A Novel Embedded Accelerator for Online Detection of Shrew DDoS Attacks

Hao Chen, Yu Chen*
Department of Electrical and Computer Engineering
State University of New York - Binghamton, Binghamton, NY 13902, USA

ABSTRACT

As one type of stealthy and hard-to-detect attack, low-rate TCP-targeted DDoS attack can seriously throttle the throughput of normal TCP flows for a long time without being noticed. The Power Spectral Density (PSD) analysis in frequency domain can detect this type of attack accurately. However, computational complexity of PSD analysis makes it impossible for software implementation at high speed network. Taking advantages of powerful computing capability and software-like flexibility, an embedded accelerator using FPGA for PSD analysis has been proposed. Optimized design in autocorrelation calculation algorithm and DFT processing distinguishes our scheme more meaningful for high speed real-time processing with limited resources. Simulation verifies that even working at very low system clock frequency, our design can still provide quality-service for malicious detection in multi-gigabyte rate network.

1. INTRODUCTIONS

Low-rate TCP-targeted distributed denial-of-service (DDoS) attacks [9], [13] are categorized as a new type of hard-to-detect, stealthy attack. This kind of attack can throttle the throughput of TCP flows to as low as 10% of its normal bandwidth usage, and may last for a long time before victims realize its existence. They are also referred as shrew attacks [13], or pulsing attacks [14] in literatures. In this paper, we will use the term "shrew attacks" for simplicity. Shrew attacks take the advantage of time-out mechanism in TCP protocol to create illusory congestions. Sending bursts with a high pulse rate while keeping a relative low average data rate, it deludes normal TCP flows to always "see" a busy link when they recover from RTO and makes them to be dropped in the end. Due to this subtle behavior, it is hard to detect shrew attacks via simple volume-monitoring in time domain.

As shown in Fig. 1, a shrew attack stream is modeled by three major parameters including *period of attack T*, *width of burst L*, and the *burst rate R*. The period T is the time interval between two consecutive attack pulses. The burst width L indicates the time period during which attackers send packets in high rate. The burst height exhibits the peak rate by which attacking flow is sent. The period T is calculated by the estimated TCP RTO timer implementation from trusted sources. During the burst with a peak rate R, the shrew pulses create a burst and severe congestion on the links to the victim. The legitimate TCP flows must decrease their sending rate as governed by rate-limiting mechanism, accordingly.

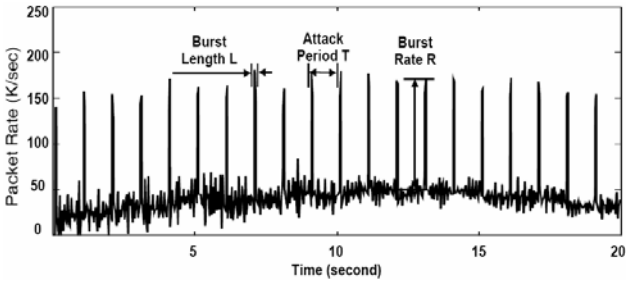

Figure 1. An illustration of typical TCP traffic flows mingled with shrew attacks.

Through Power Spectrum Density (PSD) analysis in frequency domain, shrew attacks can be easily detected [9]. The PSD analysis reveals the fact when the data traffic bears shrew attacks, majority of its energy (> 70%) appears at low frequency band (< 20 Hz). On the contrary, less energy (< 40%) is shown up at the low band. A collaborative distributed detection mechanism has been proposed in our previous work [9], which can accurately detect on-going shrew attacks using spectrum analysis. The results of previous experiment show that more than 95% shrew attacks are detected with a false positive rate of 10%. Since the PSD is obtained through the Fourier transform on the autocorrelation function, steps to achieve PSD are straightforward. We treat the arrival of packets in a time fraction as a random process. After calculating the autocorrelation, we convert it to frequency domain via Discrete Fourier Transform (DFT).

Scalability is always one of the critical concerns for real applications, since only limited resources can be shared to execute security functions in core routers. The computational complexity introduced by autocorrelation and DFT raises a new challenge for high volume data

* Manuscript submitted on April 04, 2008 to *The 2008 International Conference on Networking, Architecture, and Storage* (NAS 2008), Chongqing, China, June 12 – 14, 2008. Corresponding author: Yu Chen, Dept. of ECE, SUNY – Binghamton, Binghamton, NY 13902. E-mail: ychen@binghamton.edu, Tel.: (607) 777-6133.

processing. Meanwhile, the capability of real-time processing should be kept. Security solutions are expected to handle malicious attacks before they cause damages. It is preferable to contain attacks even before they touch the end users. In general, most software based security mechanisms are no longer viable against attacks from core networks with multi-gigabyte data rate. Like Snort, Bro, and WebSTAT, these software-based security solutions are limited to process data at rates less than 100Mbps [17].

