A Survey on the Application of FPGAs for Network Infrastructure Security

Hao Chen,Yu Chen*, Douglas H. Summerville

***Abstract*—Given the rapid evolution of attack methods and toolkits, software-based solutions to secure the network infrastructure have become overburdened. The performance gap between the execution speed of security software and the amount of data to be processed is ever widening. A common solution to close this performance gap is through hardware implementation of security functions. Possessing the flexibility of software and high parallelism of hardware, reconfigurable hardware devices, such as Field Programmable Gate Arrays (FPGAs), have become increasingly popular for this purpose. FPGAs support the performance demands of security operations as well as enable architectural and algorithm innovations in the future. This paper presents a survey of the state-of-art in FPGA-based implementations that have been used in the network infrastructure security area, categorizing currently existing diverse implementations. Combining brief descriptions with intensive case-studies, we hope this survey will inspire more active research in this area.**

***Index Terms*—Network Security, Network Infrastructure Security, Hardware-based Application, FPGA.**

I. Introduction

THE growth and success of the Internet has made it fertile ground for malicious attackers and abusers. The increase in both the number and sophistication of attacks against the network infrastructure necessitates more robust security solutions. Despite the high level of knowledge reached in the theory of network infrastructure security, successful application has been inhibited by the inability to practically implement many of these security measures [18].

The current Internet looks much different from its early appearance. The Internet has evolved from the widespread interconnection of separate physical networks. The original design focused on efficient data transmission in an environment where network resources were precious and user groups were limited to scientists and engineers with implicit trust in one another. Hence, security was not an important design requirement, especially to the protection of the network infrastructure. After decades of evolutionary development, the Internet has become a polluted place, infested with viruses and malware [48]. Today, millions of computers are connected in a complex global web supporting a wide-range of consumer, corporate and government users having expectations of a reliable communication infrastructure. However, the main infrastructure of the Internet is based on the same end-to-end paradigm as when it was initiated in the 1970's [10].

Fortunately, the research community has never stopped their efforts to reinforce the security of the network infrastructure. One of the most important achievements is the development of various Network Intrusion Detection Systems (NIDS). Unlike a network firewall, which selectively blocks all outside traffic to prevent intrusions, a NIDS is able to evaluate traffic to discern suspected intrusions from normal traffic. There are several variations of NIDS, such as Network Intrusion Prevention System (NIPS), Host-based Intrusion Detection System (HIDS), and Protocol-based Intrusion Detection System (PIDS) [90]. SNORT [89], one of the most widely used NIDS, is an example of a software-based, light-weight NIDS.

Despite the great progress that has been made, there is an ever-widening performance gap between the processing requirements of NIDS and their software implementations. The increasing number and sophistication of attacks, the performance limitations of sequential software execution and the increase in network throughput all contribute to the widening of this gap. Under these conditions, it is natural to consider the use of hardware implementations for securing the network infrastructure. The major motivation for shifting from software to hardware is to enable real-time implementation of sophisticated security functions.

At the same time, it is equally important to maintain the system flexibility provided by general purpose computing power. Typical hardware implementations are usually dedicated for a few specific accelerations and in many cases still require the assistance of software for proper operation. Since security threats are constantly evolving, defense systems also require both static and dynamic update mechanisms. Thus, reconfiguration is as important to a hardware implementation as programmability is to software. Indeed, the design boundary between hardware and software is illusory at this point [48]. Field Programmable Gate Array (FPGA) devices have commonly been proposed because they feature both the flexibility of software and the high performance of hardware [41]. The FPGA is a suitable and popular hardware platform for many network security applications, including protocol wrapper [14], packet classification [98], and intrusion detection [13, 63, 104]. The emergence of the NetFPGA platform [66] is a good example showing the high demand for incorporating FPGAs into network implementations.

In addition to performance, hardware involvement also brings opportunities for architectural or algorithm innovations. The fundamental performance difference between hard-

Manuscript received 10 October 2008; revised 30 April 2009, 23 November 2009, 6 March 2010, 22 June 2010.

Hao Chen is with Dept. of Electrical & Computer Eng., SUNY-Binghamton, Binghamton, NY 13902 (e-mail: hchen8@binghamton.edu).

Yu Chen is with Dept. of Electrical & Computer Eng., SUNY-Binghamton, Binghamton, NY 13902 (e-mail: ychen@binghamton.edu).

Douglas H. Summerville is with Dept. of Electrical & Computer Eng., SUNY-Binghamton, Binghamton, NY 13902 (e-mail: dsummer@binghamton.edu).

ware and software solutions lies in their dissimilar execution paradigms. Essentially, hardware involved implementation allows executing many operations in parallel, while the execution of pure software implementation is serial or with limited parallelism [33]. In the hardware design area, high efficiency operation execution is still an active research topic.

Applying reconfigurable hardware for network infrastructure security has been a hot topic in the network security community [12, 34, 69]. Much research effort has been devoted and many achievements have been reported. However, to the best of our knowledge there has not yet been an attempt to systematically document these efforts. The objective of this paper is to present a clear and concise overview of state-of-the-art in the use of reconfigurable hardware approaches in network infrastructure security, from individual components to systems.

The remainder of this paper is organized as follows. Section 2 provides background in network infrastructure security. A detailed survey of reconfigurable hardware-based security applications is presented in sections 3 through 6. As the fundamental components for traffic monitoring, applications on packet header classification and pattern matching are first investigated in sections 3 and 4, respectively. Since more than 85% of actual network traffic is TCP/IP protocol based [70, 94], applications on TCP stream processing are included in section 5. System applications focusing on DDoS (Distributed Denial-of-Service) attack detection and Internet worm containment are presented in section 6. After a brief discussion of certain open questions and major challenges of hardware-based security applications in section 7, we conclude our paper in section 8.

II. Background

A. Review of Current Security Situation

Network security is a critical issue for the application of new technologies in all areas of society and the economy. It is particularly important for e-transactions, where it is a prerequisite for instilling confidence in users [18]. As evidenced by the declining trend financial losses, financial and intellectual investments in network security have begun to pay off [88]. However, current and future threats to network security are still severe and should be taken seriously, which motivates continuing this investment [50].

According to a Computer Security Institute (CSI) survey, the average annual cost due to cyber-crime doubled in 2007 from the previous year. After five years of continuous decline in average estimated losses due to cyber-crime since 2001, the tide turned. Companies reported average annual losses of $350,424 in 2007, up sharply from the $168,000 they reported in 2006 [88]. However, the average loss per respondent dramatically reduced from $3,149 in 2001 to $345 in 2007 [88]. This set of data reflects both the severe situation of network security, as well as people's achievement in this battle.

Targeted attacks have become a trend in network security. A targeted attack is a malware attack aimed exclusively at an organization or organizations within a sector or market [88]. Around 20% of the respondents of the CSI survey suffered this kind of security incident. Such narrowly targeted attacks are becoming more popular than ever [112].

As one type of notorious target attacks, Denial-of-Service (DoS) attacks continue to threaten network security. Since 2000, DoS attacks have grown rapidly and have been one of the major threats to the availability and reliability of network-based services. This type of attack was listed as causing the second highest total cyber crime cost in 2004. Although the percentage of losses caused by DoS attacks has been reduced in recent years, the total losses are increasing. This is due to financial losses incurred for every minute that a site is down [88].

Distributed DoS (DDoS) attacks, evolving from DoS attacks, can cause even more significant damage. Through the exploration of asymmetry between powerful Botnets [32] and individual machines, the attacker compromises multiple machines and recruits them into a zombie army, subsequently and indirectly launching an attack towards a specific victim from these zombies [84]. The DDoS attacks against Yahoo!, eBay, Amazon.com and other popular websites in February 2000 revealed the vulnerability of even very well equipped networks [21].

In addition, the trend of malicious threats has been moving towards massively distributed Internet worms and spyware. When combined with DoS attacks, the financial damage can be profoundly high. For example, CodeRed, a well known worm that included a built-in DoS attack payload, infected more than 250,000 systems in just 9 hours on July 19, 2001 [19]; and the numerous varieties of the MyDoom worm carried time-triggered DoS attack programs as their payload, causing devastating results in 2004 [43].

B. Network Infrastructure Security

Securing the network infrastructure has become a high priority due to its underlying effects for data protection, e-commerce and even national security [44]. A reliable network should feature at least two levels of security: information security and infrastructure security. Information security is based on information theory, and primarily focuses on data protection using techniques such as authentication and encryption [20]. Although important, this topic is out of the scope of our paper. Infrastructure security, on the other hand, focuses on the protection of network resources that support information sharing [1]. Its importance and urgency have gradually come to be considered equal to information security with the emergence and growth of threats targeting the network infrastructure. In fact, infrastructure and information security are complementary; a healthy network should be secured in both dimensions for reliable operation.

From the perspective of our study, all pertinent parts connected to the network could be considered as infrastructure components, including: DNS servers, routers, buffers and end-hosts. Since these are the important components for information gathering, exchange and distribution in the network they are the most valuable attack objects. Similar to the formidable challenge of securing all possible points of entry for attacks against a nation, it is neither practical nor necessary to respond

everywhere in the network infrastructure [52]. As long as the pertinent components are secured, the entire network infrastructure can be considered as secured. In short, a meaningful network infrastructure security policy maintains the health of critical components inside the network, preventing them from being attacked, thereby, maintaining the availability, reliability and stability of network services [23].

Few security issues were taken into account and standardized as protocols in the original design of the Internet. This intrinsic deficiency has enabled many opportunities for attack. Other sources of vulnerability in the network infrastructure include implementation deficiencies, misconfiguration, and selection of topology and protocol. Though the situation has improved due to the emergence of security protocols such as IPSec (IP security) [54], it takes time for these to become widely adopted. In addition, the exponential growth of the Internet has resulted in a heterogeneous network infrastructure that may cause slow adoption of protocol solutions. As a result, security strategies are overdue to protect the network infrastructure. In order to provide quality-of-service to its users, a robust network infrastructure is imperative.

Many infrastructure security solutions are based on network traffic analysis [3, 101]. Therefore, any useful application for traffic analysis could be considered as a security application in the context of network infrastructure security. TCP flow processing, packet classification, pattern matching, shrew attack defense and worm containment represent the major application categories focusing on specific parts of network infrastructure security [82, 86]. Network Intrusion Detection Systems (NIDS) and Network Intrusion Prevention Systems (NIPS), on the other hand, focus on the security of the entire infrastructure [114].

C. Requirements for a Successful Security Application

After surveying current security applications related to network infrastructure, we observe that a successful solution usually bears following characteristics:

- *Real-Time Protection*. It is essential for an effective protection mechanism to process data at line-speed with affordable cost. All traffic is subjected for inspection in a timely manner, and alerts are generated accurately when abnormal situations occur.
- *Flexible Updating*. Constantly evolving malicious attacks require security solutions to be adaptive to retain effectiveness. The update could be of the knowledge databases (signatures) that the security analysis depends on, a new solution for resolving, or even the system itself. Updating an application will often be more practical than replacing it in practice.
- *Well Controlled Scalability*. Scalability is another critical concern for practical deployment. Many reported approaches work well on a small scale research network, but their performance deteriorates rapidly when deployed to practical scale networks, such as campus level networks or larger. The main reason for this is that system complexity usually increases at a much greater rate than the network size on which it operates.

D. Hardware-based Infrastructure Security

Hardware implementation of security solutions has become a trend as the gap between network data rates and off-the-shelf processor computing power continues to increase. With the rapid development of technology, hardware devices play increasingly important roles in network security. Among the reported research, most hardware implementations focus on the construction of platforms that perform raw data collection or analysis. These platforms may not provide complete security solutions, but they exhibit great potential [48].

The shift towards hardware-based implementation is motivated by two major issues that prevent purely software-based security applications from moving forward. First, the performance of purely software-based applications is usually inadequate for practical deployment. For example, the saturation analysis throughput of Snort [89] is only 137 Mbps [31] using the standard configuration. Other systems, like Bro [79] and WebSTAT [109], are not able to handle data rates higher than 100Mbps [92]. Second, processors have become overburdened by the bandwidth expansion of network connections. State-of-art processors such as Intel XEON Woodcrest (3.0 GHz) and Intel Itanium Montecito (1.6GHz) are unable to maintain high enough throughput while running multiple Regular Expression Engines based Snort IDS [74]. Thus, it is necessary to offload network applications to dedicated hardware [40] and free up the host processor [74].

In addition to powerful computing capability, hardware devices naturally support parallel execution of operations. Compared with mostly sequential execution of software-based implementations, this unique feature leads to more efficient data-parallelism and multi-stage processing, which is unbeatable by current software-based platforms.

In contrast to software implementations, application-oriented and highly parallel design paradigms make hardware implementations superior in terms of performance. For example, TCP Stream Reassembly and State Tracking, an Application Specific Integrated Circuit (ASIC) developed at Georgia Tech., could analyze a single TCP flow at 3.2Gbps in 2002 [76]. A FPGA-based TCP-processor developed by Open Network Laboratory (ONL) at Washington University was capable of monitoring 8 million bidirectional TCP flows at OC-48 (2.5Gbps) data rate in 2004 [92].

ASIC-based devices not only possess the advantage of high performance, achieved through circuit design dedicated to the task, but have the potential for low unit price. However, substantial cost relief from huge non-recurring engineering investment can only be achieved when ASIC devices achieve sufficiently high-volume production. Unfortunately, this may not be applicable to network security applications. Constant evolving standards and requirements make it unfeasible to fabricate ASIC-based network security applications at such a high volume [48]. Moreover, custom ASICs offer little or no reconfigurability, which could be another reason that ASICs have not been widely applied in the network security area.

