Multi-stage Packet Filtering in Network Smart Cards

HongQian Karen Lu

Smart Cards Research, Axalto, Inc., 8311 North FM 620 Road, Austin, TX 78726, USA
karenlu@axalto.com

Abstract. *Network smart cards* are smart cards with networking capabilities. They have opened new opportunities for the use of smart cards in Internet applications. At the same time, network smart cards are exposed to network security threats just as other computers on the Internet. Unfortunately, existing designs of network security mechanisms, such as packet filtering, may not be best suited for smart cards because the computing resources of the cards are too limited. This paper presents a new packet filtering approach that overcomes this difficulty. The packet filtering is performed in multiple stages. It drops unwanted packets as early as possible, starting at the I/O interrupt level. This builds a network firewall inside smart cards and reduces resource usage for packet processing. It can be used with different hardware and software configurations and with various filter rules. Advantages of this approach include better security, reduced memory usage, and enhanced performance.

1 Introduction

An exciting new phenomenon in the smart card industry is the emergence of *network smart cards*, which are smart cards with networking capabilities [1]. The network smart cards can provide services and access resources on the Internet, opening new opportunities for the smart card industry. On one hand, because of their security, portability and tamper-resistance, network smart cards provide security and convenience over the Internet, which is better than other secure tokens [2]. However, on the other hand, network smart cards are exposed to network security threats just as other computers on the Internet. Therefore, they require security protections as well. Unfortunately, existing designs and implementations of network security mechanisms, such as packet filtering, may not be best suited to network smart cards because of the cards' computing resource limitations.

Packet filtering is a key component of the network firewall technique. In the Internet world, a firewall is a network security mechanism. It is typically used to prevent unauthorized Internet users from accessing private networks connected to the Internet. Firewalls can be implemented in hardware, software, or a combination of both. Packet filtering is typically done at protocol layers. However, allocating memory for a packet, processing the packet through layers, and then filtering out the packet waste CPU time and memory resources.

Smart cards have very limited memory resources compared to other network devices or computers. For example, a network smart card may have only 6K bytes of RAM, which seems to be plenty for a smart card. However, this memory is very little

J. Domingo-Ferrer, J. Posegga, and D. Schreckling (Eds.): CARDIS 2006, LNCS 3928, pp. 192 – 205, 2006.

for a network device because it must deal with a large amount of data in real-time. The resource need is even higher when communicating over a secure channel. In addition, once connected to the network, the network smart card may face a large number of unwanted messages. If not managed properly, the card's memory buffers may be used up very quickly. Furthermore, network smart cards must protect themselves from network attacks. Therefore, new methods of packet filtering that are practical and efficient for network smart cards must be developed. This paper presents one such method called the *multi-stage packet filtering*. It is a software method that is adaptable to hardware configurations.

The new packet filtering method has two goals: security and resource management. The goals are approached by performing packet filtering as early as possible before more resources are consumed. The filtering has multiple stages starting from the I/O interrupt service routine. The amount of filtering at each stage is configured according to multiple factors, including filtering rules, the hardware configuration, hardware capability, the nature of the data link layer, memory buffering scheme, and the network stack process model.

The multi-stage packet filtering drops unwanted packets early to build a network firewall inside the network smart card, to save memory resources, and to reduce CPU usage for packet processing. It is a general framework of efficient packet filtering, which can be used with a variety of hardware and software configurations and with various filter rules. The method has several advantages over existing packet filtering designs, including better security, reduced memory usage, and enhanced performance. The approach is applicable to a variety of small resource-constrained embedded network devices.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 provides an overview of the multi-stage packet filtering method. Sections 4 through 6 present the details of the packet filtering at each of the multiple stages. The implementation is discussed in Section 7. Section 8 concludes the paper.

2 Related Work

A great deal of literature is available on network firewalls and packet filtering [3][4][5] . Many commercial products are also available. The packet filtering is typically done at Ethernet, IP, and TCP/UDP layers, that is, at the protocol processing stage. Extensive research on packet filtering in the past twenty years has produced excellent results and made many applications possible, such as network monitoring, traffic collection, performance measurement, packet classification in routers, firewall filtering and intrusion detection. The references [6][7] offer examples of the packet filtering research, which focus on flexible, extensible, and generalized filter abstractions and show how to compile the high-level abstractions to efficient implementations. These research were mostly based on modern operating systems and computing systems, such as workstations (in the past) and personal computers (at present). The packet filter is normally one module of the operating system, which executes at the protocol processing stage or is parallel to the protocol-processing module [6].

