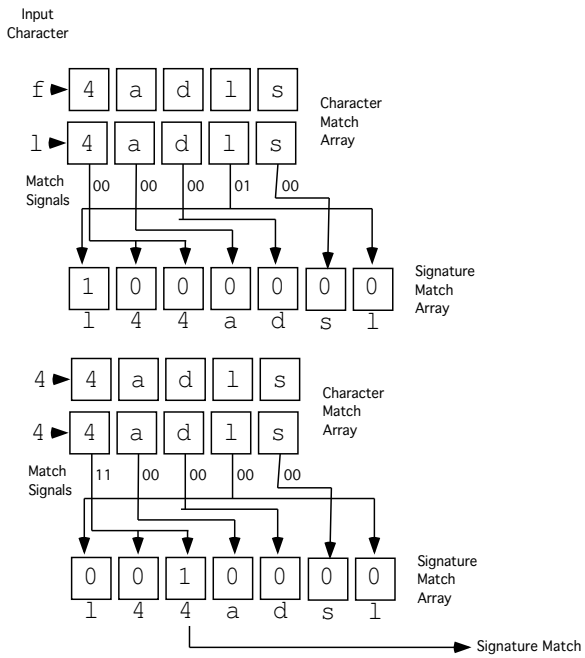

Figure 4. SMP Example.

to be useful in the context of NIDS, we also need to know which signature caused the match. It is the address output logic that finds which signature or signatures caused a match. In order to generate these addresses, we use the signature match signals generated by the signature match array. As the input string propagates through the CAM array, any time there is a signature match, there will be a signature match signal at the end position of the signature. Since there may be multiple signature matches in a single string and the signature match signals can appear on different clock cycles, it is necessary to latch each word match signal in the word match buffer. The buffer maintains the position of the signature match. When the last character of the input string has passed through, the word match buffer will then have values set at each location corresponding to the end of a matched signature.

When the last character has been reached, the match address output logic can begin processing the word match buffer entries. Thus, we need to get the beginning of the signature address from end of the signature address returned by the word match buffer. In previous methods [4] we used a start address RAM that stores the beginning address of each signature. The address output logic simply reads the RAM entries starting from end to beginning. For each start address A read from RAM address i, the position $A-1$ of the word match buffer is verified. A positive match indicates a signature match corresponding to the start address located at the $i-1$th position of the RAM. This method takes S cycles to perform the logic, where S is the number of signa-

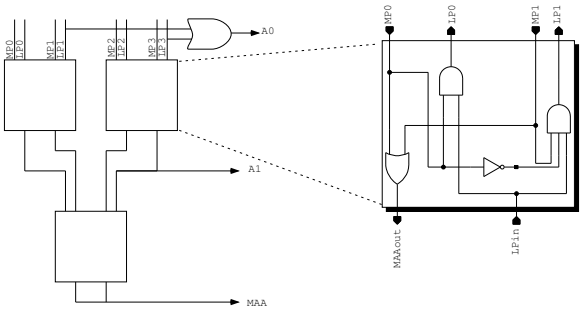

Figure 5. MAO Logic.

tures. Larger values of S would result in a longer time delay in performing the logic.

In this paper, we present a new match address output logic(MAO logic) which has the structure of a binary tree. From the word match buffer we need the start position of signature. This is accomplished by propagating the word match signal at the end position of the signature to the start position of the signature. This gives the matched position location of the signature (MP) in the CAM. Whenever the CAM character match array is updated these connections must be updated as well. The MP is given as input to MAO (matched address output) logic block which has the structure of a binary tree.

Figure 5 shows the binary tree and the logic for each of the block in the tree. MAO logic separates multiple matches for signature and decodes the start address of each matched signature. This logic generates a MAA (matched address available) signal which tells the control circuitry that there are matches. Then a LP (Leftmost Pointer) signal propagates back up the tree which is the leftmost MP signal. At each level of the tree, a bit of the address is generated from the LP signal at that level. When the LP signal reaches the top of the tree, it is used to reset the register of the MP signal that was just encoded. On the next clock cycle, the MAO can then encode the next left-most MP entry. For M matches found for a particular input string, the MAO logic takes M cycles to perform the logic. This is better than the S cycles taken by the previous RAM-based implementation of the MAO logic. We also pipelined the MAO logic to increase the clock frequency, and found that the optimal number of stages was two. Further pipelining did not improve clock frequency since the PE processing then becomes the limiting factor.

