Performance Review of Zero Copy Techniques

Jia Song *song3202@vandals.uidaho.edu*
Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 83844-1010 USA

Jim Alves-Foss *jimaf@uidaho.edu*
Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 83844-1010 USA

Abstract

E-government and corporate servers will require higher performance and security as usage increases. Zero copy refers to a collection of techniques which reduce the number of copies of blocks of data in order to make data transfer more efficient. By avoiding redundant data copies, the consumption of memory and CPU resources are reduced, thereby improving performance of the server. To eliminate costly data copies between user space and kernel space or between two buffers in the kernel space, various schemes are used, such as memory remapping, shared buffers, and hardware support. However, the advantages are sometimes overestimated and new security issues arise. This paper describes different approaches to implementing zero copy and evaluates these methods for their performance and security considerations, to help when evaluating these techniques for use in e-government applications.

Keywords: Zero Copy, Network Security, Security/Performance Tradeoffs

1. INTRODUCTION

In addition to growth of corporate servers, there has been tremendous interest and growth in the support of e-government services. This includes remote access to public information, databases, forms, guidance, training materials as well as access to personal information such as treasury holdings, social security, income taxes, and government benefits. In addition, e-government also supports internal access to government documents, support of paperless offices and workflow improvement and communication support for critical services including first responders in emergencies. Cost-effective e-government solutions will require use of shared organizational servers, such as cloud servers, and high performing computers that can provide users with timely and secure access to information. Unfortunately, we have found that there is often a disconnect between marketed performance-improving solutions and security needs. This paper addresses one “performance-improving” technology with respect to security needs. We have found that some of the solutions that have been proposed and implemented will not work with security technologies, such as encryption. The understanding of the impact of performance-improving solutions on security is important when comparing vendors' performance benchmarks and claims when evaluating systems for purchase, and is important for developers to understand when making implementation decisions.

Therefore, the intent of this paper is to bring to light an example of security concerns that we believe should be addressed in the development process and in security requirements such as those specified by US Federal Acquisition Regulation (FAR) Part 39.101 and OMB Circular A-130 while supporting e-government efforts such as those specified in the “E-Government Act of 2002”. Similar regulations exist in other countries or in corporate policies. Corporations and governments in the market for servers should pay careful attention to performance benchmarks developed by vendors.

For example, the FAR Part 39.101 requires that agencies identify their requirements pursuant to: best management practices for energy-efficient management of servers and Federal data centers and shall include the appropriate information technology security policies and requirements. Energy-efficient management often translates into performance, since a data-center can use a lesser number higher performing servers to do the same work, and with less energy. As we discuss in this paper, we need to be aware of the tradeoff between some performance technologies and security technologies. We believe there is a need for the development of a set of benchmarks or at least checklists/configuration guides that enable the acquisition officers and developers to make the most appropriate choice of technologies given the security needs of the application environment.

1.1 Performance and Security

As demand for higher performance computers grew, CPU designs developed from single core to dual-core, quad-core or even hexa-core. However, a quad-core processor does not indicate that the performance of the overall system is four times the performance of a single core processor. The performance of the system is limited by many other factors, such as access to system memory. According to Brose [1], CPU processing power improves about 50% each year, while the memory only has an average 35% increment at the same amount of time[1]. Improvement in memory usage is increasingly a critical factor for system performance. This is especially important for network applications, such as severs, which transfer a lot of data, and cannot benefit from cache size increases.

Servers used by industry and government are critical not only for their performance, but also their security. To make the government more transparent, web servers for the public are used to inform the public about new policies and the latest news from the government. What's more, the servers used among governmental employees must be secure enough to avoid leaking information and to prevent hacking. Therefore, severs used by government must ensure their efficiency and security.

Traditionally, in server environments, such as servers used in e-government, when sending data to a network, data is loaded from disk and transferred to a network card buffer. The whole procedure consumes a lot of CPU cycles and memory bandwidth, for data must be copied between application memory space and kernel memory space. However, most of these data copies are redundant and can be minimized by implementing zero copy techniques. Zero copy is a name used to refer to a variety of techniques which help reduce useless memory accesses, usually involving elimination of data copying. By implementing zero copy, less data copies are needed when data is transferred between buffers. This helps reduce the number of unnecessary memory accesses and CPU consumption, and therefore enhance overall system performance.

1.2 Zero Copy for Performance

In recent years, there have been a number of studies regarding zero copy and different kinds of zero copy techniques have been applied. Zero copy can be classified into two groups: (1) reduce data copies between devices (disk or network card) and kernel buffers, and (2) reduce data copies between kernel buffers and application buffers.

Typically, data copies between devices and kernel buffers can be reduced by using DMA (Direct Memory Access). DMA allows special purpose hardware to read or write main memory without involving the CPU, which greatly reduces CPU consumption and also eliminates redundant data copies between device and kernel buffers.

Zero copy can also reduce the number of copies between user memory space and kernel memory space. Several different approaches have been proposed to solve this problem, and they are all called zero copy. For example, memory mapping is used to map the user buffer to the

[1] This means that relative to CPU speeds, memory performance is cut in half every 7 years.

kernel buffer; in LyraNET, a modified `sk_buff` structure can be used to eliminate data copies; and buffers can be shared, so stored data can be referenced via pointers. In addition, some system calls in Linux support zero copy by reducing data copies such as the `sendfile()` and `splice()` system calls. Instead of copying data, descriptors are appended to buffers in sendfile. The system call `splice()` sets up a data transfer path between different buffers in kernel memory.

However, in addition to performance benefits of zero copy techniques, it is important to understand the security implications of using those techniques. This is important for commercial servers as well as e-government systems.

Zero copy helps reduce the number of data copies and context switches which improves the CPU performance. But it also causes some new problems that we have to understand. This paper focuses on analyzing different zero copy techniques which are used to reduce the data copies in TCP/IP transmission, which are being implemented in many commercial servers which could end up being used by government offices to gain better performance.

The remainder of this paper evaluates the use of zero copy for performance in secure environments as is laid out as follows. The traditional data transfer approach and security issues are presented in Section 2 and zero copy techniques are described in Section 3. Problems caused by zero copy are examined in Section 4; this section also discusses security issues relating to zero copy techniques, which will help developers and acquisition offices in determining the appropriate use of this technology.

2. BACKGROUND

2.1 Traditional Data Transfer Method

Traditionally, if a user/server application wants to send data to a network, the data is read from disk to kernel memory, then stored in a buffer allocated by the user application in user memory, then sent to the kernel buffer associated with the network stack, and finally sent to the network interface card (NIC) for transfer over the network. Although the whole procedure needs only two system calls -- `read()` and `write()`, from the view of kernel, it is quite inefficient, even if DMA is supported by hardware. The traditional data transfer method is shown in Figure 1. We will examine this approach step by step to illustrate the CPU copies[2] and context switches.

First, if the data is not cached in a kernel buffer, the kernel needs to load the data from disk to a kernel buffer. For this step, DMA is typically used, which allows this first copy to consume little CPU resources. However, extra effort is needed to manage the DMA copy. Then, the data needs to be transferred into the buffer allocated by the application in user memory. This step is completed by the `read()` call, which, is typically done by a CPU copy in a kernel function such as `copyout()` or `copy_to_user()` [1]. Also, the `read()` system call needs a context switch from user mode to kernel mode. After copying data to the application buffer, the `read()` system call returns, which causes another context switch from kernel mode to user mode.

From the application buffer to the corresponding network stack in the kernel buffer the data needs to be transferred using a user-to-kernel copy function, such as `copyin()` or `copy_from_user()` [1]. This kernel buffer is not the same buffer used for the initial fetch from disk. It is a new buffer which is associated with the network stack. Data is packetized according to the Internet protocol in the network stack. For this step, the use of the `write()` system call generates a context switch from user mode to kernel mode. Finally, the prepared data are sent from the kernel buffer to the buffer on the network interface card, and then transferred over the

[2] We use the term CPU copy to refer to any time the CPU reads a memory cell and writes a result back out.

network. This fourth copy can be done using DMA. Although no CPU copy is needed, the returned `write()` system call forces another context switch from kernel back to user mode.