Field Programmable Gate Array (FPGA) devices feature both capabilities of powerful computing and software-like reprogramming, providing the great flexibility for high performance implementation with fast development cycles. Hardware nature of FPGA inherits the potential for high speed pipeline and parallel processing, making the possible of innovations for dedicating architectures. Reconfiguration of FPGA allows emerging new security solutions can be easily integrated.

Fast increasing network speed pushes security tasks down to the lower hierarchy level of packet-processing for reaction. Suspicious patterns concealed in data traffic are expected to be recognized before the traffic hits the routing fabric. Dedicated accelerators are preferred for real-time detection and work in parallel with other routing jobs. Up to now, many research efforts on embedded accelerators using FPGA have been put and solid achievements have been made in recent years [1], [3], [18], [19]. It is clear that reconfigurable hardware based implementation has been the trend in this area.

In this paper, a novel embedded accelerator using FPGA has been proposed based on our previous work in shrew attack detection algorithm [9]. In fact, it is also part of our current effort in self-adaptive architecture for network infrastructure security [7], [8]. This design focuses on the kernel part of abnormal detector where intensive computation is requested. It consists of two major processing parts: autocorrelation and DFT. The rationale of design is to keep the high utilization of limited resources as well as high speed real-time processing. Two improvements have been made for both parts. Simulation results shows that even working at very low system clock frequency, like 5 MHz, the design can still provide quality-service for malicious detection in multi-gigabyte rate network.

The rest of the paper is arranged as follows: Section 2 reviews related works briefly; Section 3 presents the system architecture of accelerator after an introduction to the rationale of frequency domain shrew attack detection; Section 4 discusses the improved algorithms for autocorrelation and DFT processing; Section 5 evaluates the performance based on our simulation results; Section 6 wrap up this paper with conclusions and discussion of future works.

2. RELATED WORKS

Many researches have been reported relating to network infrastructure security using reconfigurable hardware. The common motivation moving from software-based solutions to hardware-based solutions is to keep the pace with increasing network speed. Diverse work has covered major steps of traffic analysis at high speed network for security purposes. They can be roughly categorized into three sub-areas: stream rearrangement [1], [17], header processing for packet classification [19], [20], [23], and deep content processing for anomaly detection [3], [11], [12], [21].

Stream rearrangement is dedicating for flow reorganization which reshapes the received streams to well-organized flows. During high-speed transmission in network, packets may be dropped, duplicated, or re-ordered. In addition, complicated routing topology may let packets of the same flow to be routed through different paths. As a result, received packet sequences may be quite different from the original ones. It is inefficient to perform advanced processes based on such disturbed sequences. The major job of stream rearrangements includes stream reordering, flow reassembly, and state tracking [1], [17].

Packet classification is another important application. Packets can be classified by source and destination ports, address or protocol type contained in the packet header. A common requirement for packet classification applications is that it can classify packets based on packets headers. The dedicating applications should maintain rule databases for all the rules, one for each flow to match the packet headers for efficient routing. If the information of packet header does not match the database, the packet will be considered as an anomaly packet and be shunted for further inspection or dropped.

Considerable interests are focusing on malicious detection, since it is the kernel of traffic analysis. Modern threats towards network infrastructure make header inspection insufficient for malicious detection [1]. These malicious threats can be well concealed inside the payload of packets, like DDoS attacks, worm attacks. Deep content processing is necessary for detecting the malicious packet by checking the payload.

Currently, most deep content approaches follow one of the following schemes: signature matching [3], [11], [21] and abnormal detection [12], [19]. Signature matching scheme directly compares the incoming payloads with signature patterns in inner signature database. Detection is made if any match of signature patterns is found. Anomaly detection scheme performs macro-analysis other than exact match for individual packet, so false positive rate is introduced. It compares certain parameters of incoming packets or flows with known thresholds for judgment under an acceptable false positive rate.

A plethora of hardware implementations have been reported with signature matching scheme, since this scheme is straightforward for adaptation and efficient under most cases. Researchers in Open Network Laboratory (ONL) attempt their efforts to transplant software-based SNORT to hardware-based version [1]. Though the mainstream

approaches for deep content processing follow this track, it is inefficient for shrew attacks detection.

Anomaly detection schemes also work in frequency domain. However, the conversion from time domain to frequency domain increases its computational complexity. Two analysis methods have been reported along this track, they are Power Spectrum Density (PSD) [9], [10] method and wavelet method [4]. Since the outcomes of wavelet method are highly dependent on the choice of detection parameters, it is difficult to find optimal parameters that are sensitive enough to detect low-rate distributed attacks while maintaining an acceptable false positive alarm rate [4]. Though the idea of PSD method has been mentioned for identifying normal TCP flows from adverse network environment [10], no report shows that it has been applied for reconfigurable hardware implementation in DDoS attack detection to the best of our knowledge.

3. ACCELERATOR ARCHITECTURE

Starting with an introduction to the rationale of our spectral analysis scheme to detect the existence of shrew DDoS attack flows embedded in normal traffic flows, this section also presents the framework of the embedded reconfigurable hardware accelerator.

