Chapter 3: RECONFIGURABLE FIREWALL BASED ON FPGA-BASED PARAMETERIZED CONTENT ADDRESSABLE MEMORY

Security and network monitoring applications require packet classification and filtration. Packet classification implies finding out in a set of filters the highest priority filter matching the packet. The growth of in demand bandwidth has driven the throughput over classification engines to increase exponentially. The classification task has become more complex and is based on at least five packet header fields: source and destination IP addresses and ports and protocol identifier. The type of fields values are typically prefixes for IP addresses, ranges for ports and exact values or wildcard for protocol identifier. The present chapter reports our development towards the dedicated FPGA based Packet filter based on the header field IP addresses and port numbers. Here we used the black list of ports and IPs from the standard databases declared as attacks and also experimented with the actual network of set up in our laboratory.

3.1 Introduction

Packet classification is the main function in firewalls besides the intrusion detection mechanisms and monitoring architectures. Network elements assuming these techniques operate on packet flows to insure access control. A large variety of multi-fields packet classification techniques have been reported in literature [33] – [34] but it remains difficult to find a packet classification solution that represents a good tradeoff between classification times, fast updates, memory requirements and scalability to large filters database. Advanced packet classifiers capable of examining packets at full wire-speed against large and dynamic sets of complex classification rules are essential building blocks for realizing important emerging Internet applications in the context of firewall metrics such as quality of service, web-server load balancing. Whereas the problem of searching single packet
Chapter 3: **Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory**

fields (e.g., routing table lookup) is considered to be solved, classification on multiple fields remains a difficult problem [11]. This is due to the multi-dimensional nature of the searches in combination with the large number of bits, often on the order of hundreds, that have to be inspected for each packet.

![Fig. 3.1: Structure of IP Header that needs to be parsed by the CAM](image)

A packet filter is an operating system kernel facility that classifies network packets according to criteria given by user applications, and conveys the accepted packets from a network interface directly to the designated application without traversing the network stack [13]. Packet filters have become essential to build fundamental network services ranging from traffic monitoring [14, 15] to network engineering [16] and intrusion detection [17].

The Content-Addressable Memory (CAM) implemented in this chapter is a hardware search memory implemented on a FPGA based platform. Unlike standard RAM (random access memory) in which data is stored in and retrieved from a specific address, data storage and look-up in CAM is by content. CAM is a searching device, consists an array of storage locations. A search result can be obtained in constant time through parallel matching of the input with the data in the memory array. A CAM is much similar to a
Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory

RAM both stores a data but the read mode is significantly different than the RAM [8, 9, 10]. With RAM we input an address, and get data out. With CAM we input data, and if this data is stored in the CAM, we get the address of that data out. There is an address at the output even if there is no match so with a CAM we need a Match bit to indicate if the CAM contains the input data.

3.2 Contentions related to CAM Implementations

As per the reported literature CAM based hardware packet-filters [1] are fast and support various data widths [2]. However, R. Sinnappan et. al. experienced that the size of an FPGA may limit the number of filter rules that can be implemented in hardware [3]. Our work tries to alleviate the above mentioned conception. T.K. Lee and S. Yusuf et. al. further describes two hardware structures that employ resource sharing methods to reduce hardware resource usage for packet filtering firewalls. Resource usage reduces approximately linearly with the degree of grouping of the filter rules in a rule set. These two structures, when applied to CAM based packet-filters, offer various trades-offs between speed and size under different situations involving parallel and pipelined implementations [4]. Jan van Lunteren and Ton Engbersen investigate the mechanism that determine the performance of state-of-the-art multi-field classification methods, and proposes a novel scheme called P²C for packet classification at OC-192 and OC-768 speeds [12]. P²C combines the strengths of embedded memory and ternary CAM technologies to achieve very high storage efficiency while maintaining fast incremental updates. Key feature of the new scheme is its ability to adapt to the complexity of a classification rule set, while providing effective control over the update dynamics and storage requirements at the granularity of individual rules. This makes P²C suitable for a broad range of applications.
3.3 CAM Realization for Firewall Applications: Prior Art

