Multi-core Security Defense System (MSDS)

Ashley Chonka and Wanlei Zhou
School of Engineering & Information Technology
Deakin University
Geelong, 3220, Australia

Yang Xiang
School of Management and Information Systems
Central Queensland University
Rockhampton, 4702, Australia

{ashley, wanlei}@deakin.edu.au and y.xiang@cqu.edu.au

Abstract

Today's security program developers are not only facing an uphill battle of developing and implementing. But now have to take into consideration, the emergence of next generation of multi-core system, and its effect on security application design. In our previous work, we developed a framework called bodyguard. The objective of this framework was to help security software developers, shift from their use of serialized paradigm, to a multi-core paradigm. Working within this paradigm, we developed a security bodyguard system called Farmer. This abstract framework placed particular applications into categories, like Security or Multi-media, which were ran on separate core processors within the multi-core system. With further analysis of the bodyguard paradigm, we found that this paradigm was suitable to be used in other computer science areas, such as Spam filtering and multi-media. In this paper, we update our research work within the bodyguard paradigm, and showed a marked improvement of 110% speedup performance with an average cost of 1.5ms.

Index Terms — Multicore, Ubiquitous Multicore framework, Farmer, Bodyguard Framework

1. Introduction

Computer networks, and the internet, have evolved into high-speed backbones and local-wide area networks. Through these networks, millions of end-users are linked to many critical services. Many businesses, also, rely upon these critical services to function at full capacity, in order to achieve greater customer satisfaction and greater profits. DDoS (Distributed Denial of Service) attacks are one of the most effective ways to bring these critical systems, and bring huge financial cost to bear to repair.

Currently, most defense systems, such as traceback [1][2], logging [3][4] and messaging [5][6], have difficulty in separating legitimate from illegitimate traffic. Another problem with these defense systems is that they do not really defend the system. Instead they give the means to help identify the attackers [7], without filter out the attack traffic.

In a previous paper [8], we introduced a defense system called Farmer, named after the Kevin Costner Movie 'Bodyguard'. The fundamental idea, behind the bodyguard framework was too separate out different parts of security procedures (IP reconstruction, filter attack traffic, monitoring defense system for attacks) and placed them within their own category, separate from other categories like multi-media or game development. Today's security is either placed in front of the system, like a firewall, or behind the system, like a virus scanner. Though these systems do provide good protection, they usually come with high costs. For example, with Firewalls they block inbound and outbound traffic, which is directed by the host's system administrator. But, firewalls overhead costs, such as memory usage, hard-disk space etc are particular high. With the coming of multi-core system, there is now opportunity for security applications to be placed along side other application, instead of in front or behind them. This gives applications, for example viewing youtube videos, but without the hindrance of a front end firewall scanning every packet at the beginning. Instead, aside firewall could scan along side your youtube download, in real-time, and if a packet is detected to be suspicious, then the front firewall can become active and filter out the traffic. This is just one advantage, which the bodyguard system could provide.

In this paper, our contribution is to currently update our bodyguard framework, which has now evolved into what we call ubiquitous multicore framework. What we discovered, while working with our bodyguard system, is that we could apply our methodology into other areas of computer science. For example, Spam filtering [9] and multimedia [10][11]. The rest of this paper is organized as follows. Section Two briefly covers the related work done in Multicore. The details of UM framework and how it is applied to the bodyguard framework Section Three. Section Four presents the experiments and evaluation that were conducted on our system. Lastly, Section Five covers the conclusion and future work.

2. Related Work

In this section, we discuss very briefly multicore and multimedia, and the two areas where our multicore framework has been applied.

2.1. Multicore and Multimedia

Multicore systems have two or more processing cores integrated into a single chip [12][13][14]. In such a design, processing cores have their own private cache (L1) and a shared common cache (L2). The shared cache and main memory share the bandwidth between all the processing cores. Multimedia co-processor interface was developed by [15], in which they used a multicore system to offload task management jobs from MPU or DSP. From their evaluations conducted on a JPEG file, Ou et al. achieved an overall performance increase of 57%, while they kept their overhead to 1.56% of the DSP core. The UM framework is very different from Ou et al., in which UM is more abstract, by applying applications (not separate sections of a file) to separate core processors.

2.2. Multi-classifier SPAM filter

With the use of the paper by Chonka et. al [8], Rafiq et. al [9], was able to apply the UM framework to a multi-classifier SPAM filter. What we found, was that if you ran each classifier process in parallel with each other, it greatly improved the performance of our multi-classifier architecture. It, also, reduced the false positives and increase accuracy. The other advantages we were able to achieve are as follows [9]:

Reduced computation burden of the overall mail server.