3.1 Spectral Analysis Principle

In highly distributed shrew DDoS attacks, malicious flows are embedded in the huge amount of normal traffic flows. The anomalies are not very obvious on aggregate traffic level. Meanwhile, routers cannot afford to monitor traffic on flow or packet level, where flow is defined as the set of packets with same five tuple {source/destination IP addresses, source/destination port numbers, protocol}. We proposed to monitor the traffic on a level between the two extremes. We define the term super flow to describe all packets sharing the same prefix in their destination IP address. For convenience, in following when we say flow actually we mean super flow.

For a given flow, we treat the number of packet arrivals in a time slot as a stochastic process called *packet process* [10]: $\{x(t)|\ t = n\ \Delta,\ n \in N\}$, where Δ is a constant time interval, which we assume 1 *ms*. N is a set of positive integers, and at each time point t, $x(t)$ is a random variable, representing the total number of packets arrived at a router in $(t-\Delta,\ t]$. We assume a wide sense stationary random process. We define the autocorrelation function of the random signal $x(t)$ in discrete time as follows:

$$R_{xx}(m) = \frac{1}{N-m} \sum_{n=0}^{N-m-1} [x(n)x(n+m)] \qquad \textbf{(1)}$$

$R_{xx}(m)$ captures the correlation of the packet process and itself at interval m. If there is any periodicity exists, autocorrelation function is capable of enforcing it. The next step is to figure out the periodicity embedded inside the autocorrelation functions. The Fourier transform of the autocorrelation sequence brings the useful interpretation of the frequency distribution of the signal. We convert the autocorrelation time series by DFT to generate the PSD:

$$Y(k) = \frac{1}{N} \sum_{n=0}^{N-1} R_{xx}(n) \times W_N^{kn}, \qquad k=0,1,2,...,N-1 \qquad \textbf{(2)}$$

where $W_N^{kn} = e^{\frac{-j2\pi kn}{N}}$.

Figure 2 compares the PSD for two traffic stream patterns corresponding to with and without shrew attacks. It is clear that the embedded shrew attack stream shifts the solid-line PSD curve towards the lower frequency band, while the non-attack stream has a wider frequency range of the traffic density in the dash-line curve. By studying the PSD of more than 8000 Internet traffic streams [9], we obtained the statistical energy distribution patterns. Under a hypothesis-testing framework we can detect shrew attacks with accuracy as high as 95% at the cost of 10% false positive rate.

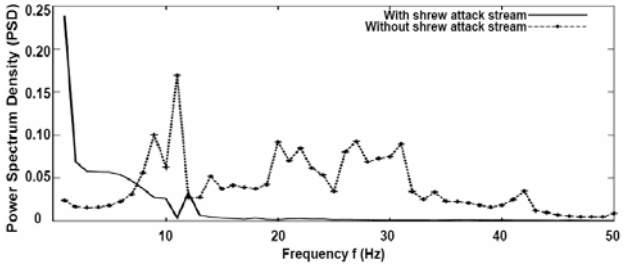

Figure 2. Comparison of the normalized PSD of traffic streams with/without embedded shrew attacks

3.2 Accelerator Architecture

Figure 3 illustrates the main function blocks in the embedded shrew DDoS attack detection accelerator system. It is connected to the network card and counting the incoming packets. The Packet Counting unit consists of several simple counters, each of them corresponding to one flow. The sampling unit records the counter values periodically, which is set as 1 *ms*. When the length of sampled series reaches 4096 for certain flow, we calculate its autocorrelation sequence and convert it into frequency domain using DFT. Comparing the PSD of sampled flow with the statistic pattern obtained, we can detect the existence of shrew attack flows. In this paper, we focus on the two most critical and time-consuming parts: the autocorrelation calculation unit and DFT convert unit.

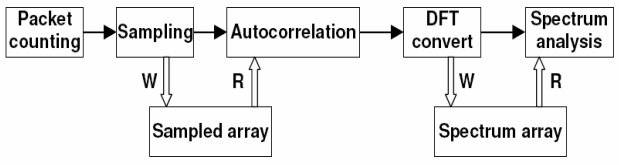

Figure 3. Function blocks of shrew DDoS attack detection system

Our previous research revealed that the length of sampled series has significant impact on the detection accuracy [9]. With the sampling period of 1 *ms*, a series of 4096 is necessary if a detection rate of 95% is desired. Therefore, we need to calculate the autocorrelation sequence and FFT of 4096 data points. The detection delay is another important concern. The shrew attacks achieve the maximum impact against TCP throughput when its period is in the range of 0.5 to 1.0 second. In addition, the TCP throughput drops quickly after several attacking pulses.