In contrast, the work described herein focuses on the design and implementation of packet filtering for small resource-constrained embedded network devices, such as

network smart cards. The main purposes of the filtering here are security and efficient resource management. The packet filtering executes at more than one stage in order to drop unwanted packets as early as possible and to best use limited computing resources.

In a previous paper, we have proposed a front-end packet filtering at the AHDLC layer for resource-constrained network devices [8]. This AHDLC packet filtering method is applicable for devices using PPP as network connections. The multi-stage packet filtering presented in this paper is a further elaboration of the front-end packet filtering concept. This elaboration includes three main aspects: (1) the front is pushed further to the I/O interrupt service routine; (2) the filtering is partitioned into stages; and (3) the main focus is moved to USB/Ethernet smart cards. The following sections describe the new method in details.

3 Method Overview

The multi-stage packet filtering method has two main purposes: network security and resource management. Both are extremely important and necessary for network smart cards. The key concept of the method is the front-end filtering, that is, to perform packet filtering as early as possible before more resources, such as memory and CPU time, are consumed. The filtering is done at multiple stages starting from the hardware I/O interrupt service routine. This front-end filtering also makes the device more secure because it blocks malicious packets upfront.

The amount and type of filtering at each of the multi-stages depend on multiple factors, including filtering rules, the hardware configuration (e.g. USB, UART), data link layers (e.g. CDC, EEM, Ethernet), hardware I/O interrupt mechanisms (e.g. byte, frame, DMA), memory buffering schemes (e.g. straight buffer, chained buffer), hardware capability, and the network stack process model. The implementation of this method also depends on these factors.

3.1 Network Smart Cards

A network smart card is a smart card that is also an Internet node. The network smart card implements standard Internet protocols and security protocols inside the card. Figure 1 illustrates a network smart card that connects to the Internet through a host computer. The smart card can provide services or access resources over the Internet. The protocol stack on the network smart card is the same as those on other Internet nodes.

A traditional smart card communicates with a host using the smart card standard ISO 7816. The host has smart card specific middleware installed in order to communicate with the card. Through the host, the smart card provides security services to the host or over the network that the host is connected to. In contrast, the network smart card communicates with the host computer or a remote computer using Internet protocols. The host does not have to be trusted [2][9]. No middleware is required on the host computer or on the remote computer in order to talk with the network smart card.

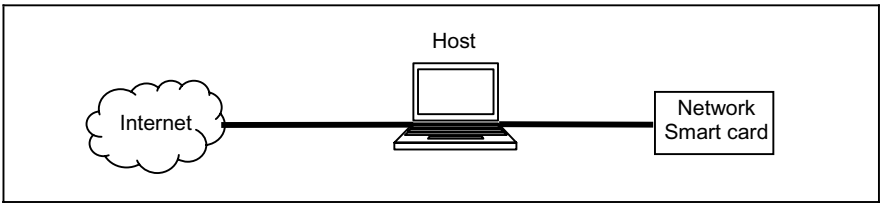

Fig. 1. A network smart card connects to the network via a host computer

Moving from proprietary computing environment to the mainstream networking environment opens many opportunities for smart cards. For example, network smart cards can establish direct secure connections with remote servers over the Internet [9]. This capability enables the cards to secure online transactions for Internet applications, such as online banking, online shopping, e-business, e-government, and e-health care.

Network smart cards present numerous engineering challenges mainly due to their computing resource limitations. For example, our first network smart card had only 6K bytes of RAM. The bandwidth was limited by the ISO 7816-3 interface and a bridging protocol [10]. The card resource is scarce considering that it must deal with real-time network traffic. The computation and communication demands for the card are even higher for secure communications because of cryptographic computations and increased network traffic. Many efforts have been made to provide network smart card functionalities in such resource-constrained environment [1][8][10]. This paper presents a continued effort to provide security and to manage limited resources for the card.

Network smart cards connect to the Internet through a host computer using USB or standard smart card interface ISO 7816. Smart cards with full speed USB interface use standard USB networking interface [11]. For smart cards that have only ISO 7816 interface, a bridging protocol, called Peer I/O, is required, which sits above the ISO 7816 layer and below the network protocols layer [10]. A device driver or a reader implements the Peer I/O on the host side to provide a full-duplex serial interface for the ISO 7816 device. Even this case does not require additional middleware because host computers know how to network through a serial interface.