2.3. Control Circuit

The control circuit manages the data flow through the SMP and also manages flow control of the incoming packet. When an incoming packet is ready to be delivered, the control circuit first resets the signature match array and also resets the word match buffer. The control circuit then takes each byte in the incoming packet and presents it to the character match array on every clock cycle. When the last character in the packet has arrived, the address output logic is enabled. Because the address output logic process is independent of the signature matching process, the control circuit can start processing the next packet immediately.

2.4. Performance Analysis

Overall, the time to process a b byte packet is $b/p+M+1$ cycles where M is the number of matches found in the packet. b/p corresponds to the time for the packet to stream through the SMP signature matches and $M+1$ is the time to do the matched address output. Since the matched address output phase could be completed in parallel with the signature match, we are left with a per-packet cycle time of $max(b/p, M+1)$. If $b/p > M+1$, which is the general case, the per-packet cycle time is b/p, and the per-byte run-time is $1/p$ cycles.

3. NIDS with SMP architecture

The previous section described the architecture of a CAM-based signature match processor. In this section, we describe how a SMP can be used in the design of a hardware based NIDS. The overall architecture is shown in Figure 6. On initialization, the SMP is loaded with signatures from an IDS ruleset. Typically, a controlling CPU or network processor would be responsible for this initialization. Packets from the network can flow into the SMP from a variety of possible sources including a MAC/PHY interface (as shown in the figure), from memory, over a GMII or SPI interface or from a TCP/IP offload engine (TOE). The latter would allow for stateful intrusion detection processing. If there are any signature matches, the SMP will interrupt the CPU in order to facilitate an alert or intrusion prevention mechanism. It is anticipated that the packets will be buffered in memory, so that the CPU can do further processing on the packet if necessary.

4. FPGA implementation

We have designed this NIDS architecture using VHDL and our initial implementation has targeted the Xilinx Virtex-II Pro FPGAs, specifically the and XC2VP230 part with -7 speed grade. The eventual goal is that we could use the embedded PowerPC in the Virtex-II Pro to perform the software components of the NIDS, such as SMP management, alerts, logging, etc. The Virtex-II Pro also has Rocket I/O transceivers that could be used to imple-

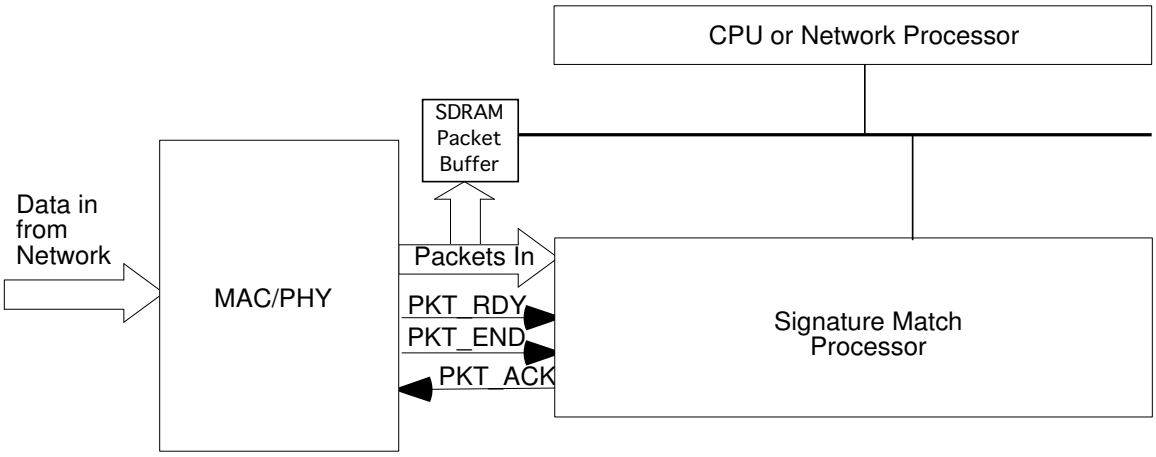

Figure 6. NIDS Architecture.

ment a gigabit MAC. Using these features, the device could be a complete NIDS solution from.