Hence, it is desired to obtain the traffic spectrum in a period higher than 0.5 second. Thus we have enough time to launch flow-filtering mechanisms to segregate malicious attacking flows from legitimate ones before damage caused. To monitor multiple flow in parallel, the smaller the area each unit occupies, higher flow resolution is achieved. Within such context, the expected performance metrics in autocorrelation and FFT calculation units design are low delay, high throughput, and area efficient, in the order of priority from high to low.

4. ALGORITHMS AND DESIGN

In this section, our approach for hardware accelerator has been discussed. This accelerator is the kernel part of anomaly detection for deep content processing in our self-adaptive architecture for network infrastructure security [7]. It consists of two parts: autocorrelation calculation unit and DFT conversion unit, and is modularized. Taking the well-organized data sequence from previous stage, the accelerator speed the PSD analysis with dedicated FPGA device and output the result for decision. Two improvements in both autocorrelation and DFT processing have been discussed in detail.

4.1 Autocorrelation Calculation Units

In order to efficiently use valuable resources, we develop a reusable mechanism based on normal autocorrelation algorithm. The idea is from the observation that some intermediate results from the previous cycle during autocorrelation can be reused to the following cycle. Since the sampled data sequences are fed with fixed time period, operation of autocorrelation will be pretty regular, which is very suitable for hardware implementation.

As defined by Equation (1), $R_{xx}(m)$ is the sum of (N - m) items. Each of the items is obtained by the sampled series' i-th entry multiplied with the (i+m)-th entry. If we calculate the autocorrelation sequence every 0.5 second, there are n = 500 samplings replaced.

Therefore, when N - $n \geq m$, there are (N – n - m) items can be reused in the calculation of next $R_{xx}(m)$ as illustrated in Fig. 4. In another word, each time for $R_{xx}(m)$, we only need to calculate n new items. When N - $n \leq m$, the new items need to calculate is N - m. Then for the whole autocorrelation sequence of length N, the total number of new items needs to be calculated is:

$$p = \sum_{m=0}^{N-n} n + \sum_{m=N-n+1}^{N} [n-(N-m)] = n(N-n) + \frac{(n-1)(n-2)}{2} = (N-\frac{3}{2})n - \frac{n^2}{2} + 1 \quad (3)$$

The total number of re-useable intermediate results is shown by Equation (4).

$$\sum_{m=0}^{N-n} (N-n-m) = \frac{(N-n)(N-n-1)}{2} \quad (4)$$

Additionally, in order to optimize the performance, it is not expected to repeat the calculation every time. We need some more storage space for these partial results. As shown in Fig. 4, when m > (N - n) the $R_{xx}(m)$ is calculated using purely new samples. So we need (N - n) storage space to save the partial results for $R_{xx}(m)$ where $m \leq (N - n)$. Therefore, the total storage space needed for intermediate results is:

$$q = \frac{(N-n)(N-n-1)}{2} + (N-n) = \frac{(N-n)(N-n+1)}{2} \quad (5)$$

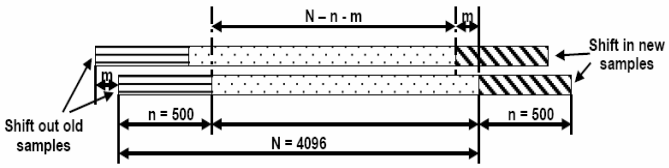

Figure 4. Re-useable entries in autocorrelation calculation (Computing the autocorrelation sequence per 0.5s, n = 500)

Based on the analysis above, we propose the algorithm to calculate the autocorrelation sequence $R_{xx}(m)$ periodically. In each cycle, the multiplication operations are needed for every new sampled entry $x(j)$. The new multiply results are added with the previously obtained partial sum. Then dividing the new summation results by the parameter indicated in Equation (1). And then the part of re-useable sums are re-generated and stored for next calculation. Using this algorithm, we can generate the autocorrelation sequence quickly. Figure 5 illustrates the autocorrelation calculation algorithm. One advantage of this design is that we saved number of multipliers tremendously.

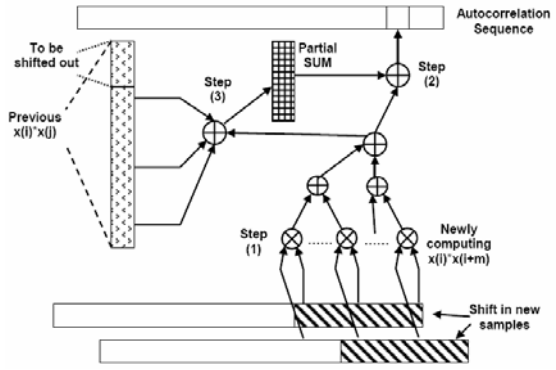

Figure 5. Autocorrelation calculation algorithm

4.2 DFT Conversion Unit

As we mentioned before, the most computational intense part for PSD analysis is DFT Conversion. Performing efficient DFT conversion with acceptable cost is truly essential to the implementation of real-time system (RTS). Hardware core offers the powerful computation capability. Taking the advantage of this capability, the major bottleneck (processing efficiency) of software implementation can be easily cracked, and the performance of DFT processing will be dramatically increased. It has become a trend using hardware DFT solution for high performance RTS design. To make the analysis clearer, we omit the factor 1/N at the beginning of equation (2). The simplified equation is:

$$Y(k) = \sum_{n=0}^{N-1} R_{xx}(n) \times W_N^{kn} \qquad (6)$$

The complexity of this equation is $O(N^2)$. It increases exponentially with respect to the length N of input sequence. The cost of direct DFT operation will be prohibitively high under the processing of high volume data sequence, even for the hardware implementation. Optimal algorithms are expected. The typical approaches use FFT based algorithms which disassemble the original input sequence into several smaller sub-sequences for DFT operation. The complexity of such FFT algorithms approximates $O(Nlog_2^N)$. Comparing the complexity with original DFT algorithm, it shrinks impressively when the length of data sequence increases. These algorithms are very suitable for high volume data sequence processing.

Another consideration is that the input data sequence only contains real values, but DFT processing contains complex operation. A better way is desired to perform the DFT processing instead of setting all imaginary parts equaling to zero. In addition, the design will be mapped to a concrete hardware device, processing capability and compatibility of that hardware core should be considered.

With above concerns in mind, we propose a modular DFT design as shown in Fig. 6, which consists of two parts, 2N-point real-valued DFT processing and inner N-point FFT processing. The function can be fulfilled through three major steps:

1) 2N-point sequence Decimation;
2) N-point FFT operation;
3) 2N-point DFT sequence generation.

It decimates a 2N-point sequence to a N-point sequence, and then performs FFT for this N-point sequence, finally recovers 2N-point DFT sequence by exploiting the symmetry properties of the N-point DFT sequence.

The first and the third steps belong to part (1), and the second step belongs to part (2). Because each part has distinguished boundary, it is easy for modular design which is desirable for hardware application. Fig. 6 illustrates the system architecture of our approach.

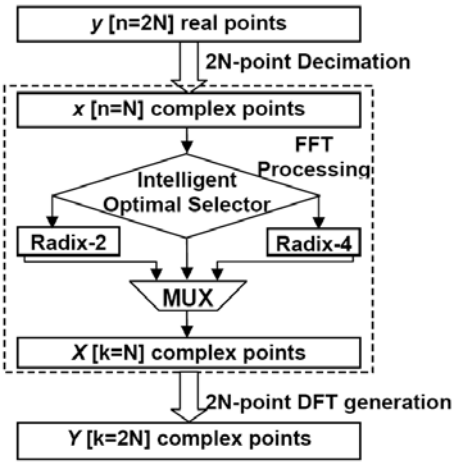

Figure 6. System architecture of DFT unit

4.2.1. 2N-point Real-valued DFT Processing

In many real applications, the input sequences only contain real-valued data. DFT is an operation working on the complex system. A naive approach to apply DFT for real-valued data is that appends zero imaginary part to the end of real part, and then carries out DFT. It is straightforward but inefficient.

Inspired by the efficient real data processing approaches discussed in [6], [15], [16], we propose a novel solution that is more efficient both from the perspective of time and area. Basically, it splits and reorganizes the original 2N-point real sequence to one N-point complex sequence; FFT is applied to this N-point sequence; the first half of original 2N-point DFT values are obtained from this N-point FFT values with additional computation and the second half values are obtained from the first half values via symmetry property.

The major steps belong to two parts haven been listed. The focus here is on the part dedicating 2N-point sequence decimation and generation, which includes step 1 and 3.

Due to the tedious equation deduction all the way in the processing, we skip the theoretical part. Only the basic principle will be presented here for a clear view of the whole picture. Interested readers can find more detailed deductions in our technical report [8].

(1) If the length of a real sequence $y(n)$ is an even number, then this sequence can always be represented as a 2N-point $y(2N)$ sequence, where $2N = n$. We also assume:
$Y(k) = \mathrm{DFT}\{y(n)\}$ $(k = 0, 1, 2, \ldots, 2N\text{-}1)$

(2) DFT is an operation based on complex-valued data, where $x(n) = a(n) + jb(n)$. Take this advantage, we decimate the $y(2N)$ to two N sequences $y(n_1)$ and $y(n_2)$ in terms of $y(2N)$ and $y(2N+1)$. Then, form an N-point complex-valued sequence:
$x(n) = y(n_1) + jy(n_2)$ $(n = 0, 1, 2, \ldots, N\text{-}1)$

(3) A FFT operation is performed on this N-point sequence. Here, we consider it as a black-box, and the FFT result is obtained:

$$X(k) = \mathrm{FFT}\{x(n)\} \qquad (k = 0, 1, 2, \ldots, N\text{-}1)$$

(4) Exploit the relations between first half of original $Y(k)$ and $X(k)$ obtained from above, an equation can be written as:

$$Y(k) = M_1 X^*(k) + M_2 X^*(N\text{-}k) \qquad (k = 0, 1, 2, \ldots, N\text{-}1)$$

Where M_1 and M_2 are can refer to pre-calculated lookup table for speedup. From this equation, the first half original DFT values (i.e.: N points) can be obtained.