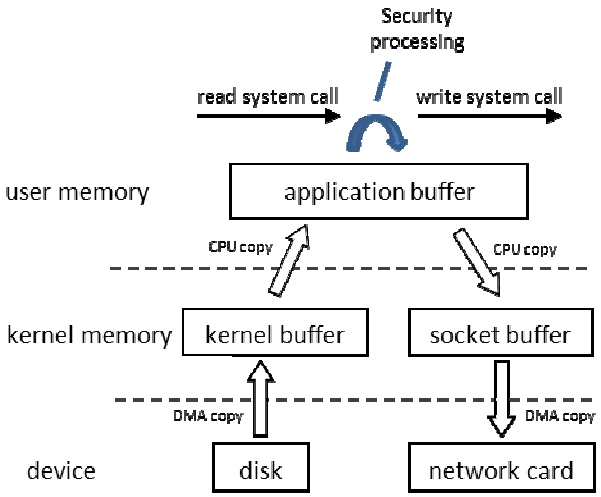

FIGURE 1: Traditional Data Transfer

2.2 Secure Communications

For secure communication across the network, applications will encrypt data packets. Depending upon the security mechanism used, encryption could occur at a high level or low level of the network stack. Servers and secure network applications often use SSL (Secure Socket Layer) for secure communication.

There are two aspects to SSL that can slow down system processing. The first is the negotiation of the encryption key and authentication. These processes typically use modular arithmetic over very large numbers and take a good amount of CPU time. There are hardware accelerators, such as SSL cards, that allow an application to offload this computation load. This set up of the encryption is beyond the scope of this paper. The second aspect is the actual encryption of the data. This requires applying an encryption algorithm to all of the data, reading the results from one buffer and writing them to another buffer (labeled *security processing* in **Error! Reference source not found.**). Even with CPU help, data copies are needed, for example in 2010 Intel introduced the AES instruction set extension to support fast encryption, and utilize 128-bit registers in the Intel architecture. These instructions require CPU access to the data and hence require a CPU copy.

In addition to encrypted communication, some security system requires filtering or guarding by the application. Typically this occurs in a secure communications environment where messages are being transmitted between different security domains. The concepts of zero copy still work, but the source is often another network device and not a disk. Filtering also occurs when data retrieved from a database is filtered before release. In these situations, the application needs to read and review the information before making the decision whether to allow the information to go out (labeled *security processing* in **Error! Reference source not found.**). We also have to make sure that no unauthorized processes get access to the sensitive information. As we examine different zero copy techniques, we will have to see if they provide any performance enhancement in encryption or filtering secure environments.

3. ZERO COPY TECHNIQUES

There are at least four data copies and four context switches during the data transmission in the traditional data transfer approach. According to Huang et al [2], in addition to the costs of data copies, software interrupts consume a great portion of CPU usage, for interrupts are generated

for each packet. As we see, some of the data copies are redundant and some of the context switches between user mode and kernel mode can be eliminated by avoiding calling the read and write system calls separately. As a result, zero copy techniques are implemented to reduce the unnecessary data copies and improve the CPU performance. Many different schemes are used to help implement zero copy techniques, such as memory remapping, shared buffers, different system calls, and hardware support.

3.1 Dynamic Remapping

Dynamically remapping memory can eliminate the data copies between the application buffer and kernel buffer. As **Error! Reference source not found.** shows, firstly, data loaded from disk is stored in a kernel buffer by DMA copy. Then the pages of the application buffer are mapped to the kernel buffer, so that the data copy between kernel buffers and application buffers are omitted.

According to Stancevic [3], to remap the memory, the Linux `mmap()` system call will be called instead of the `read()` system call. After data are copied to the kernel buffer by DMA copy, the system call `write()` causes the data to be copied from the kernel buffer to another socket buffer in kernel address space. This means that the original copies from kernel buffer to application buffer and following copies from application buffer to socket buffer are substituted by only one data copy directly from kernel buffer to socket buffer. Then data will be copied to the network card buffer by DMA. Dynamic remapping needs three CPU copies (two DMA copies and one CPU copy) and four context switches, since the system calls `mmap()` and `write()` are made during data transmission.

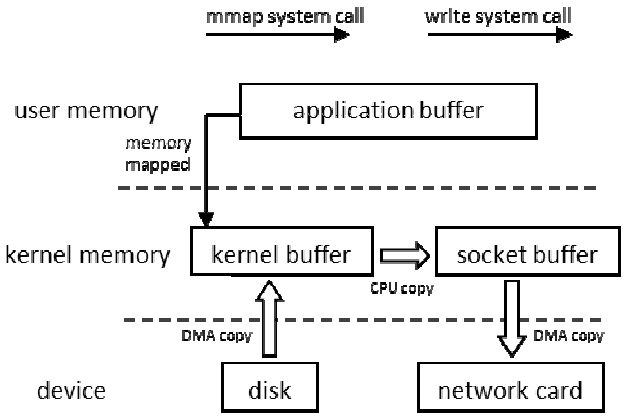

FIGURE 2: Data copy using memory remapping.

In an environment requiring encryption, unless the kernel supports an encrypt-while-copy operation, we must allow the user application to copy data from the kernel buffer to the socket buffer. In a filtering situation we must allow the user application to process the data. Therefore, unless we can link the application buffer to both the kernel buffer and the socket buffer, and have the security routine read from one to the other, we will get no savings using dynamic remapping. We also have to make sure that no other process can get access to the kernel buffer and bypass the security routine.

3.2 Shared Buffer in Kernel Memory Space

In INSTANCE-buffer [4], a memory area is shared in kernel space. As **Error! Reference source not found.** shows, there is a `buf` structure in the kernel buffer and `buf.b_data` is a pointer which points to the shared memory region which is in kernel memory space. Data fetched from disk are transferred to the shared memory area according to the `b_data` pointer. Furthermore, in the socket buffer, the m_data pointer in the `mbuf` structure points to the shared memory area.

Therefore, data are copied from the memory pointed by `mbuf.m_data` to the buffer on the network card. According to Halvorsen and Jorde et al [4], the INSTANCE-buffer scheme allocates the shared memory region and the structures (`buf` and `mbuf`) statically, which means the time cost for allocating memory and variables is reduced.

In a secure environment, this technique will cause problems. To encrypt, we have to read the data from the source buffer and write it to a destination buffer. Even if we can encrypt in place, there will still need to be a read and write of data. A simple filter process will work as long as we can be sure that data is not sent out to the network card without the filter approving the message, but we cannot use a complex filter that modifies the data. In high-assurance systems this may not provide enough separation and isolation control.

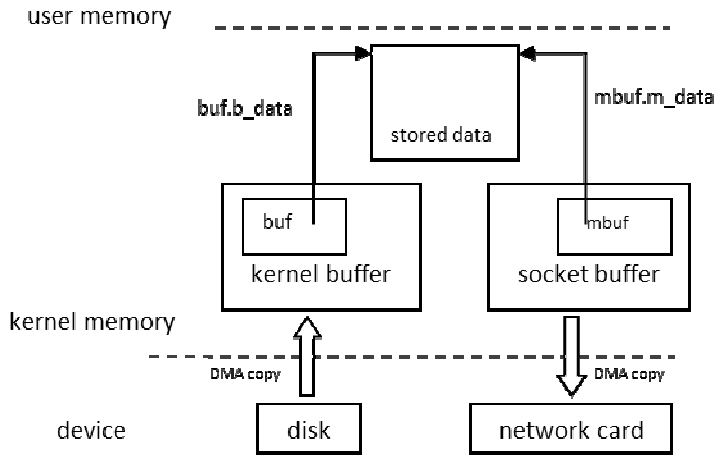

FIGURE 3: Shared buffer with structures.

3.3 Shared Buffer Between User and Kernel

In Linux, the `sk_buff` structure contains all the control information about individual network packets. Data sent to the network are copied to one or more `sk_buff` buffers. In LyraNET [5], the original `sk_buff` structure is modified to an enhanced copy elimination `sk_buff` structure, which eliminates the data copy by just passing the address of the data buffer. According to Chiang and Li [5], the modified `sk_buff` structure includes a new array with two elements, named `dataseg`, to record addresses for data without copying the data.

LyraNET is focused on reducing the data copies from user buffer to kernel buffer and from kernel buffer to network card. Using the enhanced copy elimination `sk_buff` structure, information about the protocol headers is copied to the network card and then data can be retrieved correctly according to the pointers and the data length stored in the modified `sk_buff`. LyraNET also uses DMA to copy data from the socket buffer to network card (Figure 4).

In a secure environment the security routine will have to copy data from the shared data region into a new region upon encryption. Filtering requires a CPU read of the data, but may not require a write. The system will have to ensure that the shared region is not available to the socket buffer until after it has been approved.