Reconfigurability is an essential requirement for the success of hardware-based network security applications and the availability of reconfigurable hardware has enabled the

design of hardware-based security applications [34, 118]. A reconfigurable device could be considered as a hybrid hardware/software platform since reconfigurability is used to keep the design up to date [48]. FPGAs are the most representative reconfigurable hardware devices.

A Field-Programmable Gate Array (FPGA) is a type of general-purpose, multi-level programmable logic device that can be programmed by end users [107]. At the physical level, logic blocks and programmable interconnections compose the main structure of a FPGA. A logic block usually contains a 4-input look-up table (LUT) [15] and a flip flop for basic logic operations, while programmable interconnections between blocks allow users to implement multi-level logic. At the design level, a logic circuit diagram or a high level hardware description language (HDL) [73], such as VHDL or Verilog, is used for the programming that specifies how the chip should function. In the electronics industry it is vital to reach the market with new products in the shortest possible time and to reduce the financial risk of implementing new ideas. FPGAs were quickly adopted for the prototyping of new logic designs shortly after they were invented in the mid 1980s due to their unique feature of flexibility in hardware development [15].

While the performance and size of FPGAs limited their application in the early days, advancements in density and speed have resulted in narrowing the performance gap between FPGAs and ASICs enabling FPGAs not only to serve as fast prototyping tools but also to become primary components in embedded systems [113]. Current FPGAs share the performance advantage of ASICs because they can implement parallel logic functions in hardware. They also share some of the flexibility of embedded network processors in that they can be dynamically reconfigured [47].

Combining the performance advantages of ASICs with software-like re-programmability, current FPGA technologies enable new infrastructure security applications. With millions of computers now attached to the Internet, it is infeasible to rely on every user being diligent in keeping their security provisions up to date. A feasible way to maintain the efficiency and reliability of system security is to apply security applications at a much smaller number of aggregation points [48]. Featuring powerful functionality but compact size, FPGA-based security implementations are one of the most suitable candidates for this purpose. They can be deployed anywhere within the network infrastructure for considerable improvement in security.

In practice, most network infrastructure security applications follow the strategy of being deployed adjacent to routers and work as security reinforcement to them. In this way, the original configuration of network routers can be mostly preserved, reducing sharing of valuable resources and minimizing the negative impact to already heavily burdened routers. This strategy reduces the chance of malicious traffic touching routers. Also, the balance between intrusion analyzing and resolving can be maintained [21]. While it is not possible to detect and eradicate all malicious traffic at few aggregation points, the robustness of the system can usually be guaranteed at the cost of accepting some security risk.

III. Packet Classification

The network infrastructure faces serious security challenges due to the exponential growth of the Internet. In practice, it is infeasible to have an entire network infrastructure protected due to economic, social or political reasons. It is difficult to avoid the presence of malicious users inside the network, especially the Internet. Existing infrastructure security applications are usually focused on the protection of a specific network area, such as a campus-network or an enterprise-network. As a result, most network infrastructure security applications adopt a passive defense strategy. The basic rationale is *I cannot stop you, but I can prevent you*.

Timely detection of malicious attacks is one of the essential functions for network security applications. Without effective detection, subsequent countermeasures are useless. To take full advantage of the high parallelism supported by hardware devices, signature-based detection is the most fundamental and spontaneous strategy adopted. The signature can include source/destination address, port numbers, protocols being applied, and patterns of content or any combination of these. By comparing specific strings contained in incoming packets to known signatures the detection system is able to identify previously known malicious attacks.

For network infrastructure security, packet inspection is the first approach for malicious activity detection. A packet consists of two kinds of data: control data in the packet header and user data in the payload [56]. Packet header fields are essentially constant in length and appear at fixed locations in the packet, while packet payloads can be variable in length with no fixed format. Conventionally, packet classification focuses on the processing of packet headers, while deep packet inspection focuses on processing of the payload. A combination of the two can be applied for a complete packet matching approach [98].

Packet classification is an important technology for network infrastructure security. From early firewalls to recent high-performance routers and sophisticated Intrusion Detection systems (IDS), classification plays an indispensable role. Research in packet classification has achieved substantial progress in recent years. The reported research work in this area has been well summarized and categorized into four basic types by Taylor [105]. Therefore, instead of trying to repeat the big picture, this survey focuses more on recent emerging hardware based techniques.

A. Existing Applications

To the best of our knowledge, the first hardware implementation of packet classification was developed in 2002. Called a flow classifier [117], it is a module employing a hybrid hardware-software architecture performing straightforward operations. A 16-bit flow identity, which is generated through a hash calculation of a packet's five-tuple value, is assigned to each packet [103]. With these identities, individual packets are classified to different flows for further parallel flow monitoring. The design was implemented using a Xilinx VirtexII XC2V8000 FPGA board, and reported results indicate that it is sufficient for 10 Gbps traffic rates.

Song and Lockwood developed a hardware-based approach for network intrusion detection in 2005 [98]. Combining the advantages of Ternary Content Addressable Memory (TCAM) [102, 115] and the Bit Vector (BV) algorithm [5, 58], the BV-TCAM architecture was introduced. The approach is based on the observation that while TCAM structure is efficient for direct data-lookup with prefix or exact values, it is not suitable for those that fall within a range. The complementary Bit Vector algorithm, however, is well-suited for this task. In this manner, this architecture eliminates the requirement for prefix expansion or port range lookups. With a limited embedded TCAM, packet classification can be easily implemented in currently available FPGAs. This design was prototyped in a Xilinx XCV2000E based FPX platform [64]. The evaluation results show that this application is good enough to sustain at least OC48 traffic rate (2.5Gbps).

A more recent achievement in hardware-based packet classification was reported in 2007 [69]. A Gigabit packet filter was tailored for implementation on FPGAs. Instead of choosing other advanced search algorithms, the architecture adopts a linear-search based algorithm for packet classification. The powerful hardware allows the linear-search based algorithm to achieve high performance under multi-parallel operation mode. Moreover, a linear search is the best choice when using the internal memory blocks of FPGA for storage [69]. This pipelined packet-filter architecture was implemented on a Virtex-4 FX-12 FPGA and was demonstrated to filter network traffic at layers 2 and 3 with a throughput of 1 Gbps. Evaluated with a system frequency at 125 MHz, it only took 2,300 ns for a filter to make a decision of accepting or dropping a packet.

B. Technology Analysis

In general, packet classification refers to the operation of categorizing different packets into equivalent classes based on their header information [5]. These equivalent classes, known as flows, are grouped by certain rules that include matching the source/destination addresses or ports of the packets, or the protocols being applied to them. A corresponding action is associated with each rule to indicate the subsequent processing, such as forwarding, copying or dropping. The rules are stored in a database, one rule for each flow type. When a packet arrives, at least one matched rule should be found in the database, allowing further processing to be conducted. If more than one rule is matched by a packet, an arbiter makes a decision according to a predefined policy, usually the longest pattern match. While each of these steps of packet classification is an important technology concern, the most important is how to perform effective rule matching.

Intensive research on packet classification has been carried out and many algorithms and architectures have been developed for this purpose. Taylor's survey in 2004 [105] summarized the major packet classification techniques. He framed each technique as employing one or more of four high-level approaches: Exhaustive Search, Decision Tree, Decomposition, and Tuple Space. Our analysis follows the same categorization and is complementary to Taylor's work. While Taylor's survey focuses on the description of techniques, we focus on the analysis of hardware-based application of these techniques.

Exhaustive search is intuitively the most straightforward approach to find a matching rule by examining all the rules in database. The basic linear search looks for filter rules sequentially until the first rule matches. The matching priority of the search is usually implicit. It is easy to modify the linear search to run in parallel operation mode, and a fully parallelized search can be achieved by using a Content Addressable Memory (CAM) [42] or similar data-addressable memory-structure. Ternary Content Addressable Memory (TCAM) [115] is commonly used, as it can find a matching rule in constant time.

Although CAM-based approaches are popular and practical methods for the implementation of packet classification, they are limited by two major drawbacks. First, the hardware complexity results in storage inefficiency, high power consumption and limited scalability for long input-key searching. The second issue is the inefficient representation of filters with port ranges. The emergence of Extend-TCAM (ETCAM) [102] improves the performance; however, the cost of the E-TCAM approach doubles in terms of gate count [69].

Decision-tree based approaches require the pre-construction of a decision tree based on rule- databases, and then bit-keys obtained from packet header fields are used to traverse the tree from root to leaf. The searching time highly depends on the length of search key. The Bit Vector algorithm [58] is a classic tree-based search algorithm. More sophisticated tree-based algorithms introduce the concept of 'cut'. A 'cut' in multi-dimensional space is isomorphic to a branch in a decision tree [105]. These branch decisions in cutting algorithms are more complicated than single bit decision in a bit-vector, but the principles are the same. The BV-TCAM [85][98] approach is a hybrid of the Bit-Vector algorithm and TCAM structures. It combines the advantages of both for performance improvement. However, the search of a decision tree inherits a serial nature precluding it from fully parallel implementation.

Decomposition achieves performance improvement through the reduction of search complexity. Parallel operation can be performed by decomposing multiple-field searches into multiple instances of single-field searches. In this manner the total search time can be reduced. This approach leverages the common case in which a packet rarely matches multiples rules [46]. After searching, a mechanism is needed to collect the disjoint results for final matching evaluation. In general, decomposition based algorithms are known to have high memory requirements [69]. For instance, the Distributed Cross-producing of Field Labels (DCFL) approach provides higher throughput, but at the cost of exponential memory requirements. For a set of N filters containing d fields each, the size of the cross-product table could be $O(N^d)$ [106]. An improved bit-vector scheme called Aggregated Bit Vector (ABV) was developed from this idea. By searching a constant number of memory words in each field instead of examining all the leaves, it reduces memory access from N to ${log_A}^N$ [5].

Unlike decision-tree based approaches that perform the rule-search by exploring the relationships among rules, tuple-space based approaches can quickly narrow down the scope of a

multiple field search by partitioning the rule-set into similar rules.

A tuple defines a series of specified bit numbers in each rule field [105]. It can be considered as a hash-like value to check whether a packet matches specific rules [69]. The motivation for using tuples arises from the fact that the number of distinct tuples is much less than the number of rules in the rule set [103]. The more similar the rules are the more they can be grouped into tuples. After scope-narrowing, the complexity of the search is reduced. Since the number of tuples depends on the number of characters, the search time is predictable and usually short.

While each technique has its own set of suitable applications, the brute-force linear search is the best for hardware implementation mainly because of its compatible features with hardware devices. Brute-force based linear search algorithms are usually considered inefficient and as a result are cautiously used in software-based applications. However, the simple search mechanism allows for simpler implementation while high parallelism enables optimization in hardware-based applications. Modern high-performance hardware devices provide powerful functionality that can greatly accelerate processing speed. In addition, parallel processing can be realized without much overhead.

Linear search employs common RAM architectures for sequential rule database search. Since incorporation of RAM-based memory is relatively common within FPGAs, efficient utilization can be achieved. On the contrary, CAM-based approaches are able to achieve highly parallel search but the CAM itself is a complex structure that is more difficult to efficiently implement on a hardware device like a FPGA. The decision tree and decomposition techniques require memory structures to store and access the decision tree efficiently. So far, efficiently implementing decision tree in hardware is still an open problem [69].

Packet inspection consists of packet classification using header information and deep packet inspection using payload trunk. Indeed, many search or match techniques are orthogonal to each other.

IV. Pattern Matching Application

With the emergence of application level network attacks, inspection of packet headers alone is no longer sufficient [27]. Deep packet inspection has been introduced to reinforce network security and is dedicated to the task of payload inspection. The inspection of packet headers is relatively easy due to the explicit format specified by protocols. However, the inspection of payload contents is more challenging since the payload contents can be any format that is determined by the applications. Therefore, it is more complex and expensive to perform deep packet inspection at payload level.

Pattern matching is the most popular approach for payload inspection [62]. It is one of the fundamental functions for anomaly detection and has been widely applied for security purposes in the network infrastructure, including Firewalls and Intrusion Detection Systems (IDS). The major task of pattern matching is straightforward; incoming packets are compared with a large number of patterns (or signatures), with subsequent processing based on the matching pattern [30]. In order to conduct comparison operations efficiently, it is essential to update the pattern database frequently, even dynamically if possible, since the security environment constantly evolves.

The performance of pattern matching is often described by two metrics: throughput and scalability [51]. Indeed, the speed of pattern searching determines the acceptable processing throughput, since all meaningful operations should be performed in real time [6]. Scalability evaluates how well a design fits to a real implementation. In practice, available resources are always limited. A design with higher processing throughput, but at the cost of significantly increased resource utilization, is considered to possess poor scalability. In terms of hardware pattern matching applications, better scalability implies that more patterns could be accommodated for a given amount of memory while the Quality-of-Service (QoS) of pattern matching operations is maintained. The speed of pattern matching is improved by accommodating more patterns on chip, which results in improved overall performance.

The introduction of reconfigurable hardware, such as FPGA devices, enables great strides for pattern matching applications providing powerful processing capability that software-based solutions cannot match. Techniques specialized for hardware implementation of pattern matching have been developed [12, 28, 63, 77]. In the following sub-sections, four typical hardware-based pattern matching techniques will be discussed: Finite Automata (FA), Content Addressable Memory (CAM), Bloom-filter, and min-cut partition.

