SEPCOM: Customizable Zero Copy Model *

Kai Chen
PHD Candidate
State Key Laboratory of Information Security, Graduate University of the Chinese Academy of Sciences
Beijing 100049, China
86-01062661709
chenk@is.iscas.ac.cn

Purui Su
Associate Professor
State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences
Beijing 100080, China
86-01062661709
supurui@is.iscas.ac.cn

Yingjun Zhang
PHD Candidate
State Key Laboratory of Information Security, Graduate University of the Chinese Academy of Sciences
Beijing 100049, China
86-01062661709
yjzhang@is.iscas.ac.cn

Dengguo Feng
Professor
State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences
Beijing 100080, China
86-01062661709
feng@is.iscas.ac.cn

ABSTRACT
Fast Ethernet packages management has become a hot topic in the world since the bandwidth is approaching Gigabit and the magic speed of worm spreading. It is necessary for a server such as IDS and firewall to manage packages in an extremely fast way. In the procedure of transmitting packages, CPU copies each package twice. In order to make CPU spend less time in the copy procedure, zero-copy mechanism has been brought forward. However, in former implementations, almost all the user-face software needs to be modified and no one has made theoretical analysis of those implements. We propose SEPCOM (State basEd Partly Customizable zero cOpy Model) to speed up the package managing with the maximum compatibility. We also make a mathematic model to maximize the speed. In the end of the paper, to test our model, we propose an application to monitor network flows, which is one of the most popular applications requiring high speed of package managing.

Categories and Subject Descriptors
C.2.3 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Operations – *Network management, Network monitor*

General Terms
Management, Measurement, Performance, Design, Security.

Keywords
SEPCOM, zero copy, package management, fast Ethernet, system architecture

* This work was supported by Hi-Tech Research and Development Program of China (863 Program) under Grant No. 2006AA01Z412, 2006AA01Z437 and 2006AA01Z433 and by the National Natural Science Foundation of China under Grant No.60403006.

Valuetools'07, October 23-25, 2007, Nantes, France.

1. INTRODUCTION
Today, with the development of network and increasing speed of worm spreading, handling packages quickly is much more important than before, especially in the fields of IDS, firewall and network managing of the backbone network[16] [17] and so on. In some cases, high speed up to 10G is required. However, if the system's own network protocol stack is used, the network flow can not be managed in the speed of 1 Gigabit for one simple reason: In the process of transmitting package, it requires CPU to copy every package twice. First, the package is copied from Network Interface Card (NIC) to memory. Second, it is copied from system space to user space. In many cases, these two copy operations will cause CPU not having time to do normal transactions. To avoid this, a method called zero copy has been brought forward. However, as the zero copy method has to modify the driver program of NIC, almost all of the software in the user space needs to be modified. We proposed SEPCOM, which is not only a model of high speed packages management but also compatible with existing systems. It makes it possible to customize any software into zero copy one and achieve almost the utmost speed of the NIC.

We proceed in the remainder of the paper as follows. In Section 2, we review the related work. We continue in Section 3 by introducing our SEPCOM. Next, in Section 4, we make mathematic model for our SEPCOM and evaluate it before quantity analysis in Section 5. We provide one example of

application in the field of monitoring and filtrating and make some comparison in Section 6; and conclude in Section 7.

2. RELATED WORK

In the past, a lot of efforts have been made to increase the efficiency of packages transporting. The best way which has minimal data transfer overhead is that the network memory can be accessible to user or kernel space. However, this requires difficult hardware support and many changes of the software. Since the memory of NIC is so limited, we can hardly do any large scale use of them. Furthermore, if the software can not manage the memory well such as leaking memory, it may easily deplete the interface memory available for use [2]. Cooper [1] used some analogical ways to do this.