Xilinx's ISE design environment was used for all parts of the design flow including synthesis, mapping, and place and route. We have implemented from the Snort rule database signature sets ranging from 94 rules comprising 1024 characters to 1237 rules comprising 16347 characters. The entire Snort rule database contains 2758 rules comprising 37680 rules. Unfortunately, Xilinx's ISE synthesis tool was not able to process that large a design. Table 1 shows the resource utilization for different levels of parallelism. As parallelism increases, the throughput increases as expected. However, the efficiency in terms of operating frequency decreases mainly because the PE becomes more complex as p increases. It is also interesting to note that as the signature set size increases the number of logic cells per character decreases and the operating frequency decreases as well. This is because as the signature set size increases, each CAM character cell is connected to more PEs and thus creating increased fanout on each cell.

The design using binary tree structured MAO logic uses roughly 1.5 registers and 1.5 LUTs per CAM character compared to 2 flip-flop and 3 LUTs per CAM character of SMP using a start address RAM to implement the MAO logic. The register is used to implement the PE register and the word match buffer. The LUTs correspond roughly to the CAM, the PE logic, and the matched address output logic. The flip-flops and LUTs can be mapped to almost 1 slice per character.

Table 2 shows a comparison of our work with other recent related work in FPGA implementations of signature matching. The performance metric is the ratio between throughput and logic cells/char and is similar to that introduced in [3, 6] to evaluate the trade offs between area and performance. Our design is comparable in performance to other work in the area, particularly other comparator based designs [21, 6, 5]. What is particularly notable is that the number of logic cells/character is significantly smaller than any other comparator based design. With increased parallelism, the throughput should increase significantly.

5. Conclusions and Future Directions

In this paper, we have introduced a novel architecture for a hardware based network intrusion detection system (NIDS) using an innovative CAM-based signature match processor. Based on the current implementation of the SMP, we can process incoming streams at rates of over 5 Gbps. This is more than sufficient to handle intrusion detection on current gigabit networks. We have also presented a unique design of a priority address encoder that will generate addresses even in cases when there are multiple matches within a packet.

The SMP design also opens opportunities in other applications besides NIDS. Any lookup that is based on non-fixed-size keys seems to be an ideal candidate to take advantage of these SMPs. Some examples include directory lookup in network storage applications, DNS lookup, and LDAP processing. These are all applications that require large amounts of computational power to perform string matching lookup based operations. We are investigating the use of SMPs in these applications as well as developing extensions of the SMPs to support wildcards and approximate word matching capabilities as well. Other research directions include improving the power characteristics of the SMPs.

References

[1] K. Anagnostakis, S. Antonatos, E. P. Markatos, and M. Polychronakis. E^2XB: A domain-specific string matching algorithm for intrustion detection. In *Proceedings of IFIP Inter-*

	XC2VP30		
	1021 characters (94 rules)		
Level of parallelism	1	2	4
Slices	456	689	1119
Frequency (MHz)	389.3	244.0	200.3
Throughput (Gb/s)	3.11	3.90	6.41
	2044 characters (246 rules)		
Level of parallelism	1	2	4
Slices	835	1331	2124
Frequency (MHz)	392.3	203.8	179.2
Throughput (Gb/s)	3.13	3.26	5.73
	8163 characters (770 rules)		
Level of parallelism	1	2	4
Slices	3833	4314	6768
Frequency (MHz)	206.1	154.4	125.5
Throughput (Gb/s)	1.65	2.47	4.02
	16347 characters (1237 rules)		
Level of parallelism	1	2	4
Slices	6777	5656	11176
Frequency (MHz)	281.7	147.8	94.8
Throughput (Gb/s)	2.25	2.36	3.03