A. Finite Automata (FA) Technique

Finite Automata (FA) describes a class of models of computation that are characterized as having a finite number of states [59]. This concept has been widely applied and is prevalent in the digital logic design area. As it applies to pattern matching, the result of a FA processed input string is either accepted or rejected. A successful matching occurs when the string of input characters match the labeled patterns, or, more precisely, the regular expressions, on any path of the FA that leads from the initial state to the final state. Currently, there are two types of FA approaches reported for hardware implementation: Non-deterministic Finite Automata (NFA) and Deterministic Finite Automata (DFA) approaches.

1) Existing Applications: In recent years, several Non-deterministic Finite Automata (NFA) implementations have been reported. Sidhu and Prasanna [95] mapped the NFA logic for regular expression onto a Xilinx Virtex FPGA and the Self-Reconfigurable Gate Array (SRGA) to perform fast pattern matching in 2001. The proposed approach takes $O(n+m)$ time and $O(n^2)$ space to find matches to a regular expression of length n in text of length m. Using this method, Hutchings and Franklin compiled patterns of the open-source NIDS system Snort using JHDL [11], a Java-based design tool, and then converted them to a FPGA bit-stream for implementation [49]. The result shows that the FPGA-based string matcher exceeds the performance of the software-based system by a factor of 600 for large patterns.

Clark and Schimmel further developed the NFA generator with a similar approach in 2003 [29], but focused more on complex regular expressions. They reported that using their approach, the entire Snort rule database consisting of over 1,500 rules and 17,000 characters can be fit on a single one-million-gate FPGA board while keeping pattern matching at gigabit traffic rate. In 2004, they proposed a scalable pattern matching using multi-character decoder NFA technology [30]. This approach offers flexible trade-offs between character capacity, throughput, and data bus width and rate. A wide range of pattern set sizes can be covered by high-performance circuits. This approach enables current-generation FPGAs to match a large number of complex patterns at network data rates from 1Gbps to 100Gbps.

Compared to hardware application of the NFA approach, there are relatively few hardware Deterministic Finite Automata (DFA) implementations being reported. Moscola and Lockwood at Washington University translated regular expressions into DFA in 2003 [75]. The content scanner was implemented on their Field-programmable Port Extender (FPX) platform and tested using real Internet traffic on the Washington University Gigabit Switch (WUGS). The pattern matching can be guaranteed at speeds of 1.184 Gbps for twenty-one regular expressions each with 20 characters, and exceeding speeds of 2.5 Gigabits/second for smaller numbers of similar regular expressions.

In 2004, Bos and Huang [13] reported a DFA approach on the IXP1200 network processor with support for large rule sets. They employed the Aho-Corasick algorithm in a parallel fashion, where each micro-engine processes a subset of traffic for pattern matching. In order to allow for large patterns as well as a large number of rules, the patterns are stored in off-chip memory. The evaluation results show that the processing speed is only 200Mbps, though the system is capable of handling full content scan for realistic threats.

2) Technology Analysis: Although the specific implementations of Finite Automata (FA) are different, the main approaches are similar in that they build an efficient Finite State Machine (FSM) for pattern matching. Of course, NFA and DFA feature their own special properties.

A regular expression is a pattern that is used to match a set of strings according to certain syntax rules [97]. An FA can be constructed to recognize strings matching a regular expression. An FA can be expressed as a directed graph where nodes represent states and edges are labeled as characters. One state is designated as the initial state and the remaining states are intermediate or accepting (or final) states. When the last input symbol has been received it will report the matching status depending on whether the DFA/NFA is in an accepting state or not.

For each input symbol, the next state of the current DFA state is uniquely determined. No state in DFA has more than one outgoing edge with the same label [95]. Unlike DFA, for any input symbol the next state of NFA may not be uniquely determined. It could be any one of several possible states depending on its constituent sub-expressions. An extension of an NFA is an NFA-epsilon, which allows epsilon (ε) transitions that represent a transition to a new state without consuming any input symbols. Hence, the edge of an epsilon-NFA may be labeled with a single character or [95].

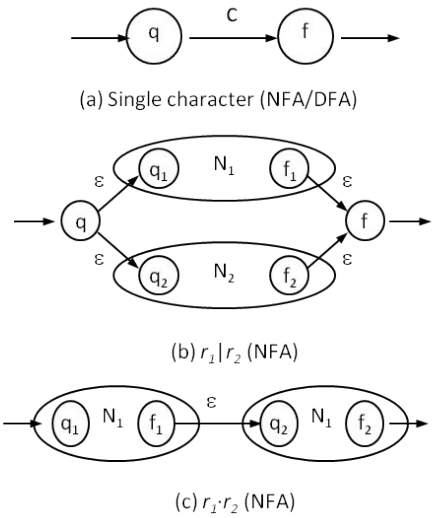

Fig. 1. NFA/DFA for regular expressions

Fig. 1, reproduced from [95], provides some logic structures for NFA/DFA models. Fig. 1(a) illustrates the simplest structure for single character matching. It has an initial state q and a final state f, and is applicable to both NFA and DFA. Fig. 1(b) and (c), however, are NFA exclusive structures for the computing of $r_1 \mid r_2$ and $r_1 \cdot r_2$, i.e.: to match either r_1 or r_2, or to match both of them, respectively. N_1 and N_2 are NFAs for regular expressions of r_1 and r_2. They each have an initial state q and final state f. If the next state cannot be determined immediately from current state, processing will register the current status and move to all next possible states for joint prediction until reaching the final state, otherwise the matching fails. The feature of joint prediction gives NFA higher throughput for pattern matching than DFA.

Although the functionality of NFA and DFA are transferable, they feature different hardware behaviors [95]. For FPGA implementation, their construction time and relative memory occupation are much different. The construction technique shown in [95] can dynamically generate an NFA for a length n regular expression in $O(n)$ time with $O(n^2)$ memory; while constructing an equivalent DFA requires significantly more time and memory area, both at $O(2^n)$. After construction, both NFA and DFA are able to process one character per cycle. Assuming a pattern matching operation between a length n regular expression and length m input data, a serial machine requires $O(mn)$ time to reach the goal and requires $O(n)$ memory occupation. It takes total $O(2n+m)$ time and $O(2^n)$ memory occupation for the equivalent DFA and $O(n+m)$ time and $O(n^2)$ memory occupation for the equivalent NFA to process.

From the perspective of hardware implementation, construction time causes great impact to the overall performance of FA. Though the time complexity of NFA looks moderate, it actually requires dynamic update, which may result in even heavier construction overhead. In practice, optimized

DFA construction techniques can be fast. However, for any DFA based matcher there will always be cases that cause exponential blowup in the time and memory required [95]. Since it is impossible to eliminate the configuration time of hardware, a feasible way is to minimize it. Within this context, the preferred approach may be to implement the FA construction algorithm as a program that outputs the FA logic as a placed and routed netlist and subsequently use vendor tools for generating configuration bits [95].

In terms of logic level design, there are two standard methods for FA implementation on FPGAs [95]. One is through the use of binary encoding for state storage. However, this method is not efficient for NFA implementation, since the memory can look up only one next state at every clock cycle. For NFA construction, the one-hot encoding (OHE) scheme offers a better solution. It allows NFA to simultaneously look up any possible next state. The key is that each state of the NFA is associated with one flip-flop and that state is uniquely indicated by an output of one from that flip-flop; thus, multiple states can be encoded by having multiple ones with no conflict.

In comparison, NFA possesses shorter processing time and can be implemented in smaller overall area than DFA, but at the cost of dynamic NFA update. If the size of the completed automaton is the most critical concern, the DFA approach contains up to $O(2^n)$ states, where n is the number of characters in the expression. However, the structure of DFA are simpler than NFA since all the states are explicit, but at the cost of higher memory consumption. This is more prominent in hardware applications when the size of patterns increases and the scalability issue dominates the overall performance of pattern matching applications. A good tradeoff between complexity and memory consumption is desired.

B. Content Addressable Memory (CAM) Technique

The most common hardware approach for pattern matching is regular expression based finite-automata (FA), either NFA or DFA, which results in designs with low cost but modest throughput [100]. Though the use of parallelism in FA implementations has been attempted, it is difficult in general. The implementations of FA are built with the implicit assumption that the input data are processed sequentially. Thus, the overall latency increases proportionally with the number of patterns. In addition, FA implementations are restricted in operating frequency by the complexity of combinational logic needed for state transitions, where complex expressions result in multilevel implementations [100]. In recent years, approaches based on Content Addressable Memory (CAM) have had renewed interest due to their unique ability to apply fast pattern comparison without memory addressing [45].

1) Existing Applications: To the best of our knowledge, Gokhale and Duois are the first who implemented Snort rules with a CAM in 2002 [42]. Their implementation achieved a throughput of 2.2 Gbps on a Xilinx Virtex XCV1000E device running at 68 MHz with 32-bit data each clock cycle.

A Ternary Content Addressable Memory (TCAM) based multiple-pattern matching approach was introduced by Yu and Katz in 2004 [115]. It features a don't care ('?') state in addition to the '0' and '1' states used in classical CAM approaches. The don't care state can be used as a mask, allowing one input to match multiple patterns simultaneously. Therefore, the TCAM approach is capable of handling complex patterns, such as arbitrarily long patterns, correlated patterns and patterns with negation. For the ClamAv [55] virus database with 1768 patterns whose sizes vary from 6 bytes to 2189 bytes, the proposed approach can operate at 2 Gbps with a 240 kB TCAM.

Sourdis and Pnevmatikatos proposed a CAM implemented using discrete comparators in 2003 [99]. They adopted a scalable, low-latency architecture with extensive fine-grain pipelining to tackle the fan-out, match and encode bottlenecks. Their design achieved a processing bandwidth of 11 Gbps with operating frequencies at 340 MHz for fast Virtex devices. To increase throughput, they applied multiple comparators for parallel matching of multiple search strings. The evaluation of timing and area reports presented show that the match cost per search pattern character is between 4 and 5 logic cells. For a lower cost, they improved their implementation using shared comparators and developed the Decoded CAM (DCAM) in 2004 [100]. The results showed the area cost per search pattern character is less than 1.1 logic cells and an operating frequency of about 375 MHz (3 Gbps) on a Virtex2 device. When using quad parallelism to increase the matching throughput, the area cost was further decreased to less than one cell with a throughput of 10Gbps.

Hardware-based approaches are able to deal with pattern matching under tremendously high speeds but have high resource requirements. As the number of patterns increases, sufficient storage resources are essential. A Binary Decision Diagram (BDD) based CAM method was introduced by Yusuf and Luk in 2005 [116] to solve the problem of size limitation while retaining the speed advantage of hardware. Their pattern-matching engine is based on the tree-based CAM structure. Exploiting logic optimizations for multiple strings in the form of BDD, the approach involves hardware sharing at the bit level [116]. This method has been used to implement the entire SNORT rule set with approximately 12% of the area on a Xilinx CC2V8000 FPGA. The design can run at a rate of about 2.5 Gbps and is about 30% smaller than another related approach [8]. The main obstacle in optimizing for speed is large fan-out, which implies long latency. This is a common obstacle for tree-based structures. Adopting an optimized multi-stage pipeline may be a good solution for mitigation.

2) Technology Analysis: The core part of pattern matching is performing fast comparison between input data and patterns. Since all the patterns are stored in memory devices, it is necessary to perform a memory search to obtain patterns for comparison. Traditional memory search techniques based on RAM technology require generating addresses to locate the patterns. As a result, the speed of addressing often becomes the limiting factor for memory search and further delays the comparison.

CAM is a class of parallel pattern matching circuits that is an outgrowth of RAM (Random Access Memory) technology [83]. In RAM circuits, an address is necessary to access the corresponding data. The number of address lines limits the

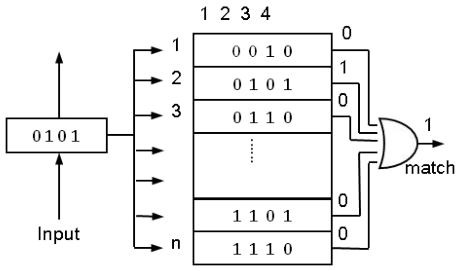

Fig. 2. Binary CAM operation in match mode with packet length =1

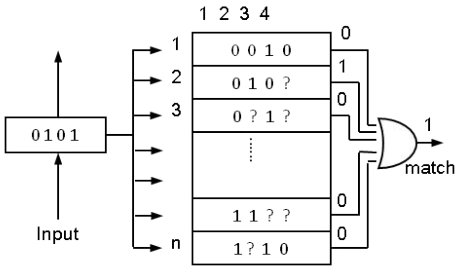

Fig. 3. TCAM operation in match mode with packet length =1

depth of a memory, but the width of the memory can be extended as far as desired if possible. CAM circuits can also work in dual mode. In one mode, the CAM operates like a standard RAM circuit where data may be written to and read from the device [111]. In the other mode, the CAM circuits have a powerful parallel match mode: the entire memory array can be searched in parallel using hardware that compares each stored entry to an input value. If a matching value is found in any of the memory locations, a match signal is generated. Since no address line is required in match mode, the depth of a memory system using CAM can be extended as far as desired. As a further extension, a CAM circuit can provide the addresses of matched patterns in match mode when necessary, which is the opposite functionality of a standard RAM.