To evade the complicated hardware changes, a different approach emerged. A block of memory is shared between the network interface and the kernel, and then use DMA to transmit packages [3], [4], [5], [6]. Unfortunately, it requires us to modify network software, the kernel and interface driver. Special attention needs to be paid to the memory of network software, which is expensive and not easy to be done. In addition to the complicacy of this approach, there is another copy operation between the kernel space and the user space, which diminishes the efficiency.

Then, in order to avoid the two copy operations, Druschel and Peterson [7] proposed fast buffers (fbufs). In this new approach, mapping scenario was added between the user and kernel and DMA was used to transmit packages between the network interface and the memory. Although this method eliminates the overhead of the additional copy operation, the software, kernel and driver also need to be modified. Moreover, all the software uses its own memory buffer which is pre-mapped to the kernel. So it may cause the memory corruption and resource competition and will render the efficiency.

Chu [8] made another schema which remaps memory by editing MMU table and makes use of copy-on-write techniques. Although it allows less modification of the software, there are several flaws in it such as the VM operations' expense and the boundary problems just as the author himself says.

On the other hand, SCAMPI[12]和 MAGNeT[13] mapped the kernel events to the user space to capture packages, which is unfortunately difficult to deploy. SNORT[14] can also be managed to capture packages, but it can not exceed the upper limit of 64000pps[15].

In this paper, we propose SEPCOM, which uses only one block of kernel buffer instead of allocating for each program. By doing this, we can easily manage the buffer by modifying the network interface driver but not the kernel. As the block is managed only by driver, the memory corruption will be maximally avoided. Then the overall block is divided into small units on which some flags can be added to indicate the different states and different software. Since the mapping mechanics of the kernel is not changed, we can customize any software we want to be the zero copy one without any changes to other software. In this way, the compatibility is maximized. DMA mode and mapping schema are also used to eliminate the two copy operations of CPU. With zero copy and little overhead of memory management, we achieve great speed in the tests. We also make mathematic model for our approach, which can help us optimize our model and maximize the speed.

3. SEPCOM

When applying zero copy, the cost of managing memory and modifying software should be taken into account. If complex managing mechanics of the user-kernel mapping, memory or software are used, the overhead of these operations will counteract the efficiency we get from zero copy. Our SEPCOM overcomes the deficiencies mentioned above. In order to show how SEPCOM works clearly, we will introduce our model from two angles of view: the overall framework of SEPCOM and the package transmitted process.

3.1 Framework of SEPCOM

Figure 1 shows the architecture of SEPCOM. It can be classified into three strata: user space, kernel space and hardware. Network interface is working in the hardware stratum. It receives the package from the network link and sends the packages of the network software such as FTP Server. Network Driver works in the kernel space as a module. In this way, the operating system kernel does not need to be changed, which makes our approach more compatibility. In the user space, many software run here including the network ones and others. Network software can be divided into two types: zero copy software and other software which do not need zero copy. As we all know, in large scale network, the server is almost acting as only one role. For example, the file server is always used to supply the function of file access. The log server is always used to log the network or services information. So we can only change those software which need fast package management to the zero copy ones while the other software use the system tcp/ip stack without any changes. In this way, we reduce the cost of changing all the network software. Even if the server serves as more than one role, we only need to make the most used two or more network software to be zero copy ones. Then I will explain how we achieve this.

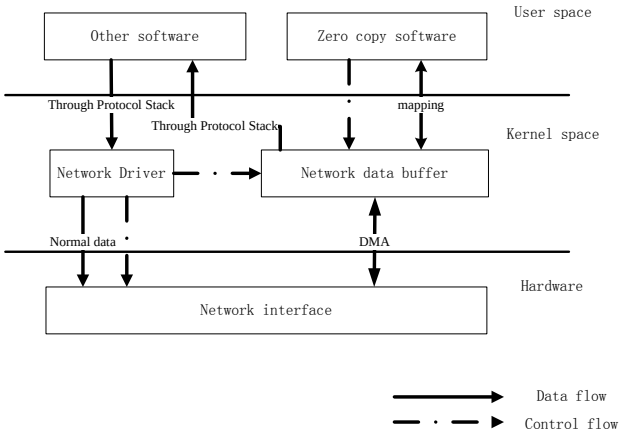

Figure 1. The system architecture of SEPCOM

Firstly, the network driver allocates one block of buffer in the kernel and remapped it to the user space. The overall block can be divided into n pieces of unit equally. When data come from the network interface, they will be transferred to one of the unit through DMA. Next, the driver will try to find whether the package belongs to zero copy software. If it does, the driver flags the package and does not take care of them any more. If the