Common applications of CAM are network routing tables and cache memories, among others. In a traditional CAM, the search data must be an exact match of the data stored in the CAM. In a ternary CAM, mask may be used to specify matching of only certain data bits. Note that CAM performance is a constant number of cycles. For example, the Micron T-CAM [5] can contain over 16,000 entries with individual masks that can be searched in one clock cycle when data is pipelined and in four cycles using individual searches. The FPGA technology mapping problem can be reduced to pattern matching, for which CAM is an ideal tool. Technology mapping translates an input circuit of Boolean logic gates onto a specific implementation technology that typically includes LUT’s, as well as more sophisticated hardware structures. Occurrences of circuit patterns can be mapped onto target technology components.

Joshua M. Lucasa et. al. described a methodology for space-optimal combinational technology Lunar Replacement Unit (LURU), specifically designed for FPGA based CAM architectures. LURU, leverages the parallel search capabilities of CAM’s to outperform traditional mapping algorithms in both execution time and quality of results. This LURU algorithm is fundamentally different from other techniques for technology mapping in that it uses textual representations of circuit topology in order to efficiently store and search for circuit patterns in a CAM. A circuit is mapped to the target LUT technology using exact, inexact, or hybrid matching techniques. Common sub-circuit expressions (CSE’s) are also identified and used for architectural optimization [6].

Alistair A. McEwan and Jonathan S. describe the technique for implementing a CAM on FPGA as a pipelined packet detection algorithm which is highly parameterizable, allowing varying word widths, memory depths and operations to be implemented depending upon the requirements of the target application [7].

A System-On-Programmable-Chip (SOPC) Internet firewall has been implemented by John W. Lockwood et al. [35] that protects high-speed networks from present and future threats. In order to protect networks against current threats, the SOPC firewall parses Internet protocol headers, scans packet payloads, filters traffic, and performs per-flow
queuing. In order to protect against future threats, the SOPC is highly extensible. The present work also reports the CAM implementation for the firewall applications with a value addition of realization at the highest level of abstraction using Handel C. The following section covers the packet filters and the rules to be embedded in the context of firewall.

3.4 Embedding the Packet Filters and Rules

Packet classifying criteria are called as filters and in terms of firewall terminology these are called as rules [21]. A filter is formally an ordered pair of Boolean expression ‘F’ and ‘A’ the action to be taken if the expression is true. Each packet header contains the information relevant to classification. For instance, the relevant fields of an IPv4 packets could be the destination IP address (32 bits), the source IP address, the transport level protocol field (8 bits), the source port (16 bits), the destination port (16 bits), and the TCP flags (8 bits). The filter database, also known as access list, consists of a set of filters. Each filter requires the matching of a number of fields.

There are two administrative approaches for using the same for firewall:
1. Which is not expressly forbidden is permitted and that what is not expressly permitted is forbidden. If none of rules in the access list matches a packet, that packet is accepted.
2. Whereas in the second approach the packet is rejected.

The second approach is more conventional and secure, and so used by many existing firewalls as default. Further Molitor describes a general architecture for packet filtering [18]. He defines an access policy as a collection of lines of text, each line describing a certain class of data (for example, TCP packets from host A to host B) and reactions to take to such a data (e.g. drop it and log an access violation to host D). He also insists that the definition of data and the reaction should be left as open-ended as possible, so that a perfect architecture for implementing access policy would permit anything whatsoever to appear in those imaginary lines of text. He stresses more on flexibility than on the performance issues, giving the reason that performance is in general adequate, which unfortunately is not true anymore.
Based on the above conjectures the conceptualization of our packet filter implementation is portrayed in fig. 3.2

Table 3.1 Exemplifying partial set of rules for Firewall Configuration

<table>
<thead>
<tr>
<th>RULE</th>
<th>DIRECTION</th>
<th>TYPE</th>
<th>SRC_ADDR</th>
<th>DEST_ADDR</th>
<th>DEST_PORT</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OUT</td>
<td>TCP</td>
<td>*</td>
<td>128.245.9.29</td>
<td>ANY</td>
<td>DENY</td>
</tr>
<tr>
<td>2</td>
<td>IN</td>
<td>TCP</td>
<td>145.45.*</td>
<td>192.168.10.1</td>
<td>ANY</td>
<td>PERMIT</td>
</tr>
<tr>
<td>3</td>
<td>IN</td>
<td>ANY</td>
<td>220.78.*</td>
<td>ANY</td>
<td>ANY</td>
<td>DENY</td>
</tr>
<tr>
<td>4</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>DENY</td>
</tr>
</tbody>
</table>
3.5 Packet Classification schemes and algorithms