Content Addressable Memory (CAM) technology is optimal for memory search without addressing since it can directly compare the input data against entire list of patterns being stored in the memory simultaneously. This unique feature offers CAM based approaches superior speed for fast comparison. It results in an order-of-magnitude reduction in search time, compared with other memory search methods such as binary or tree-based searches or look-aside tag buffers [83].

Fig. 2 illustrates the basic Binary CAM (BCAM) [42] operation for pattern matching based on FPGA implementation. As implied by its name, only '0' or '1' is stored in a BCAM. The width of CAM depends on how many LUTs are contained in one entry; it also determines the width of input data upon which the lookup is based. A typical FPGA logic block contains a 4-input lookup table (LUT), which means an individual LUT can handle at most 4-bit input data. For simplicity, each entry in Fig. 2 contains only one LUT, so the width of CAM is 4 bits. In Fig. 2, 4 bits of input data are fetched for matching during each processing cycle. CAM circuits compare the input with all the pattern entries in parallel and 'OR' the results for final match indication.

In general, a CAM requires nw bytes of memory space to store n entries, each w bytes wide. The simple scenario of an input packet of length kw can always be processed in deterministic time $O(k)$ if the pattern contained in each processing entry is independent [115]. However, in many practical applications this is not the case since a pattern may cross processing entries. In addition, several patterns may be correlated to one another. In that case, more sophisticated mechanisms must be applied. As shown in Fig. 2, shift registers may be applied for data holding to ensure that all possible patterns can be detected. Instead of simply passing bits four by four, shifting bits one by one will never miss a pattern spanning two processing entries.

Besides the basic BCAM approach, two other improved CAM approaches have been proposed: Ternary CAM (TCAM) [115] and Decoded CAM (DCAM) [99]. To perform exact matching with wildcards using a BCAM, the wildcards must be expanded to all possible matching patterns, and a BCAM entry is required for each. A more efficient method is to match only the interesting parts of the input data with patterns stored in each entry. The TCAM approach is dedicated for this purpose, as illustrated in Fig. 3. In addition to the '0' and '1' states, it has a don't-care state, represented as '?' in the figure. With the application of wildcards, the TCAM masks-off uninteresting bits at each entry for faster comparison.

Decoded CAM (DCAM) is a variety of basic CAM optimized for parallel matching. It was proposed in [99]. Unlike a basic BCAM, which employs a single large CAM to perform parallel comparison of all stored patterns, the DCAM is implemented with discrete 'comparators' for partitioned comparison. Patterns that would otherwise be stored in a large BCAM are first decomposed into elementary patterns according to certain rules at the character-level and are stored in smaller CAM circuits. Each comparator implements the functionality of such a small CAM. Comparisons are then performed between the input data and individual comparators in parallel. Finally, these intermediate results are processed by encoders for a matching decision. Taking advantage of pipelining and redundancy in patterns, the approach has the potential to increase the efficiency of the comparison operation dramatically. This approach was proposed in [99] and further developed in [100]; these enhancements are discussed in section 4.3.

In summary, CAM circuits feature regular structures and are suitable for pattern matching applications in hardware, especially for parallel implementation. The unique feature of searching the entire memory contents simultaneously makes this approach faster than any RAM based searching approach. However, this speed comes at the cost of larger resource consumption than other approaches. With respect to FPGA implementation, the construction of CAM circuits relies on flip-flops for data storage, so the size of the implemented circuits is restricted by the availability of flip-flops [45]. Thus, as pattern matching circuits become more and more complicated, scalability issues emerge.

C. Optimization for Scalability

Usually, scalability is not an issue for small or simple systems, but can be the dominant factor in large complex systems. Hardware based approaches can significantly improve the performance of pattern matching, but at the cost of increased resource consumption. Without good system scalability, limited hardware resources can be quickly exhausted. Moreover, any modification in FPGA implementations, especially the constant update of pattern databases, requires synthesizing, mapping, and programming to the target device, which can be time consuming. As hardware based pattern matching applications become increasingly sophisticated, scalability becomes a large concern. In recent years, several approaches have been dedicated to maintaining scalability in pattern matching. Here, we introduce two popular techniques: Bloom filter and min-cut partition.

1) Existing Applications: A basic structure using Bloom filter for packet payload inspection has been reported by a research team from Washington University in 2004 [36]. Basically, it consists of two stages: the first stage uses hardware Bloom filters to isolate all packets that potentially contain predefined signatures and the second stage eliminates false positives produced in the first. A prototype based on the FPX platform [64] has been built. Evaluation results show that this design can support up to 10,000 pattern matching operations at a line speed of OC-48 (2.4 Gbps). They have reported extensions of the design in [4] and [35], which provide detailed implementation. They further improved their work to handle arbitrarily large pattern matching operations at the cost of slightly more on-chip memory in 2006 [37]. The idea of handling arbitrary length patterns is to split up longer strings into multiple shorter fixed-length segments that can be handled by the basic Bloom filter.

Suresh and Guo proposed an automatic compilation framework in 2006 [104]. They used ROCC, a C to VHDL compiler, to generate a Bloom-filter based intrusion detection system on FPGAs. It is the first work that automatically generates VHDL for Bloom filter code written in C. Their synthesized hardware application can run at 73MHz and handle a throughput of 18.6 Gbps, while occupying approximately 8% of the area of a Xilinx XC2V8000 FPGA.

Researchers at University of Southern California have dealt with the scalability issue using a partition based approach. They have adopted a graph-based min-cut partition technique to decompose a large pattern database into certain basic pattern sets in 2004 [6]. After a reasonable partition is found, the redundancy contained in a pattern database can be reduced to reduce the corresponding consumption of hardware resources. Since the partition problem is achieved at the Register Transfer Level (RTL) rather than the gate level during the logic design phase, the system is affected less by synthesis tools, which facilitates pre-synthesis performance estimation [6].

In practice, pattern decomposition can be achieved via parallel hardware decoders. This operation is called "pre-decode" at the system level because pattern decomposition is done prior to pattern matching. The idea is to compare each input character to a set of known values of interest.

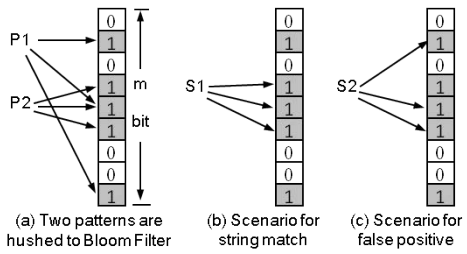

Fig. 4. Basic bloom filter operations

The comparator outputs are fed through a network of shift-register delay elements to maintain the temporal relationship among characters; the outputs of these shift registers become the individual match lines that are combined to perform pattern matching. This concept is an extension of the approach [100] mentioned in the previous section. Reported results indicate at least 8 times more area efficiency than other shift-and-compare architectures, and 2 times more efficiency than other pre-decoded architectures [6].

A subsequent work, reported in 2005 [7], further optimized the approach by combining graph-based partitioning and tree-based matching of large pattern databases. Moreover, an optimized incremental design strategy was introduced for place-and-route based on the min-cut partition approach. Usually, full place-and-route is required when any modification is made in FPGA design, no matter how small the change. The proposed strategy needs only to perform partial place-and-route, since other partition patterns are not affected.

2) Technology Analysis:

a) Bloom Filter based Technique: A Bloom filter is a data structure allowing representation of a set of elements with probabilistic membership queries [36]. Patterns are stored in Bloom filters compactly via the computation of multiple hash functions. An important property of Bloom filters is that the computation time for pattern searching is independent of the number of patterns contained in the database. In addition, the required memory space for hashed pattern storage is independent of the size of real patterns, though a capacious memory is preferred. Both membership queries and memory space depend on the number of hash functions. Hence, the Bloom filter technique naturally possesses good scalability.

Assume that a Bloom filter contains k independent hash functions and a memory with m bits. As shown in Fig. 4, for example, $k = 3$ and $m = 9$. Each hash function maps to a single location for a given key (pattern). During initialization, the Bloom filter sets k bits out of m memory bits according to the hash functions for each stored pattern, as shown in Fig. 4(a). While testing for membership, if any of the k memory bits is zero, the pattern is not stored; however, ambiguities exist when all bits are one. Fig. 4(b) and (c) illustrate one such scenario. S_1 and S_2 represent the corresponding hash values of incoming data. Though S_1 and S_2 both hit bits that have already been set, only S_1 is a true match since there is no such pattern mapping to the exact location where S_2 points. The erroneous matching result of S_2 is called a false positive [2]. Thus, each query result may be a match or a

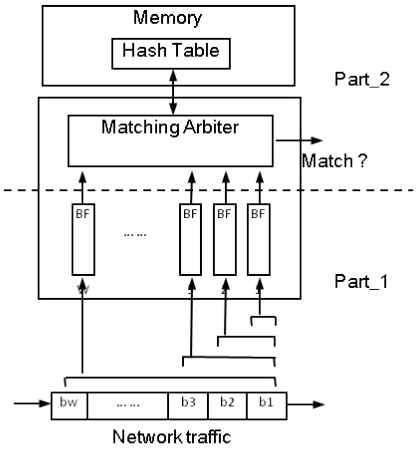

Fig. 5. Basic unit for pattern matching application with bloom filters

false positive but never a false negative. Bloom filters may over-report matches, but never miss any.

Due to the existence of false positives, Bloom filter based approaches usually contain two basic stages: one for Bloom filter querying and another for false positive elimination. A model adapted from [35] presents a good illustration, as shown in Fig. 5.

The lower portion of the figure, labeled Part_1, is for Bloom filter querying. This model includes w Bloom filters each having a different number of hash functions. Hashed patterns with the same length are stored in the same Bloom filter unit. In the figure, the rightmost Bloom filter unit contains all hashed patterns with the shortest length.

For simplicity both the shortest length of hashed patterns, and thus the number of corresponding hash functions, are one in this case. From the right to the left, the number increases by one. This indicates that the model has the ability to process different length patterns simultaneously. With data traffic connected, the size of operation window is w bytes. At each clock cycle, one byte of data is shifted through the window and all subsets of this w byte sequence of data are subjected to simultaneous query. In the case of multiple hits a potential match is chosen by a pre-defined policy; usually, either the longest match or the shortest will be considered as the interesting one.

As shown in Fig. 5, the functions of Part_2 include false positive elimination and final commitment of a match. It consists of two main components: a matching arbiter and a hash table. The hash table may be stored in the on-board RAM. The matching arbiter buffers the hash value of potential matched data. Then, it recalls the hash values of interesting patterns from the hash table and compares them. If they are the same, a match is committed and a match signal is output. The whole procedure requires temporary freezing of current Bloom filter querying for processing, so as to cause certain delay to the incoming traffic. Since the potential matching rate is expected to be low in real traffic, this kind of delay may have little effect to system performance. A proper applied data buffer for traffic holding could further mitigate this risk.

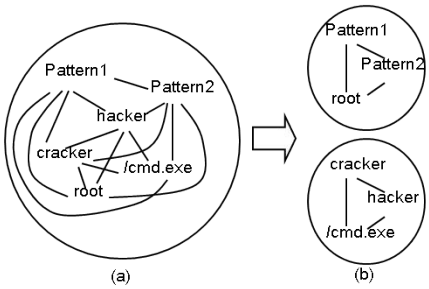

Fig. 6. (a). Un-partitioned graph (b).Partitioned graph

Hardware based Bloom filter implementations are quite attractive for quick pattern matching. In addition to possessing good scalability, other possible security enforcements may be added to the process of hash value generation. It is practical for special purpose hardware to handle those tasks. However, some researchers argue that the main weakness of the Bloom filter approach lies in its fixed length pattern processing. Without optimization, accommodating n different length patterns requires n Bloom filter units.

b) Min-cut Partition Based Technique: The min-cut partition technique stems from graph theory. With respect to pattern matching applications, the partition focuses on the pattern database, with the goal of exploiting redundancy for hardware optimization. Patterns can be considered as combinations of different characters and some elementary characters tend to repeat across patterns. According to the study in [6], Baker pointed out that only about 100 different characters are ever matched against within the whole set of the Snort database, if more than 2000 patterns could be completely decomposed into single characters.

The rationale for the partition is that the number of repeated characters within a group be maximized, while the number of characters repeated between groups is minimized. With min-cut, the entire pattern database is partitioned into several small groups. Fig. 6, reproduced from [6], illustrates this concept. Fig. 6(a) represents the original pattern database, which is a densely connected graph. Each node is a pattern and the edges between them represent shared characters. Fig. 6(b) represents the result after partition.

In terms of hardware implementation, this result implies that the number of pipeline registers can be decreased, since small groups naturally contain fewer characters, which reduces the length of pipeline stage. Meanwhile, the average utilization of each character is increased, because like characters more frequently appear within few groups. In addition, more parallel operations can be performed since more pattern units are created due to the min-cut partition. Considering these advantages, redundant hardware resources for pipelined operations can be released or be used to support more parallel operations if possible.

Through reasonable min-cut partitioning, the pattern matching operation can be achieved efficiently and compactly. However, rules for partitioning are usually heuristic; thus, it is difficult to find a single algorithm to maintain efficiency across applications. Fortunately, the emergence of design automation

tools enables efficient application of trial-and-error.

