An Optimized Design of Reconfigurable PSD Accelerator for Online Shrew DDoS Attacks Detection

[†]Hao Chen, [†]Yu Chen, [†]Douglas H. Summerville, [‡]Zhou Su

[†]Dept. of Electrical and Computer Engineering, SUNY - Binghamton, Binghamton, NY 13902, USA
[‡]Dept. of Computer Science, Waseda University, Ohkubo 3-4-1, Shinjyuku, Tokyo 169-8555, Japan

Abstract

Shrew Distributed Denial-of-Service (DDoS) attacks are stealthy, concealing their malicious activities in normal traffic. Although it is difficult to detect shrew DDoS attacks in the time domain, the existent energy exposes them in frequency domain. For this purpose, online Power Spectral Density (PSD) analysis necessitates real-time PSD data conversion. In this paper, an optimized FPGA based accelerator for real-time PSD conversion is proposed, which is based on our innovative component-reusable Auto-Correlation (AC) algorithm and the adapted 2N-point real-valued Discrete Fourier Transform (DFT) algorithm. Further optimization is achieved through the exploration of algorithm characteristics and hardware parallelism for this case. Evaluation results from both simulation and synthesis are provided. The overall design can be easily placed in a Xilinx Virtex2 Pro FGPA.

Keywords

Power Spectral Density, FPGA Design, Shrew DDoS Attack Detection, real time processing.

1. INTRODUCTION

The low-rate TCP-targeted Distributed Denial-of-Service (DDoS) attack, also known as Shrew DDoS attack [10] or pulsing attack [12], takes advantage of the time-out mechanism of the TCP protocol to create illusory congestions. By sending bursts at a high pulse data rate while keeping a low average data rate, these attacks can throttle the throughput of legitimate TCP flows to as low as 10% of normal bandwidth usage, and last indefinitely until detected [5, 6].

Analyzing the power spectral density (PSD) of monitored traffic can be an efficient way to detect DDoS attacks. PSD describes the power distribution of a signal in the frequency domain. By counting arriving network packets during fixed intervals, a sequence of data points is collected over a specific time interval. Treating the data sequence as a random process, the PSD of traffic flow can be obtained through the calculation and subsequent conversion of its autocorrelation results to the frequency domain. The PSD difference between real-time monitored traffic flows and their normal statistical energy distribution exposes concealed malicious activities without the need for deep packet payload inspection [2].

Several research efforts have been reported that use PSD analysis for the detection and containment of DDoS attacks. Cheng *et al* [7] first reported the concept of utilizing PSD of network flow to detect general TCP SYN flood. Chen *et al* [4, 6] proposed using PSD of network flow to detect shrew attacks. Hashim *et al* [8] further extended PSD analysis based detection to the next generation mobile network (NGMN). They claimed that their proposed algorithm can accurately classify traffic as normal, DoS and DDoS. He *et al* [9] proposed a method for remote detection of bottleneck links, which is applicable to the detection of any network congestion, including DDoS attacks and Internet worms in theory.

However, a major challenge that hampers the real-time application of these approaches is the lack of an appropriate data converter to bear the intensive computing requirements of producing real-time PSD data for analysis. Software solutions are incapable of coping with the high data rates, which naturally leads to pursuing hardware solutions.

Having both the flexibility of software and high parallelism of hardware, reconfigurable hardware devices, such as Field Programmable Gate Arrays (FPGAs), have been widely applied for many network security applications including protocol wrapping, packet classification, and intrusion detection [3]. The growing number of FGPA Intellectual Property (IP) modules not only makes it easy for the implementation of diverse DSP functions to circuits, but provide excellent design balance between high performance and lower power consumption. Given these considerations, FPGA is the obvious choice for implementation of this PSD converter.

In this paper, an optimized FPGA based real-time PSD converter is proposed, which converts data of interest into frequency domain for analysis. Our innovative embedded converter is able to close the gap between supply and demand of real-time PSD data. Taking advantage of processing continuous data sequences with partial overlap, the component-reusable Auto-Correlation (AC) algorithm is capable of significantly reducing computing workload in comparison with that of the conventional AC algorithm. In conjunction with the employment of an FFT IP core, the adapted 2*N*-point real-valued Discrete Fourier Transform (DFT) algorithm not only reduces the length of the operational data sequence to half, but fully utilizes the dual input channels of the FFT core to achieve an optimized design. For performance comparison, a design based on the conventional approach was also implemented.

The remainder of this paper is structured as follows. In section 2, the overall system architecture and algorithms are described. Section 3 depicts further implementation

details. Evaluation results are then explained in Section 4. After discussing some development issues in Section 5, Section 6 summarizes this paper.