package belongs to other software, the driver reconstructs the package to the suitable structure (in Linux, it is the sk_buff structure) and sends it through system TCP/IP stack. In this way, the zero copy software in user space can access the data without any copy operation of CPU. What we talk above is about receiving operations, and then we will talk about sending operations.

When zero copy software wants to send a package, it can fulfill the package to the unit of kernel buffer and flag it to be sent. The network driver has a timer, which makes the driver to inspect every unit of buffer in a regular time period. When it finds the package be ready to send, it makes NIC to send it by DMA. Other software sends packages only in their usual way just like no zero copy mechanism. To better understand SEPCOM, we can look at it from the view of the package flag, which shows the different states when the package is transferred.

3.2 State of Package Transition

When the first package reaches, it is flagged to indicate different stages of being managed. NFA (Non-Deterministic finite automaton) M is helpful to describe the process.

Definition 1. $M = (Q, \sum, \delta, q_0, F)$

In this definition, Q is the set of states. Q={Flag_No_Use，Flag_Alloc，Flag_Recv_0，Flag_Read_0，Flag_Send_1，Flag_In_Send_1，Flag_Drop}. $\sum$ is the set of inputs. δ is a function of state change. The detailed information about $\sum$ and δ will be talked below. q_0 is the start state of M and F is the final state. In this case, $q_0 = F$ = Flag_No_Use. When the state reaches F, it means not only the end of one package transference but a new start of next. All the states and switches among them can be seen in figure 2 below.

For simplicity, we draw the state-transition figure under the condition of only one zero copy software. Two or more software can be distinguished by using additional flags. If software only sends or receives packages, some parts of figure 2 can be used to indicate this.

At first, the initial state is Flag_No_Use, which means the unit of buffer is of no use. When the unit is allocated for receiving packages, the state will be changed to Flag_Alloc. After a new package comes, the driver judges whether it belongs to zero copy software. If so, the state will be changed to Flag_Recv_0 (the number 0 means the package is received from the first network interface). Otherwise, it will be changed back to Flag_No_Use. When the zero copy software reads its package, the state will be changed to Flag_Read_0. Then if the software wants to send a respondent package, it can modify the package and set the state to Flag_Send_1 (the number 1 means to the package is sent from the second network interface). If the software does not want to make any response, it can just set the state to Flag_Drop. When the driver finds a package flagged Flag_Send_1, it will try to send the package and change the state to Flag_In_Send_1. When finding Flag_Drop, it simply reclaims the package and sets the state to Flag_No_Use. At last, the driver changes the state to Flag_No_Use after sending successfully or it sets the state back to Flag_Send_1 and continues to send for a second time. If still failed, the driver sets the flag to Flag_Drop and drops it. A package ends its life in the above process.

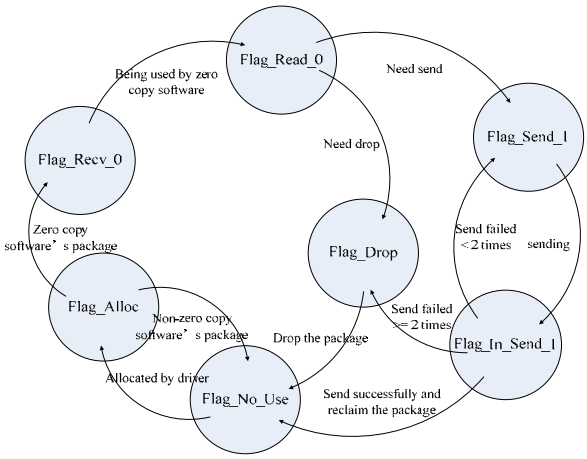

Figure 2. The state-transition figure

Now, we have known the architecture of SEPCOM and how to manage packages. Next we will analyze and maximize the efficiency of SEPCOM by means of constructing mathematic model.