The multi-stage packet filtering approach does not require a particular hardware interface. It is applicable to a variety of hardware and software configurations. For the convenience of discussion, we use USB smart cards as an example. In Section 7 we discuss the case of network smart cards with standard ISO 7816 interface.

Figure 2 illustrates an example of the protocol stacks for a USB network smart card and a host computer. The hardware connection between the smart card and the host computer is the USB. On top of the USB driver is a USB EEM (Ethernet Emulation Model) driver, which carries Ethernet frames using USB packages [11].

The network layer is the Internet Protocol (IP). Ethernet frames carry IP datagrams. The transport layer is TCP or UDP. IP datagrams carry TCP or UDP segments. Figure 3 illustrates the protocol encapsulations.

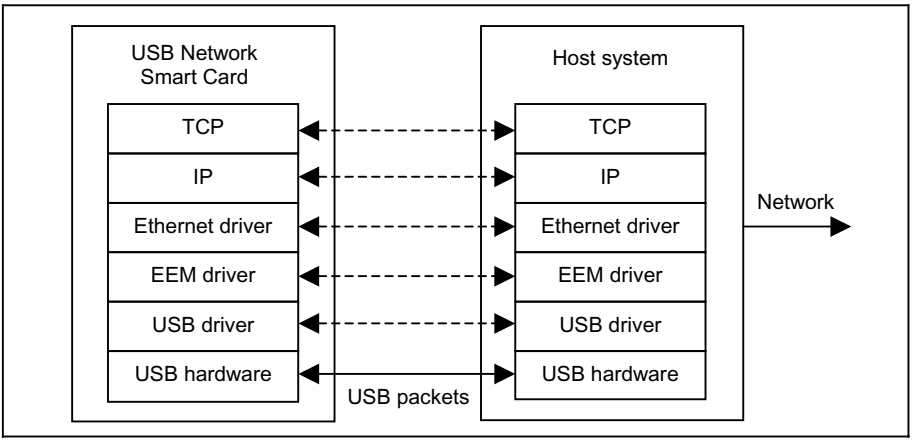

Fig. 2. Network stacks in a USB network smart card and the host computer[1]

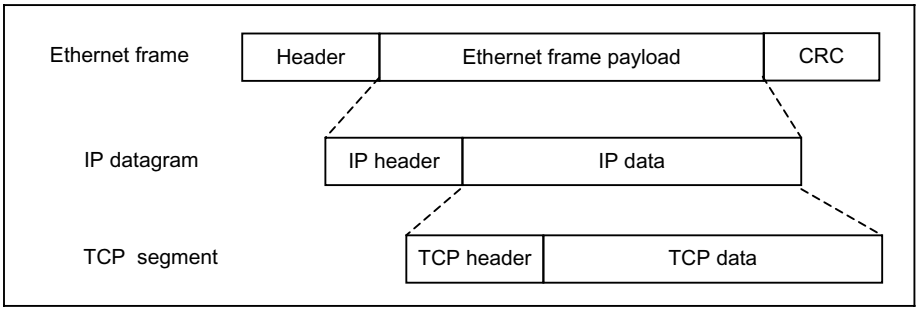

Fig. 3. Protocol Encapsulations

The TCP/IP network is a packet-switched network. Messages are divided into *packets* before they are transmitted. Each packet contains a source address and a destination address. Packets can follow different routes to their destinations. Once all packets forming a message arrive at the destination, they are recompiled into the message. In short, the TCP/IP network transmits messages via packets. Packet filtering filters packets to decide whether or not to let the packets pass, or to classify the packets. Packet filtering can be performed on in-bound packets as well as out-bound packets. This paper focuses on filtering of in-bound packets for security purposes.

3.2 Filter Rules

Packet filtering has been studied and used for over twenty years for network monitoring, firewall, and other purposes. Filter rules specify how packet filtering should be

[1] The full speed USB interface and network stack for smart cards are being proposed as an ETSI smart card standard.

performed. For security purposes, the basic idea is to block all packets, except those that the filter rules allow to pass [4].

In general, filter rules specify packet pass or drop conditions based on information in protocol headers. Packet filters look at protocol headers of a packet and check against filter rules to decide whether or not to let the packet pass. A network stack normally does not look into the payload (or user data) of a packet.

Some of the filter rules are static; others are dynamic. For example, the network device's MAC address is normally fixed, and thus the filter rule associated with the address is a static rule. In contrast, the permissible target IP address list is dynamic and, hence, its corresponding filter rules are also dynamic.