A packet filter operates properly if the rules are properly set. Each rule specifies some criterion based on packet headers. Researcher McKeown et. al. represent an alternative way of viewing the rule R is the set of all packets with headers that could match R [22]. This set is theoretic view of rule allow to define some terms that are useful in discussing the rules. Two distinct rules are said to be either partially overlapping, non-overlapping, or one is a subset of the other. In many access lists, not all rules are disjoint. In such a case, the filters are prioritized according to their ordering or in some other ways, and the highest priority rule is said to have matched a packet. Good number of researchers published different articles on packet filter dependency factors on suitability of an algorithm such as the number of rules, number of dimensions, the target environment (e.g. hardware or software), expected volume of traffic, etc. Therefore, an algorithm developed for a particular situation may not be the best choice for another situation. Many research worked in the past have focused on the single-field packet classification, used by many routers for packet forwarding [23, 24, 25, 26, and 27].

While it is difficult or impossible to examine the algorithms and approaches used by many of the commercial implementations of packet filters, there are some algorithms and their implementations that are published in the literature, which can be used to explore packet filtering techniques. Not all of those algorithms were designed or optimized for hardware implementation. Some of the most popular packet classification algorithms are listed below with short descriptions:

? **Sequential matching:** For each arriving packet, each rule is evaluated sequentially until a rule is found that matches all the packet headers that are of concern. Despite its simplicity and efficient use of memory, it has poor scaling properties since the time to perform classification grows linearly with the number of rules.

? **Hardware algorithms using ternary CAMs:** Ternary CAMs (content addressable memories) can store words with three-valued-digits: ‘0’, ‘1’ and ‘*’ (wildcard). Each rule is represented by one or more entries in the CAM. Since the CAM does not support some operators such as greater than or less than, some rules need to be broken down
into multiple entries in the CAM to be represented by the available three-digit based patterns. The CAMs also have built in priority mechanism so that the entries with lower address values have higher priority in the case of multiple matches. The comparison is performed in parallel, which makes it an attractive alternative to the linear search algorithm.

? **Grid of tries**: [28] proposes a scheme in which tried data structure is extended to two fields. This provides a good solution if there are only two fields, however, for more than two fields, it does not extend very well. They also describe another more general solution called ‘Crossproducting’, which can handle more than two fields. According to their experiment, it uses 1.5 MB of memory for 50 rules. For more rules, they suggest a form of caching technique.

? **Bit-parallelism in hardware**: [29] describes a scheme optimized for hardware implementation, which employs bit-level parallelism to match multiple fields concurrently. The authors implemented five dimensional packet classification in a high speed router prototype using a FPGA device and five 1M-bit SRAMs. They achieved a classification rate of 1 million packets per second (in the worst case) for 512 rules. Two variations of the algorithms have been suggested, one is space efficient while other is time efficient.

? **RFC classification**: [30] points out that in reality the packet filtering rule sets contain a considerable structure and redundancy that can be exploited by the classification algorithms. They estimate that a multi-stage algorithm, called RFC (recursive flow classification) can be used to classify 30 million packets per second in pipelined hardware, or one million packets per second in software.

With the background of the packet filter in place we now describe our implementation related to the packet filter.

**3.6 Packet Filter Implementation**

Although the design of a packet filtering system is primarily influenced by its use, more flexibility and efficiency are almost always desired. Efficient use of available resources and maintainability of the system are of high priority in implementation of all packet filtering systems.
These issues are widely covered in the literature [19, 20, and 21]. Some of the selected ones are shown in fig. 3.3.