4. MATHEMATIC MODEL OF SEPCOM

Before we can analyze the efficiency of our model, we will firstly model it.

Definition 2. package managing time t_M : indicates the average life time of a package. From the view of state transition, it means the average time interval between the start from Flag_No_Use and the end in Flag_No_Use.

Definition 3. zero copy package managing time t_I : indicates the average life time of a zero copy package.

Definition 4. non-zero copy package managing time t_B : indicates the average life time of a non-zero copy package.

From Definition 2, we can get that:

$$t_M = \rho \cdot t_I + (1-\rho) \cdot t_B \quad (1)$$

where ρ is the propostion of zero copy software's packages to all packages.

We can also get:

$$t_B = C_{Alloc} + C_{DMA} + C_{NM} \quad (2)$$

where C_{Alloc} is the time we need to allocate one unit of buffer, which is the time interval between Flag_No_Use and Flag_Alloc; C_{DMA} is the time needed by DMA to transfer the package from network to kernel memory buffer; C_{NM} is the managing time of non-zero copy package after received by driver. All the three parameters above are constant. Then we can get the expression of tI from Definition 3:

$$t_I = \omega \cdot t_{IT} + (1-\omega) \cdot t_{ID} \quad (3)$$

where ω is the sending probability of a zero copy package; t_{IT} is the time if the package is sent and t_{ID} is the time if the package is dropped.

We can easily conclude that when there are no sufficient units of buffer, the new-coming packages will be dropped by network interface, which is different from the situation of enough units of buffer.

Condition 1. There are enough units of buffer that no package will be dropped.

Under the Condition 1 and Definition 2, we can get (4) from the state-transition figure:

$$t_{IT} = C_{Alloc} + C_{DMA} + \beta + \frac{1}{2n}\alpha + \frac{1}{n}\left(\sum_{i=1}^{n}\left(i \cdot C_{Scan}\right)\right) + C_{Send} \quad (4)$$

where β is the managing time of zero copy software; n is the number of units; α is the time interval between the two times of buffer scanning by driver; C_{Scan} is the time taken by driver to scan one unit of buffer; C_{Send} is the time interval between the state of Flag_Send_1 and the state of Flag_No_Use. Note that random probability is 1/2. We can also get t_{ID}:

$$t_{ID} = C_{Alloc} + C_{DMA} + \beta + \frac{1}{2n}\alpha + \frac{1}{n}\left(\sum_{i=1}^{n}\left(i \cdot C_{Scan}\right)\right) + C_{Drop} \quad (5)$$

where C_{Drop} is the time to release a buffer. That is the time interval between Flag_Drop and Flag_No_Use. From (3) to (5), we can get:

$$\begin{aligned} t_I &= \omega \cdot t_{IT} + (1-\omega) \cdot t_{ID} \\ &= C_{Alloc} + C_{DMA} + \beta + \frac{1}{2n}\alpha + \frac{1}{n}\left(\sum_{i=1}^{n}\left(i \cdot C_{Scan}\right)\right) + \omega \cdot C_{Send} + (1-\omega) \cdot C_{Drop} \end{aligned} \quad (6)$$