2. System Architecture and Algorithms

2.1 System Architecture

Figure 1 shows the overall architecture of the embedded data conversion accelerator, as outlined by the dashed box. It consists of two major modules: AC and DFT. The AC module takes a batch of sampled data and calculates its auto-correlation, i.e.: the similarity to its previous batch. The DFT module converts this AC result from the time domain to the frequency domain for PSD analysis. One can treat the number of arriving packets within a fixed time slot as a random process:

$$\{x(t) |\ t = n \cdot \Delta,\ n \in N\} \tag{1}$$

where Δ is a constant time interval, N is the set of natural numbers and n is a set of positive integers. $x(t)$ is a random variable that refers to the number of packets sampled within a unit time interval at time t, i.e. $(t\text{-}\Delta,\ t]$. If we consider Δ is a unit time slot, Eq. (1) could be reduced to:

$$\{x(t) |\ t = n,\ \ n \in N\} \tag{2}$$

Also, $x(n)$ represents the total number of packets sampled at the *n-th* sampling period. The sampling process is assumed as a Wide-Sense Stationary (WSS) random process. The AC function is defined as:

$$A(m) = \frac{1}{L-m} \sum_{n=0}^{L-m-1} [x(n) \cdot x(n+m)] \tag{3}$$

where m=0, 1, 2, …, L-1.

L refers to the length of the data sequence, i.e. the total number of sample points participating in a batch of the AC process. $A(m)$ is the AC with a shift of m points from the *n-th* sampling point, where $\{0 \le m < L\}$. After AC processing, the DFT is applied to convert the intermediate data to the frequency domain for PSD analysis by:

$$Y(k) = \sum_{m=0}^{L-1} A(m) \times W_L^{km} \quad (k=0,\ 1,\ 2,\ldots,\ L-1) \tag{4}$$

where $W_L^{km} = e^{\frac{-j2\pi km}{L}}$.

Eq. (4) gives a standard DFT formula and more detail can be found in [14]. As shown by Eq. (3) and Eq. (4), the data conversion process is computationally intensive. To reduce the delay, a hardware-based approach is introduced, which is based our innovative component-reusable AC algorithm and the Adapted 2N-point Real-valued DFT algorithm.

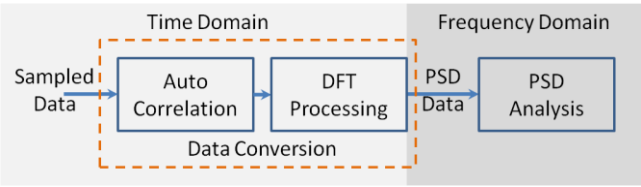

Figure 1. Data Conversion Steps.

2.2 C*omponent*-Reusable AC algorithm

According to Eq. (3), the essential part of the AC process is the convolution between $x(n)$ and $x(n+m)$. With a specified m, it takes L - m multiplications per round to calculate the corresponding $A(m)$, as shown in Fig. 2. To achieve a full batch of $A(m)$ with $\{0 \le m < L\}$, a fixed number of multiplications are required, as:

$$\sum [L + (L-1) + \ldots + 2 + 1] = \frac{L \cdot (L+1)}{2} \tag{5}$$

For instance, in Fig. 2, it is assumed that the length of the operational data sequence (L) is 10 and the current shift (m) is 2. The convolution result of the i^{th} batch is:

$$C_{x(n) \cdot x(n+m)} = x(0) \cdot x(2) + x(1) \cdot x(3) + \ldots + x(7) \cdot x(9) \tag{6}$$

which includes 8 multiplications per round as represented by the area combining C_1 and C_2 in the figure.

In practice, the workload of AC process on a continuous data series can be reduced if convolution results are reusable. As long as not all the data points are replaced for the following batch's AC process, existing partial convolution results can be reused, so as to reduce the overall computational complexity. To describe the replacement of data points for the continuous AC process, a new parameter r is introduced.

As illustrated in Fig. 2, the replacement (r) is 3 for the $(i+1)^{th}$ batch's convolution. The three oldest data points are removed and the same number of new data points is appended to keep the same data length (L) for operation. Moving from the i^{th} batch, the operational data points are now from 3 to 12. The current convolution result is represented by the combined area of C_2 and C_3. C_1 is excluded, since the corresponding old points have been removed. Obviously, C_2, which was generated in the i^{th} batch, can be reused as partial current result to avoid

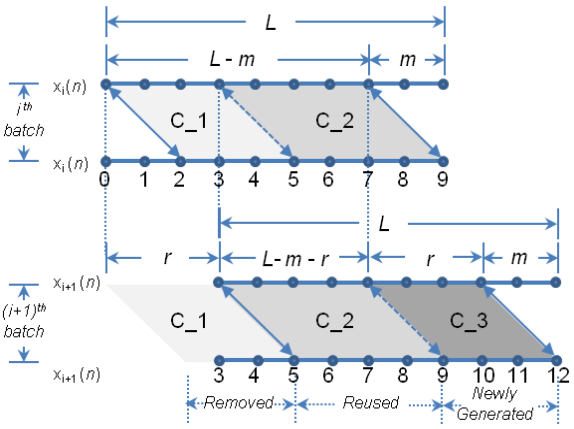

Figure 2. Convolution with reusable parts.

recalculation. Only C_3 requires calculation, which is 3 multiplications in this case.