Reduced memory storage, email messages are processed independently from other classifiers.

When one of the classifiers becomes idle it will directly go into training mode, thereby optimizing resource usage.

3. Background

3.1 Farmer System Design

In our original paper [8], the bodyguard framework was distributed on each router in the network. This was done, in order to provide overall protection (Figure 1). Each bodyguard is a source end (provides security before traffic leaves the router) and destination end protector (provides security as the traffic enters the network). Also in covered in Figure 1, was each bodyguard was in communication with each other. There are three main reasons for this; to allow bodyguards to send updated security information to each other (new attacks that each has encountered, for example), send security information down to the next hop for checking application data as it comes into the router (This is to provide better performance, by breaking up the security and application data), monitors the performance of each other (So if a successful attack brings down a bodyguard, the next hop router is prepared to handle the security). Farmer's objectives are as follows:

1. To protect the system, while allowing applications full performance potential.
2. If an attack is discovered, the front bodyguard sub-system will be initiated, which will affect the performance ratio of the application, but will not affect the other applications on the host. The affected application performance ratio will be kept to a minimum, while the security issue is resolved.
3. That all security process generated by side and front bodyguard sub-system are handled by the Security Cores.

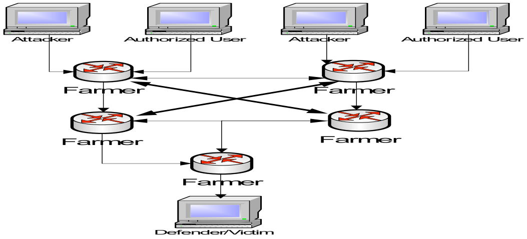

Figure 1. System Architecture of Farmer

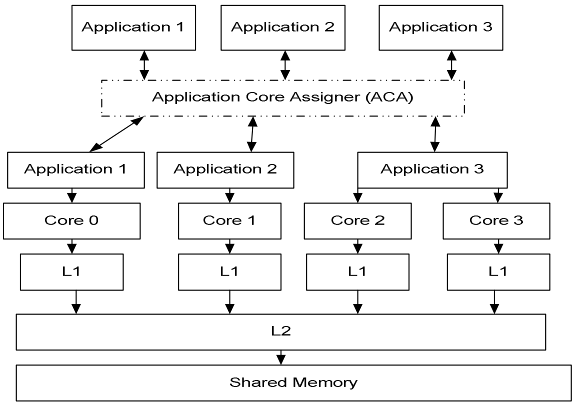

Figure 2. Ubiquitous Multicore Framework

Ubiquitous Multicore (UM) Framework

The Ubiquitous Multicore Framework is built from a divide-and-conquer approach [16], by dividing our applications and placing them on separate core processors (Figure 2). [Note: UM is not the new SMP] Each application will run in parallel with each other, exchanging information when necessary. The application core assigner (ACA), assigns the application either on behalf of the user, or the user can select from the core(s) that are available. Once an application is assigned to a core, depending on the application program, a number of jobs or threads can then be executed on this core processor.

Applied UM Algorithm to Bodyguard Framework

Our contribution in this paper, to give a further analysis of Mathematical Partition model [10]. We follow this, by conducting a short experiments of the UM Framework, through the use of MPI. The Mathematical Partition Model essentially, consists of only computation, but we do assume a minimum of communication is required for the 4 applications.

Mathematical Partition Model

The MPM is adapted and modified from the partition analysis of [19][20], in which they analysed the speedup performance, computation and communication cost and execution times of their partition. We briefly covered the partition model in [10], in which we now further extend.

$$t_{com1} = \frac{n(cp-1)}{cp} t_{msgdata} \qquad (1)$$

where $t_{msgdata}$ is the transmission time for a data message sent over broadcasting, and n is the number of computations used, and cp is the core processor that have been selected to be used. Computational time is represented by counting the number of computational steps, usually if all processors are used then just one process computation is necessary, it is as follows:

$$t_{comp} = f(n, cp) \qquad (2)$$

Communication Time is depended upon the number of messages, size of the message, communication infrastructure (communication and network):

$$t_{com} = t_{init\_startup} + w t_{msgdata} \qquad (3)$$

$t_{init\_startup}$ is the message latency, which is the time it takes for a message to be sent with no data. The data messages sent via each partition is found in the formula:

$$t_{com2} = (\log cp) t_{msgdata} \qquad (4)$$

For the total communication time is as follows:

$$t_{tcom} = t_{com1} + t_{com2} = \frac{n(cp-1)}{cp} t_{msgdata} + (\log cp) t_{msgdata} \qquad (5)$$

