Report Number: WUCSE-2003-68

2003-08-11

Secure Remote Control and Configuration of FPX Platform in Gigabit Ethernet Environment

Haoyu Song

Because of its flexibility and high performance, reconfigurable logic functions implemented on the Field-programmable Port Extender (FPX) are well suited for implementing network processing such as packet classification, filtering and intrusion detection functions. This project focuses on two key aspects of the FPX system. One is providing a Gigabit Ethernet interface by designing logic for a FPGA which is located on a line card. Address Resolution Protocol (ARP) packets are handled in hardware and Ethernet frames are processed and transformed into cells suitable for standard FPX application. The other effort is to provide a secure channel to enable...
Read complete abstract on page 2.

Secure Remote Control and Configuration of FPX Platform in Gigabit Ethernet Environment

Haoyu Song

Complete Abstract:

Because of its flexibility and high performance, reconfigurable logic functions implemented on the Field-programmable Port Extender (FPX) are well suited for implementing network processing such as packet classification, filtering and intrusion detection functions. This project focuses on two key aspects of the FPX system. One is providing a Gigabit Ethernet interface by designing logic for a FPGA which is located on a line card. Address Resolution Protocol (ARP) packets are handled in hardware and Ethernet frames are processed and transformed into cells suitable for standard FPX application. The other effort is to provide a secure channel to enable remote control and configuration of the FPX system through public internet. A suite of security hardware cores were implemented that include the Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES), Hashed Message Authentication Code (HMAC), Message Digest Version 5 (MD5) and Secure Hash Algorithm (SHA-1). An architecture and an associated protocol have been developed which provide a secure communication channel between a control console and a hardware-based reconfigurable network node. This solution is unique in that it does not require a software process to run on the network stack, so that it has both higher performance and prevents the node from being hacked using traditional vulnerabilities found in common operating systems. The mechanism can be applied to the design and implementation of re-motely managed FPX systems. A hardware module called the Secure Control Packet Processor (SCPP) has been designed for a FPX based firewall. It utilizes AES or 3DES in Error Propagation Block Chaining (EPBC) mode to ensure data confidentiality and data integrity. There is also an authenticated engine that uses HMAC. to generate the acknowledgments. The system can protect the FPX system against attacks that may be sent over the control and configuration channel. Based on this infrastructure, an enhanced protocol is addressed that provides higher efficiency and can defend against replay attack. To support that, a control cell encryption module was designed and tested in the FPX system.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2003-68

Secure Remote Control and Configuration of FPX Platform in Gigabit Ethernet Environment - Master's Project, August 2003

Authors: Song, Haoyu

August 11, 2003

Abstract: Because of its °exibility and high performance, reconfigurable logic functions implemented on the Field-programable Port Extender (FPX) are well suited for implementing network processing such as packet classi¯cation, ¯ltering and intrusion detection functions.

This project focuses on two key aspects of the FPX system. One is providing a Gigabit Ethernet interface by designing logic for a FPGA which
is located on a line card. Address Resolution Protocol (ARP) packets are handled in hardware and Ethernet frames are processed and transformed into cells suitable for standard FPX application.

The other effort is to provide a secure channel to enable remote control and configuration of the FPX system through public internet. A suite of security hardware cores were implemented that include the Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES), Hashed Message Authentication Code (HMAC), Message Digest Version 5 (MD5) and Secure Hash Algorithm (SHA-1).

An architecture and an associated protocol have been developed which provide a secure communication channel between a control console and a hardware-based reconfigurable network node. This solution is unique

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SECURE REMOTE CONTROL AND CONFIGURATION OF THE FPX PLATFORM IN GIGABIT ETHERNET ENVIRONMENT

by

Haoyu Song

Prepared under the direction of Professor John W. Lockwood

A project report presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

Master of Science

August, 2003

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

SECURE REMOTE CONTROL AND CONFIGURATION OF THE FPX PLATFORM IN GIGABIT ETHERNET ENVIRONMENT

by Haoyu Song

ADVISOR: Professor John W. Lockwood

August, 2003
Saint Louis, Missouri

Because of its flexibility and high performance, reconfigurable logic functions implemented on the *Field-programable Port Extender* (*FPX*) are well suited for implementing network processing such as packet classification, filtering and intrusion detection functions. This project focuses on two key aspects of the *FPX* system. One is providing a *Gigabit Ethernet* interface by designing logic for a *FPGA* which is located on a line card. *Address Resolution Protocol* (*ARP*) packets are handled in hardware and *Ethernet* frames are processed and transformed into cells suitable for standard *FPX* application.