A further study reveals that, when $L - m \geq r$, results of $(L - m - r)$ rounds of multiplication can be reused for the calculation of the next batch's $A(m)$. In terms of the number of multiplications, that is:

$$P_{reusable} = \sum_{m=0}^{L-r}(L-r-m) = \frac{(L-r)\cdot(L-r+1)}{2} \qquad (7)$$

Only r new multiplications need to be calculated, as illustrated in Fig. 2. However, when $L - m < r$, $(L - m)$ multiplication per need to be calculated. Therefore, the total number of multiplications required for the calculation of a full batch AC process is:

$$\begin{aligned} P_{total} &= \sum_{m=0}^{L-r} r + \sum_{m=L-r+1}^{L-1}(L-m) \\ &= r(L-r+1) + \frac{(r-1)\cdot r}{2} = (L+\frac{1}{2})r - \frac{r^2}{2} \end{aligned} \qquad (8)$$

When $r \geq L$, since all the old data points are replaced, P_{total} reaches its maximum, as:

$$p_{total\_\max} = (L+\frac{1}{2})\cdot L - \frac{L^2}{2} = \frac{L\cdot(L+1)}{2} \qquad (9)$$

This equation also applies to the initial batch's data sequence for AC processing, since all the data points are new. In fact, $P_{reusable}$ is the difference between P_{total_max} and P_{total} given $0 \leq r \leq L$.

According to Eq. (8) and Eq. (9), P_{total} calculated on top of a continuous data series is determined by L and r; and its upper bound is only determined by L. Figure 3 gives an intuitive demonstration of their relationship. The replacement rate R (R= r/L) is introduced for the evaluation of the impact of this component-reusable AC algorithm on the overall computational complexity. In Fig. 3, the X axis represents the length of data sequence (L), and Y axis represents the number of required multiplication operations (N). Ten curves are presented that illustrate the relationship between L and P_{total} under different R values, bottom up from 10% to 100%, respectively. For L ranging from 10 to 150, the impact of different Rs to the number of required multiplications N is presented. Without any reusable part, i.e.: $R \geq 100\%$, multiplication is applied to all the available data points, which is given by the upper curve. This curve overlaps the results of $P_{\text{total\_max}}$ from Eq. (9), which also validates our algorithm.

With reusable part, N, depends on the percentage of reuse. The smaller R is, the less the N, since more elements are reusable for AC computation. Consider a series of N values when L = 120, for example. As labeled on the plot, N is 1374 when R = 10%, and it goes to 3690 when R = 30%. In comparison with N = 7260 when R = 100%, approximately 81% and 50% multiplications are saved, respectively, when R is within 10-30%. However, N grows much faster with the increase of R. It rises to 5430 when R goes to 50%, and becomes much closer to the maximum N when R crosses 70%. Considering the corresponding overhead, it is more inappropriate to adopt this algorithm with high R for AC processing.

Figure 4 further evaluates our component-reusable AC algorithm in terms of the increment rate (I) of N with respect to L. The X axis represents the length of data sequence (L), with a scale from 0 to 1500. The Y axis represents the increment rate (I) which is defined as:

$$I(l+1) = \frac{\Delta N}{N(l)} = \frac{N(l+1)-N(l)}{N(l)} \qquad (l>0) \qquad (10)$$

where l and l + 1 refer to L on two adjacent stages, respectively. ΔN = $N(l$ + 1) - $N(l)$, which refers to the increment of N between two adjacent stages.

Corresponding to different R values, ten curves are plotted illustrating the trends of I with the increase of L. Starting from L = 10, all curves achieve their peaks at L = 11 with I at approximately 20. The maximum vertex is 22.222 for R = 30%. In fact, I(11) is the first available I according to Eq. (10). Then, the curves decrease quickly

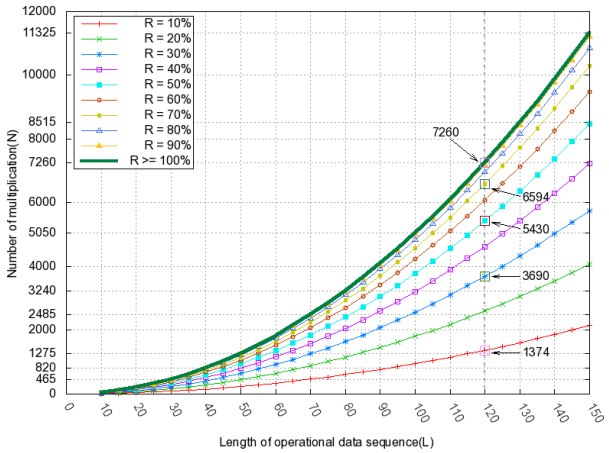

Figure 3. The impact of sequence length L replacement rate to the computational complexity of multiplication.

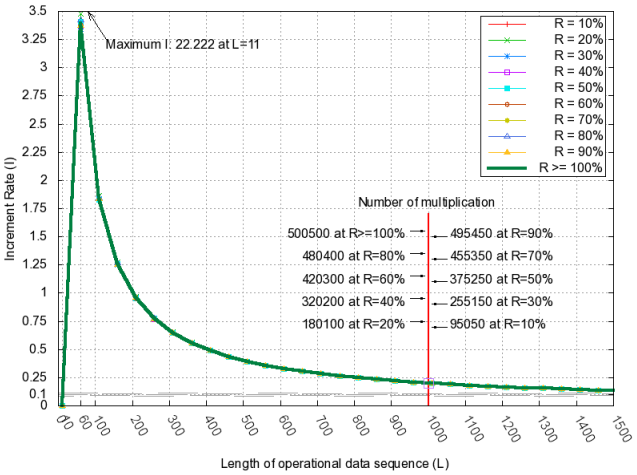

Figure 4. The trend of increment rate for multiplication.

and finally trend to 0.1 based on the evaluation of L up to 1500 points. Despite wide variation of N, curves under different R values show high consistency in I as illustrated in Fig. 3.