It is difficult to meet all above mentioned packet filtering system; existing packet filter can’t meet all the requirements. Some packet filters might focus more on some of the above aspects while ignoring other aspects that they consider less important or are too difficult to implement. With these issues covered we now give our implementation of the packet filter in Handel C.
Fig. 3.4: Flow Diagram for logic Implementation
3.7 Handel C Implementation of Packet Filter

Content Addressable Memory (CAM) is the heart of the present implementation. It concurrently compares lookup table matches with incoming packet fields like IP and Ports etc. from the attacker. Fig. 3.4 reveals the logic through the flowchart of our implementation, while fig. 3.5 portrays topological placement of the Handel C modules developed by us.

![Flowchart of Handel C Modules]

**Fig. 3.5: Topology of Handel C Modules**

The related details in this context are presented in the following subsections.
3.7.1 Initialization of PAL console and Ethernet

The Platform Abstraction Layer (PAL) is a component of the Platform Developer’s Kit. PAL is an Application Programming Interface (API) for peripherals. The API offers a standard interface to hardware, enabling users to write portable Handel-C applications that can run on different FPGA/PLD boards without modification. PAL is implemented as a thin layer on top of a Platform Support Library (PSL) [31]. Celoxica PAL not only include the application functionality, but also the code necessary to interface to required all of the external devices such as Ethernet, keyboard, LED etc. By providing an OS type environment for Handel-C FPGA/PLD designs, the user saves time by focusing on adding value to the design rather than detailed hardware interfacing issues [32].

*************************************************************************
Pseudo Code for initialization of PAL consol API and Ethernet MAC
*************************************************************************

#define PAL_TARGET_CLOCK_RATE
PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pal_console.hch"
#include "pal_keyboard.hch"
#include "main.hch"

/* * Main program* */
PalConsole *ConsolePtr;
PalKeyboard *KeyboardPtr;
void main (void)
{
  PalVersionRequire (1, 0);
  PalVideoOutRequire (1);
  PalPS2PortRequire (2);
  //ethernet
Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory

PalEthernetRequire (1);
/*  * Run  */
par
{
  PalConsoleRun (&ConsolePtr, PAL_CONSOLE_FONT_NORMAL,
  PalVideoOutOptimalCT (ClockRate), ClockRate);
  PalKeyboardRun (&KeyboardPtr, PalPS2PortCT (1), ClockRate);
  PalEthernetRun (Ethernet, 0x12345678dead, ClockRate);
seq
{
  par
  {
    PalConsoleEnable (ConsolePtr);
    PalKeyboardEnable (KeyboardPtr);
    PalEthernetEnable (Ethernet);
  }
  RunFirewall(ConsolePtr, KeyboardPtr);
}  }}

*************************************************************************

3.7.2 Implementation of EDIT IP/Port rule and Packet Capture with RUN firewall module

The functionalities shown by 1 and 2 in figure 3.5 are implemented concurrently by using the ‘par’ construct of the Handel C. The parameters viz. console pointer and Ethernet are passed to the function “ReadPacket” for reading the entire packet attributes. Out of this the ‘header’ is passed to the “PacketAnalyzer” function for analysis. The above said packet analyzer is implemented using the RCAM. After comparison the packet is dropped in case the match is found. This main program loops endlessly for all the incoming packets. The forever loop implemented through while(1) construct.
Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory

Source code for Implementation of EDIT IP/Port rule and Packet Capture with RUN firewall module

static macro proc RunFirewall (ConsolePtr, KeyboardPtr)
{
    static rom unsigned char EDIT[128] =   "\n"
    "-----------------------------\n"   " Edit Rule\n"
    "-----------------------------\n"
    static rom unsigned char MAINMANU[255] =   "\n"
    "================================\n"   " Intelligent Firewall Menu\n"
    "================================\n"
    " 1 = For EDIT RULE\n":
    "2 = For Run Firewall\n":
    static rom unsigned char Run[128] =   "\n"
    "-----------------------------\n"   " Firewall Running \n"
    "-----------------------------\n"
    unsigned Char;
    /*
     * Dump characters from the keyboard to the console */
    PalConsolePutString(ConsolePtr, MAINMANU);
    PalKeyboardReadASCII (KeyboardPtr, &Char);
    PalConsolePutChar (ConsolePtr, Char);
    do
    {   if(Char=='1')
        {
            PalConsolePutString(ConsolePtr,EDIT);
        }
        ...
Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory

else
{
    ReadPacket (ConsolePtr, Ethernet);
    CAMFilter(ConsolePtr, Ethernet);
    PalConsolePutString(ConsolePtr,Run);
}
} while (1);