Then, from (1), (2), (6), we can know the package managing t_M:

$$t_M = \rho \cdot t_I + (1-\rho) \cdot t_B = \frac{1}{2n}\rho \cdot \alpha + \frac{1}{2} \cdot \rho \cdot C_{Scan} \cdot (n+1) + C \quad (7)$$

where

$$\begin{aligned} C &= C_{Alloc} + C_{DMA} + \rho \cdot \beta + \rho \cdot \left(\omega \cdot C_{Send} + (1-\omega) \cdot C_{Drop}\right) \\ &+ (1-\rho) \cdot C_{NM} \approx \rho \cdot \beta + (1-\rho) \cdot C_{NM} \end{aligned} \quad (8)$$

From (7), we can find that C, C_{Scan} and ρ are constant. Then the less value α and n, the less time driver need to manage a package. However, we can not reduce α and n with no limit for the following reasons.

① Too low the value of α will cause the scan procedure running almost every time, which will not allow CPU do any other jobs.

② Too few of n will make the units of buffer be used up and no units else can be ready for new coming network packages. In this situation, Condition 1 will not take effect, which means all the reasoning above will have no sense.

So it is very important to get the restriction of n. We can do this from the following two aspects.

All packages should be received. It means p (the sending speed of network packages) should not be higher than the speed of managing. That is $p \leq 1/ t_M$. In this way, if we could estimate the upper speed of p, we can get the range of n. From (7) and the limit of t_M:

$$\begin{aligned} &\frac{1}{2n}\rho \cdot \alpha + \frac{1}{2}\rho \cdot C_{Scan} \cdot (n+1) + C \\ &\leq \frac{1}{p} \Rightarrow \frac{1}{n}\alpha + C_{Scan} \cdot n \leq \frac{2}{\rho}\left(\frac{1}{p} - C\right) - C_{Scan} \end{aligned}$$

Let $C' = \frac{2}{\rho}\left(\frac{1}{p} - C\right) - C_{Scan}$, we may get:

$$\begin{aligned} &\frac{1}{n}\alpha + C_{Scan} \cdot n \leq C' \Rightarrow C_{Scan} \cdot n^2 - C' \cdot n + \alpha \leq 0 \\ &\Rightarrow \frac{C' - \sqrt{\Delta}}{2 \times C_{Scan}} \leq n \leq \frac{C' + \sqrt{\Delta}}{2 \times C_{Scan}} \end{aligned} \quad (9)$$

where $\Delta = (C')^2 - 4 \times C_{Scan} \times \alpha$.

Although we construct the mathematic model in the form of a single package, the packages are managed in block in reality. So the number of units we allocate should be larger than that of the packages generated in the time interval between two times of driver scan. Hence,

$$\begin{aligned} &n \geq p \cdot \left(\alpha + \left(t_I - C_{DMA} - \beta\right)\right) \\ &\Rightarrow n \geq \frac{2p}{2 - p \cdot C_{Scan}} \cdot \left(C_{Alloc} + \alpha + \frac{C_{Scan}}{2} + \omega \cdot C_{Send} + (1-\omega) \cdot C_{Drop}\right) \\ &\Rightarrow n > \frac{2p}{2 - p \cdot C_{Scan}} \cdot (\alpha) > p \cdot \alpha \end{aligned} \quad (10)$$

Up to now, we know the limit of n under the Condition 1. However, when package sending speed is higher than the managing speed, not all of the packages can be received. Then we will assess the loss of the packages.

Condition 2. Too high the sending speed makes packages dropping happen.

Under Condition 2, there is no unit else for the new coming package. If the burst flow of packages makes m pieces of buffers in the network interface can not be mapped to kernel units, the driver can not receive any more packages by DMA. Then the average recovery time is

$$t_H = \frac{1}{2}m \cdot t_M = \frac{1}{2}m \cdot \left(\frac{1}{2n}\rho \cdot \alpha + \frac{C_{Scan}}{2} \cdot \rho \cdot (n+1) + C\right) \quad (11)$$

The number of loss packages in t_H is

$$N_S = p \cdot t_H = \frac{1}{2}p \cdot m \cdot \left(\frac{1}{2n}\rho \cdot \alpha + \frac{C_{Scan}}{2} \cdot \rho \cdot (n+1) + C\right) \quad (12)$$

Up until now, we have made a mathematic model for SEPCOM in the above. We also limit n from the model under Condition 1 and assess the recovery time and the number of loss packages under Condition 2. In the next section, we will optimize the managing time tM and make some numerical computation.