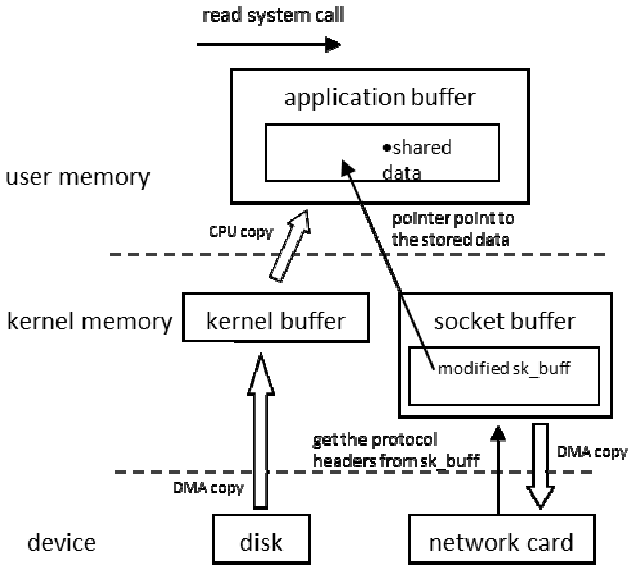

FIGURE 4: Shared user and kernel buffer

3.4 Sendfile

According to Stancevic [3], in Linux kernel version 2.1, the `sendfile()` system call was introduced to simplify the transmission of data over the network and between two local file descriptors. As Figure 5 shows, when calling the `sendfile()` system call, data are fetched from disk and copied into a kernel buffer by DMA copy. Then data are copied directly from the kernel buffer to the socket buffer. Once all data are copied into the socket buffer, the `sendfile()` system call will return to indicate the completion of data transfer from the kernel buffer to socket buffer. Then, data will be copied to the buffer on the network card and transferred to the network.

The `sendfile()` method is much more efficient than the combination of `read()` and `write()` system calls [6]. Compared with the traditional data transmission method, it replaces the `read()` and `write()` system calls, which reduces the number of context switch from four to two. Compared with dynamic remapping, `sendfile()` reduces the costs for virtual memory management. However, it still needs three data copies (two DMA copies and one CPU copy) to finish the data transmission.

This approach to zero copy bypasses the application completely, avoiding any application specific security routine or CPU processing of the data, making encryption and filtering impractical.

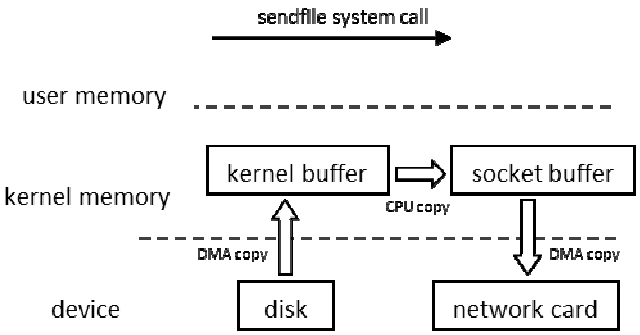

FIGURE 5: Data transfer using sendfile.

3.5 Sendfile With DMA Scatter/Gather Copy

The method that enhances the `sendfile()` system call with DMA scatter/gather copy can eliminate the CPU copy between kernel buffer and socket buffer found in the `sendfile()` method. Different from the DMA which maps each buffer one by one and then do the operation, DMA scatter/gather copy maps the whole list of buffers at once and transfer them in one DMA operation [7]. In this method, support from hardware is needed, for the network interface gathers data from various memory spaces (Figure 6). When calling `sendfile()`, data on disk are loaded into a kernel buffer using DMA transfer. Then, only the buffer descriptor is sent to the socket, instead of coping all of the data to the buffer. The descriptor contains information about the length of the data and where it is. Using this information, header and trailer of the data packet can be generated. Then by using the DMA scatter/gather operation, the network interface card can gather all the data from different memory locations and store the assembled packet in the network card buffer.

Hardware which supports DMA scatter/gather copy eliminates the CPU copy between kernel buffer and socket buffer. It means that no CPU copies occur during the data transmission from disk to network card, which helps to increase the performance of the CPU. Only one system call, `sendfile()`, is made in this approach, so there are only two context switches. Additionally, since hardware supports the DMA scatter/gather operation, data in memory are not required to be stored in consecutive memory spaces.

This approach to zero copy bypasses the application completely, avoiding any application specific security routine or CPU processing of the data, making encryption and filtering impractical.

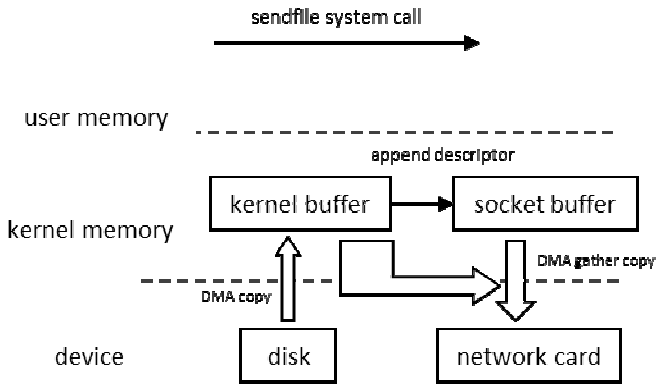

FIGURE 6: Sendfile and DMA scatter/gather copy.

3.6 Splice

According to the Linux man page [8], the Linux 2.6.17 kernel has a new system call, `splice()`. This call takes two file descriptors and an integer as parameters. It copies data, which size is specified by the integer, from one file descriptor to the other file descriptor using a pipe. By doing this, it does not need to copy data between kernel space and user space.

When using this approach, data are copied from disk to kernel buffer first. Then the `splice()` system call allows data to move between different buffers in kernel space without the copy to user space, for only file descriptors are transferred. In some zero copy implementations, `splice()` is called to move data from the kernel buffer to socket buffer which is also in kernel space. Then data are copied from the socket buffer to the network card buffer by DMA transfer. The system call splice eliminates the data copy between kernel buffer and socket buffer.

Unlike the method `sendfile()` with DMA scatter/gather copy, `splice()` does not need support from hardware. In addition, two context switches are bypassed compared to the

traditional method, therefore only two context switches are needed. Instead of coping data to an application buffer, `splice()` works as a pipe between two kernel buffers to allow direct data transfer. However, there must be two file descriptors opened to both the input and output devices (Figure 7).

Fall and Pasquale [9] indicated that, in the best case, the performance of using `splice()` is at 1.8 times than the maximum throughput of the traditional data transfer method. This approach to zero copy bypasses the application completely, avoiding any application specific security routine, making encryption and filtering much more difficult.

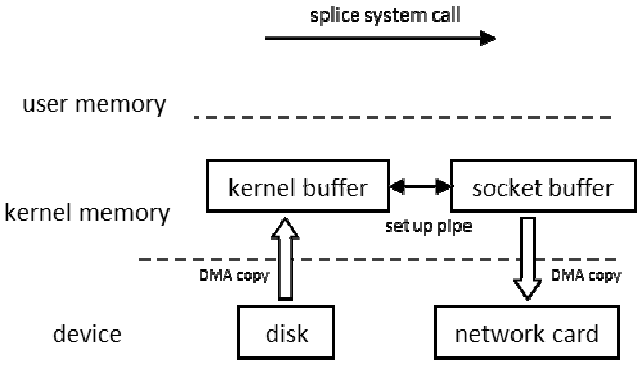

FIGURE 7: Data transmission using splice.

4. POTENTIAL PROBLEMS AND SECURITY CONCERNS

The implementation of zero copy technique helps to decrease overhead so as to release CPU cycles. Redundant data copies are reduced which saves more CPU and memory resources. However, the benefits of zero copy are sometimes overestimated, for the implementation of zero copy brings other new problems which may nullify the improvement of the system. For example, memory mapping needs costly virtual memory operations, the method of `sendfile()` with DMA scatter/gather copy needs support from hardware to gather data from various memory locations. In addition, zero copy techniques ignore or even introduce security concerns as well.

4.1 Dynamic Remapping Problem

Dynamic remapping uses `mmap()` instead of the `read()` system call, which reduce one copy during the data transmission. However, data might be modified by the user program while being prepared for transmission, this may cause security problems.

This problem can be prevented by implementing the Copy-on-write (COW) technique. COW is a simple technique. When the pages of the application buffers are mapped into kernel space, these pages can be written by the user. After the `mmap()` system call returns, both application and socket buffers can then only read the kernel buffer. The user can then start writing to the kernel buffer when the `write()` system calls return, which means the required data are sent to socket buffers. COW is implemented when using page remapping. It helps to ensure that while data is transferring, write permission is controlled. So the content of the physical memory page cannot be changed. However, COW is a costly event which consumes more CPU resource than a simple CPU copy [10].