*************************************************************************

Fig. 3.6: window Packet Capture module capturing the inbound packets
Chapter 3: **Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory**

---

**Fig. 3.7:** window firewall in edit mode to modify the rule base

**Fig. 3.8:** window firewall in running mode

16
3.8 Handel C implementation of Packet Extractor

The following module of the Handel – C extracts the 32 bit source IP from the frame which is stored in buffer by the previous module. The mechanism used is as follows:
The TCP/IP header is extracted from the frame by the module described in the previous section. Out of this module the source IP starts from the 12th byte and ends at 15th byte. In order to obtain this 32 bit IP address, a shift by 8 bit and add technique has been employed for four times.

*****************************************************************************************

Source Code for Packet Extractor:
*****************************************************************************************

/* Extract The Source IP, Destination IP, and Port number from Packet Buffer */
macro proc PacketExtractor(value)
{
    unsigned i;
    SrcIP[0]=value[12];
    SrcIP[1]=value[13];
    SrcIP[2]=value[14];
    SrcIP[3]=value[15];
    DstIP[0]=value[16];
    DstIP[1]=value[17];
    DstIP[2]=value[18];
    DstIP[3]=value[19];
    Source=SrclP[0]@SrcIP[1]@SrcIP[2]@SrcIP[3];
    Destination=DstIP[0]@DstIP[1]@DstIP[2]@DstIP[3];
    PalConsolePutString(ConsolePtr,SRCIP);
    DisplayIP (ConsolePtr, Source);
    PalConsolePutString(ConsolePtr,DSTIP);
    DisplayIP (ConsolePtr, Destination);

    Ver=value[0];
    if(Ver==0x45)
    {
        printf("packet is Ethernet base having version IPV4\n");
    

Chapter 3: Reconfigurable Firewall Based on FPGA-Based Parameterized Content Addressable Memory

//packet is ethernet
} else
{
 printf("packet is not ethernet base\n");
}
}

macro proc CAMFilter(ConsolePtr, Ethernet)
{
 unsigned 1 HIT;
 unsigned 8 i;
 CAM.ReadWrite[i] = 0xc0;
 par (i=0;i<255;i++)
{
    if(SrcIP[0]==CAM.Read[i])
    {
       HIT=1;
       printf("MAtch Found\n");
    }
    else
    {
       HIT=0;
       printf("MAtch Not Found\n");
    }
}    }

macro proc DisplayIP (ConsolePtr, IPAddress)
{
 unsigned 4 Count;
 unsigned 32 Temp;
 unsigned 8 ASCII;
 par
{    Count = 0;
    Temp = IPAddress;
}

}
{     DisplayByte (ConsolePtr, Temp[31:24]);
     par
     {         PalConsolePutChar (ConsolePtr, ',
         Count++;
         Temp <<= 8;
     } }
while (Count != 4);
} void DisplayByte (PalConsole *ConsolePtr, unsigned 8 Byte)
{
    if ((Byte \ 4) < 10)
    {
        PalConsolePutChar (ConsolePtr, (0 @ (Byte \ 4)) + '0');
    }
    else
    {
        PalConsolePutChar (ConsolePtr, (0 @ (Byte \ 4)) + 87);
    }
    if ((Byte <= 4) < 10)
    {
        PalConsolePutChar (ConsolePtr, (0 @ (Byte <= 4)) + '0');
    }
    else
    {
        PalConsolePutChar (ConsolePtr, (0 @ (Byte <= 4)) + 87);
    }
}
3.9 Handel C implementation of Packet frame write module

This module is implemented to write the packet frame in the output buffer and console output.