Table 1. SMP Resource Utilization

Design	Device	Throughput (Gb/s)	No. characters	Logic cells/char	Performance (Mb/s/cell)
Singaraju et al. (p=2)	Virtex 2VP30-7	2.36	16347	0.7	3424
Singaraju et al. (p=4)	Virtex 2VP30-7	6.41	1021	2.2	2933
Baker-Prasanna (8 byte) [3, 2]	Virtex 2VP100-7	10.3	19584	2.0	5150
Baker-Prasanna (Tree) [2]	Virtex 2VP100-7	2.0	19584	0.4	4848
Cho-Mangione-Smith (ROM) [5]	Spartan3-2000	3.2	6805	0.9	3556
Sourdis-Pnevmatikatos [21]	Virtex 2-6000	9.7	18000	3.6	2694
Clark-Schimmel et al. [6]	Virtex2-8000	7.0	17537	3.1	2245
Cho-Mangione-Smith (RDL) [5]	Spartan3-2000	3.2	19021	1.6	2000
Hutchings et al. [10]	VirtexE-2000	0.4	16028	2.5	160
Gokhale et al. [9]	VirtexE-1000	2.2	640	15.2	145

Table 2. Comparison NIDS FPGA Designs

national Information Security Conference, pages 217–228, May 2003.

[2] Z. K. Baker and V. K. Prasanna. Automatic synthesis of efficient intrusion detection systems on FPGAs. In *Proceedings of International Conference on Field Programmable Logic and Applications*, 2004.

[3] Z. K. Baker and V. K. Prasanna. A methodology for synthesis of efficient intrusion detection systems on FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 135–144, Apr. 2004.

[4] L. Bu and J. A. Chandy. FPGA based network intrusion detection using content addressable memories. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2004.

[5] Y. H. Cho and W. H. Mangione-Smith. Deep packet filter with dedicated logic and read only memories. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 125–134, Apr. 2004.

[6] C. Clark and D. Schimmel. Scalable multi-pattern matching on high-speed networks. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2004.

[7] C. J. Coit, S. Staniford, and J. McAlerney. Towards faster string matching for intrusion detection or exceeding the

speed of Snort. In *Proceedings of DARPA Information Survivability Conference and Exposition II*, pages I:367–373, 2001.

[8] M. Fisk and G. Varghese. Fast content-based packet handling for intrusion detection. Technical Report CS2001-0670, Department of Computer Science, University of California, San Diego, May 2001.

[9] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett. Granidt: Towards gigabit rate network intrusion detection technology. In *Proceedings of International Conference on Field Programmable Logic and Applications*, pages 404–413, 2002.

[10] B. L. Hutchings, R. Franklin, and D. Carver. Assisting network intrusion detection with reconfigurable hardware. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 111–120, 2002.

[11] G. Judge. FPGA architecture ups intrusion detection performance. *CommsDesign.com*, Sept. 2003.

[12] A. J. McAuley and P. Francis. Fast routing table lookup using CAMs. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1382–1391, Mar. 1993.

[13] T. Moors and A. Cantoni. Cascading content addressable memories. *IEEE Micro*, 12(3):56–66, May/June 1992.

[14] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos. Implementation of a content-scanning module for an internet firewall. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, 2003.

[15] M. Motomura, J. Toyoura, K. Hirata, H. Ooka, H. Yamada, and T. Enomoto. A 1.2-million transistor, 33 MHz, 20-b dictionary search processor (DISP) ULSI with a 160-kb CAM. *IEEE Journal of Solid-State Circuits*, 25(5):1158–1164, Oct. 1990.

[16] M. Motomura, H. Yamada, and T. Enomoto. A 2k-word dictionary search processor (DISP) with an approximate word search capability. *IEEE Journal of Solid-State Circuits*, 27(6):883–891, June 1992.

[17] A. Mukhopadhyay. Hardware algorithms for string processing. *IEEE Computer*, pages 508–511, 1980.

[18] M. Roesch. Snort – Lightwieight intrusion detection for networks. In *Proceedings of the USENIX LISA Conference*, Nov. 1999.

[19] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2003.

[20] R. Sidhu and V. K. Prasanna. Fast regular expression matching using FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2001.

[21] I. Sourdis and D. Pnevmatikatos. Pre-decoded CAMs for efficient and high-speed NIDS pattern matching. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2004.