Some of the filter rules are stateless while others are stateful. For example, the TCP layer maintains a state machine for each connection. Rules for checking addresses or protocol types are stateless, because they do not require any state information. Rules that depend on the state of a connection are stateful rules. Figure 4 illustrates a classification of filter rules. The stateful rules may also be classified into static and dynamic rules. For the purpose of the multi-state filtering, this further classification is unnecessary.

There are different ways to model the packet filtering, including a Boolean expression tree and a directed acyclic control flow graph (CFG) [6][7]. The two models are computationally equivalent. Figure 5 illustrates a filter example with these two representations. Research has shown that CFG leads to more efficient implementations [7].

Packet filter rules are hierarchical, as shown in their representations. Once one filter rule decides to drop the packet, the remaining rules need not be checked; the packet is dropped. After a packet passes one filter rule, it still needs to pass other rules down the hierarchy in order to get to its destination.

The multi-stage packet filtering method is a general packet filtering framework. It does not depend on a particular filter rule set. It can be used for various filter rule configurations. The filter rules specification is out of the scope of this paper.

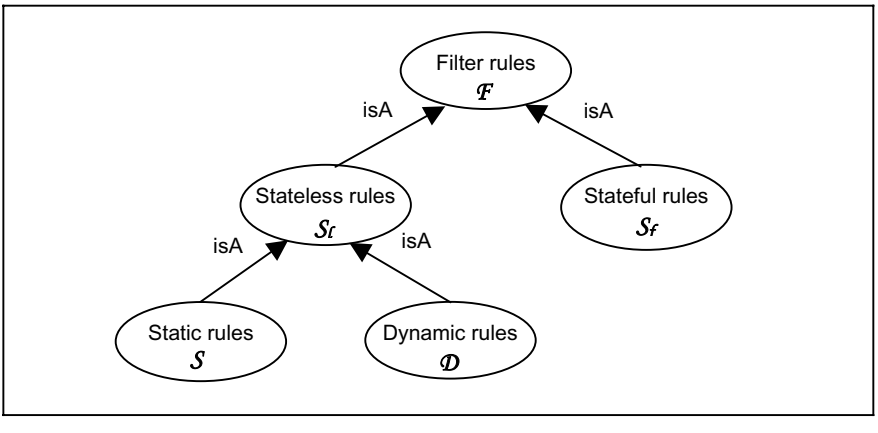

Fig. 4. A classification of filter rules

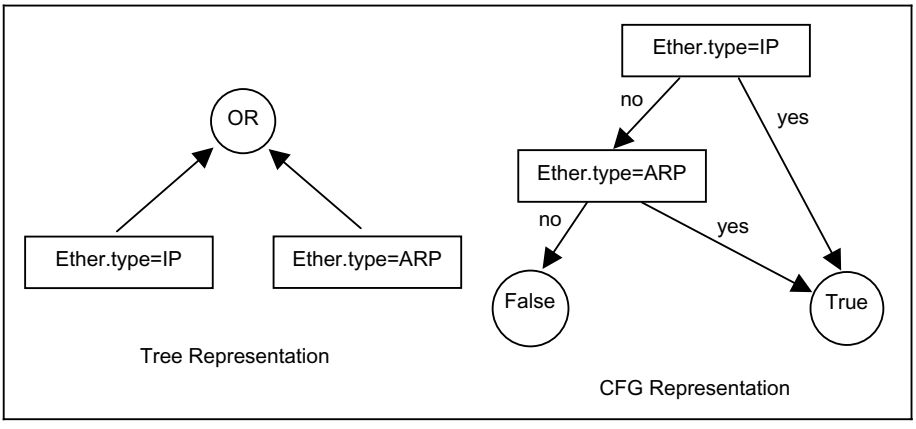

Fig. 5. Filter Function Representations

3.3 Software Models

This section examines several stages of an embedded system in which the multi-stage packet filtering may be performed. When a packet comes into a network smart card, the I/O hardware of the chip generates an interrupt. The corresponding interrupt service