From the above analysis, although our component-reusable AC algorithm does not optimize I in terms of R, it decreases the value of N significantly with the increase of L. For L = 1000, a saving of 81% N could be achieved in comparison with different R values, at R = 10% *versus* R = 100%. However, determining the right R for a perfect fit is application dependent. On the other hand, it affects the efficiency of using this algorithm.

2.3 Adapted 2*N*-point Real-valued DFT algorithm

After AC processing, the operational data sequence $\{A(m)|\ 0 \le m \le L\text{-}1\}$ from Eq. (3) needs to be converted from the time to the frequency domain. The DFT is employed for this conversion. However, the implementation of DFT conversion in hardware is complicated and time-consuming. Employing existing Intelligent Property (IP) cores for this purpose is more practical than developing custom modules in FPGA design, not only facilitating prototyping, but achieving optimized performance and resource utilization on the targeted platform.

Applying a Fast Fourier Transform (FFT) IP core is a common approach for implementing the DFT in and FPGA. Since the processing of the general purpose DFT is based on complex-valued data, the Xilinx FFT IP core features dual-channel input for taking both parts of a complex number simultaneously, i.e. one channel dedicating for real part, and the other for imaginary part. However, the data sequence output from $A(m)$ is real-valued only. Fetching them directly to the FFT IP core for processing leaves the input channel for imaginary-valued data unused. By fetching the real-valued data sequence into both input channels, not only could the throughput of DFT processing be doubled but the operation timing of the FFT IP core can be cut due to the length reduction of the operational sequence. Based on this idea and inspired from [1, 13, 14], an adapted 2N-point Real-valued DFT algorithm is proposed.

This algorithm only applies to the data sequence with even length, i.e. $L = 2N$. The basic idea is to split the $2N$-point real data sequence to form an N-point complex data sequence for regular FFT processing. Then, the results of the other half are reconstructed by application of the symmetry conjugate property of complex numbers, rather than calculation. The following description outlines this procedure.

(1) Decimate the real data sequence $\{A(m) \mid 0 \le m \le 2N\text{-}1\}$ into two parts to form a complex data sequence with length N.

$$x(n) = A(2n) + jA(2n+1),\ (n = 0, 1, 2, \dots, N-1) \quad (11)$$

(2) Perform an FFT operation on the N-point sequence. Since the FPGA IP core is employed, this step is considered as a black-box process.

$$X(k) = \mathrm{FFT}\{x(n)\},\ (k,n = 0, 1, 2, \dots, N-1) \quad (12)$$

(3) Calculate the first N-point results of $Y(k)$ by:

$$Y(k) = M_A \cdot X(k) + M_B \cdot X^*(N\text{-}k),\ (k = 0, 1, 2, \dots, N-1) \quad (13)$$

where M_1 and M_2 are pre-calculated values for lookup.

$$\begin{cases} M_A(k) = \frac{1}{2}(1 - jW_{2N}^k) \\ M_B(k) = \frac{1}{2}(1 + jW_{2N}^k) \end{cases},\ (k = 0, 1, 2, \dots, N-1) \quad (14)$$

(4) Obtain the remaining N-point result of $Y(k)$ by using symmetry conjugate property of complex numbers:

$Y(2N-k) = Y^*(k)$, $(k = 1, 2, \dots, N-1)$; and

$$\begin{cases} Y_r(k) = X_r(0) - X_i(0) \\ Y_i(k) = 0 \end{cases},\ (k = N) \quad (15)$$

(5) Complete a batch of DFT conversion for $A(m)$ with $L = 2N$. The results should be the same as that obtained through direct processing via $2N$-point DFT:

$$Y(k) = \mathrm{DFT}\{A(m)\},\ (k, m = 0, 1, 2, \dots, 2N\text{-}1) \quad (16)$$

The benefit of this approach is two-fold. Not only is the dual-channel input of the IP core fully utilized, but the workload of essential FFT processing is cut by at least half. Considering the complexity of the FFT operation for a length L data sequence is $O(L^*\log(L))$, a sequence with length $N = L/2$ is $O((L/2)^*\log(L/2))$. The reduction rate (R) is:

$$R = 1 - \frac{\frac{L}{2}\cdot\log\frac{L}{2}}{L\cdot\log L}\cdot 100\% = \frac{1}{2}\cdot(1 + \log_L{}^2)\cdot 100\% \quad (17)$$

which represents how much the original work load is reduced.

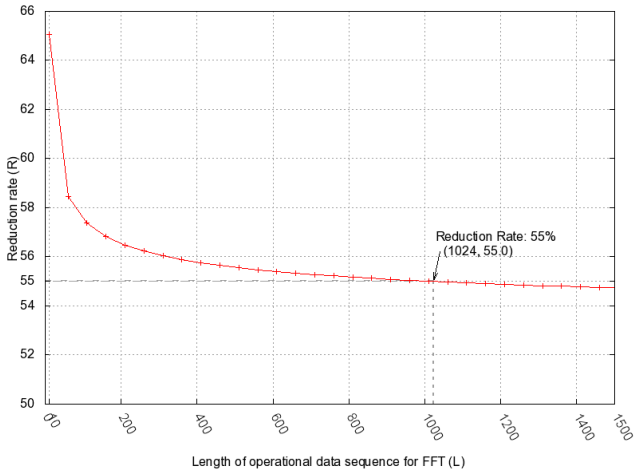

Figure 5. The reduction trend of FFT workload.