V. TCP Stream Preprocessing Application

TCP is a connection-oriented protocol which provides guaranteed transport for network applications over unreliable network environments. In real network environments, such as the Internet, more than 85% of traffic uses the TCP protocol [54, 94]. With TCP, packets can be dropped, duplicated or re-ordered during transmission. Hence, data streams observed at network connection points may deviate from their original patterns. Since this situation can be exploited to conceal malicious activity, for security monitoring it is necessary to reveal it through continuous TCP session monitoring [76].

A Network Intrusion Detection System (NIDS) is a type of network security application that provides more sophisticated protection than firewalls. TCP session monitoring is one of the basic functions of a NIDS. To achieve TCP session monitoring in a NIDS it is necessary to perform packet reordering, stream reassembly and state tracking operations of TCP traffic [87]. In this paper, we consider these operations as preprocessing of TCP traffic since most security applications require well organized TCP streams for analysis.

Modern security applications that focus on the network infrastructure will be inefficient or less effective without the capability of high-speed TCP traffic handling. The data rate of communication links on which current Internet backbones operate is in the range from OC-3 (155 Mbps) to OC-768 (40 Gbps) [92]. Many advanced infrastructure security services require high-speed TCP traffic manipulation, including packet inspection, content-based routing, Internet worm detection, DDoS attack resistance and spam removal. Hardware-based implementations are essential to most security applications for adequate performance.

However, hardware resources are generally limited in practice and efficient resource utilization is highly desired. Modular design for hardware offers an opportunity to separate TCP stream preprocessing from security functions. Since many security functions require TCP stream preprocessing, separating this function allows subsequent applications to be relieved of this task, which can result in more resources being available for security functions. These recovered resources can be utilized to support other tasks, enabling high quality network services at Gigabit rates.

Thus far, the research in hardware based TCP stream preprocessing has not received sufficient recognition as only a few papers related to this topic have been published. The reasons for this could be twofold. On the one hand, the processing of TCP streams itself may seem to be fairly straightforward, so it may not be considered to be worth the effort. On the other hand, the benefit of modular design for hardware TCP stream preprocessing may not be clearly identified. In fact, implementing this function as an individual module is meaningful to practical applications. In our opinion, a modular TCP stream reassembly block can not only save valuable resources for subsequent blocks, but also optimize the system architecture in a concise way. This property is highly desirable to implement complicated security applications at the system level.

A. Existing Applications

A TCP stream reassembly and state tracking module was reported in 2002 [76]. They proposed to implement portions of the NIDS functions to a high performance reconfigurable network card. Its feasibility has been demonstrated by a reassembly module designed in VHDL and implemented on the Xilinx XCF1000 FGPA platform. It operates at a 100 MHz system frequency with 32 bit data width, so the maximum throughput achieved is 3.2 Gbps for raw TCP/IP traffic.

After successfully adapting the reassembly function from software to hardware with a single thread, feasibility with multi-threaded execution was subsequently explored. This led to the question of how to manage multi-threaded processing. Though they did not present any detailed implementation, it is conceivable that the proposed design works for multi-threaded TCP reassembly processing under modest network circumstances.

In 2003, Li and Torresen [61] implemented a reconfigurable hardware architecture to replace a software based STREAM4 [38] preprocessor in Snort, which performs TCP stateful inspection and reassembly functions. Through logging and analyzing the TCP stream packet by packet, it is possible to predict what will happen next based on the status of current and past packets. This function is very helpful for detection of certain abnormal activities.

The major innovation is the implementation of a Server TCP stream reassembly block and a Client reassembly block. Unlike the implementation in [76] where reassembled flows for opposite directions are simply stored in two different RAMs at the end of the processing, their approach segregates unprocessed server-oriented streams and client-oriented streams at the very beginning. The design was implemented on a Xilinx Virtex XCV1000-6 FPGA. The core part of the TCP stream reassembly unit consists of two 32 bit comparators and one 32 bit adder. Experimental results showed that this design achieved a throughput of 3.06 Gbps.

Another similar implementation with a slight modification was reported in [60], which was evaluated on top of the FPX platform from Washington University at St. Louis (WUSTL) [65]. The whole system has been placed and routed into a XCV2000E FPGA with throughput at 2.75 Gbps. Only a total of 2.5 percent of the SLICEs (496 of 19600 SLICEs) of an XCV2000E-6 were required to implement this system [60].

Schuehler and Lookwood reported their TCP-splitter in 2002 [91]. TCP-splitter was designed as a lightweight, high performance circuit module that provides a consistent TCP data stream to client application systems. It consists of two logical sections: TCP-input, which handles ingress IP frames and most of the job of a TCP-splitter; and TCP-output, which is responsible for packet routing and frame delivery to either the IP stack or its client applications for further security processing. Their synthesized design on a Xilinx Virtex 1000E-7 FPGA demonstrated a throughput of 3.2 Gbps at 100 MHz clock frequency.

A TCP processing Engine has been developed based on the splitter [93]. The most significant improvement in the design is the appearance of off-chip memory. Off-chip memory dramatically expands the available resources for TCP stream pro-

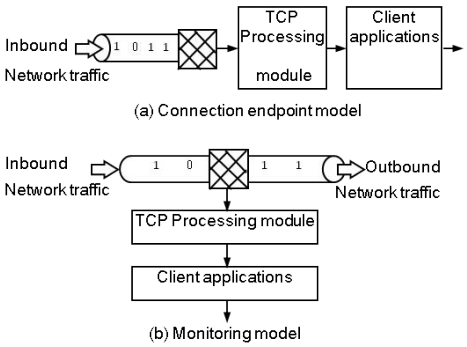

Fig. 7. Basic Models for TCP stream processing

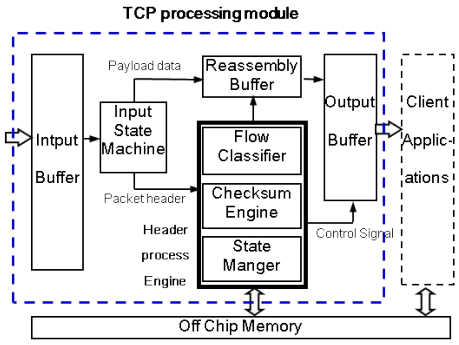

Fig. 8. Block diagram for TCP stream preprocessing

cessing. In addition, with the assistance of off-chip memory, cooperation between different functional modules becomes feasible. Information from higher-level applications can be used to assist servicing of future TCP stream processing. In this way, the processing of a TCP stream can be evaluated at a higher level, incorporating this intelligence with self-updating.

A structure for parallel TCP stream scanning was also discussed [93]. The deigned circuit was targeted for the Xilinx XCV2000E FPGA on the FPX platform. This architecture is able to monitor eight million simultaneous TCP flows at OC48 rates (2.5 Gbps). In addition, the cooperation issue was discussed in a subsequent publication in 2004 [92], in which they added a set of encoding and decoding circuits for the cooperation among multiple FPGA devices.

B. Technology Analysis

Among the published TCP stream processing applications, most adhere to one of two models. One model acts as a connection endpoint of network traffic that accepts and processes all the incoming network traffic; the other one acts as a monitor that inspects the TCP streams passing through. Each model has distinct characteristics that can complement the other. Selection of a suitable model for implementation depends on specific design requirements. Resource consumption is always a big concern that affects the practical implementation.

Usually, the endpoint model is suitable for fine TCP stream processing at end-system networks with moderate TCP connections, such as deep state inspection. Necker and Contis's design [76] follows this model. By instantiating multiple processing circuits, up to 30 TCP/IP connections can be processed simultaneously on a single FPGA board. On the other hand, the monitoring model is suitable for coarse TCP stream processing in high throughput networks with massive TCP connections, such as reassembly, reordering, or counting. Following the monitoring model, it is able to monitor all TCP flows passing through the TCP-splitter [91]. Fig. 7(a) represents the connection endpoint model, and Fig. 7(b) represents the monitoring model.

Applications for TCP stream preprocessing require placement at locations where the interesting traffic passes. Indeed, all packets associated with a TCP/IP connection must be available for inspection. Only in this manner can consistent security protection be guaranteed to all the packets within a TCP connection. In general, the best deployment locations for TCP stream applications are at places where edge routers are located [91]. It is extremely difficult to process traffic in the middle of the network in practice, since packets may subsequently be dropped, duplicated or re-ordered along the way. However, it is not difficult to process them at the location of connected points. In addition, the traffic passing through the network is not widely distributed. In fact, it is usually concentrated to flows over a limited number of routers [39].

To correct the TCP stream defects produced during transmission, TCP stream applications mainly focus on the inspection of the packet header. Although packet classification also focuses on packet header, it concentrates on malicious inspection other than the typical TCP defects inspection. Fig. 8 demonstrates the basic procedure of TCP steam preprocessing.

When an incoming data stream enters the processing module via an Ethernet line card, an input buffer holds its packets for synchronization. Buffered packets are then decomposed into packet headers and payloads by an input state machine. Corresponding tags are assigned to both parts for later association. Subsequently, payloads are dispatched to another buffer to wait for reassembly, while packet headers are dispatched to the header processing engine for advanced processing. During inspection, all information contained in a packet header is subject to checking, including IP address, port number, sequence number, acknowledgement number, and flags.

A state manager works as a coordinator handling all components of the header processing engine, including the checksum engine and flow classifier. In addition, it assumes the responsibility for off-chip memory access for updating. Since the off-chip memory may contain useful information from other applications, continuous updating not only improves the resilience of the header processing engine, but also makes it more intelligent. The header processing engine may feature the capability to segregate or drop verified malicious flows at this point. Therefore, the boundary between typical TCP preprocessing and packet classification is blurred due to this cooperation. Of course, this function is optional; its use depends on specific application requirements.

After being inspected, eligible headers are rearranged into flows according to certain rules. Individual packets are re-

assembled using the association numbers previously assigned. Meanwhile, certain useful control signals may be assigned to these flows for future use. Finally, reorganized TCP flows are stored in an output buffer for further processing by subsequent client applications.

VI. Internet Worms and DDoS Attack Detection and Containment

Internet worms and Distributed Denial-of-Service (DDoS) attacks have been identified as the two major security threats towards the network infrastructure [16]. Since both aim at vulnerabilities of the network infrastructure, our daily activities are more sensitive to this kind of attack than other higher level attacks. MyDoom, an Internet mailer worm first detected on 26 January 2004, is considered one of the fastest-spreading and possibly the most dangerous worm in history [85]. In the first 36 hours after release, it reached more 160 countries and infected over 100 million emails [110]. Such widespread infection can be even more severe if malicious DDoS content, such as a "Trojan horse", is piggybacked onto the payloads of worm packets for a joint attack. Unfortunately, this became the case when approximately 500,000 computers worldwide were infected by MyDoom.a, which then launched a Denial of Service (DoS) attack on SCO Systems services at www.sco.com on 1st February 2004, according to a report form CRN.COM [53].

In practice, security applications can only protect a limited network area and have little information about the network as a whole. Even during a single attack, security strategies may need to vary depending on the specific attack stage in progress. Though different attacks exhibit diverse behaviors, security strategies against them usually can be categorized as pre-attack prevention, under-attack containment and post-attack update.

Currently, most security applications against worm attacks are applied during the under-attack stage. Good disguise and quick outbreak make worms difficult to prevent at the pre-attack stage. Due to the property of self-propagation, worms can spread quickly and have the potential to evolve into massive attacks over the whole network within a short time. Though many worms make no attempt to modify the systems that they pass through, they may quickly exhaust the resources of and finally crash these systems. It is essential to contain the propagation of worms as early as possible during their outbreak.

Compared with worm containment, security strategies against DDoS attacks are more complicated due to the two-phase discipline of DDoS attacks [84]. The first phase of a typical DDoS attack is to compromise a large number of computers and recruit them into a zombie army; the second phase is to indirectly launch DDoS attacks towards a specific target through these zombies [84]. The joint attack of DDoS and worms can be even more destructive. If attackers take the advantage of worm spreading to comprise thousands or even millions of computers and subsequently launch a DDoS attack, none of today's network infrastructure will be able to survive under such an aggressive attack without proper protection.

Exploring this two-phase attack discipline, security applications can limit or reduce the number of zombies by filtering the malicious content intended for this purpose at the pre-attack stage. However, under-attack containment is the core part of the defense against a DDoS attack once it begins. Finally, post-attack update should not be ignored, since it is the most convenient way to maintain the resilience of security applications.

A. Existing Applications

Although the importance of defending against Internet worms and DDoS attacks has been widely recognized and many efforts have been reported, few of these have addressed defense approaches based on hardware. Excluding the fact that certain applications may not be released due to national security or commercial concerns, current academic research in this area is still at the initial stage.

Lockwood and Moscola reported their first hardware-based implementation of a worm containment scheme in 2003 [67]. This design uses FPGA devices for worm scanning at high speed Internet traffic rates. The system consists of three interconnected components. The major component is the Data Enabling Device (DED) which is deployed at key traffic aggregation points for packet inspection. Content Matching Service (CMS) updates the signature database and automatically generates the corresponding binary code for the reconfiguration of DED circuits. Regional Transaction Processor (RTP) works as a coordinator that manages the system to maintain proper operation.

The Field Programmable Port Extender (FPX) card [64] functions as the interface between network traffic and the hardware device. It is the heart of DED. Through the utilization of layered protocol wrappers [14], application-level flows can be decomposed for further analysis. Implementing four signature detection modules in parallel, the FPX is able to process data at the rate of 2.4 Gbps with a single Xilinx Virtex 2000E FPGA. At this rate, the DED module can achieve full throughput for IP packet lengths ranging from 40 bytes to 1500 bytes. A more detailed design can be found in another paper published in 2003 [68].