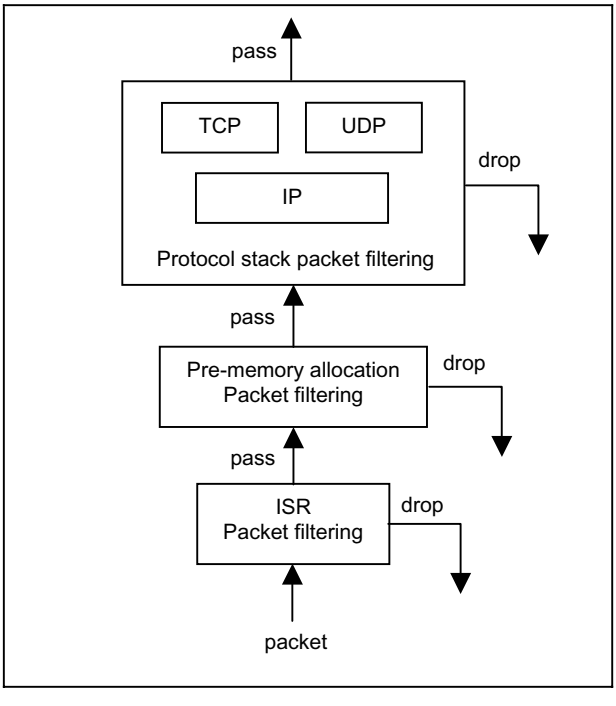

Fig. 6. Multi-stage packet filtering

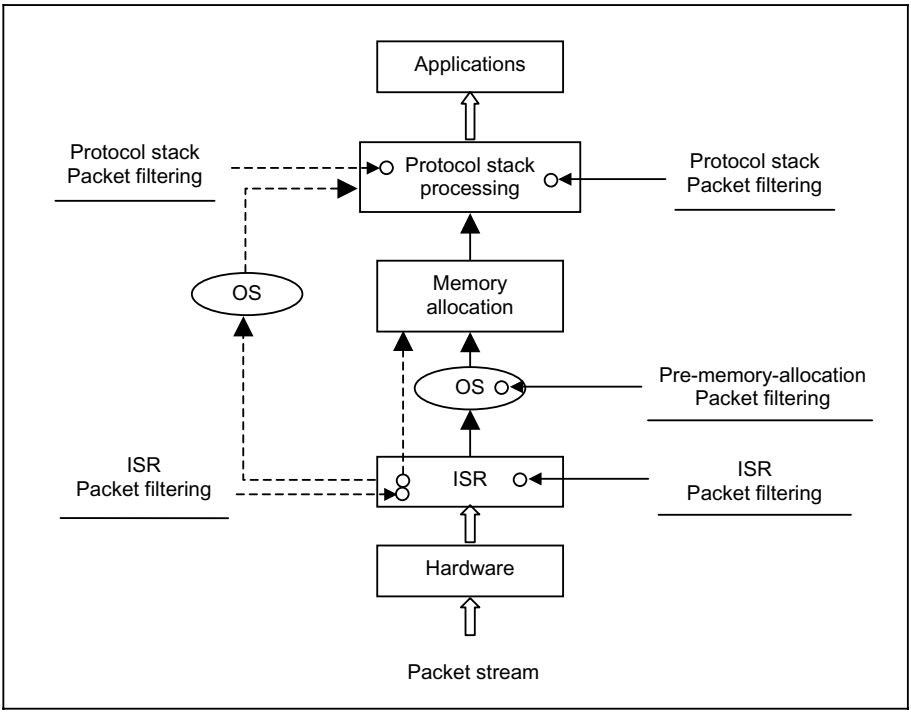

Fig. 7. Stages of the packet filtering depend on the software configuration

routine (ISR) handles the interrupt to get the incoming data. This is the first stage of packet handling. We may start filtering inside the ISR. This is called the *ISR packet filtering*. Then, a memory buffer, e.g. a byte array or a buffer chain, is allocated to store the packet for processing. This is a second stage. The filtering may be done just before the memory allocation, which is called the *pre-memory allocation packet filtering*. The protocol stack processes the packet, making a third stage. We call the filtering at this stage the *protocol stack packet filtering*. Depending on the interrupt handling, memory buffer scheme, and protocol stack, these three stages may not be completely separated. Figure 6 illustrates these three general stages. It should be noted that the protocol stack packet filtering might be further distributed among protocol layers.

Figure 7 illustrates two examples of the multi-stage packet filtering. The solid arrow path shows one example of a software configuration in which the memory allocation for the in-coming packet is outside of the ISR. This example uses a three-stage packet filtering. The dashed arrow path shows another example of a software configuration, with which the memory allocation is inside the ISR. This example uses a two-stage packet filtering method in which the ISR filtering is executed before the memory allocation.