4.2 Pre-allocated Buffer Problem

Some zero copy techniques use pre-allocated buffers to store data. By using pre-allocated buffer, the costly virtual memory operations are done only once. In addition, the pre-allocated buffer eliminates the exchange of the size of data between sender and receiver, but it causes another problem: the pre-allocated buffer space in receiver might be exhausted. If the receiver's pre-

allocated buffers are exhausted, the receiver has to wait until some buffers are usable. According to Yamagiwa et al [11] even if there are a lot of pre-allocated buffers, the receiver still might be exhausted since these buffers cannot be paged out.

4.3 Sendfile System Call Problem

According to Brose [1], the `sendfile()` interface requires two file descriptors. The first one is a readable memory mappable file such as regular file or a block device. The second one can be a writable file or a network socket. The `sendfile()` system call can only transmit data which has specified length. The other disadvantage of the sendfile approach is that it is hard to implement `sendfile()` on the receivers' end, for network transfers are asynchronous and the `sendfile()` system call is not a standard system call among different Linux or UNIX systems. In some systems, `sendfile()` is routed to the `transferto()` system call, which makes the implementation of this approach more difficult.

4.4 Sendfile With DMA Scatter/gather Copy Problem

The method that uses `sendfile()` with DMA scatter/gather copy enhances the performance of the CPU by avoiding CPU copies. Since data can be copied from various memory areas to the network card buffer by DMA scatter/gather copy, which is supported by hardware, cache coherency is not required anymore. However, according to Brose [1], the source buffer may be a part of the page cache, which means it is available for the generic read functionality and may be accessed in traditional way as well. As long as the memory area can be accessed by the CPU, the cache consistency has to be maintained by flushing the caches before DMA transfers.

5. CONCLUSION

5.1 Zero-Copy Impacts

Over the past decade we have seen tremendous growth in e-government services at the state and federal level. The OMB report “FY 2009 Report to Congress on the Implementation of The E-Government Act of 2002” highlights federal agencies' efforts to implement e-government. These activities have included performance enhancing and cost saving consolidation of services into server farms and clouds, and increased availability of private and secure information. There are other reports that highlight similar efforts by the states. These reports indicate that there is a growth in the use and availability of e-government.

Vendors developing solutions for data servers will use new technologies if they can increase performance, such as zero copy. Although the implementations of zero copy techniques help reducing the redundant data copies between the kernel memory space and user memory space, they also bring us new problems. Memory remapping approach needs virtual memory operations and COW which might nullify the improvement of the CPU performance. When using the pre-allocated buffer or static buffer, size of the buffer becomes a problem. Because the buffer overflow attack and the condition that the buffers are exhausted need to be considered.

	CPU copy	DMA copy	System call	Context switches	Advantages and disadvantages	App-level security processing
Traditional data transfer	2	2	*read* *write*	4	Redundant data copies consume many CPU cycles. 2 system calls cause 4 context switches. Large consumption of memory bandwidth.	Yes
Memory remapping	1	2	*mmap* *write*	4	Needs costly virtual memory operations. Needs COW to be implemented which costs a lot.	Yes

Shared buffer (INSTANCE -buffer)	0	2		0	Buffers need to be released after using. Pre-allocated buffer reduces the exchange of the data size, since the buffer size is fixed. Pre-allocated buffer needs less virtual memory operations, for the virtual memory operations only used when creating the buffer.	Yes
LyraNET	1	2	*read*	2	Need costly virtual memory operations. LyraNET still needs data copy from kernel buffer to application buffer.	No
Sendfile	1	2	*sendfile*	2	Reduces the costs for virtual memory management. *Sendfile* hard to implement due to asynchronous network transfers. *Sendfile* system call is not a standard system call among different Linux or UNIX systems.	Yes
Sendfile with DMA scatter/gather copy	0	2	*sendfile*	2	Reduces the costs for virtual memory management. Needs hardware which supports DMA scatter/gather copy. *Sendfile* hard to implement due to asynchronous network transfers. *Sendfile* system call is not a standard system call among different Linux or UNIX systems.	No
Splice	0	2	*splice*	2	When using *splice* system call, there must has two file descriptors which opened to both the input and the output devices.	No

TABLE 1: Comparison of different zero copy techniques.

Therefore, the implementation of zero copy techniques regard to network is limited by many factors. That is why zero copy techniques have not been widely adopted in the operating systems. Although these new techniques need more evaluation, zero copy is worth analyzing and implementing carefully for critical e-government services. Removing redundant CPU copies can improve the performance of servers, and this is especially useful for commercial and government servers which require high efficiency. The advantages and disadvantages of different zero copy techniques are compared in
TABLE 1. Any zero-copy technique that bypasses the application will limit the ability to perform security processing. Therefore the `sendfile()` and `splice()` system calls are not recommended for use in applications that require security processing. The right most column indicates whether security processing can be used with each zero-copy technique, providing some performance improvement.

5.2 Understanding the Implications

To better understand the implications of security on performance improving solutions, we recommend that the acquisition officer evaluate the proposed solution using questions similar to the following list:

1) Will the server need to provide encrypted communication?
   a) Can the encryption be performed at the connection-level by dedicated hardware, or will you need application-level processing?

   b) Will the server need to provide multiple encryption services, supporting different algorithms and protocols?
2) Will the service need to provide application-level security processing/filtering on the data being serviced?
   a) Will that processing be service specific (same processing for all users) or user-specific?
   b) Will the service require multiple filtering processes?
3) Does the server need to support multiple concurrent services (such as cloud servers)?
4) Does the proposed solution support the level of encryption and application processing required?

The answers to these questions will help acquisition officers and developers better understand the expected use cases of the system and better evaluate the use of performance-improving technologies such as zero-copy. Any system that requires the examination or processing of the contents of the message, such as encryption, or filtering will require either a hardware solution, or application-level processing.

A hardware solution, such as a cryptographic hardware module, allows the use of all of the zero-copy techniques by passing the data packet directly to the cryptographic hardware. However, if the system supports multiple encryption services, a hardware solution may become impractical, thus forcing the system to rely on an application-level software solution.

Any software-based solution that requires access to the full data packet, cannot utilize the zero-copy features with a "no" in the final column of TABLE 1, Lyrannet, Sendfile with DMA scatter/gather copy and Splice. Other techniques will allow application level security processing.

We believe that it would be beneficial to develop a set of benchmark scenarios that address encryption and filtering technologies at the connection and application level and the recommend that the vendors provide performance data with respect to these benchmarks. This can be done to help address a wide-variety of proposed performance-improving technologies, which may not work well for security-enabled environments, contrary to vendor's claims.

6. REFERENCES

[1] E. Brose. "ZeroCopy: Techniques, Benefits and Pitfalls," http://kbs.cs.tu-berlin.de/teaching/ws2005/htos/papers/streams-zcpy.pdf, 2005, [last accessed May 30, 2012]

[2] C. Huang, C. Chen, S. Yu, S. Hsu, and C. Lin. "Accelerate in-line packet processing using fast queue," in Proc. 2010 IEEE Region 10 Conference, 2010, pp. 1048–1052.

[3] D. Stancevic. "Zero Copy I: User-Mode Perspective," *Linux Journal,* vol. 2003 no 105, Jan. 2003, pp. 3.

[4] P. Halvorsen, E. Jorde, K. Skevik, V. Goebel, T. Plagemann, "Performance Tradeoffs for Static Allocation of Zero-Copy Buffers," in Proc. Euromicro Conference, 2002, pp. 138-143.

[5] M. Chiang and Y. Li, "LyraNET: A Zero-Copy TCP/IP Protocol Stack for Embedded Operating Systems," in Proc. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Aug. 2005, pp. 123-128.

[6] Linux Programmer's Manual, http://www.unix.com/man-page/Linux/2/sendfile/ Feb. 2010.

[7] J.Corbet, A.Rubini, and G.Kroah-Hartman. "Memory Mapping and DMA" in *Linux Device Drivers*, third edition, O'REILLY, Feb, 2005, pp.450.

[8] Linux Programmer's Manual, http://www.unix.com/man-page/Linux/2/splice/ Sep. 2009.

[9] K. Fall and J. Pasquale, “Exploiting In-Kernel Data Paths to Improve I/O Throughput and CPU Availability,” in Proc. Winter 1993 USENIX Conference, 1993, pp. 327-333.

[10] H.K.J. Chu. “Zero-copy TCP in Solaris,” Proc. USENIX Annual Technical Conference, 1996, pp. 253–264.

[11] S. Yamagiwa, K. Aoki, and K.Wada. “Active zero-copy: A performance study of non-deterministic messaging,” in Proc. International Symposium on Parallel and Distributed Computing, 2005, pp. 325–332.