Figure 5 gives an intuitive demonstration. The *X* axis shows the length (*L*) of data sequence for FFT operation ranging from 10 to 1500, and the *Y* axis the reduction rate (*R*). For a sequence with *L*=1024, a reduction of 55% the original FFT workload can be achieved. The reduction approaches nearly half as *L* increases further.

3. Implementation

3.1 Design Overview

The block diagrams in Fig. 6 further refine the data conversion process presented in Fig. 1. Figure 6(a) illustrates a conventional architecture. Besides the AC and FFT processing blocks, memory blocks are introduced for data storage so as to achieve parallel execution. Memory M1 holds the input data sequence, memory M2 the intermediate results from the AC process, and memory M3 and M4 the real and imaginary parts of processed DFT data, respectively. The depth of these memories is set to 2*N*, the length of the operational data sequence.

Figure 6(b) illustrates our innovative data conversion architecture. In comparison with the previous architecture, both the AC and DFT blocks are modified substantially. In the AC part, the reuse logic and its corresponding M5 for holding reusable data are added. In the DFT part, the direct 2*N*-point FFT is replaced by the process of the *N*-point FFT plus the 2*N*-point rebuild. The intermediate data from the FFT are stored in M6, while the static parameters are stored in M7. The process of 2*N*-point rebuilding starts when both data sets are available.

Based on the data traveling through the conversion process, the overall design is organized into three parts, as outlined in Fig. 6(b). The first part includes the AC process and the following *N*-point FFT process. It starts with reading the data from M1 and ends with writing the data to M6. The second part is the parameter generating process, which consists of the parameter generator and M7. The generator only runs during initialization: after all the necessary parameters are produced and stored in M7 for subsequent access, it sleeps until reset. The third part consists of the remaining components that take data from both M6 and M7 for 2N-point rebuild and write to M3 and M4. Therefore, Part 1 and Part 3 set up the main data path.

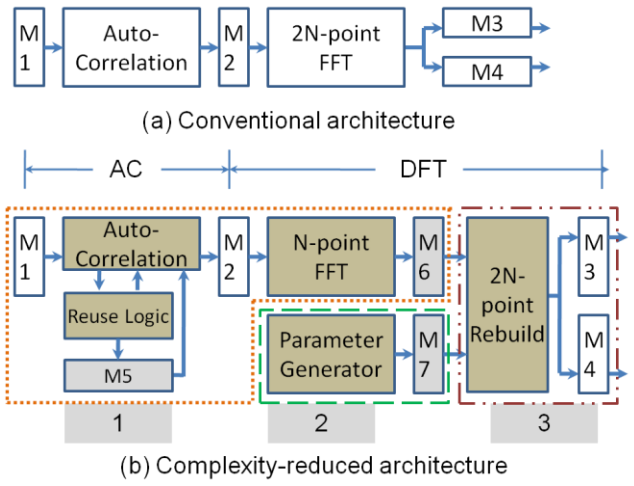

Figure 6. Refined architectures for PSD data conversion.

3.2 Design Implementation

Essentially this design reduces the volume of necessary computational workload. Part 1 emphasizes this goal with the help of three memory blocks. The key block is employed to hold the reusable data, while the other two auxiliary blocks are used to store the insertion points of reusable data for the calculation of new AC results as well as to store a duplication of M1 for expediting the generation of reusable data. They constitute Memory M5.

In the implementation of regular AC calculation according to Eq. (3), we found that the operation of division IP core takes a fixed amount of time for processing. Until the division operation is finished, the data entry on the critical data path will not move forward. A timer-like mechanism is implemented for parallel assistance. The length of the timer is set based on the time a division operation takes. Working as a clock signal that controls the critical data path, extra control logic is not needed. Hence, delay of the overall process is reduced.

The design of the parameter generation block is straightforward, accomplished by implementing Eq. (14) and storing the results to M7 as indicated in Fig. 6(b). A key step is applying sine-cosine values for the production of M_A and M_B. With careful exploration of equation characteristics, a simplified design is achieved. With the replacement of twiddle factor W_{2N}^{k} by sinusoids through Euler's formula, Eq. (14) can be rewritten as:

$$\begin{cases} M_A(k)=\frac{1}{2}[1-j(\cos(\frac{\pi}{N}k)-j\sin(\frac{\pi}{N}k))]=\frac{1}{2}(1-\sin(\frac{\pi}{N}k))+j\cos(\frac{\pi}{N}k) \\ M_B(k)=\frac{1}{2}[1+j(\cos(\frac{\pi}{N}k)-j\sin(\frac{\pi}{N}k))]=\frac{1}{2}(1+\sin(\frac{\pi}{N}k))+j\cos(\frac{\pi}{N}k) \end{cases}$$

$$(k = 0, 1, 2, \ldots, N-1) \qquad (18)$$

Since hardware cannot directly support complex algorithms, it is helpful to express them separately as real and imaginary parts for implementation.