The above design is a typical application of signature detection. All decisions are made depending on prior attack knowledge. However, this technology is not sufficient for unknown attack detection. In 2004, an anomaly detection based approach for worm containment was proposed [72]. It was inspired by the idea that a worm detection system should keep looking for frequently occurring contents [96]. The system can process traffic at full line rate and was implemented on the FPX platform. The circuit runs at a frequency of 91.5 MHz and network traffic is fetched into the device with 32 bit-width. The circuit fits well with Xilinx Virtex 2000 FPGA device and allows the processing at OC-48 line speed on the FPX platform. With a pipeline delay of 7 clock cycles, the circuit introduces a delay of only 70 ns. This implies that the system is feasible for real-time worm containment.

Due to the fact that even legitimate data packets, such as SYN-ACK, RST or ICMP packets in TCP flows, can be used for DDoS attacks [21], it is often difficult to discern malicious activities via signature-based detection. However,

TABLE I
APPLICABILITY OF BASIC SECURITY APPLICATIONS

Technology Category	Sub-category	Best Time to Apply		
		Pre-attack	Under-attack	Post-attack
Signature detection	Packet header classification	x	x	
	Deep payload inspection	x	x	
Anomaly detection	Direct content counting	x	x	
	Spectral analysis	x	x	
Knowledge update		x	x	x

the flooding property of DDoS attacks makes them relatively easy to detect via anomaly detection. A recently published DDoS detection application by Oh and Park in 2007 is based on the inspection of network traffic rates [78]. A double token-bucket mechanism was applied for bandwidth control.

This double token-bucket mechanism sets two essential thresholds, one for the first bucket size and the other for the sum of both bucket sizes. During the continuous flow of incoming traffic, the current token level of traffic bandwidth varies. If the token level exceeds the threshold of the first bucket size but not the second one, a warning will be issued for notification. If the token level exceeds the second threshold, packets will be dropped due to the limitation of throttling bandwidth. At the cost of dropping packets, valuable bandwidth can be saved. This design has been implemented on a security board that integrates Xilinx Vertex II Pro FPGAs with a set of two gigabit Ethernet interfaces for evaluation. It can serve up to 2 Gbps Ethernet on bidirectional traffic and full 1 Gbps bidirectional traffic without loss [78].

The theory of using spectral analysis for DDoS attack defense has been proposed for years [9, 25]. However, few corresponding hardware-based approaches have been developed due to the infeasibility of using hardware for implementation and the cost of hardware devices. A Power Spectral Density (PSD) analysis based hardware application was recently developed [22]. The design is specifically targeted for online detection of shrew attacks [57, 71], a type of stealthy DDoS attacks. The idea stems from the observation that malicious shrew attack flows are more distinguishable in the frequency domain. With a proper DFT conversion, it is more convenient and accurate to perform the anomaly detection in the frequency domain [24]. The key procedure to achieve this goal is the consecutive calculation of autocorrelation and Power Spectrum Density (PSD) of sampled data. This application was designed on the Xilinx ISE platform with the Virtex-4 XC4VFX12 as the targeted device. Under extreme experimental conditions, with a system frequency of only 5 MHz and a long TCP flow window of 4096 packets (normal TCP-flow length is less than 1024 packets), it takes only 6.88 ms to achieve the major processing required.

B. Technology Analysis

Defending against Internet worms or DDoS attacks is a systematic process that integrates various technologies and strategies. The choice of applying different security technologies and adopting corresponding strategies depends on the specific application. Observing existing security solutions against Internet worm and DDoS attacks, we find that many basic technologies are shared among applications. Though we will only focus on the defense of the two types of attacks here, the above observation is generally applicable to most security solutions. For convenience, Table I associates the major technologies with their best application time. Following this categorization, this section summarizes the technologies adopted for reconfigurable hardware based implementations.

From the perspective of defenders, threats to the network infrastructure can be classified into known and unknown types. The known threats imply that these threats have been captured and their properties have been studied. Their signatures are usually collected and stored in a knowledge database for future reference. On the contrary, unknown threats include all that have not been identified.

For known threats, signature detection is the most intuitive and straightforward technique for quick and accurate identification. It can be applied at both the pre-attack and under-attack stages for prevention and containment. Any known type of worm and malware containing a malicious DDoS attack payload can be easily detected through packet header classification and deep payload inspection. The intensive computing required of signature comparison can be accomplished on high performance hardware. An analysis of hardware-based applications of these has been presented in the previous sections. Anomaly detection is another option for known threat detection, but is more popular for unknown threat detection.

In addition to techniques developed directly for maliciousness detection, update techniques for the resiliency of security applications is equally important. Knowledge update mainly focuses on the update of all kinds of databases, especially the signature database. Due to the complex procedure of hardware reconfiguration, a dedicated mechanism is desired.

1) Anomaly Detection: Generally speaking, anomaly detection can run faster than signature detection. Instead of performing exact matching of a large number of signatures, as in signature detection, modeling normal behavior and evaluating deviation from thresholds is typically less computationally intensive. However, this benefit is achieved at the cost of generally higher false positive rates. Anomaly detection is widely applied for the detection of unknown malicious activities at the pre-attack and under-attack stages. Due to constantly changing malicious threats, it is impossible to have all knowledge available before new threats emerge. Through continuous observation and modeling of normal behavior, anomaly detection offers a way to find possible threats via deviation from the normal model without knowing signatures. The key is to choose reasonable thresholds so the balance between performance and false positives can be maintained.

Most Internet worms and DDoS attacks exhibit the common characteristic of flooding during attack, with the difference that Internet worms spread randomly while the DDoS attacks target specific victims. Nearly all prevalent Internet worms and DDoS attacks had not exposed themselves before they broke out. Taking advantage of anomaly detection, these kinds of

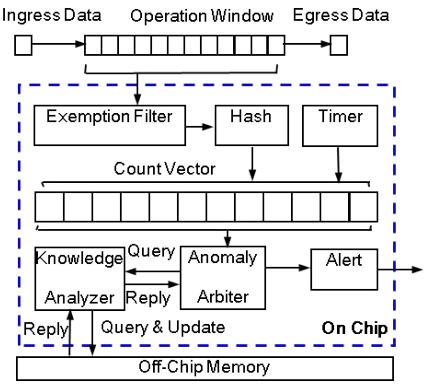

Fig. 9. Block diagram for payload anomaly detection

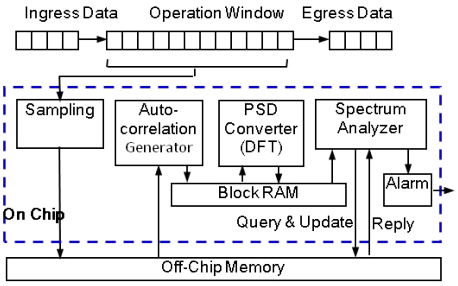

Fig. 10. Block diagram for PSD based anomaly detection

threats can be effectively detected, introducing the possibility of stopping further propagation and curtailing wider impact on the network infrastructure. In the following subsections, two application modules are presented. In order to explain their operation, both are assumed to be run within single thread. However, higher throughput can be achieved using multi-threaded operation via multiple-module instantiation.

a) Direct Content-Counting Technique: Direct content-counting is the most intuitive and widely used technique for anomaly detection. Any observable features are valid for counting, including packet payloads, addresses and port numbers. Regardless of the specific content counted, an anomaly can be detected as long as the counts of inspected objects deviate far enough from the normal range. Fig. 9 shows a block diagram of a typical logic module for anomaly detection [72]. It focuses on the detection of anomalous packet payloads.

With a suitable data rate, all packets are expected to be inspected. A fixed-length operating window limits the number of packets being processed simultaneously. Specifically, a window of only one data byte is assumed in this example. An exemption filter works as a coarse filter excluding many typical patterns with high frequency upfront. Through the hash functions, data patterns are hashed into specific locations of a counter vector, which adds one to the corresponding counters each time. If the counter of a pattern goes beyond the suspicious thresholds stored in the anomaly arbiter, an anomaly is detected. A timer is employed to refresh the count vector periodically. Proper timer length is important to efficiently capture as many anomaly patterns as possible. An important issue related to start-up is how to set the initial counter values. A typical way to handle this problem is to subtract an average value from all the counters, instead of resetting all the counters to zeros [72].

In practice, it is not wise to make decisions only on the result of anomaly arbitration due to its high false positive rate. It is both desirable and necessary to reduce this uncertainty in practice. We call the block performing this function a knowledge analyzer, as shown in Fig. 9. The goal is to reduce the false positive rate of suspected anomalies. The knowledge analyzer not only contains signatures or statistic estimations for reference, but also provides other available knowledge for support. Instead of occupying limited on-chip memory, off-chip memory provides a flexible space for knowledge storage. In addition, associating with off-chip memory is convenient due to its potential capability of sharing with other functions and even other systems.

b) Power Spectral Density based technique: Other than direct content-counting based anomaly detection, spectral analysis based anomaly detection has begun to emerge in recent years. The rationale is that certain complicated processes in the time domain may behave more concisely in the frequency domain. A key step for spectral analysis is converting time domain data into the frequency domain. This operation is very expensive from the perspective of computing power, especially for real time applications. Though many hardware devices can provide the necessary computing power, the high cost has prevented them from gaining widespread use. The rise of spectral analysis based approaches benefit from advancing hardware technologies, especially the emergence of high-performance cost-efficient FPGA devices.

With state-of-the-art technologies, there are now more hardware devices with powerful hardware cores and larger on-chip memories than ever. The traditional processing bottleneck of the Fourier transform and its relatively high memory access requirements is being eliminated. In addition, hardware vendors are willing to release diverse pre-designed intellectual property (IP) cores that are optimized for these hardware devices. The existence of IP cores offers designers great convenience. From the perspective of hardware applications, the Power Spectral Density (PSD) based technique is quite mature and very suitable for hardware implementation.

Power Spectral Density (PSD) based anomaly detection exploits the property that normal TCP flows possess a periodic Round-Trip Time (RTT) property during transmission, while anomalous TCP flows do not [26]. It is efficient to reveal this phenomenon by checking the PSD of corresponding packet flows in the frequency domain. However, network traffic is naturally sampled in the time domain and it is necessary to convert the sampled sequences into the frequency domain in order to examine the PSD properties. In practice, the whole procedure is treated as discrete signal processing; N samples are taken within a fixed-length time interval for processing. A detailed explanation can be found in [26] and [22].

An abstract diagram of a PSD based anomaly detection module is given in Fig. 10. A fixed-length operating window is set for data acceptance. At each time interval, the same

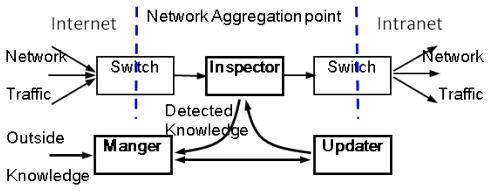

Fig. 11. Block diagram for system architecture

quantity of data will shift in and out of the window. Due to the high operating frequency of hardware devices and low sampling rates of real data, it is typically better to buffer the raw data in off-chip memory so that valuable on-chip resources can be reserved for more important tasks. After one unit of N sampled data points is gathered they are fetched to an autocorrelation generator. From there, the data is processed in a high-speed DFT hardware device. After the DFT operation, the power spectral densities of fixed length packets are generated and ready for analysis.

The spectrum analyzer performs anomaly analysis in the frequency domain. It usually contains some normal frequency-patterns of regular packet flows or specific thresholds for decision. Additional knowledge residing in off-chip memory is available to provide extensive support. Since this mechanism exploits the periodicity of TCP flows, it is sensitive to abnormal deviation. Low false positive rates make it possible to eliminate the corresponding mitigation required in other approaches.

2) System Update: Techniques for knowledge-update mainly focus on the update of different kinds of databases, especially those storing signatures. Through continuous update, the resilience of security applications can be maintained at a high level. Usually, the majority of the update is scheduled routinely. However, dynamic updating during attack requires time-efficient update to mitigate losses. In addition, corresponding hardware circuits may require certain modifications to accommodate the updated knowledge. With regards to FPGA devices, this process is referred to as reprogramming or reconfiguration. It is a multi-step process, including synthesis, mapping, place-and-route and finally downloading the binary code to the target device.

An abstract block-diagram in Fig. 11 illustrates such an updatable architecture at the system level [67]. The system consists of three major modules: inspector, updater and manager. Each module performs specific functions. In order to emphasize the process of update, all the functions corresponding to packet inspection are abstracted in the inspector module. The manager module takes charge of the update process and the updater module performs the hardware reconfiguration.

The manager module has two sources of knowledge update: one is from the outside the system, such as existing knowledge databases; the other is from the system itself, utilizing the detection results of the system. At least two update modes are required: by-request update and routine update. Using these modes, a system administrator can easily manage the update. However, both modes are relatively static. A dynamic update mode that can update the system online, especially the signature database and related knowledge, is highly desired. That way, any newly detected maliciousness can be quickly incorporated for the inspection of future packets.

Dynamically updating hardware-based applications within a restricted time period is nontrivial, however, due to the complex procedure of hardware reconfiguration. Certain tradeoffs can be made to handle this problem. An intuitive method is to use half of a FPGA to actively run operations leaving the inactive half ready for reprogramming. When an update has been made in the inactive half, it becomes active and the previously active half is released for the next reprogramming. This approach, however, may be na?ve for practical implementation. In general, dynamic hardware update is still an open question.