Performance Review of Zero Copy Techniques

Jia Song *song3202@vandals.uidaho.edu*
Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 83844-1010 USA

Jim Alves-Foss *jimaf@uidaho.edu*
Center for Secure and Dependable Systems
University of Idaho
Moscow, ID 83844-1010 USA

Abstract

E-government and corporate servers will require higher performance and security as usage increases. Zero copy refers to a collection of techniques which reduce the number of copies of blocks of data in order to make data transfer more efficient. By avoiding redundant data copies, the consumption of memory and CPU resources are reduced, thereby improving performance of the server. To eliminate costly data copies between user space and kernel space or between two buffers in the kernel space, various schemes are used, such as memory remapping, shared buffers, and hardware support. However, the advantages are sometimes overestimated and new security issues arise. This paper describes different approaches to implementing zero copy and evaluates these methods for their performance and security considerations, to help when evaluating these techniques for use in e-government applications.

Keywords: Zero Copy, Network Security, Security/Performance Tradeoffs

1. INTRODUCTION

In addition to growth of corporate servers, there has been tremendous interest and growth in the support of e-government services. This includes remote access to public information, databases, forms, guidance, training materials as well as access to personal information such as treasury holdings, social security, income taxes, and government benefits. In addition, e-government also supports internal access to government documents, support of paperless offices and workflow improvement and communication support for critical services including first responders in emergencies. Cost-effective e-government solutions will require use of shared organizational servers, such as cloud servers, and high performing computers that can provide users with timely and secure access to information. Unfortunately, we have found that there is often a disconnect between marketed performance-improving solutions and security needs. This paper addresses one “performance-improving” technology with respect to security needs. We have found that some of the solutions that have been proposed and implemented will not work with security technologies, such as encryption. The understanding of the impact of performance-improving solutions on security is important when comparing vendors' performance benchmarks and claims when evaluating systems for purchase, and is important for developers to understand when making implementation decisions.

Therefore, the intent of this paper is to bring to light an example of security concerns that we believe should be addressed in the development process and in security requirements such as those specified by US Federal Acquisition Regulation (FAR) Part 39.101 and OMB Circular A-130 while supporting e-government efforts such as those specified in the “E-Government Act of 2002”. Similar regulations exist in other countries or in corporate policies. Corporations and governments in the market for servers should pay careful attention to performance benchmarks developed by vendors.

For example, the FAR Part 39.101 requires that agencies identify their requirements pursuant to: best management practices for energy-efficient management of servers and Federal data centers and shall include the appropriate information technology security policies and requirements. Energy-efficient management often translates into performance, since a data-center can use a lesser number higher performing servers to do the same work, and with less energy. As we discuss in this paper, we need to be aware of the tradeoff between some performance technologies and security technologies. We believe there is a need for the development of a set of benchmarks or at least checklists/configuration guides that enable the acquisition officers and developers to make the most appropriate choice of technologies given the security needs of the application environment.

1.1 Performance and Security

As demand for higher performance computers grew, CPU designs developed from single core to dual-core, quad-core or even hexa-core. However, a quad-core processor does not indicate that the performance of the overall system is four times the performance of a single core processor. The performance of the system is limited by many other factors, such as access to system memory. According to Brose [1], CPU processing power improves about 50% each year, while the memory only has an average 35% increment at the same amount of time[1]. Improvement in memory usage is increasingly a critical factor for system performance. This is especially important for network applications, such as severs, which transfer a lot of data, and cannot benefit from cache size increases.

Servers used by industry and government are critical not only for their performance, but also their security. To make the government more transparent, web servers for the public are used to inform the public about new policies and the latest news from the government. What's more, the servers used among governmental employees must be secure enough to avoid leaking information and to prevent hacking. Therefore, severs used by government must ensure their efficiency and security.

Traditionally, in server environments, such as servers used in e-government, when sending data to a network, data is loaded from disk and transferred to a network card buffer. The whole procedure consumes a lot of CPU cycles and memory bandwidth, for data must be copied between application memory space and kernel memory space. However, most of these data copies are redundant and can be minimized by implementing zero copy techniques. Zero copy is a name used to refer to a variety of techniques which help reduce useless memory accesses, usually involving elimination of data copying. By implementing zero copy, less data copies are needed when data is transferred between buffers. This helps reduce the number of unnecessary memory accesses and CPU consumption, and therefore enhance overall system performance.

1.2 Zero Copy for Performance

In recent years, there have been a number of studies regarding zero copy and different kinds of zero copy techniques have been applied. Zero copy can be classified into two groups: (1) reduce data copies between devices (disk or network card) and kernel buffers, and (2) reduce data copies between kernel buffers and application buffers.

Typically, data copies between devices and kernel buffers can be reduced by using DMA (Direct Memory Access). DMA allows special purpose hardware to read or write main memory without involving the CPU, which greatly reduces CPU consumption and also eliminates redundant data copies between device and kernel buffers.

Zero copy can also reduce the number of copies between user memory space and kernel memory space. Several different approaches have been proposed to solve this problem, and they are all called zero copy. For example, memory mapping is used to map the user buffer to the

[1] This means that relative to CPU speeds, memory performance is cut in half every 7 years.

kernel buffer; in LyraNET, a modified `sk_buff` structure can be used to eliminate data copies; and buffers can be shared, so stored data can be referenced via pointers. In addition, some system calls in Linux support zero copy by reducing data copies such as the `sendfile()` and `splice()` system calls. Instead of copying data, descriptors are appended to buffers in sendfile. The system call `splice()` sets up a data transfer path between different buffers in kernel memory.

However, in addition to performance benefits of zero copy techniques, it is important to understand the security implications of using those techniques. This is important for commercial servers as well as e-government systems.

Zero copy helps reduce the number of data copies and context switches which improves the CPU performance. But it also causes some new problems that we have to understand. This paper focuses on analyzing different zero copy techniques which are used to reduce the data copies in TCP/IP transmission, which are being implemented in many commercial servers which could end up being used by government offices to gain better performance.

The remainder of this paper evaluates the use of zero copy for performance in secure environments as is laid out as follows. The traditional data transfer approach and security issues are presented in Section 2 and zero copy techniques are described in Section 3. Problems caused by zero copy are examined in Section 4; this section also discusses security issues relating to zero copy techniques, which will help developers and acquisition offices in determining the appropriate use of this technology.

2. BACKGROUND

2.1 Traditional Data Transfer Method

Traditionally, if a user/server application wants to send data to a network, the data is read from disk to kernel memory, then stored in a buffer allocated by the user application in user memory, then sent to the kernel buffer associated with the network stack, and finally sent to the network interface card (NIC) for transfer over the network. Although the whole procedure needs only two system calls -- `read()` and `write()`, from the view of kernel, it is quite inefficient, even if DMA is supported by hardware. The traditional data transfer method is shown in Figure 1. We will examine this approach step by step to illustrate the CPU copies[2] and context switches.

First, if the data is not cached in a kernel buffer, the kernel needs to load the data from disk to a kernel buffer. For this step, DMA is typically used, which allows this first copy to consume little CPU resources. However, extra effort is needed to manage the DMA copy. Then, the data needs to be transferred into the buffer allocated by the application in user memory. This step is completed by the `read()` call, which, is typically done by a CPU copy in a kernel function such as `copyout()` or `copy_to_user()` [1]. Also, the `read()` system call needs a context switch from user mode to kernel mode. After copying data to the application buffer, the `read()` system call returns, which causes another context switch from kernel mode to user mode.

From the application buffer to the corresponding network stack in the kernel buffer the data needs to be transferred using a user-to-kernel copy function, such as `copyin()` or `copy_from_user()` [1]. This kernel buffer is not the same buffer used for the initial fetch from disk. It is a new buffer which is associated with the network stack. Data is packetized according to the Internet protocol in the network stack. For this step, the use of the `write()` system call generates a context switch from user mode to kernel mode. Finally, the prepared data are sent from the kernel buffer to the buffer on the network interface card, and then transferred over the

[2] We use the term CPU copy to refer to any time the CPU reads a memory cell and writes a result back out.

network. This fourth copy can be done using DMA. Although no CPU copy is needed, the returned `write()` system call forces another context switch from kernel back to user mode.