(5) Take the advantage of complex symmetry property, the rest half of 2N-point can be obtained from the first half points:

$$Y(2N\text{-}k) = Y^*(k) \qquad (k = 0, 1, 2, \ldots, N\text{-}1)$$

Trough these fives steps, the DFT processing of a 2N-point real sequence has been achieved. The advantage of this processing stems from that it explores the characters of complex-valued data to reduce the total operation points by half at the very begin; and only *N* points need to do real FFT processing.

4.2.2. Inner N-point FFT processing

The inner N-point FFT processing part is a unit dedicating FFT operation. It runs independently after receiving an N-point sequence from the interface. Depending on different criterions, the intelligent selector will choose the proper algorithm applying for FFT operation.

Due to the heavy computing load and prohibitive cost, it is not a recommendation to perform the DFT processing with long data sequence directly. Instead, many algorithms have been proposed to reduce the complexity. FFT algorithms have been great developed and widely applied. They derive from the idea called Divide-and-Conquer which decomposes long input sequence to several short sub-sequences for processing. In this manner, the complexity can be reduced to $O(\frac{n}{N}\log_N^n)$ instead of $O(n^2)$ from the original one.

The well-known FFT algorithms are Radix-2N serial algorithms, which perform FFT operations on even number sequences. Radix-2 is the most compatible Radix-2N algorithm. A restriction for compatible issue is that any FFT processing applicable for high Radix-2N will always be applicable for the lower Radix-2N algorithms, but it may not be true from the reverse side. Because the length sequence of 8-powerd value can always be disassembled to 4-power or 2-powered value, but it may not true from the reverse side. Radix-4 algorithm is considered as the good balance between high efficiency, low cost and better compatibility. It requires no multiplications but performs 4 points operation simultaneously on the sequence with a length of 4-powered value.

Radix-2N algorithms have inherent commonness. They share the same pre-calculated coefficients and the same storage memory which is called in-place storage. It means different Radix-2N algorithms can be applied to the same data sequence without changing any outside conditions as long as the preliminary restriction is met. Taking this advantage, we can selectively choose the optimal Radix-2N algorithm for certain purpose.

As mentioned, Radix-2 has the best compatibility, and Radix-4 contains much better efficiency. Combing both together with good balance, it is convinced that more reliable and flexible hardware implement can be designed.

Based on this idea, we develop an intelligent inner FFT processing unit. This unit consists of a set of logic mechanisms and a set of Radix-2N algorithm parts. The former deals with optimal selection; and the later handles detail calculation. Radix-2 and Radix-4 algorithms are always integrated inside for above reasons; other Radix-2N algorithms can also be plugged in if necessary. As long as the interfaces are the same, it is convenient to do such modification because all Radix-2N algorithms can be considered as black-boxes.

We employ the Xilinx IP core as a black box to perform inner FFT processing. Taking the great advantage of system supported IP core, the performance has been noticeably improved. The logic design takes the maximum consideration of the situations which will be met under real environment. We believe this kind of design style has strong compatibility.

5. SIMULATION RESULT

We coded and simulated our design on Xilinx ISE 7.1 platform with Vertex-4 XC4VFX12 as designated FPGA device. Since this accelerator is only one part of self-adaptive architecture for network infrastructure security [7], its interface is designed for data flows that have been pre-processed by stream rearrangement unit. In order to evaluate the design and obtain the performance, we setup the test environment where accelerator can directly run.

The assumption as follows: (1). Data traffic fed into the accelerator is well-organized. Since accelerator has no ability for rearrangement, we input integrated data flows for simplicity. (2) The sampling period for is 1 *ms*, and 4096 points are sampled each cycle. 1*ms* sampling is fast enough to most real sensors, this setting is fair for simulation. Normal TCP flows contains less 1024 packets, 4096 points' sampling capacity is big enough to hold all the packets. (3) The replacement for the next sequence is 500 points. This is most common case when performs packets sampling during inspection, not all the sampling points will be replaced at a time. (4) The design is running with 5MHz frequency. It is a courteous frequency in terms of modern hardware device.

We run it with such a low frequency in order to know whether the accelerator still works under relative poor

conditions. With the assumption of fixed sampling period we made above, low frequency system clock means less processing cycles that the accelerator can have to perform the operations. For example, a system running under 5MHz for 1 *ms* has only 2000 clock cycles for use, but the system running under 50MHz has 20000 clock cycles for use during the same time period. The guideline of our simulation assumptions is to provide a relative fair test environment. In fact, PowerPC 405 Core embedded in Xilinx Virtex-4 FPGA runs up to 450 MHz, which provides enough power to our design.