Due to time limitation and other constraints of an interrupt service routine, a limited filtering is done at the interrupt service stage. If the memory allocation is outside of the interrupt service routine, much of the stateless filtering is performed before the memory buffer allocation so that unwanted packets do not use additional memory buffers. The protocol stack filtering applies remaining filter rules, especially stateful

rules, to the packets. Other software models may have additional stages, which may perform packet filtering. The key of this method is to drop unwanted packets as early as possible. This blocks malicious packets up front, avoids allocating memory buffers for these packets, and avoids or reduces processing time for the packets. The next few sections describe each of the filtering stages in more detail.

4 ISR Packet Filtering

This section describes the packet filter in the input event interrupt service routine (ISR). This is also called the front-end packet filtering. We first discuss the constraints of the ISR. We then describe what an ISR packet filter must do to live with these constraints.

4.1 Constraints

An I/O interrupt service routine is a software routine that handles I/O events. How an I/O event triggers an interrupt and how the microprocessor invokes the interrupt service routine depend on the chip architecture, the I/O hardware, and the software/hardware interface that the chip manufacture provides. Some chips let a software programmer write hardware interrupt service routines. Other chips provide a hardware/software interface layer to deal with hardware interrupts in which case a software programmer writes interrupt service routines triggered by the interface layer. The interrupt service routine may be called when a byte arrives, when a packet arrives, or when a larger amount of data arrives. For example, with USB devices, the interrupt service routine is typically invoked when a USB packet arrives. With full speed USB bulk data transfer, this may mean that 64 bytes of data have just arrived.

An interrupt service routine normally does some quick and simple things to handle the interrupt. The program goes back to the routine that was interrupted as soon as possible. Typically, the ISR has timing constraints. For example, the ISR must finish before the next input event happens. For USB full speed bulk data transfer on an otherwise idle bus, the maximum possible speed per pipe is nineteen 64-byte transactions per frame, where one frame is 1 millisecond. This takes about 82% of the bus bandwidth. Hence, the minimum time interval between the arrivals of two consecutive USB data packets is 43 microseconds. The ISR must finish within this time.

Another constraint for an ISR is the availability of other resources. For example, the input interrupt may happen when the CPU is doing a non-volatile memory write. In this case, typically the ISR cannot do a non-volatile memory write. In general, the ISR should avoid any non-volatile memory write.

A third constraint for an ISR is variable access. An I/O interrupt may happen when the program is changing a variable. If the ISR tries to access this variable or, worse, to change the variable, the result is unpredictable. This is known as the data-sharing problem. Therefore, either the ISR should try to avoid accessing or changing a variable or the variable must be protected, for example, using critical sections. To avoid the data-sharing problem and to reduce the interrupt latency, the ISR packet filtering must not access any variables, which means only checking static filter rules.

4.2 Packet Filtering

The input event ISR extracts Ethernet frames (called packets) from the underlying link layer protocols. Performing packet filtering inside the ISR is feasible because protocol headers, such as headers of EEM, Ethernet, and IP, are at fixed positions within their outer protocol packets. For example, an EEM packet has a two-byte header; the Ethernet packet header has fourteen bytes; and the IP header starts immediately after the Ethernet header. With such fixed positions, the ISR can access header elements directly.

The basic packet filtering rules are very simple and involve only constants. The following is an example of a set of basic filter rules. These are static rules to be applied first and can be done in the ISR.

Rule 1:	If (Ethernet destination address == my Ethernet address) Pass the packet.
Rule 2:	If the packet passed Rule 1, and if Type == IP Pass the packet.
Rule 3:	If (Ethernet destination address == ff:ff:ff:ff:ff:ff) and (Type == ARP) Pass the packet.
Rule 4:	If the packet passed Rule 3, and if Target IP address == my IP address Pass the packet.
Rule 5:	If the packet passed Rule 2, and if Destination IP address == my IP address Pass the packet.
Rule 6:	If the packet passed Rule 5, and if (Protocol type == TCP) or (Protocol type == UDP) or (Protocol type == ICMP) Pass the packet.

The amount of packet filtering in an ISR depends on the CPU speed, the timing constraints for the ISR, and the amount of necessary work that the ISR must do. For the example mentioned earlier, the ISR has as little as 43 microseconds to do its job. One of our implementations can check the above filter rules in 1.67 microseconds in the worst-case scenario. That is a sufficiently short execution time to fit into the ISR. Section 7 provides more details about our implementations.

During software development, one could measure the time needed for the normal ISR work without the packet filtering. The difference between the allowable time for the ISR and the time needed for the ISR function is the time interval that the packet filter can use. Some chips may only have time for checking one filtering rule for an Ethernet packet header; while other chips may have enough time for checking all static filter rules inside the ISR.