Finally, it is the responsibility of the update module to set up the hardware-based inspector through a reconfiguration procedure. Since the detailed reconfiguration procedure depends on specific hardware devices and software kits, it is beyond the scope of this paper. In a security oriented system, however, any potential vulnerability should be considered. The last major step for reconfiguration of the FPGA is downloading the configured bit-stream to the target device. If a remote operation is performed through the network, some new security issues regarding the protection of bit-streams during transmission need to be considered.

VII. Discussion

The application of hardware for network infrastructure security has become an active area of research. With the support of powerful hardware devices, many previously infeasible computationally intensive security tasks in software-based applications now become feasible. At the same time, however, hardware-based applications also face severe performance challenges due to fiercely rising demands to handle higher traffic rates, more intensive analysis, real-time operation and even the collapse of Moore's law for sequential processing [80]. Without a carefully thought out design, valuable hardware resources and powerful computing capabilities could be quickly exhausted.

Researchers have pointed out that now is the time to rethink the use of hardware for security applications [80]. They presented their vision to pursue the high parallelism, stating: "the key is devising an abstraction of parallel process that will allow us to expose the parallelism latent in semantically rich, stateful analysis algorithms." [80] Their proposal includes three major steps: 1) using high-level language to express rich forms of network analysis; 2) executing compiled elements of the language with a powerful parallel abstraction; and 3) compiling the results from the abstraction to concrete hardware implementations. Through decomposition and dispatching, the implementation of a sophisticated parallel process can be efficiently assigned to suitable devices. In addition to potentially high system performance, the utilization of available hardware resources can be maximized. Furthermore, an architecture exploiting multi-core processors to parallelized Network Intrusion Prevention (NIP) was proposed recently [81]. These proposals have introduced certain new design concepts that

are currently not available; thus they are still open issues. In addition to issues related to hardware implementation of local security applications, the challenge of system-wide collaboration lies in the global cooperation among these applications. This goes well beyond the scope of local hardware implementation and must consider high-level issues of systems engineering. The idea of system collaboration was proposed in [16, 17]. System-wide collaboration offers to individual applications a global view of dynamic security situations. Since individual single-point-deployed security applications can only cover limited areas of the network infrastructure, they generally do not have global knowledge. With multi-point-collaboration, a security shield which covers a wider area of the network infrastructure can be established without major modifications to the current single-point-deployed architecture. In this manner, applications at individual nodes have more ability to handle sophisticated security problems.

From our viewpoint, achieving system-wide collaboration requires addressing collaboration at two levels. The first level is collaboration between different single-point-deployed security applications within the same system. The second is collaboration between different single-point-deployed security applications in different systems. Then, the corresponding hardware implementation changes should follow. Until now, collaboration technologies have been based on the use of an overlay network within the same security system [16]. Through effective information sharing and updating, system performance can be improved substantially.

Collaboration at the system level can not only further improve the performance of current security applications but is essential to build intelligent security systems, such as neuron-network-like security systems. The prospective benefits stimulate research interests to develop distributed systems [108]. However, this research is still at the initial stages. The collaboration among different security systems is non-trivial, and up to now there has been no reported mechanism to effectively coordinate different security systems. In fact, many questions regarding collaboration are open. However, the potential benefits justify more effort.

VIII. Conclusion

In this paper, we reviewed existing hardware-based techniques for network infrastructure security. Through the analysis of general network security, we presented the background of network infrastructure security. Then, we introduced the major fundamental techniques being applied for signature detection, including packet classification and pattern matching. Subsequently, we focused on TCP-stream based security applications, since TCP-based streams dominate network traffic.

After considering the pre-processing of TCP-streams, we focused on security applications protecting the end-system level of the network infrastructure, since users are more sensitive to malicious activities at this level. This includes the detection of Internet worms and DDoS attacks. In addition to classical techniques for signature detection, anomaly detection techniques were reviewed. Direct content-counting and PSD based techniques are two useful anomaly detection approaches. In order to maintain the resilience of security applications, continual system update is necessary. With respect to hardware, the principal challenge is how to dynamically reconfigure hardware devices. Finally, we concluded with a brief discussion regarding open-questions such as high-parallelism and system collaboration.

Our ongoing research focuses on anomaly detection for DDOS attacks using spectral analysis. This approach is based on the Xilinx Virtex 4 FPGA platform with VHDL coding. After achieving some encouraging simulation results, we plan to evaluate our design remotely on the testbed of the Open Network Laboratory (ONL) at Washington University in St. Louis. With their newly developed FPX interface, we are able to directly upload our configured bit-stream code for evaluation. Based on our research results, we are also interested in more intelligent security systems. Collaboration is one potential approach and a dedicated protocol set is the key to achieve reliable and effective collaboration.

In short, this paper presents a survey of state-of-the-art FPGA based implementations in the network infrastructure security area, covering the major categories of currently existing diverse implementations. It is our hope that this survey presents a comprehensive coverage of the current status in this area and will inspire active research in this area.

References

[1] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein, "A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 2, pp. 106–124, 2009.

[2] D. Allchin, "Error types," *Perspectives on science*, vol. 9, no. 1, pp. 38–58, 2001.

[3] A. Callado, C. K. G. Szab, B. P. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, 2009.

[4] M. Attig, S. Dharmapurikar, and J. Lockwood, "Implementation results of bloom filters for string matching," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004, pp. 322–323.

[5] F. Baboescu and G. Varghese, "Scalable packet classification," *Networking, IEEE/ACM Transactions on*, vol. 13, no. 1, pp. 2–14, 2005.

[6] Z. K. Baker and V. K. Prasanna, "A methodology for synthesis of efficient intrusion detection systems on fpgas," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004, pp. 135–144.

[7] Z. K. Baker and V. K. Prasanna, "High-throughput linked-pattern matching for intrusion detection systems," in *Architecture for networking and communications systems, 2005. ANCS 2005. Symposium on*, 2005, pp. 193–202.

[8] Z. K. Baker and V. K. Prasanna, "Automatic synthesis of efficient intrusion detection systems on fpgas," *Dependable and Secure Computing, IEEE Transactions on*, vol. 3, no. 4, pp. 289–300, 2006.

[9] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. Marseille, France: ACM, 2002, pp. 71–82.

[10] S. M. Bellovin, D. D. Clark, A. Perrig, and D. Song, "A clean-slate design for the next-generation secure internet," in *NSF workshop at CMU*, 2005.

[11] P. Bellows and B. Hutchings, "Jhdl-an hdl for reconfigurable systems," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, 1998, pp. 175–184.

[12] I. Bonesana, M. Paolieri, and M. D. Santambrogio, "An adaptable fpga-based system for regular expression matching," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 1262–1267.

[13] H. Bos and K. Huang, "A network instruction detection system on ixp1200 network processors with support for large rule sets," in *Technical Report 2004-02*. Leiden Univeristry, 2004.

[14] F. Braun, J. Lockwood, and M. Waldvogel, "Protocol wrappers for layered network packet processing in reconfigurable hardware," *Micro, IEEE*, vol. 22, no. 1, pp. 66–74, 2002.

[15] F. R. R. J. V. Z. Brown, S.D., *Field-Programmable Gate Arrays*, 1st ed., ser. The Springer International Series in Engineering and Computer Science Ser. New York: Springer-Verlag New York, LLC, 1992, vol. 180.

[16] M. Cai, K. Hwang, Y.-K. Kwok, S. Song, and Y. Chen, "Collaborative internet worm containment," *Security & Privacy, IEEE*, vol. 3, no. 3, pp. 25–33, 2005.

[17] M. Cai, K. Hwang, J. Pan, and C. Papadopoulos, "Wormshield: Collaborative worm signature detection using distributed aggregation trees," in *tech. report TR 2005-12*. Univ. of Southern California, 2005.

[18] L. S. Cardoso, "Internet security and critical infrastructures," 2004. [Online]. Available: http://www.eurescom.de/message/messagesep2004/Internet_security_and_critical_infrastructure.asp

[19] CERT, "Overview of attack trends," CERT? Coordination Center, Carnegie Mellon University, Tech. Rep., 2002.

[20] A. Chakrabarti and G. Manimaran, "Internet infrastructure security: a taxonomy," *Network, IEEE*, vol. 16, no. 6, pp. 13–21, 2002.

[21] R. K. C. Chang, "Defending against flooding-based distributed denial-of-service attacks: a tutorial," *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 42–51, 2002.

[22] H. Chen and Y. Chen, "A novel embedded accelerator for online detection of shrew ddos attacks," in *Networking, Architecture, and Storage, 2008. NAS '08. International Conference on*, 2008, pp. 365–372.

[23] Y. Chen and H. Chen, "Neuronet: An adaptive infrastructure for network security," *Journal of Information, Intelligence and Knowledge*, vol. 1, no. 2, pp. 143–168, 2009.

[24] Y. Chen and K. Hwang, "Collaborative change detection of ddos attacks on community and isp networks," in *Collaborative Technologies and Systems, 2006. CTS 2006. International Symposium on*, 2006, pp. 401–410.

[25] Y. Chen, K. Hwang, and Y.-K. Kwok, "Filtering of shrew ddos attacks in frequency domain," in *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, 2005, pp. 8 pp.–793.

[26] C.-M. Cheng, H. T. Kung, and K.-S. Tan, "Use of spectral analysis in defense against dos attacks," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 3, 2002, pp. 2143–2148.

[27] Y. H. Cho and W. H. Mangione-Smith, "Deep packet filter with dedicated logic and read only memories," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004, pp. 125–134.

[28] Y. H. Cho and W. H. Mangione-Smith, "Deep network packet filter design for reconfigurable devices," *Trans. on Embedded Computing Sys.*, vol. 7, no. 2, pp. 1–26, 2008.

[29] C. R. Clark and D. E. Schimmel, "Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns," in *Proc. 11th ACM/SIGDA Int. Conf. Field-Program. Logic Appl. (FPL)*, 2003, p. 956.

[30] C. R. Clark and D. E. Schimmel, "Scalable pattern matching for high speed networks," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004, pp. 249–257.

[31] M. Colajanni, D. Gozzi, and M. Marchetti, "Enhancing interoperability and stateful analysis of cooperative network intrusion detection systems," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*. Orlando, Florida, USA: ACM, 2007, pp. 165–174.

[32] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: understanding, detecting, and disrupting botnets," in *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*. Cambridge, MA: USENIX Association, 2005, pp. 6–6.

[33] D. E. Culler, A. Gupta, and J. P. Singh, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers Inc., 1997.

[34] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An fpga-based network intrusion detection architecture," *Information Forensics and Security, IEEE Transactions on*, vol. 3, no. 1, pp. 118–132, 2008.

[35] S. Dharmapurikar, "Design and implementation of a string matching system for network intrusion detection using fpga-based bloom filters," in *Proc. of 12 th Annual IEEE Symposium on FieldProgrammable Custom Computing Machines*, 2004.

[36] S. Dharmapurikar and J. W. Lockwood, "Deep packet inspection using parallel bloom filters," *Micro, IEEE*, vol. 24, no. 1, pp. 52–61, 2004.

[37] S. Dharmapurikar and J. W. Lockwood, "Fast and scalable pattern matching for network intrusion detection systems," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 10, pp. 1781–1792, 2006.

[38] S. Egorov and G. Savchuk, "Snortran: An optimizing compiler for snort rules," Fidelis Security Systems, Inc., Tech. Rep., 2002.

[39] S. G. Eick, J. W. Lockwood, R. Loui, J. Moscola, and D. J. Weishar, "Transformation algorithms for data streams," in *Aerospace Conference, 2005 IEEE*, 2005, pp. 1–10.

[40] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda, "Performance characterization of a 10-gigabit ethernet toe," in *High Performance Interconnects, 2005. Proceedings. 13th Symposium on*, 2005, pp. 58–63.

[41] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, pp. 13–13, 2006, 1288236.

[42] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, "Granidt: Towards gigabit rate network intrusion detection technology," in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*. Springer-Verlag, 2002, pp. 404–413.

[43] L. A. Gordon and I. Computer Security, "2004 csi/fbi computer crime and security survey," 2004, (San Francisco, Calif.). [Online]. Available: http://i.cmpnet.com/gocsi/db%5Farea/pdfs/fbi/FBI2004.pdf

[44] T. Grandison and M. Sloman, "A survey of trust in internet applications," *IEEE Communications Surveys & Tutorials*, vol. 3, no. 4, pp. 2–16, 2000.

[45] S. Guccione, D. Levi, and D. Downs, "A reconfigurable content addressable memory," in *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*. Springer-Verlag, 2000, pp. 882–889.

[46] P. Gupta and N. McKeown, "Packet classification on multiple fields," Cambridge, Massachusetts, United States, pp. 147–160, 1999.

[47] S. Hauck, "The roles of fpgas in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[48] P. Hunter, "Hardware-based security: Fpga-based devices," *Computer Fraud & Security*, vol. 2004, no. 2, pp. 11–12, 2004.

[49] B. L. Hutchings, R. Franklin, and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," in *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, 2002, pp. 111–120.

[50] V. M. Igure and R. D. Williams, "Taxonomies of attacks and vulnerabilities in computer systems," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 6–19, 2008.