The computation formula for the 4 partition applications at the end of the partition phase (step 6) is as follows:

$$t_{comp} = \frac{n}{cp} \qquad (6)$$

This gives us the Overall Execution Time for the 4 partition applications in the following formula:

$$t_p = \left[\frac{n(cp-1)}{cp} + \log cp\right] t_{msgdata} + \frac{n}{cp} + \log cp \qquad (7)$$

The very best speedup we could expect, when the 4 partitioned applications have completed their computations, is as follows:

$$\frac{n-1}{((n/cp)(cp-1)+\log cp)t_{msgdata} + n/cp + \log cp} \qquad (8)$$

The actually speedup will be less than this due to partition phase; computation/communication (c/c) ratio is as follows:

$$\frac{n/cp + \log cp}{((n/cp)(cp-1) + \log cp)t_{msgdata}} \qquad (9)$$

For load balancing we use the Mandelbrot computation [16], in which if the maximum performance (mp) is reached for the processor, it will then search for another core processor to continue the work.

$$T_s \le mp * m \qquad (10)$$

To partition the application correctly we use the following three phases:

Phase 1:

$$t_{comm1} = (p-1)(t_{startup} + t_{data}) \tag{11}$$

Phase 2:

$$t_{comp} \le \frac{mp * n}{p-1} \tag{12}$$

Phase 3:

$$t_{comm2} = u(t_{startup} + vt_{data}) \tag{13}$$

In order to maintain the highest speedup and computation/communication ratio we use the Overall Execution Time(14), Speedup factor (15), C/C ratio (16):

$$t_p \le \frac{mp * n}{p-1} + (p-1)(t_{startup} + t_{data}) + k \tag{14}$$

$$\frac{t_s}{t_p} = \frac{mp * n}{\frac{mp * n}{p-1} + (p-1)(t_{startup} + t_{data}) + k} \tag{15}$$

$$\frac{mp * n}{(p-1)((p-1)(t_{startup} + t_{data}) + k)} \tag{16}$$

4. Performance Evaluation

4.1 Performance Analysis

To assess the performance of our multicore system, we used the same performance outline in [10]. In which, we compared the two kernel benchmarks. The hardware we used was, an Intel Core 2 Quad Q6600 2.4GHz Quad Core Processor, 2 GB of RAM and 2 300GB SATA hard-drives. The kernel under measurement was 2.6.22.14.72 fc6. To gather computational data, we included timers with our application, in order to record execution times. Communication time is depended upon the number of messages, the size of the message and the interconnection speed. We have decided to set the standard to 1ms and computational data is assumed to be .1ms less then execution time.

4.2. Simulation Setup

4.2.1 Benchmark factors

Our benchmarks are also from our [10], in which the execution times t_s, computational time t_{com}, and communication time t_{com}, can be used to establish the speedup factor (17) and computation/communication ratio (18) from a single core to multicore system.

$$\frac{t_s}{t_{cp}} = \frac{t_s}{t_{comp} + t_{com}} \tag{17}$$

Where t_s will stand for execution time on a single core processor (t_{cp}), this includes computation time and communication time.

$$\frac{t_{comp}}{t_{com}} \tag{18}$$

Apart from speedup and the Computation and Communication ratios, we also evaluate the UM algorithm, through the use of Time Complexity or "big-oh", also referred to as "order of magnitude" [12].

$$f(x) = O(g(x))$$
$$[0 \le f(x) \le cg(x)] \text{ for all } x \ge 0 \tag{19}$$

Where f(x) and g(x) are functions of x. A positive constant, c, has to exist for all $x \ge x_0$ otherwise it is zero. To evaluate Time complexity, we use the total sum of computation and communication (formula 11)

$$(n/cp+1)+(2t_{startup}+(n/cp+1)t_{msgdata} \tag{20}$$

Where n is the number of threads on each core processor.

4.3. Experimentation

To give us a baseline of comparison, we wrote a program using MPI (19), in which the program gives us a "perfect" example of parallel programming. The results shown in table 1 are speedup that we were able to achieve, which was 110%. From table 1 we then do a comparison of previous results from paper [8], in which we selected the best of our results and show them along side table 1 (See Table 2). To make the comparison fair, we used the same computation and communication time from the MPI program, for our multicore program. What the results from Table 2 shows, is that our MPI program use's multicore technology with greater efficiency, then our previous multi-core program. But the reason for this greater efficiency, was due to the fact that we wrote our multicore program in C++ only.