The packet filtering at ISR is especially useful if the input event ISR allocates memory buffers for incoming packets. The filtering should be done before the memory allocation. Regardless whether the allocated memory is a single contiguous memory or a chained memory buffer, once the ISR decides to drop the packet according to

filtering rules, it will require no memory allocation and no further processing to this packet. This leads to a reduced memory usage and enhanced performance. For zero-copy protocol stack implementations, being able to drop packets at ISR still prevents further processing of the unwanted packets. This again enhances the performance of the system.

In addition to reduced memory usage and enhanced performance, the unwanted packet does not go further into the system. This makes the system less susceptible to network attacks and, hence, results in a more secure system.

5 Pre-Memory-Allocation Packet Filtering

For some hardware and software configurations, the interrupt service routine or the Direct Memory Access (DMA) puts the incoming packets into a fixed contiguous memory location. Outside of the ISR, the network protocol stack processes and queues the packet. Before or during this process, the packet is taken out from the fixed memory location and put into a dynamically allocated memory buffer or a buffer chain. The contiguous memory is ready for the ISR or DMA to put in the next packet. This provides another opportunity for early packet filtering, which filters the packets before the memory allocation.

This pre-memory-allocation packet filtering, if performed outside the ISR, can check against all the remaining stateless filter rules, including static rules that were not checked by the ISR packet filter and dynamic rules. Once one rule decides to drop the packet, the remaining rules need not be checked; the packet is dropped. The packet filtering at this stage, again, prevents allocation of memory buffer for unwanted packets.

One example of dynamic filter rules that can be performed at the pre-memory-allocation stage checks the destination port number of an incoming packet. Each TCP or UDP packet contains a destination port number P_d. For example, an http server has a well-known port number 80; a secure http server has a well-known port number 443. The network smart card maintains a permissible destination port number list, L_d, which contains port numbers that the card allows the incoming packets to target at a given time. Then, we have the following rule:

Permissible destination port number rule:

$$\text{If } P_d \in L_d, \text{ then pass the packet.}$$

If an incoming packet's destination port number P_d is not in the list L_d, the packet is dropped. This list is static if the network smart card is a network server only. The list is dynamic if the card can be a client as well as a server.

For example, a network smart card provides a secure web server. The permissible destination port number list L_d initially has only one entry 443. The card is also an Internet client or an agent. When the card initiates a connection to a remote server using an ephemeral port number x, then x is added to L_d. When this connection finishes, the x is removed from L_d. Therefore, the list L_d changes; the associate filter rule is dynamic.

If a smart card chip has DMA (Direct Memory Access), the incoming data stream is transferred directly to a pre-specified contiguous memory location without passing

through the CPU. The packet filtering may be performed from the DMA memory directly to decide whether or not to drop a packet. Note that the packet filtering in this case may or may not be inside the ISR. If the filtering is inside the ISR, it should leave the check of the dynamic filtering rules to the next filtering stage.

6 Protocol Stack Packet Filtering

The protocol stack includes a data link layer (e.g. Ethernet), a network layer (IP), and a transport layer (e.g. TCP, UDP). Conventional packet filters work on the protocol stack or side-by-side to the protocol stack [6]. With the multi-stage method, the packet filtering at the protocol processing stage is reduced because of the filtering already done at previous stages. The filtering at this stage checks remaining filter rules. The stateful filtering is always done here because it requires state information. The amount of filtering at this stage depends on how much has been performed in previous stages. The following are three examples.

1. The protocol stack packet filter does the entire packet filtering work. (There has been neither ISR nor pre-memory-allocation packet filtering.) This is the conventional packet filtering.
2. The protocol stack packet filter does a part of the stateless static filtering, stateless dynamic filtering, and stateful filtering. (There is an ISR packet filter, but no pre-memory-allocation packet filter.)
3. The protocol stack packet filter does stateful filtering only. (There is an ISR and a pre-memory-allocation packet filtering.)

7 Implementations

Several smart card companies, such as Axalto, Giesecke & Devrient, and Gemplus, have demonstrated network smart cards, which have been called Internet smart cards or web cards, at various conferences in the past few years. We have implemented a network smart card on a smart card chip from Samsung, which was demonstrated at Cartes in 2003 and other smart card conferences.