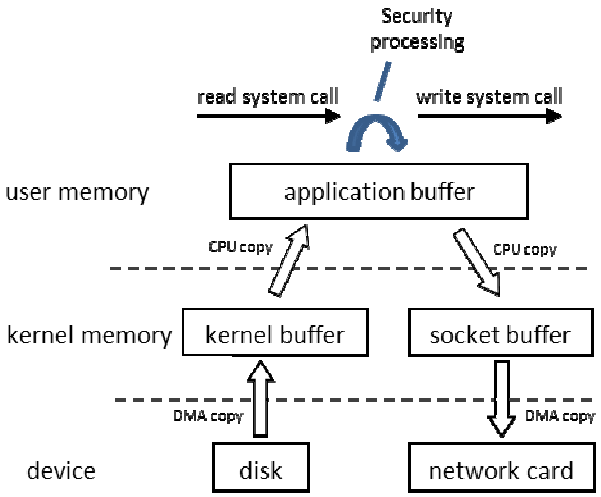

FIGURE 1: Traditional Data Transfer

2.2 Secure Communications

For secure communication across the network, applications will encrypt data packets. Depending upon the security mechanism used, encryption could occur at a high level or low level of the network stack. Servers and secure network applications often use SSL (Secure Socket Layer) for secure communication.

There are two aspects to SSL that can slow down system processing. The first is the negotiation of the encryption key and authentication. These processes typically use modular arithmetic over very large numbers and take a good amount of CPU time. There are hardware accelerators, such as SSL cards, that allow an application to offload this computation load. This set up of the encryption is beyond the scope of this paper. The second aspect is the actual encryption of the data. This requires applying an encryption algorithm to all of the data, reading the results from one buffer and writing them to another buffer (labeled *security processing* in **Error! Reference source not found.**). Even with CPU help, data copies are needed, for example in 2010 Intel introduced the AES instruction set extension to support fast encryption, and utilize 128-bit registers in the Intel architecture. These instructions require CPU access to the data and hence require a CPU copy.

In addition to encrypted communication, some security system requires filtering or guarding by the application. Typically this occurs in a secure communications environment where messages are being transmitted between different security domains. The concepts of zero copy still work, but the source is often another network device and not a disk. Filtering also occurs when data retrieved from a database is filtered before release. In these situations, the application needs to read and review the information before making the decision whether to allow the information to go out (labeled *security processing* in **Error! Reference source not found.**). We also have to make sure that no unauthorized processes get access to the sensitive information. As we examine different zero copy techniques, we will have to see if they provide any performance enhancement in encryption or filtering secure environments.

3. ZERO COPY TECHNIQUES

There are at least four data copies and four context switches during the data transmission in the traditional data transfer approach. According to Huang et al [2], in addition to the costs of data copies, software interrupts consume a great portion of CPU usage, for interrupts are generated

for each packet. As we see, some of the data copies are redundant and some of the context switches between user mode and kernel mode can be eliminated by avoiding calling the read and write system calls separately. As a result, zero copy techniques are implemented to reduce the unnecessary data copies and improve the CPU performance. Many different schemes are used to help implement zero copy techniques, such as memory remapping, shared buffers, different system calls, and hardware support.

3.1 Dynamic Remapping

Dynamically remapping memory can eliminate the data copies between the application buffer and kernel buffer. As **Error! Reference source not found.** shows, firstly, data loaded from disk is stored in a kernel buffer by DMA copy. Then the pages of the application buffer are mapped to the kernel buffer, so that the data copy between kernel buffers and application buffers are omitted.

According to Stancevic [3], to remap the memory, the Linux `mmap()` system call will be called instead of the `read()` system call. After data are copied to the kernel buffer by DMA copy, the system call `write()` causes the data to be copied from the kernel buffer to another socket buffer in kernel address space. This means that the original copies from kernel buffer to application buffer and following copies from application buffer to socket buffer are substituted by only one data copy directly from kernel buffer to socket buffer. Then data will be copied to the network card buffer by DMA. Dynamic remapping needs three CPU copies (two DMA copies and one CPU copy) and four context switches, since the system calls `mmap()` and `write()` are made during data transmission.

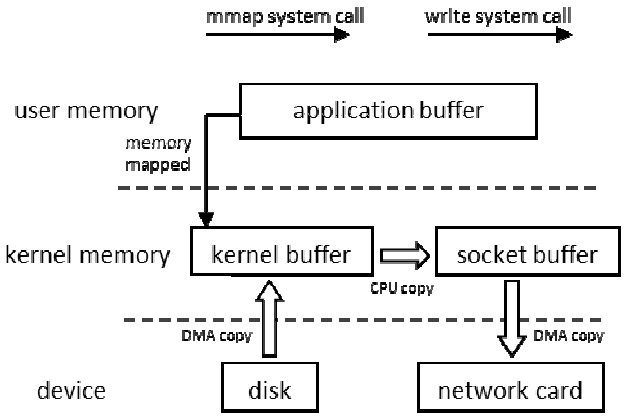

FIGURE 2: Data copy using memory remapping.

In an environment requiring encryption, unless the kernel supports an encrypt-while-copy operation, we must allow the user application to copy data from the kernel buffer to the socket buffer. In a filtering situation we must allow the user application to process the data. Therefore, unless we can link the application buffer to both the kernel buffer and the socket buffer, and have the security routine read from one to the other, we will get no savings using dynamic remapping. We also have to make sure that no other process can get access to the kernel buffer and bypass the security routine.

3.2 Shared Buffer in Kernel Memory Space

In INSTANCE-buffer [4], a memory area is shared in kernel space. As **Error! Reference source not found.** shows, there is a `buf` structure in the kernel buffer and `buf.b_data` is a pointer which points to the shared memory region which is in kernel memory space. Data fetched from disk are transferred to the shared memory area according to the `b_data` pointer. Furthermore, in the socket buffer, the m_data pointer in the `mbuf` structure points to the shared memory area.

Therefore, data are copied from the memory pointed by `mbuf.m_data` to the buffer on the network card. According to Halvorsen and Jorde et al [4], the INSTANCE-buffer scheme allocates the shared memory region and the structures (`buf` and `mbuf`) statically, which means the time cost for allocating memory and variables is reduced.

In a secure environment, this technique will cause problems. To encrypt, we have to read the data from the source buffer and write it to a destination buffer. Even if we can encrypt in place, there will still need to be a read and write of data. A simple filter process will work as long as we can be sure that data is not sent out to the network card without the filter approving the message, but we cannot use a complex filter that modifies the data. In high-assurance systems this may not provide enough separation and isolation control.

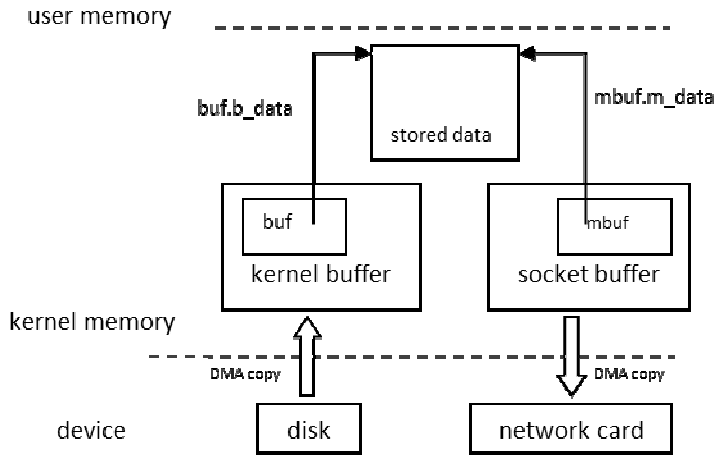

FIGURE 3: Shared buffer with structures.

3.3 Shared Buffer Between User and Kernel

In Linux, the `sk_buff` structure contains all the control information about individual network packets. Data sent to the network are copied to one or more `sk_buff` buffers. In LyraNET [5], the original `sk_buff` structure is modified to an enhanced copy elimination `sk_buff` structure, which eliminates the data copy by just passing the address of the data buffer. According to Chiang and Li [5], the modified `sk_buff` structure includes a new array with two elements, named `dataseg`, to record addresses for data without copying the data.

LyraNET is focused on reducing the data copies from user buffer to kernel buffer and from kernel buffer to network card. Using the enhanced copy elimination `sk_buff` structure, information about the protocol headers is copied to the network card and then data can be retrieved correctly according to the pointers and the data length stored in the modified `sk_buff`. LyraNET also uses DMA to copy data from the socket buffer to network card (Figure 4).

In a secure environment the security routine will have to copy data from the shared data region into a new region upon encryption. Filtering requires a CPU read of the data, but may not require a write. The system will have to ensure that the shared region is not available to the socket buffer until after it has been approved.