From the perspective of power and resource efficiency, major system resources will be "sleeping" or be reserved for other purposes during data sampling. A specific data buffer accepts and holds the valid data points. It copies the data set to a high speed Block RAM on board. Meanwhile, system resources are re-invoked to be ready for full power processing. Taking the advantage of these periods, several look-up tables are set up to accommodate the high-frequently used data during DFT processing. Instead of on-the-fly-calculation, FPGA devices are much good at look-up-table searching in nature. By reducing the computing complexity, the process can be further speeded up.

Roughly speaking, the upper limit time boundary for "real-time" processing should be the interval between two start points of different data sequences. In our case, the total available processing time is 4096 *ms* for the first sequence or the replacement points multiplied by the sampling rate for the rest sequences, which is 500 *ms*.

During Autocorrelation, efforts have been made to continue reducing resource consumption and processing time. With our method, only the first set of autocorrelation need to be calculated in full as the normal one, the rest sets can always take partial results from their ancestors. The evidence is much clear when the ratio of length and replacement points keeps large. For length equal to 4096 points and replacement for each time is 500 points. The normal fix length multiplication takes 6.963 *ms* for one set of autocorrelation, while our method only takes 1.881 *ms* except the initial sequence, running under the same evaluation environment.

In order to convert 2-N real points to N complex points, conversion coefficients are indispensable. Since these coefficients are only depended on the length of sequence, they could be fixed as long as the length of operation sequence is determined. Building such kind of coefficient table for explicit conversion, it can be used for good. We perform this step at the beginning after system reset or reboot. Most system resources are at idle at that time. The simulation result shows that this step can be done at 1.008 *ms*.

Our experiment compared both Radix_2 and Radix_4 FFT IP cores. With Radix_4, it takes around 3 *ms* to process the 2N sequence of 4096 points. With Radix_2, it takes around 1.5 *ms* to process N sequence of 2048 points. We adopted Radix_2 in our project, even though the total time of using Radix_2 IP core is longer than that using Radix_4. The major concern is still the efficiency, to keep the cost low with the job done.

Under the processing of recovery 2N-point, the first N points are obtained from the multiplication of values from M lookup table and the data from N-point FFT. It only takes four multipliers and 3 adders to perform this step. The rest N points is obtained by explore the symmetry properties of complex number. Comparing to FFT operation, dramatic resources can be saved, but the cost is its delay. Our simulation result shows that 3.5 *ms* is needed to perform a 2N-point sequence with the length equation to 4096, including 2.5 *ms* for the first N points and 1 *ms* for the second N points.

Finally, the total time spending on major parts of our design would be:

$$T_{TOTAL} = T_{AC} + T_{FFT} + T_{RECOVERY} = 6.88\,ms$$

With 6.88 *ms* processing time out of 500 *ms* interval, it is good enough to handle the real-time detection.

6. CONCLUSIONS

In this paper, an embedded reconfigurable hardware accelerator has been designed for the detection of shrew DDoS attack in high-speed network. The principle of this design is based on the Power Spectrum Density (PSD) analysis in frequency domain. The proposed accelerator architecture consists of modularized autocorrelation unit and DFT conversion unit. Through two optimized design, our algorithms can drastically improve the performance in the processing of PSD analysis as well as the utilization of limited resources. The synthesis and simulation results verified that our design can cope with high data rate in today's network.

In fact, in this paper we just report our preliminary results. This work is part of our effort in exploring a self-adaptive architecture for network infrastructure security [7]. The anomaly and malicious attack detection based on statistical traffic flow analysis is a critical components in an adaptive network infrastructure design. Actually, this is merely the initial work towards real-time processing using reconfigurable hardware. Further improvements are to be done and through which a better performance can be achieved by optimizing the structure of design.

In our ongoing efforts, we are integrating our accelerator into the Field Programmable Port Extender (FPX) test-bed developed by the Open Network Laboratory (ONL) at Washington University in St. Louis [24], where we will evaluate the performance of the accelerator in real network environment.

Our future efforts will focus on using techniques combing pipeline and parallelism to relieve time constraint in the critical path and perform multi-threads execution; other algorithm or architecture innovations will be convinced to explore the unique features of hardware

design. On the other hand, a more sophisticated test-bed is under construction, which will be great help for testing. Currently we have expanded the LISAR testbed at SUNY – Binghamton with NetFPGA boards, which is developed at Stanford University as a reconfigurable hardware platform optimized for high-speed networking.

The NetFPGA includes the all of the logic resources, memory, and Gigabit Ethernet interfaces necessary to build a complete switch, router, and/or security device [25], [26]. Because the entire datapath is implemented in hardware, the system can support back-to-back packets at full Gigabit line rates and has a processing latency measured in only a few clock cycles. With this test-bed, more complicated scenarios can be investigated in more detail.