$$\begin{cases} M_{A\_r}(k)=\frac{1}{2}(1-\sin(\frac{\pi}{N}k)) \\ M_{A\_i}(k)=-\frac{1}{2}\cos(\frac{\pi}{N}k) \end{cases}$$

$$\begin{cases} M_{B\_r}(k)=\frac{1}{2}(1+\sin(\frac{\pi}{N}k)) \\ M_{B\_i}(k)=\frac{1}{2}\cos(\frac{\pi}{N}k) \end{cases} \quad (k = 0, 1, 2, \ldots, N-1) \qquad (19)$$

The range of *k* in the above equations with respect to $\sin(\frac{\pi}{N}k)$, which is actually $\sin(\frac{2\pi}{2N}k)$, is {k = 0, 1, 2, …,

$N-1\}$. This implies that only half number of sinusoid values are required, that is N out of $2N$ points. In addition, taking the advantage of triangular conversion formula,

$$\begin{cases} \sin(x+\frac{\pi}{2}) = \cos x \\ \cos(x+\frac{\pi}{2}) = -\sin x \end{cases}, \text{ when } 0 \le x \le \frac{\pi}{2}, \quad (20)$$

the following equations are obtained:

$$\begin{cases} M_{A\_r}(k+\frac{\pi}{2}) = \frac{1}{2}(1-\cos(\frac{\pi}{N}k)) \\ M_{A\_i}(k+\frac{\pi}{2}) = \frac{1}{2}\sin(\frac{\pi}{N}k) \end{cases}$$

$$\begin{cases} M_{B\_r}(k+\frac{\pi}{2}) = \frac{1}{2}(1+\cos(\frac{\pi}{N}k)) \\ M_{B\_i}(k+\frac{\pi}{2}) = -\frac{1}{2}\sin(\frac{\pi}{N}k) \end{cases}, (k = 0, 1, 2, \ldots, N/2-1) \quad (21)$$

Therefore, the required number of sinusoid values can be further cut by half. With only having the first quarter's sinusoid values, i.e. for $\{k = 0, 1, 2, \ldots, N/2\text{-}1\}$, two N-point parameter vectors M_A and M_B are able to be calculated and stored in M7, respectively.

Once one batch of data from the N-point FFT process is ready in M6, the $2N$-point DFT results are rebuilt in conjunction with parameters stored in M7. This process fulfills the operation of Eq. (13) and Eq. (15). Similar to what was done above, Eq. (13) is separated into real and imaginary part for processing, as:

$$\begin{aligned} Y_r(k) &= X_r(k)\cdot M_{A\_r}(k) - X_i(k)\cdot M_{A\_i}(k) \\ &\quad + X_r(N-k)\cdot M_{B\_r}(k) + X_i(N-k)\cdot M_{B\_i}(k) \\ Y_i(k) &= X_i(k)\cdot M_{A\_r}(k) + X_r(k)\cdot M_{A\_i}(k) \\ &\quad + X_r(N-k)\cdot M_{B\_i}(k) - X_i(N-k)\cdot M_{B\_r}(k) \end{aligned}$$

$$(k = 0, 1, 2, \ldots, N-1) \quad (22)$$

And, further extend Eq. (15) to:

$$\begin{cases} Y_r(2N-k) = Y_r(k) \\ Y_i(2N-k) = -Y_i(k) \end{cases} \quad (k = 1, 2, \ldots, N-1) \quad (23)$$

$$\begin{cases} Y_r(k) = X_r(0) - X_i(0) \\ Y_i(k) = 0 \end{cases} \quad (k = N)$$

With Eq. (22) and Eq. (23), $2N$-point DFT values are calculated and finally stored in M3 and M4 as real and imaginary parts, respectively. It completes the function of Part 3 as depicted in Fig. 6(b).

4. Experimental Evaluation

4.1 Experimental Setup

The implementation of this data converter targeted a Xilinx Virtex-2 Pro XC2VP50 FPGA, with Xilinx ISE Design Suite 10.1 on Redhat Enterprise Linux 5.8 OS as its development environment. As part of our auto-defensive infrastructure for network security [7], this data converter is designed to have the capability of processing data at Gigabit network rates. According to Sinha *et al.*'s study in [15], the size of internet packets varies from 40 to 1500 Bytes and its distribution is strongly bimodal at both ends with 40% and 20%, respectively. Theoretically, the expected maximum throughput of Gigabit network is 2^{17} *byte*/*ms*, which corresponds to 88 to 3277 *packets*/*ms* according to the packet sizes indicated above. The 16-bit input data bus is capable of accepting data at rates up to 64*k*, which is enough to accommodate the number of incoming packets collected in a time interval of 20 *ms*.

The network traffic embeded with Shrew DDoS attacks concentrates its energy on the low frequency area less than 50 *Hz*, while healthy traffic shows a much wider energy distribution [5, 6]. For a clear distinction of DDoS attacks in frequency domain, it is necessary to set the sampling rate of incoming network traffic great than 50 *Hz*. Setting it to 512 *Hz* extends the observable spectral range 9 times wider than that of 50 *Hz*, which enables an accurate PSD analysis. Converting this frequency rate to the time domain yields 1.95 *ms* per sample. For a data sequence with L = 512, it takes one second to refresh all its data points, where each data point represents the number of incoming packets counted during a time interval of 1.95 *ms*.