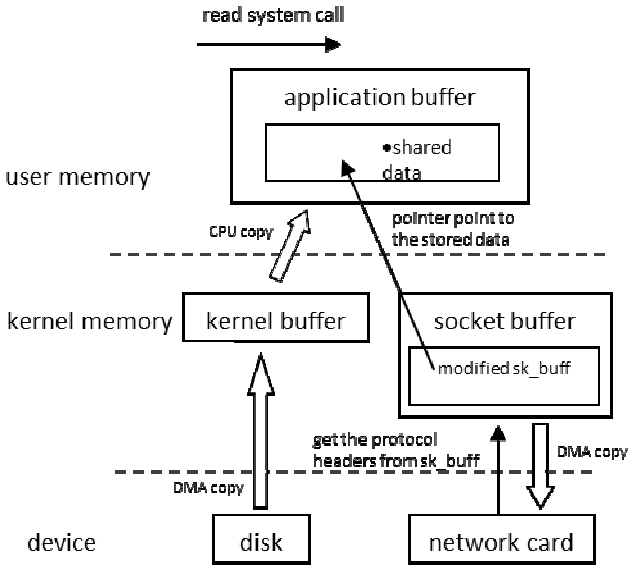

FIGURE 4: Shared user and kernel buffer

3.4 Sendfile

According to Stancevic [3], in Linux kernel version 2.1, the `sendfile()` system call was introduced to simplify the transmission of data over the network and between two local file descriptors. As Figure 5 shows, when calling the `sendfile()` system call, data are fetched from disk and copied into a kernel buffer by DMA copy. Then data are copied directly from the kernel buffer to the socket buffer. Once all data are copied into the socket buffer, the `sendfile()` system call will return to indicate the completion of data transfer from the kernel buffer to socket buffer. Then, data will be copied to the buffer on the network card and transferred to the network.

The `sendfile()` method is much more efficient than the combination of `read()` and `write()` system calls [6]. Compared with the traditional data transmission method, it replaces the `read()` and `write()` system calls, which reduces the number of context switch from four to two. Compared with dynamic remapping, `sendfile()` reduces the costs for virtual memory management. However, it still needs three data copies (two DMA copies and one CPU copy) to finish the data transmission.

This approach to zero copy bypasses the application completely, avoiding any application specific security routine or CPU processing of the data, making encryption and filtering impractical.

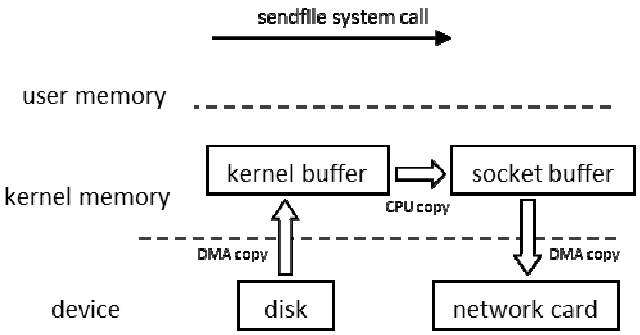

FIGURE 5: Data transfer using sendfile.

3.5 Sendfile With DMA Scatter/Gather Copy

The method that enhances the `sendfile()` system call with DMA scatter/gather copy can eliminate the CPU copy between kernel buffer and socket buffer found in the `sendfile()` method. Different from the DMA which maps each buffer one by one and then do the operation, DMA scatter/gather copy maps the whole list of buffers at once and transfer them in one DMA operation [7]. In this method, support from hardware is needed, for the network interface gathers data from various memory spaces (Figure 6). When calling `sendfile()`, data on disk are loaded into a kernel buffer using DMA transfer. Then, only the buffer descriptor is sent to the socket, instead of coping all of the data to the buffer. The descriptor contains information about the length of the data and where it is. Using this information, header and trailer of the data packet can be generated. Then by using the DMA scatter/gather operation, the network interface card can gather all the data from different memory locations and store the assembled packet in the network card buffer.

Hardware which supports DMA scatter/gather copy eliminates the CPU copy between kernel buffer and socket buffer. It means that no CPU copies occur during the data transmission from disk to network card, which helps to increase the performance of the CPU. Only one system call, `sendfile()`, is made in this approach, so there are only two context switches. Additionally, since hardware supports the DMA scatter/gather operation, data in memory are not required to be stored in consecutive memory spaces.

This approach to zero copy bypasses the application completely, avoiding any application specific security routine or CPU processing of the data, making encryption and filtering impractical.

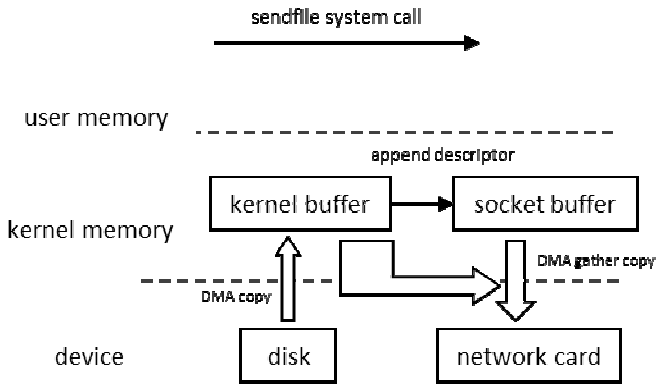

FIGURE 6: Sendfile and DMA scatter/gather copy.

3.6 Splice

According to the Linux man page [8], the Linux 2.6.17 kernel has a new system call, `splice()`. This call takes two file descriptors and an integer as parameters. It copies data, which size is specified by the integer, from one file descriptor to the other file descriptor using a pipe. By doing this, it does not need to copy data between kernel space and user space.

When using this approach, data are copied from disk to kernel buffer first. Then the `splice()` system call allows data to move between different buffers in kernel space without the copy to user space, for only file descriptors are transferred. In some zero copy implementations, `splice()` is called to move data from the kernel buffer to socket buffer which is also in kernel space. Then data are copied from the socket buffer to the network card buffer by DMA transfer. The system call splice eliminates the data copy between kernel buffer and socket buffer.

Unlike the method `sendfile()` with DMA scatter/gather copy, `splice()` does not need support from hardware. In addition, two context switches are bypassed compared to the

traditional method, therefore only two context switches are needed. Instead of coping data to an application buffer, `splice()` works as a pipe between two kernel buffers to allow direct data transfer. However, there must be two file descriptors opened to both the input and output devices (Figure 7).

Fall and Pasquale [9] indicated that, in the best case, the performance of using `splice()` is at 1.8 times than the maximum throughput of the traditional data transfer method. This approach to zero copy bypasses the application completely, avoiding any application specific security routine, making encryption and filtering much more difficult.

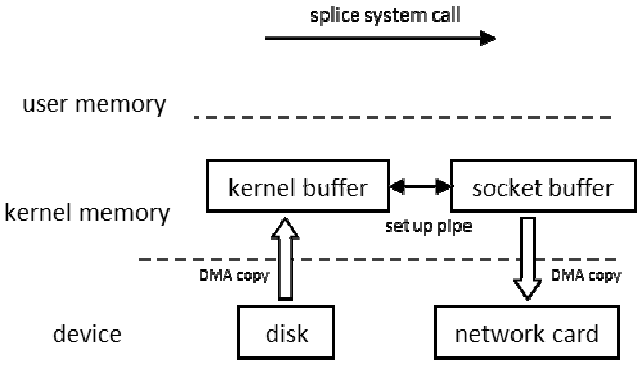

FIGURE 7: Data transmission using splice.

4. POTENTIAL PROBLEMS AND SECURITY CONCERNS

The implementation of zero copy technique helps to decrease overhead so as to release CPU cycles. Redundant data copies are reduced which saves more CPU and memory resources. However, the benefits of zero copy are sometimes overestimated, for the implementation of zero copy brings other new problems which may nullify the improvement of the system. For example, memory mapping needs costly virtual memory operations, the method of `sendfile()` with DMA scatter/gather copy needs support from hardware to gather data from various memory locations. In addition, zero copy techniques ignore or even introduce security concerns as well.

4.1 Dynamic Remapping Problem

Dynamic remapping uses `mmap()` instead of the `read()` system call, which reduce one copy during the data transmission. However, data might be modified by the user program while being prepared for transmission, this may cause security problems.

This problem can be prevented by implementing the Copy-on-write (COW) technique. COW is a simple technique. When the pages of the application buffers are mapped into kernel space, these pages can be written by the user. After the `mmap()` system call returns, both application and socket buffers can then only read the kernel buffer. The user can then start writing to the kernel buffer when the `write()` system calls return, which means the required data are sent to socket buffers. COW is implemented when using page remapping. It helps to ensure that while data is transferring, write permission is controlled. So the content of the physical memory page cannot be changed. However, COW is a costly event which consumes more CPU resource than a simple CPU copy [10].