REFERENCES

[1] M. Attig and J. Lockwood, "A Framework For Rule Processing in Reconfigurable Network Systems", *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, April 17-20, 2005.

[2] F. Baboescu and G. Varghese, "Scalable Packet Classification", SIGCOMM'01, August 2001, San Diego, CA.

[3] Z. Baker and V. Prasanna, "A Methodology for the Synthesis of Efficient Intrusion Detection Systems on FPGAs", In *Proceedings of the Twelfth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2004* (FCCM '04), 2004.

[4] P. Barford, J. Kline, D. Plonka, and A. Ron, "A Signal Analysis of Network Traffic Anomalies," *Proc. Internet Measurement Workshop*, 2002.

[5] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A Fast Single-Chip Implementation of 8192 Complex Point FFT," *IEEE Journal of Solid-Sate Circuits*, Vol. 30, No. 3, March 1995.

[6] C.S. Burrus and T.W. Parks Fds, "DFT/FFT and convolution algorithms: theory and implementation", *Wiley*, 1985.

[7] Y. Chen and H. Chen, "NeuroNet: An Adaptive Infrastructure for Network Security", *International Journal on Adaptive Infrastructures, Special Issue on Intelligent Systems for Adaptive Infrastructures*, accepted to be published in 2008.

[8] H. Chen and Y. Chen, "A Real Time Shrew DDoS Attack Detection Accelerator using Reconfigurable Hardware Devices", *Technical Report, Dept. Electrical and Computer Engineering,* SUNY – Binghamton, September 2007.

[9] Y. Chen and K. Hwang, "Collaborative Detection and Filtering of Shrew DDoS Attacks using Spectral Analysis," *Journal of Parallel and Distributed Computing, special issue on Security in Grids and Distributed Systems*, Vol. 66. No. 9, September 2006.

[10] C. M. Cheng, H. T. Kung, and K. S. Tan, "Use of spectral analysis in defense against DoS attacks," *Proc. of 2002 IEEE GLOBECOM*, Taipei, Taiwan.

[11] C. Clark and D. Schimmel, "Scalable pattern matching for high speed networks", in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM'04), 20-23 April 2004 Page(s):249 – 257

[12] B. Hutchings, R. Franklin, and D. Carver, "Assisting Network Intrusion Detection with Reconfigurable Hardware", in *Proceedings of 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, April 22-24, 2002, pp. 111 – 120

[13] A. Kuzmanovic and E. W. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks—The Shrew vs. the Mice and Elephants," in *Proceedings of ACM SIGCOMM 2003*, Aug. 2003.

[14] X. Luo and R. Chang, "On a New Class of Pulsing Denial-of-Service Attacks and the Defense," *Proc. of NDSS'05*, San Diego, CA., Feb. 2-5, 2005.

[15] R. Matusiak, "Implementing Fast Fourier Transform Algorithms of Real-Valued Sequences With the TMS320 DSP Platform", Texas Instruments, August 2001

[16] J. Proakis and D. Manolakis, "Digital signal processing: principles, algorithms, and applications", Pearson Prentice Hall, 2007.

[17] D. Schuehler and J. Lockwood, "A Modular System for FPGA-based TCP Flow Processing in High-Speed Networks", *14th International Conference on Field Programmable Logic and Applications* (FPL), Springer LNCS 3203, Antwerp, Belgium, August 2004.

[18] M. Sima, S. Vassiliadis, S. Cotofana, J. van Eijndhoven, and K. Vissers, "Field-Programmable Custom Computing Machines: A Taxonomy", in *12th Conference on Field Programmable Logic and Applications*, Montprllier, France, 2002

[19] H. Song and J.W. Lockwood, "Efficient Packet Classification for Network Intrusion Detection using FPGA", *International Symp. on Field-Programmable Gate Arrays* (FPGA'05), Monterey, California, Feb 20-22, 2005.

[20] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," in *Proceedings of IEEE International Conference on Network Protocols* (ICNP), 2003.

[21] Y. Sugawara, M. Inaba, and K. Hiraki, "Over 10Gbps String Matching Mechanism for Multi-stream Packet Scanning Systems", *14th International Conference on Field Programmable Logic and Applications* (FPL), Springer LNCS 3203, Antwerp, Belgium, August 2004.

[22] D. Taylor, "Survey & Taxonomy of Packet Classification Techniques," *Tech. Rep. WUCSE-2004-24, Department of Computer Science & Engineering*, Washington University in Saint Louis, May 2004

[23] D. Taylor and J. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field Labels", *IEEE INFOCOM* 2005

[24] Open Network Laboratory, Washington University in St, Louis, http://onl.arl.wustl.edu

[25] J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA – An Open Platform for Gigabit-rate Network Switching and Routing," *IEEE International Conference on Microelectronic Systems Education* (MSE'2007), June 3 – 4, 2007, San Diego, CA., USA

[26] NetFPGA official homepage, http://yuba.stanford.edu/NetFPGA/, as of Feb. 1, 2008