5. QUANTITATIVE ANASYSIS OF SEPCOM

In this section, we will firstly take our IBM server as an example to show some concrete limits of model and the optimum value of n. Then we will assess the number of packages we will lose in reality under the Condition 2.

5.1 Limit of β

We will estimate some of the parameters we talked above in order to limit n. The machine we test has two 3GHz Intel Pentium IV processors. So C_{Scan} is approximately 10^{-9}. Suppose $p=2\times10^5$pps (packages per second), $\rho=0.9999\approx1$, α=1ms. We can get from (9) that

$$\begin{aligned}\Delta &= (C')^2 - 4\times C_{Scan}\times\alpha \\ &\approx \left(\frac{2}{\rho}\left(\frac{1}{p}-(\rho\cdot\beta+(1-\rho)\cdot C_{NM})\right)-C_{Scan}\right)^2 - 4\times C_{Scan}\times\alpha \\ &\approx \left(2\left(\frac{1}{2\times10^5}-\beta\right)-10^{-9}\right)^2 - 4\times10^{-9}\times10^{-3}\end{aligned} \quad (13)$$

Since $\Delta\geq0$, $\left(2\left(\frac{1}{2\times10^5}-\beta\right)-10^{-9}\right)^2-4\times10^{-9}\times10^{-3}\geq0\Rightarrow\beta<4\times10^{-6}$.

So the managing time of zero copy software for one package would not be more than 4 μ s, or we will lose the consequent packages.

5.2 Limit of n

Suppose β=2×10-6s, from (9), we can get

$$\Delta\approx\left(2\left(\frac{1}{2\times10^5}-\beta\right)-10^{-9}\right)^2-4\times10^{-9}\times10^{-3}\approx32\times10^{-12}$$

$$\therefore\frac{C'-\sqrt{\Delta}}{2\times C_{Scan}}\leq n\leq\frac{C'+\sqrt{\Delta}}{2\times C_{Scan}}\Rightarrow172<n<5828$$

.

From (10), we will get

$$n>p\cdot\alpha=2\times10^5\times10^{-3}=200$$

.

In sum, we can conclude that 200<n<5828. If n is not in the range, there will be loss of packages.

5.3 Optimum Value of n

To minimize t_M, consider (7).

$$t_M=\frac{1}{2n}\rho\cdot\alpha+\frac{1}{2}\cdot\rho\cdot C_{Scan}\cdot(n+1)+C\geq\rho\sqrt{\alpha\cdot C_{Scan}}+\frac{1}{2}\rho\cdot C_{Scan}+C$$

If and only if $\frac{1}{2n}\rho\cdot\alpha=\frac{1}{2}\cdot\rho\cdot C_{Scan}\cdot n$, t_M has its minimum value. At this time, $n=\sqrt{\alpha/C_{Scan}}=10^3$ and $t_M=\rho\sqrt{\alpha\cdot C_{Scan}}+\frac{1}{2}\rho\cdot C_{Scan}+C\approx3\times10^{-6}$. So if we divide the kernel buffer to 1000 units, we will get the most efficiency.

5.4 Assess the Loss of the Packages

Under the Condition 2, suppose n=1000, consider (12).

$$N_S=\frac{1}{2}p\cdot m\cdot\left(\frac{1}{2n}\rho\cdot\alpha+\frac{C_{Scan}}{2}\cdot\rho\cdot(n+1)+C\right)\approx0.3m$$

So if the burst of package flow makes no unit of kernel memory for DMA transmission, it will cause 0.3 times package loss. Note that if α > 15ms, N_s > m. The avalanche effect will make SEPCOM hard to recover.