We are currently using a faster USB smart card chip that has a 33 MHz microprocessor, 16K of RAM, 128K of ROM and 64K of EEPROM. The network smart card uses the new USB networking standard, EEM, for the lower link layer to carry Ethernet frames [11]. For the multi-stage packet filtering, from the software implementation perspective, the most critical part is the filtering in the interrupt service routine. For USB bulk data transfer with full speed USB, the ISR has a little less than 43 microseconds. Our implementation of an ISR packet filter using the sample rules, listed in Section 0, executes in 55 machine cycles in worst-case scenarios. The ISR packet filter was programmed using the C language. With the chip's 33 MHz microprocessor, this takes 1.67 microseconds. Even assuming a 20 MHz practical processor speed, the ISR filtering, in the worst- case scenario, takes 2.75 microseconds. It could take less time if coded in an assembly language. Therefore, the proposed packet filtering approach is practical and effective.

In a previous work, we have proposed packet filtering at the AHDLC layer [8]. This is especially useful for smart cards that have ISO 7816 interface and do not have USB. For example, our first network smart card prototype used Samsung's S3FC9BJ smart card chip, which had only ISO 7816 interface. In this case, the network smart card uses PPP [12], instead of Ethernet, as the data link layer to carry TCP/IP Internet protocol packets. The AHDLC layer does framing for the PPP layer [13]. The multi-stage packet filtering method is also applicable in this situation. If the AHDLC processing is performed during the interrupt service routine, the filtering that can be performed in the AHDLC layer is under the constraints of the ISR filtering, which is described in Section 4. Otherwise, if the AHDLC processing is outside of the interrupt service routine, stateless packet filtering can be done during the AHDLC processing as described in the reference [8]; stateful packet filtering is done at the upper layer protocol processing stage. In both cases, there can be at least two stages of packet filtering.

8 Conclusions

The multi-stage packet filtering method presented in this paper builds a network firewall inside network smart cards. It drops unwanted packets as soon as possible to save memory resources and to reduce CPU usage for packet processing. This is a general framework of efficient packet filtering for network smart cards. It can be used with a variety of hardware and software configurations and with various filter rules. The method has several advantages over existing packet filtering designs, including better security, reduced memory usage, and enhanced performance. The approach is applicable to a variety of small resource-constrained embedded network devices to enhance their security and success on the Internet.

References

[1] Montgomery, M., Ali, A., and Lu, H.K., "SECURE NETWORK CARD - Implementation of a Standard Network Stack in a Smart Card," Sixth Smart Card Research and Advanced Application IFIP Conference (Cardis), Toulouse, France, August 23-26, 2004.

[2] Ali, A. and Montgomery, M., "Secure Internet Access and the Role of Network Smart Card," Proc. of the 4th IASTED Int. Conf. on Communications, Internet and Information Technology. Cambridge, MA, USA. Oct 31 - Nov 02, 2005, page 259-265.

[3] Cheswick, W.R., Bellovin, S.M. and Rubin, A.D., *Firewalls and Internet Security*, Addison-Wesley, 2003.

[4] Lockhart, A., *Network Security Hacks*, O'Reilly, 2004.

[5] Zwicky, E.D., Cooper, S. and Chapman D.B., *Building Internet Firewalls*, O'Reilly, 2000.

[6] McCanne, S. and Jacobson V., The BSD Packet Filter: A New Architecture for User-level Packet Capture**.** In *Proceedings of the Winter 1993 USENIX Conference*, pages 259-290, January 1993.

[7] Mogul, J., Rashid, R. and Accetta. M., The Packet Filter: An Efficient Mechanism for User-level Network Code. In *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 39-51, November 1987.

[8] Lu, H.K., "Firewall at AHDLC Layer," The 2005 International Conference on Embedded Systems and Applications, June 27-30, 2005, Las Vegas, USA.
[9] Lu, H.K. and Ali, A., "Prevent Online Identity Theft - Using Network Smart Cards for Secure Online Transactions," 7th Information Security Conference (ISC), Palo Alto, CA, USA, September 27-29, 2004.
[10] Lu, H.K., "New Advances in Smart Card Communication," International Conference on Computing, Communications and Control Technologies (CCCT), Austin, TX, USA, August 14-17, 2004.
[11] Universal Serial Bus Communications Class Subclass Specification for Ethernet Emulation Model Devices, http://www.usb.org/developers/devclass_docs/CDC_EEM10.pdf.
[12] PPP – RFC 1662.
[13] Calson, J., PPP Design, Implementation, and Debugging, Addison-Wesley, 2000.