4.2 Simulation Results

Simulation provides an intuitive timing assessment of processing full length data sequences with L = 512. The replacement r was set to 100, roughly R = 0.2. With clock frequency at 5 *MHz* for simulation, our improved approach takes 102.4 *us* to store the initial 512-point's data sequence to M1, according to Fig. 6(b). The AC process then takes 26,679.0 *us* for operation until storing the intermediate data to M2. The 256-point FFT process takes 329.0 *us* from retrieving the data in M2 to storing the processed data to M6. The final 512-point DFT rebuild process takes 307.6 *us* to complete the entire batch of data conversion and store the output to M3 and M4. The time it takes for the production and storage of M_a and M_b parameters according to Eq. (21) to M7 is 51.8 *us*. Since this step runs independently at the initial stage, it has no impact with the delay on the main data path. The overall processing delay for the initial sequence is 27,418.0 *us*.

For a fair comparison, a hardware design of the conventional approach was also developed. As shown in Fig. 6, all necessary modules required by the conventional approach in Fig. 6(a) are all included in our improved approach in Fig. 6(b). The only difference is the length of operational sequences between $2N$-point FFT and N-point FFT modules. This could be easily covered through parameter alteration without additional logic modification. Taking the advantage of modularized design, adapting our improved approach to the conventional approach is relatively straightforward. Meanwhile, the shared base of

both designs makes the evaluation results more convincible.

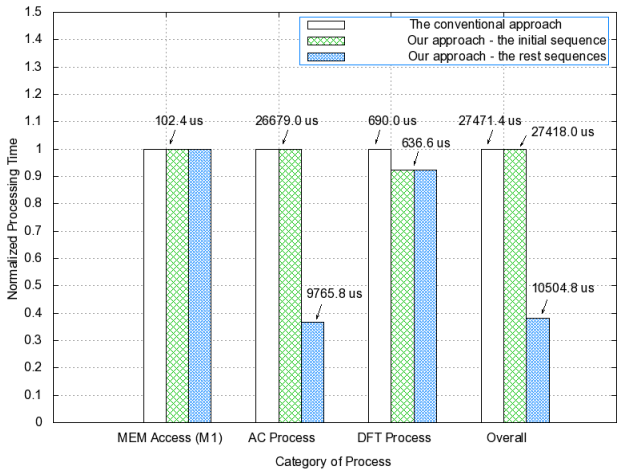

Figure 7. Timing analysis of simulation results between the improved approach and the conventional approach.

The graphs in Fig. 7 illustrate the measured results from both our improved approach as well as the conventional approach depicted in Fig. 6(a) and Fig. 6(b), respectively. The left bar in each set represents the processing time under the conventional approach, while the remaining two bars represent that under our improved approach. The middle bar indicates the processing time for the initial data sequence, and the right bar represents the processing time for the following sequences. To be readable, the processing time is normalized, with the original measured values indicated on the top of each bar for reference.

The first (leftmost) set depicts the processing time of memory access to M1 for writing the input data sequence. It takes 102.4 *us* and there is no difference between both approaches. The second set shows the delay for the AC process. Taking the conventional approach according to Eq. (3), the same amount time, 26679.0 *us*, always applies to the processing of each batch's data sequence. With our component-reusable AC algorithm, this amount of time only applies to the processing of the initial batch's data sequence. When reusable data is available for the processing of the following sequences, only 9,765.8 *us* is required to process. A significant delay reduction of 63.4% is achieved.

The third set represents the delay for DFT processing. The processing time of direct 512-point DFT conversion according to Eq. (4) is 690.0 *us*, while our approach combining 256-point FFT plus 512-point DFT rebuild takes 636.6 *us* for completion. A saving of 7.7% is obtained. Since the DFT processing time is only equivalent to 2.6% or 6.5% of the AC processing time with respect to the conventional or our approach, respectively, this reduction has trivial contribution to the overall processing time which is concluded in the fourth histogram. Overall, a maximum reduction of 61.8% processing time from 27471.4 *us* to 10504.8 *us* is achieved in this instance.

According to the above timing analysis, it is the AC process that dominates the delay of data conversion. More than 97.3% of the overall processing time is taken for this purpose, in terms of our improved approach. On the other hand, the adapted 2*N*-point real-valued DFT conversion does not show much time saving on DFT process. However, the avoidance of another 256 points participating in direct FFT conversion does relieve the workload as analyzed in Fig. 5, which cannot be simply evaluated through timing analysis.

4.3 Synthesis Analysis

After simulation, the design for our improved approach was synthesized for the assessment of resource utilization as well as more accurate timing report. Table 1 lists the summary of device utilization. Though the available resources that the Xilinx Virtex-2 Pro FPGA could provide is conservative in comparison with the latest Virtex family devices, it is still spacious enough to accommodate our design according to this summary. The timing report indicates that the maximum frequency of this board at Speed Grade: Virtex-7 is 91.893*MHz*. This means that all evaluation results from the above simulation with 5 *MHz* clock frequency could be sped up roughly 17 times. Considering the clock frequency at 90 *MHz*, the overall processing delay for the initial sequence is 1.523 *ms*, and the delay from the subsequent sequences is 583.6 *us*. Therefore, all the conversion process can be done within the time interval of sampling one data point at 1.95 *ms* in this case.