6. EXPERIMENTS AND EVALUATION

In order to test the efficiency of SEPCOM and our analysis, we modify the network interface driver and correspondingly one zero copy software. The software's aim is to monitor and filter the traffic flow. Our testing computer is one IBM server which is equipped with two 3GHz Intel Pentium IV processors, 4GB memory, two Intel e1000 network interface cards and the operating system is Linux 2.6.11. Since we need to generate the extremely large number of traffic flows, we use SmartBits [11] as package generator and the equipped software SmartFlow, which controls SmartBits to generate flows automatically. We choose two types of test: throughput test and latency test.

6.1 Throughput Test

The Throughput test determines the maximum transmission rate at which the DUT can forward IP traffic with no frame loss, or at a user-specified acceptable frame loss [9]. In this test, we will make a comparison between our SEPCOM and simply use libpcap [10] to capture packages. We can see figure 3 below. The upper line is the result of SEPCOM and the lower line is the result of libpcap.

We can easily find out that SEPCOM achieves the utmost of the network interface card when the size of package comes to 450 Byte. However, when using libpcap, even the package size is 1518 Byte, the maximum size of Ethernet, it can not achieve the utmost speed. We have no zero copy implements for Gigabit NIC of others, so we can not make some comparisons between them.

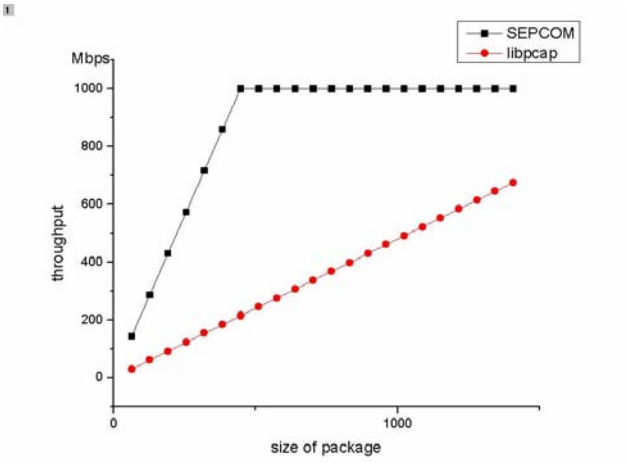

Figure 3. the throughput test results

6.2 Latency Test

Latency test is to measure the latency (transmission delay) introduced by the DUT [9]. In this test, our goal is to check the analysis above. So we choose different n, by comparison the latency results of the tests, we can find the optimum value of n.

In figure 4, the lower line is the result of our computation under the assumption that α=1ms and $\beta=2\times10^{-6}$s. The upper line is the result we get from SmartFlow. From the results above, we can find our analysis is very precise. The reason why the theoretical value is less than the testing value may be the result of omitting some tiny parameters when computing.

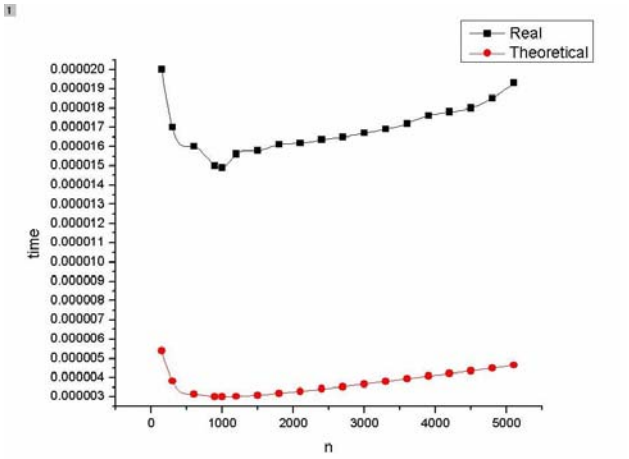

Figure 4. the latency test results

7. CONCLUSION AND FUTURE WORK

In this paper, we present SEPCOM, a zero copy model for extremely fast package management. SEPCOM achieves significant improvements in speed and compatibility over the former implements. It can give great help to increase the managing speed of IDS and firewall. We also make mathematic analysis of the model and optimize some parameters, which help us to limit some key parameters when programming. We also confine the time limit of zero copy software and assess the loss of packages after a burst of flow, which may also help us in the real situation.