*************************************************************************

Source code for writing Ethernet frame on console output
*************************************************************************

macro proc WriteFrame (Ethernet)
{
    unsigned 11 index;
    unsigned 48 Destination;
    Destination= 0x111122223333;
    PalEthernetEnable(Ethernet);
    do {
        PalEthernetWriteBegin(Ethernet, Destination, Type, DataByteCount, &Error);
    } while (Error == 0);
    index = 0;
    do {
        PalEthernetWrite (Ethernet, Data, &Error);
        index++;
    } while (Error == 1);
    do {
        PalEthernetWriteEnd(Ethernet, &Error);
    } while (Error == 0); }

*************************************************************************
Chapter 3: RECONFIGURABLE FIREWALL BASED ON FPGA-BASED PARAMETERIZED CONTENT ADDRESSABLE MEMORY

3.10 CAM Implementation Results from Device Utilization point of View

The higher level schematic of the CAM is as shown in figure 3.9. The structural details are as follows:

- The input to the system is the 32 bit IP address
- The search word is broadcast on the 8 bit search lines maintaining the least and most significant byte sequences.
- In case of hit the ‘Match’ signal is asserted to logic ‘1’ indicating attack signal.
- The table size for ‘IP look up is 256 configured in a semi-automatic mode.
Chapter 3: Reconfigurable Firewall Based on FPGA-Based Parameterized Content Addressable Memory

3.10.1 Utilization summaries

The prototyping of the 32*256 CAM was done on the Xilinx Spartan 3e FPGA after converting the Handel C code into VHDL through Xilinx Webpack Version 12.4. The results pertaining to the device utilization are given below:

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Used</th>
<th>Available</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>100</td>
<td>9,312</td>
<td>1</td>
</tr>
<tr>
<td>Slice Flip Flops</td>
<td>171</td>
<td>9312</td>
<td>1</td>
</tr>
<tr>
<td>occupied Slices</td>
<td>158</td>
<td>4,656</td>
<td>3</td>
</tr>
<tr>
<td>Slices containing only related logic</td>
<td>158</td>
<td>158</td>
<td>100%</td>
</tr>
<tr>
<td>Slices containing unrelated logic</td>
<td>0</td>
<td>158</td>
<td>0</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>184</td>
<td>9312</td>
<td>1</td>
</tr>
<tr>
<td>bonded IOBs</td>
<td>68</td>
<td>232</td>
<td>29</td>
</tr>
<tr>
<td>BUFGMUXs:</td>
<td>1</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>IOs</td>
<td>4008</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>BRAMs</td>
<td>4</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

CAM Device Utilization report 32*256 bit CAM

- Target Device: xc3s500e
- Target Package: fg320
- Target Speed: -4
3.10.2 Synthesis View

Figure 3.10 reveals the top level view of the RTL Schematic of the CAM.

Fig. 3.10: Top level RTL Schematic of CAM

Fig. 3.11: Higher level Schematic of CAM
Chapter 3: Reconﬁgurable Firewall Based on FPGA-based Parameterized Content Addressable Memory

Fig. 3.12: Schematic of CAM

Fig. 3.13: Higher level Schematic of CAM
Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory

3.11 Conclusions

This chapter has presented design and development of a CAM based Packet filter on FPGA to filter the Black and white IP addresses as well as port numbers as a parallel pattern matching circuits. The implementation exploits the powerful match capability of the CAM with an inherent parallelism mode in which the entire memory array is searched in one clock cycle to generate a ‘deny/accept’ signal indicating a black/white listed IP responsible for attack.

However, an additional advantage in using the CAM for the firewall is the only requirement of getting the match signal unlike in other applications, where it is enviable to identify exactly where in the memory address space this data was located. The FPGA based CAM implementation accomplishes the match and detect operation in the same time irrespective of the number of search items thus provides a better real time analysis and throughout equivalent to the line speed.
Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory

References:


Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory


35. John W. Lockwood, Christopher Neely, Christopher Zuver, James Moscola, Sarang Dharmapurikar, and David Lim, An Extensible, System-On-Programmable-Chip, Content-Aware Internet Firewall, Field-Programmable Custom Computing
Chapter 3: Reconfigurable Firewall Based on FPGA-based Parameterized Content Addressable Memory