Table 1. Device Utilization Summary.

Selected Device : 2vp50ff1148-7				
Number of Devices		Used	Total	Utilization Rate
Slices		2711	23616	11%
Slice Flip Flops		2901	47232	6%
4 input LUTs	Overall	4162	47232	8%
	used as logic		4081 out of 4162 (98%)	
	used as Shift registers		81 out of 4162 (2%)	
IOs		52		
bonded IOBs		52	812	6%
BRAMs		14	232	6%
MULT18X18s		13	232	5%
GCLKs		1	16	6%

5. Discussion

Although this data converter is developed for our Shrew DDoS attack detection system, it could be applied to any application requiring real-time PSD data. Modularized design and IP core utilization enable easy adaptation. During development, it was noticed that even the definition of auto-correlation varies slightly depending on application

background. Thus, supporting logic modification to one functional block without impacting other blocks is highly preferred. Segmenting individual modules is thus a good solution to both support design reuse and facilitate modification. The utilization of IP cores aids in managing data storage requirements as well as enabling FFT processing for new applications. With simple parameter adjustment, such as depth/length of B-RAM and FFT cores, a desired memory and FFT size can be achieved. In conjunction with a modularized interface, the customization of data converter can be completed efficiently.

An important reason for our choosing the Xilinx Virtex-2 Pro FPGA as the target device is that it is also the device that NetFPGA-1G [11] is based on. With four Gigabit Ethernet Network Interface Controllers (NICs) and FPGA chips integrated together, the NetFPGA board seamlessly bridges the network traffic to digital signal processing. Meanwhile, many valuable open source solutions targeted to NetFPGA have been created for diverse applications. Configuring NetFPGA to be an open core gigabit router is one of the most popular applications. Implementing a data converter on top of this platform further expands its adaptability for network related applications.

Since the implementation of the above referenced router to xc2vp50 consumes more than half of the available FPGA resources, the feasibility of implementing the data converter on the same chip remains an open question at this point. An alternative approach could be to implement the data converter in another xc2vp50, while pulling the sampled data from NetFPGA through the SATA interface for sharing. If successful, this approach will not only overcome the potential problem of resource shortages, but will actually lead to solving the issue of inter-board function extension. It is also a meaningful preparation for our planned detection system.

6. Conclusions

In this paper, a FPGA based real-time Power Spectral Density (PSD) converter for Shrew DDoS attack detection has been described. The innovative component-reusable AC algorithm and the adapted 2*N*-point real-valued DFT algorithm are the foundation of this work. With the inclusion of the component-reusable AC algorithm, the computing burden of AC processing on top of partially overlapped data sequences is significantly reduced. Both theoretical analysis and experimental study demonstrate the benefit of our approach in comparison with the conventional approach.

REFERENCES

[1] C. S. S. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms: Theory and Implementation*: John Wiley & Sons, Inc., 1991.

[2] H. Chen and Y. Chen, "A Novel Embedded Accelerator for Online Detection of Shrew DDoS Attacks," in *Proceedings of the 2008 International Conference on Networking, Architecture, and Storage*, 2008, pp. 365-372.

[3] H. Chen, Y. Chen, and D. H. Summerville, "A Survey on the Application of FPGAs for Network Infrastructure Security," *Communications Surveys & Tutorials, IEEE,* vol. 13, pp. 541-561, 2011.

[4] Y. Chen and H. Chen, "NeuroNet: An Adaptive Infrastructure for Network Security," *Journal of Information, Intelligence and Knowledge,* vol. 1, pp. 143-168, 2009.

[5] Y. Chen and K. Hwang, "Collaborative Change Detection of DDoS Attacks on Community and ISP Networks," in *the IEEE International Symposium on Collaborative Technologies and Systems (CTS'06)*, Las Vegas, NV., USA, 2006, pp. 401-410.

[6] Y. Chen and K. Hwang, "Spectral Analysis of TCP Flows for Defense against Reduction-of-Quality Attacks," in *the 2007 IEEE International Conference on Communications (ICC'07)*, lasgow, Scotland, 2007, pp. 24-28.

[7] 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*, 2002, pp. 2143-2148.

[8] F. Hashim, M. R. Kibria, and A. Jamalipour, "Detection of DoS and DDoS attacks in NGMN using frequency domain analysis," in *Communications, 2008. APCC 2008. 14th Asia-Pacific Conference on*, 2008, pp. 1-5.

[9] X. He, C. Papadopoulos, J. Heidemann, r. Mitra, and U. Riaz, "Remote detection of bottleneck links using spectral and statistical methods," *Comput. Netw.*, vol. 53, pp. 279--298, 2009.

[10] 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, 2003, pp. 75-86.

[11] 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.

[12] 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.

[13] R. Matusiak, "Implementing fast Fourier transform algorithms of real-valued sequences with the TMS320 DSP family," in *Application Report, SPRA291 - August 2001* texas instruments, 2001.

[14] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications (4th Edition)*: Prentice Hall, 2006.

[15] R. Sinha, C. Papadopoulos, and J. Heidemann, "Internet Packet Size Distributions: Some Observations," in *Technical Report ISI-TR-2007-643*, ed. Los Angeles: USC/Information Sciences Institute, 2007.