SEPCOM is a young model. We will optimize the algorithm of memory management. For example, we can use queue to manage packages instead of poll method. We will also find a better algorithm for choosing the right zero copy software if there is more than one. Finally, we will continue to make clear the relationship between the CPU and α and to choose the best value of α.

8. ACKNOWLEDGMENTS

We would like to thank Shengyong Li for his constructive comments and suggestions. We would also like to thank the anonymous reviewers for their insightful feedback.

9. REFERENCES

[1] E. Cooper, P. Steenkiste, R. Sansom, and B. Zill: Protocol Implementation on the Nectar Communication Processor, Proceedings of SIGCOMM'90 Conference on Comm. Architectures, Protocols and Applications, Aug. 1994.

[2] B. Traw: Applying Architectural Parallelism to High Performance Network Subsystems, Ph.D. Dissertation, University of Pennsylvania, 1995.

[3] C. Dubnicki, E.W. Felten, L. Iftode and K. Li: Software support for virtual memory-mapped communication, in: Proc. 10th IEEE Int. Parallel Proc. Symp., Honolulu, HI, April 1996, pp. 372–381.

[4] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke and C. Dodd: The virtual interface architecture, IEEE Micro 18(2) (March–April 1998) 66–76.

[5] T. von Eicken, A. Basu, V. Buch and W. Vogels, U-Net: A user-level network interface for parallel and distributed computing, in: Proc. of 15th Symposium on Operating Systems Principles (SOSP-15), Cooper Mountain, CO, USA, December 1995 (ACM).

[6] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley: Afterburner - A network-independent card provides architectural support for high-performance protocols, IEEE Network, July 1993.

[7] P. Druschel, L Peterson. Fbufs: A High-Bandwidth Cross-Domain Transfer Facility, Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, Dec. 1993.

[8] H.K. Jerry Chu: Zero-Copy TCP in Solaris, in: Proc. of the USENIX 1996 Annual Technical Conference, San Diego, CA, USA, January 1996 (The USENIX Association) pp. 253–264.

[9] Spirent Communications, Inc: the User Guide of SmartFlow version 4.5, pp. 296, 334

[10] Tim Carstens: Programming with pcap. http://www.tcpdump.org/pcap.htm

[11] Spirent: Spirent SmartBits Trusted Industry Standard for Router and Switch Testing, http://www.spirentcom.com/analysis/technology.cfm?media=7&WS=325&SS=110&wt=2

[12] SCAMPI, a Scalable Monitoring Platform for the Internet, Lieden University (in collaboration), Mar. 2002, http://www.ist-scampi.org.

[13] M. K. Gardner, W. Feng, and J. R. Hay, "Monitoring Protocol Traffic with a MAGNeT," in Passive & Active Measurement Workshop, Fort Collins, Colorado, 3 2002.

[14] Martin Roesch and Chris Green, "SNORT: The Open Source Network Intrusion Detection System 1.9.1," Nov. 2002, http://www.snort.org/.

[15] A. Moore, J. Hall, C. Kreibich, et al. Architecture of a Network Monitor. In PAM, 2003.

[16] Ubik. S, Smotlacha. V, Simar. N, Performance monitoring of high-speed networks from the NREN perspective, Terena Networking Conference. 2004

[17] J. Hall, I. L. Ian Pratt, and A. Moore, "The Effect of Early Packet Loss on Web Page Download Times," in Passive &Active Measurement Workshop 2003 (PAM2003), Apr. 2003.