In our second experiment we trained up Farmer, which contains our Back Propagation Neural Network Filter (placed on core 2), in order to detect and filter DDoS attack traffic. To train up our Neural Network, we used the dataset from the week 2, 1998 DARPA intrusion detection evaluation set at Lincoln Laboratory, MIT [17]. The data sets from MIT came in TCP dump format, so we extracted the features we needed and insert them into a MySQL database. These features included SrcIP, DestIP, SrcPort, DestPort and

	Core 1	Core 2	Core 3	Core 4
Exe Time	1.5ms	1.4ms	1.3ms	1.4ms
Comp Time	.3ms	.3ms	.2ms	.3ms
Comm Time	1ms	1ms	1ms	1ms
Speed Ratio	115%	108%	108%	108%
C/C	0.3	0.3	0.2	0.3
Time Complex	3.5	3.5	3.5	3.5
Cost	1.5	1.4	1.3	1.4
Cost-Optimal	3.7	3.7	3.7	3.7

Table 1. Results of speedup and the costs, which show an average increase of 110% at the average cost of 1.4ms

MPI (MC)	**Core 1**	**Core 2**	**Core 3**	**Core 4**
Exe Time	1.10ms	1.15ms	1.11ms	1.11ms
Comp Time	0ms	.04ms	.01ms	.01ms
Comm Time	1ms	1ms	1ms	1ms
Speed Ratio	110%	111%	110%	110%
C/C	0.3	0.3	0.2	0.3
Cost	1.5	1.4	1.3	1.4

Table 2. Results of speedup and the costs, which show an average increase of 110% at the average cost of 1.4ms for the MPI over our previous Multicore result.

the length of time. We added an extra field to the table for the decision, 0 for legitimate and 1 for illegitimate. The result shown in Figure 3, was that, we were able to achieve a +90% of the known attack traffic, with an average of 6 false positives per test. This means that our security detection is quite sensitive in detecting and Network against the test data provide by [18].
filtering out DDoS attack traffic. To confirm this result we then further our experiment by testing our Neural

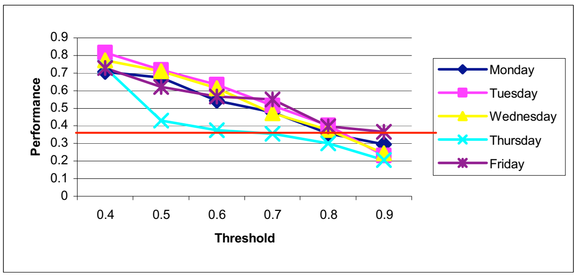

Figure 3. Test Results of our Neural Network archived a better than average (94%) result, with an average of 6 false positives per test (5 days of tests were conducted from the MIT Dataset).

5. Problem and Future Work

In this section, we discuss briefly a problem within our UM Framework in regards to our defense system, and a possible solution to this problem. One of the area's of defense systems is accuracy [18]. Though, the result shown in Figure 3, displays a high accuracy rate, this result was not due to use of UM framework but instead due to efficiency of Back Propagation Model of our Neural Network. What we propose is a prediction that accuracy can be improved using the UM Framework by monitoring the efficiency of other core processors. When a core is either below an efficiency level of an assumed 30%, then redundant Neural Network would then use these other core processors in training and detecting DDoS attack traffic. In other words we have redundant detection filters setup behind the main programs on the core processors, ready to startup when the efficiency level has been reached (or when an emergency arises due to high attack rates for example).

This would have three major benefits; firstly, accuracy would be improved upon (This needs to be confirmed). Secondly, greater efficiency of core processors, and lastly, having these redundant filters available during a particular difficult DDoS flood attack would ease the resources on the main filter (placed on core 2). The problem with this future work is, does it violate the UM Framework? We would answer, No, to the question because we still have the main programs each assigned to the core processors. We just have the redundant systems in place, to "replace" the main programs until such time that the main program efficiency level is low, thereby freeing up core processing time.