[51] H.-J. Jung, Z. K. Baker, and V. K. Prasanna, "Performance of fpga implementation of bit-split architecture for intrusion detection systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, 2006, p. 8 pp.

[52] C. Kaufman, R. Perlman, and M. Speciner, *Network security : private communication in a public world*, 2nd ed., ser. Prentice Hall series in computer networking and distributed systems. Upper Saddle River, N.J.: Prentice Hall PTR, 2002.

[53] G. Keizer, "Mydoom dos attack on microsoft falters," February 03 2004. [Online]. Available: http://www.crn.com/security/18831398

[54] S. Kent and K. Seo, "Rfc 4301:security architecture for the internet protocol," 2005. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4301.txt

[55] T. Kojm, "Clamav." [Online]. Available: http://www.clamav.net.

[56] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[57] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. Karlsruhe, Germany: ACM, 2003, pp. 75–86.

[58] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," 1998, 285283 203-214.

[59] M. V. Lawson, *Finite Automata*. Boca Raton: Chapman & Hall/CRC, 2004.

[60] S. Li, J. Torresen, and O. Soraasen, "Improving a network security system by reconfigurable hardware," in *proc. of 22nd Norchip Conference*, Oslo, Norway, 2004.

[61] S. Li, J. Torresen, and O. Sorasen, "Exploiting stateful inspection of network security in reconfigurable hardware," in *Field-Programmable Logic and Applications*. Springer Berlin / Heidelberg, 2003, vol. 2778/2003, pp. 1153–1157.

[62] P.-C. Lin, Y.-D. Lin, T.-H. Lee, and Y.-C. Lai, "Using string matching for deep packet inspection," *Computer*, vol. 41, no. 4, pp. 23–28, 2008.

[63] C.-T. D. Lo, Y.-G. Tai, and K. Psarris, "Hardware implementation for network intrusion detection rules with regular expression support," in *Proceedings of the 2008 ACM symposium on Applied computing*. Fortaleza, Ceara, Brazil: ACM, 2008, pp. 1535–1539.

[64] J. Lockwood, N. Naufel, J. Turner, and D. Taylor, "Reprogrammable network packet processing on the field programmable port extender (fpx)," in *FPGA*, 2001, pp. 87–93.

[65] J. W. Lockwood, "An open platform for development of network processing modules in reprogrammable hardware," pp. WB–19, January 2001.

[66] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and L. Jianying, "Netfpga–an open platform for gigabit-rate network switching and routing," in *Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on*, 2007, pp. 160–161.

[67] J. W. Lockwood, J. Moscola, M. Kulig, D. Reddick, and T. Brooks, "Internet worm and virus protection in dynamically reconfigurable hardware," in *Military and Aerospace Programmable Logic Device (MAPLD)*, 2003, p. 10.

[68] J. W. Lockwood, J. Moscola, D. Reddick, M. Kulig, and T. Brooks, "Application of hardware accelerated extensible network nodes for internet worm and virus protection," pp. 44–57, 2003.

[69] J. Loinig, J. Wolkerstorfer, and A. Szekely, "Packet filtering in gigabit networks using fpgas," in *Austrochip 2007 - Proceedings of the 15th Austrian Workshop on Microelectronics (2007)*, 2007, pp. 53 – 60.

[70] S. Low and J. Walrand, *Transmission Control Protocol*. San Rafael: Morgan & Claypool Publishers, 2007.

[71] X. Luo and R. K. C. Chang, "On a new class of pulsing denial-of-service attacks and the defense," in *Network and Distributed System Security Symp. (NDSS)*, 2005, pp. 61–79.

[72] B. Madhusudan and J. Lockwood, "Design of a system for real-time worm detection," in *Proceedings of the High Performance Interconnects, 2004. on Proceedings. 12th Annual IEEE Symposium*. IEEE Computer Society, 2004, pp. 77–83.

[73] J. P. Mermet, *Fundamentals and Standards in Hardware Description Languages: Proceedings of the NATO Advanced Study Institute, in Ciocco, Barga, Italy, April 16-26, 1993*. Kluwer Academic Publishers, 1993.

[74] A. Mitra, W. Najjar, and L. Bhuyan, "Compiling pcre to fpga for accelerating snort ids," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*. Orlando, Florida, USA: ACM, 2007, pp. 127–136.

[75] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, "Implementation of a content-scanning module for an internet firewall," in *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, 2003, pp. 31–38.

[76] M. Necker, D. Contis, and D. Schimmel, "Tcp-stream reassembly and state tracking in hardware," in *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, 2002, pp. 286–287.

[77] J. Ni, C. Lin, Z. Chen, and P. Ungsunan, "A fast multi-pattern matching algorithm for deep packet inspection on a network processor," in *Parallel Processing, 2007. ICPP 2007. International Conference on*, 2007, pp. 16–16.

[78] J.-T. Oh, S.-K. Park, J.-S. Jang, and Y.-H. Jeon, "Detection of ddos and ids evasion attacks in a high-speed networks environment," *International Journal of Computer Science and Network Security*, vol. 7, no. 6, pp. 124–131, 2007.

[79] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Computer Networks*, 1999, pp. 2435–2463.

[80] V. Paxson, K. Asanovic, S. Dharmapurikar, J. Lockwood, R. Pang, R. Sommer, and N. Weaver, "Rethinking hardware support for network analysis and intrusion prevention," in *Proceedings of the 1st USENIX Workshop on Hot Topics in Security*. Vancouver, B.C., Canada: USENIX Association, 2006, pp. 11–11.

[81] V. Paxson, R. Sommer, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," in *Sarnoff Symposium, 2007 IEEE*, 2007, pp. 1–7.

[82] P. Li, M. Salour, and X. Su, "A survey of internet worn detection and containment," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 20–35, 2008.

[83] M. Peng and S. Azgomi, "Content-addressable memory (cam) and its network applications," in *International IC Taipei Conference Proceedings*, Taipei, 2001. [Online]. Available: www.eetasia.com/ARTICLES/2000MAY/2000MAY03_MEM_NTEK_TAC.PDF

[84] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Comput. Surv.*, vol. 39, no. 1, p. 3, 2007.

[85] J. Pescatore, M. Reynolds, and A. Hallawell, "How to limit damage from the mydoom worm," Gartner, Inc., Tech. Rep., 2004.

[86] P.-C. Lin, Z.-X. Li, Y.-D. Lin, Y.-C. Lai, and F. C. Lin, "Profiling and accelerating string matching algorithms in three network content security applications," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 2, pp. 24–37, 2006.

[87] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Secure Networks Inc, Tech. Rep., 1998. [Online]. Available: http://handle.dtic.mil/100.2/ADA391565

[88] R. L. Richardson and I. Computer Security, *CSI survey 2007 : the 12th annual computer crime and security survey*. [San Francisco, Calif.]: Computer Security Institute, 2007, (San Francisco, Calif.).

[89] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX conference on System administration*. Seattle, Washington: USENIX Association, 1999, pp. 229–238.

[90] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," National Institute of Standards and Technology, Tech. Rep., 2007. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf

[91] D. V. Schuehler and J. Lockwood, "Tcp-splitter: A tcp/ip flow monitor in reconfigurable hardware," in *High Performance Interconnects, 2002. Proceedings. 10th Symposium on*, 2002, pp. 127–131.

[92] D. V. Schuehler and J. W. Lockwood, "A modular system for fpga-based tcp flow processing in high-speed networks," in *14th International Conference on Field Programmable Logic and Applications (FPL)*, 2004, pp. 301–310.

[93] D. V. Schuehler, J. Moscola, and J. Lockwood, "Architecture for a hardware based, tcp/ip content scanning system," in *High Performance Interconnects, 2003. Proceedings. 11th Symposium on*, 2003, pp. 89–94.

[94] S. Shalunov and B. Teitelbaum, "Tcp use and performance on internet2," in *ACM SIGCOMM Internet Measurement Workshop*, San Francisco, USA, 2001.

[95] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using fpgas," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, 2001, pp. 227–238.

[96] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. San Francisco, CA: USENIX Association, 2004, pp. 4–4.

[97] M. Sipser, *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[98] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using fpga," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. Monterey, California, USA: ACM, 2005, pp. 238–245.

[99] I. Sourdis and D. Pneumatikatos, "Fast, large-scale string match for a 10gbps fpga-based network intrusion detection system," in *Lecture notes in computer science*. Berlin; New York: Springer Berlin / Heidelberg, 2003, vol. Volume 2778/2003, pp. 880–889.

[100] I. Sourdis and D. Pnevmatikatos, "Pre-decoded cams for efficient and high-speed nids pattern matching," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, 2004, pp. 258–267.

[101] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, 2010.

[102] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended tcams," in *Proceedings of the 11th IEEE International Conference on Network Protocols*. IEEE Computer Society, 2003, p. 120.

[103] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*. Cambridge, Massachusetts, United States: ACM, 1999, pp. 135–146.

[104] D. C. Suresh, Z. Guo, B. Buyukkurt, and W. A. Najjar, "Automatic compilation framework for bloom filter based intrusion detection," in *Lecture notes in computer science.* Berlin; New York: Springer-Verlag, 2006, pp. 413–418.

[105] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.
[106] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducting of field labels," in *IEEE INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005, pp. 269–280.
[107] S. M. Trimberger, *Field-Programmable Gate Array Technology*. Kluwer Academic Publishers, 1994.
[108] A. K. Tummala and P. Patel, "Distributed ids using reconfigurable hardware," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–6.
[109] G. Vigna, W. Robertson, K. Vishal, and R. A. Kemmerer, "A stateful intrusion detection system for world-wide web servers," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, 2003, pp. 34–43.
[110] D. v. Vuuren, "Death, taxes and worms white paper," Dimension Data Holdings plc, Tech. Rep., 2004. [Online]. Available: http://www.dimensiondata.com/DocumentLibrary/WhitePapers/SecuritySolutions/DeathTaxesWormsWhitePaper.htm
[111] N. H. E. Weste and K. Eshragian, *Principles of Cmos Vlsi Design: A Systems Perspective*, 2nd ed. Addison-Wesley, 1993.
[112] T. Wilson, "Targeted attacks on the rise," *DarkReading.com*, Apr 18 2007. [Online]. Available: http://www.darkreading.com/security/perimeter/showArticle.jhtml?articleID=208804471
[113] S. Wong, S. Vassiliadis, and S. Cotofana, "Future directions of (programmable and reconfigurable) embedded processors," in *In Embedded Processor Design Challenges, Workshop on Systems, Architectures, Modeling, and Simulation - SAMOS*. Marcel Dekker, Inc, 2002.
[114] Y.-D. Lin, H.-Y. Wei, and S.-T. Yu, "Building an integrated security gateway: Mechanisms, performance evaluations, implementations, and research issues," *IEEE Communications Surveys & Tutorials*, vol. 4, no. 1, pp. 2–15, 2002.
[115] F. Yu, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet pattern-matching using tcam," in *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, 2004, pp. 174–183.
[116] S. Yusuf and W. Luk, "Bitwise optimised cam for network intrusion detection systems," in *Field Programmable Logic and Applications, 2005. International Conference on*, 2005, pp. 444–449.
[117] S. Yusuf, W. Luk, M. Sloman, N. Dulay, and E. Lupu, "A combined hardware-software architecture for network flow analysis," in *IEEE International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'05)*, Las Vegas, USA, 2005.
[118] S. Yusuf, W. Luk, M. Sloman, N. Dulay, E. C. Lupu, and G. Brown, "Reconfigurable architecture for network flow analysis," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 1, pp. 57–65, 2008.

Hao Chen is a Ph.D. student majoring in Electrical Engineering at the State University of New York (SUNY) at Binghamton. He received his B.S. degree from Zhejiang University, China in 2003. He earned his M.S. degree in Electrical Engineering from SUNY - Binghamton in 2007. His current research is focusing on the Network Security, Novel Infrastructure for the Internet, Reconfigurable Hardware-based Internet Infrastructure Security. Mr. Hao Chen could be reached at hchen8@binghamton.edu.

Dr. Yu Chen is an Assistant Professor of Electrical and Computer Engineering at the State University of New York - Binghamton. He received the Ph.D. in Electrical Engineering from the University of Southern California (USC) in 2006 under the supervision of Professor Kai Hwang. His current research interest lies in Network Security, Distributed/Grid Computing, and Reconfigurable/Embedded Computer Architectures. Particularly, his work covers security in networks and large distributed/Grid computing systems; Internet security protocols; infrastructure security; trust, security and privacy in wireless, embedded and pervasive computing. He has authored or coauthored more than 40 scientific papers in refereed journals, conferences, and book chapters. He is a member of ACM, IEEE and SPIE.

Dr. Douglas H. Summerville is an Associate Professor in the Department of Electrical and Computer Engineering at the State University of New York at Binghamton. He was formerly an Assistant Professor in the Department of Electrical Engineering at the University of Hawaii at Manoa. He received the B.E. Degree in Electrical Engineering in 1991 from the Cooper Union for the Advancement of Science and Art, and the M.S. and the Ph.D. degrees in Electrical Engineering from the State University of New York at Binghamton in 1994 and 1997, respectively. He has authored over 35 journal and conference papers and two textbooks on embedded systems design. He is a senior member of the IEEE and a member of the ASEE. He received the State University of New York Chancellor's Award for Excellence in Faculty Service and two teaching excellence awards. His research and teaching interests include microcontroller systems design, digital systems design and computer and network security, and tamper detection.