4.2 Pre-allocated Buffer Problem

Some zero copy techniques use pre-allocated buffers to store data. By using pre-allocated buffer, the costly virtual memory operations are done only once. In addition, the pre-allocated buffer eliminates the exchange of the size of data between sender and receiver, but it causes another problem: the pre-allocated buffer space in receiver might be exhausted. If the receiver's pre-

allocated buffers are exhausted, the receiver has to wait until some buffers are usable. According to Yamagiwa et al [11] even if there are a lot of pre-allocated buffers, the receiver still might be exhausted since these buffers cannot be paged out.

4.3 Sendfile System Call Problem

According to Brose [1], the `sendfile()` interface requires two file descriptors. The first one is a readable memory mappable file such as regular file or a block device. The second one can be a writable file or a network socket. The `sendfile()` system call can only transmit data which has specified length. The other disadvantage of the sendfile approach is that it is hard to implement `sendfile()` on the receivers' end, for network transfers are asynchronous and the `sendfile()` system call is not a standard system call among different Linux or UNIX systems. In some systems, `sendfile()` is routed to the `transferto()` system call, which makes the implementation of this approach more difficult.

4.4 Sendfile With DMA Scatter/gather Copy Problem

The method that uses `sendfile()` with DMA scatter/gather copy enhances the performance of the CPU by avoiding CPU copies. Since data can be copied from various memory areas to the network card buffer by DMA scatter/gather copy, which is supported by hardware, cache coherency is not required anymore. However, according to Brose [1], the source buffer may be a part of the page cache, which means it is available for the generic read functionality and may be accessed in traditional way as well. As long as the memory area can be accessed by the CPU, the cache consistency has to be maintained by flushing the caches before DMA transfers.

5. CONCLUSION

5.1 Zero-Copy Impacts

Over the past decade we have seen tremendous growth in e-government services at the state and federal level. The OMB report “FY 2009 Report to Congress on the Implementation of The E-Government Act of 2002” highlights federal agencies' efforts to implement e-government. These activities have included performance enhancing and cost saving consolidation of services into server farms and clouds, and increased availability of private and secure information. There are other reports that highlight similar efforts by the states. These reports indicate that there is a growth in the use and availability of e-government.

Vendors developing solutions for data servers will use new technologies if they can increase performance, such as zero copy. Although the implementations of zero copy techniques help reducing the redundant data copies between the kernel memory space and user memory space, they also bring us new problems. Memory remapping approach needs virtual memory operations and COW which might nullify the improvement of the CPU performance. When using the pre-allocated buffer or static buffer, size of the buffer becomes a problem. Because the buffer overflow attack and the condition that the buffers are exhausted need to be considered.

	CPU copy	DMA copy	System call	Context switches	Advantages and disadvantages	App-level security processing
Traditional data transfer	2	2	*read* *write*	4	Redundant data copies consume many CPU cycles. 2 system calls cause 4 context switches. Large consumption of memory bandwidth.	Yes
Memory remapping	1	2	*mmap* *write*	4	Needs costly virtual memory operations. Needs COW to be implemented which costs a lot.	Yes

Shared buffer (INSTANCE -buffer)	0	2		0	Buffers need to be released after using. Pre-allocated buffer reduces the exchange of the data size, since the buffer size is fixed. Pre-allocated buffer needs less virtual memory operations, for the virtual memory operations only used when creating the buffer.	Yes
LyraNET	1	2	*read*	2	Need costly virtual memory operations. LyraNET still needs data copy from kernel buffer to application buffer.	No
Sendfile	1	2	*sendfile*	2	Reduces the costs for virtual memory management. *Sendfile* hard to implement due to asynchronous network transfers. *Sendfile* system call is not a standard system call among different Linux or UNIX systems.	Yes
Sendfile with DMA scatter/gather copy	0	2	*sendfile*	2	Reduces the costs for virtual memory management. Needs hardware which supports DMA scatter/gather copy. *Sendfile* hard to implement due to asynchronous network transfers. *Sendfile* system call is not a standard system call among different Linux or UNIX systems.	No
Splice	0	2	*splice*	2	When using *splice* system call, there must has two file descriptors which opened to both the input and the output devices.	No

TABLE 1: Comparison of different zero copy techniques.

Therefore, the implementation of zero copy techniques regard to network is limited by many factors. That is why zero copy techniques have not been widely adopted in the operating systems. Although these new techniques need more evaluation, zero copy is worth analyzing and implementing carefully for critical e-government services. Removing redundant CPU copies can improve the performance of servers, and this is especially useful for commercial and government servers which require high efficiency. The advantages and disadvantages of different zero copy techniques are compared in
TABLE 1. Any zero-copy technique that bypasses the application will limit the ability to perform security processing. Therefore the `sendfile()` and `splice()` system calls are not recommended for use in applications that require security processing. The right most column indicates whether security processing can be used with each zero-copy technique, providing some performance improvement.

5.2 Understanding the Implications

To better understand the implications of security on performance improving solutions, we recommend that the acquisition officer evaluate the proposed solution using questions similar to the following list:

1) Will the server need to provide encrypted communication?
   a) Can the encryption be performed at the connection-level by dedicated hardware, or will you need application-level processing?

   b) Will the server need to provide multiple encryption services, supporting different algorithms and protocols?
2) Will the service need to provide application-level security processing/filtering on the data being serviced?
   a) Will that processing be service specific (same processing for all users) or user-specific?
   b) Will the service require multiple filtering processes?
3) Does the server need to support multiple concurrent services (such as cloud servers)?
4) Does the proposed solution support the level of encryption and application processing required?

The answers to these questions will help acquisition officers and developers better understand the expected use cases of the system and better evaluate the use of performance-improving technologies such as zero-copy. Any system that requires the examination or processing of the contents of the message, such as encryption, or filtering will require either a hardware solution, or application-level processing.

A hardware solution, such as a cryptographic hardware module, allows the use of all of the zero-copy techniques by passing the data packet directly to the cryptographic hardware. However, if the system supports multiple encryption services, a hardware solution may become impractical, thus forcing the system to rely on an application-level software solution.

Any software-based solution that requires access to the full data packet, cannot utilize the zero-copy features with a "no" in the final column of TABLE 1, Lyrannet, Sendfile with DMA scatter/gather copy and Splice. Other techniques will allow application level security processing.

We believe that it would be beneficial to develop a set of benchmark scenarios that address encryption and filtering technologies at the connection and application level and the recommend that the vendors provide performance data with respect to these benchmarks. This can be done to help address a wide-variety of proposed performance-improving technologies, which may not work well for security-enabled environments, contrary to vendor's claims.

6. REFERENCES

[1] E. Brose. "ZeroCopy: Techniques, Benefits and Pitfalls," http://kbs.cs.tu-berlin.de/teaching/ws2005/htos/papers/streams-zcpy.pdf, 2005, [last accessed May 30, 2012]

[2] C. Huang, C. Chen, S. Yu, S. Hsu, and C. Lin. "Accelerate in-line packet processing using fast queue," in Proc. 2010 IEEE Region 10 Conference, 2010, pp. 1048–1052.

[3] D. Stancevic. "Zero Copy I: User-Mode Perspective," *Linux Journal,* vol. 2003 no 105, Jan. 2003, pp. 3.

[4] P. Halvorsen, E. Jorde, K. Skevik, V. Goebel, T. Plagemann, "Performance Tradeoffs for Static Allocation of Zero-Copy Buffers," in Proc. Euromicro Conference, 2002, pp. 138-143.

[5] M. Chiang and Y. Li, "LyraNET: A Zero-Copy TCP/IP Protocol Stack for Embedded Operating Systems," in Proc. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Aug. 2005, pp. 123-128.

[6] Linux Programmer's Manual, http://www.unix.com/man-page/Linux/2/sendfile/ Feb. 2010.

[7] J.Corbet, A.Rubini, and G.Kroah-Hartman. "Memory Mapping and DMA" in *Linux Device Drivers*, third edition, O'REILLY, Feb, 2005, pp.450.

[8] Linux Programmer's Manual, http://www.unix.com/man-page/Linux/2/splice/ Sep. 2009.

[9] K. Fall and J. Pasquale, “Exploiting In-Kernel Data Paths to Improve I/O Throughput and CPU Availability,” in Proc. Winter 1993 USENIX Conference, 1993, pp. 327-333.

[10] H.K.J. Chu. “Zero-copy TCP in Solaris,” Proc. USENIX Annual Technical Conference, 1996, pp. 253–264.

[11] S. Yamagiwa, K. Aoki, and K.Wada. “Active zero-copy: A performance study of non-deterministic messaging,” in Proc. International Symposium on Parallel and Distributed Computing, 2005, pp. 325–332.