5. Conclusion

In this paper, we further extended upon our previous work within multicore defense system, by applying the UM Framework to our Bodyguard Defense System. The goal of such a security system is to use the new multicore machines that are coming out, but also, with these machines they can be used to solve some of the many problems of computer security. Based on the results, we have showed our defense system has an improved performance average of 110%, through the use of MPI [19]. We, also, showed our test results of Farmer's side bodyguard (Back Propagation Neural Network), which would than tell the forward bodyguard to filter the attack traffic detected. The results show, based on 10 tests that we conducted over 4 hours of training the system. The result we achieved was an average of 94% of attack traffic detected, with an average of 6 false positives per test.

10. References

[1] Savage, S., Wetherall, D., Karlin, A., and Anderson, T., (2001), '*Practical Network Support for IP Traceback*', SIGCOMM'00, Stockholm, Sweden, 2000

[2] Belenky, A.,and Ansari, N., '*Tracing Multiple Attackers with Deterministic Packet Marking (DPM)'*, Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing

[3] Snoeren, A.C., et al., (2002), "Single-Packet IP Traceback," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, 2002, pp. 721–734.

[4] Baba, T., and Matsuda, S., (2002). "Tracing Network Attacks to Their Sources," *IEEE Internet Computing*, vol. 6, no. 3, 2002

[5] Bellovin, S., Leech, M., and Taylor, T., (2003), 'ICMP Traceback Messages,' Internet Draft, Internet Eng. Task Force, 2003; work in progress.

[6] Mankin, A., Massey, D., Wu, C.L., Wu S.F and Zhang, L., (2001), "On Design and Evaluation of 'Intention-Driven' ICMP Traceback," *Proc. IEEE Int'l Conf. Computer Comm. and Networks*, IEEE CS Press, 2001. pp. 159–165.

[7] Aljifri, M., (2003), '*IP Traceback: A NewDenial-of-Service Deterrent?*' Published By The IEEE Computer Society 1540-7993/03 2003

[8] Chonka, A Zhou, W Knapp, K and Xiang, Y, (2008), "Protecting Information Systems from DDoS Attack Using Multicore Methodology", *IEEE 8th International Conference on Computer and Information Technology*, IEEE, 2008.

[9] Islam, R. M.D, Singh, J, Zhou, W., and Chonka, A., (2008) , "Multi-Classifier Classification of Spam Email on a Multicore Architecture", Proceedings of IFIP International Conference on Network and Parallel Computing, 2008

[10] Chonka, A, Zhou, W, and Ngo, L, (2008), "Multi-core Defense System (MSDS) for Protecting Computer Infrastructure against DDoS attacks", IEEE Proceedings of PDCAT2008.

[11] Chonka, A, Zhou, W, and Ngo, L, (2008), "Ubiquitous Multicore (UM) Methodology for Multimedia, Proceeding of International Symposium on Computer Science and its Applications

[12] Multi-Core from Intel – Products and Platforms. http: //www.intel.com/multi- core/products.htm, 2006.

[13]AMD Multi-Core Products. http://multicore.amd.com/en/Products/, 2006.

[14] Gorder, P.M, (2007), *'Multicore processors for science and engineering*', IEEE CS and the AIP, 1521-9615/07/,March/April 2007

[15] Ou, S.H., Lin, T.J., Deng, X.S., Zhuo, Z.H., Liu, C.W., (2008), "Multithreaded coprocessor interface for multi-core multimedia SoC', Proceedings of the 2008 conference on Asia and South Pacific design automation, Seoul, Korea SESSION: University LSI design contest, Pages 115-116, ISBN:978-1-4244-1922-7, 2008

[16] JaJa, J. (1992), '*An Introduction to Parallel Algorithms*", Addison Wesley, Reading, MA

[17] MIT 1998 DARPA Intrusion Detection Evaluation Data Set, http://www.ll.mit.edu/mission/communications/ist/index.html

[18] Xiang, Y., and Zhou, W., (2004), 'Trace IP packets by flexible deterministic packet marking (FDPM)**'**, IP Operations and Management, 2004. Proceedings IEEE Workshop on 11-13 Oct. 2004

[19] Gropp, W., Lusk, E, Skjellum, A, (1996), "*Using MPI: Portable Parallel Programming with the Message-Passing Interface*", Massachusetts Institute of Technology, 1994.

[20] Foster, I, (1994), "*Designing and Building Parallel Programs: concepts and tools for parallel software engineering*", Addison-Wesley Publishing Company, (1994)

[21] Wilkson, B & Allen, M, (2005), "*Parallel Programming: Techniques and Applications using network workstations and parallel computers*", Pearson Education, Pearson Prentice Hall, (2005)