The other effort is to provide a secure channel to enable remote control and configuration of the *FPX* system through public internet. A suite of security hardware cores were implemented that include the *Advanced Encryption Standard* (*AES*),

Triple Data Encryption Standard (*3DES*), *Hashed Message Authentication Code* (*HMAC*), *Message Digest Version 5* (*MD5*) and *Secure Hash Algorithm* (*SHA-1*). An architecture and an associated protocol have been developed which provide a secure communication channel between a control console and a hardware-based reconfigurable network node. This solution is unique in that it does not require a software process to run on the network stack, so that it has both higher performance and prevents the node from being hacked using traditional vulnerabilities found in common operating systems. The mechanism can be applied to the design and implementation of remotely managed *FPX* systems. A hardware module called the *Secure Control Packet Processor* (*SCPP*) has been designed for a *FPX* based firewall. It utilizes *AES* or *3DES* in *Error Propagation Block Chaining* (*EPBC*) mode to ensure data confidentiality and data integrity. There is also an authenticated engine that uses *HMAC* to generate the acknowledgments. The system can protect the *FPX* system against attacks that may be sent over the control and configuration channel. Based on this infrastructure, an enhanced protocol is addressed that provides higher efficiency and can defend against replay attack. To support that, a control cell encryption module was designed and tested in the *FPX* system.

Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to thanks my project advisor Dr. John W. Lockwood. This research would not have been possible without him. It is he who kept me motivated and encouraged me to work on this project.

I also thank Dr. William D. Richard, John DeHart and Fred Kuhns who have been involved with the Gigabit Ethernet Line Card project. Their initial work and diligent help during testing make it possible to complete this project smoothly and quickly.

I'd like to thank Todd Sproull for his help in the lab. Thanks are due to James Moscola for his work on the frame wrapper which is a part of the Gigabit Ethernet project and his software implementation of the secure control packet processor. Thanks are due to Jing Lu for her work in hardware design of the security cores. Thanks are due to Dave Lim for his cooperation in the testing and development of the Gigabit Ethernet Card.

This research is supported in part by grants from the National Science Foundation and Global Velocity.

Haoyu Song

Washington University in Saint Louis
August 2003

Chapter 1

Motivation

As the Internet grows, more and more networking applications will use reconfigurable hardware devices to provide both high performance and flexibility. The *Field-programable Port Extender* (*FPX*) is an open, reconfigurable, high performance and extensible *IP* packet processing platform [24][23]. It processes packets in hardware to perform content filtering, intrusion detection and other customizable functions. All logic on the *FPX* is implemented in *FPGAs*. *FPGAs* accelerate processing while preserving flexibility. Several packet processing engines can be deployed in parallel or in pipelines, and it is possible to achieve multiple gigabit per second throughput.

The *FPX* platform is highly reconfigurable. New bitfiles can be downloaded on demand within seconds and start being tested at once. Moreover, the hardware can be reprogrammed via the Internet without physical access to the device. Consequently, remote control and configuration of the *FPX* are feasible and convenient for a network administrator.

The *FPX* platform can be deployed throughout a backbone network or at an access point to an enterprise network. Gigabit rate network links are common today. To meet this bandwidth requirement, the *FPX* has a broadband line interface.

To allow the *FPX* to be easily connected to standard networks, a *Gigabit Ethernet* interface was implemented. *Ethernet* is an evolving technology that will play an important role not only in *Local Area Networks* (*LANs*) but also in *Wide Area Networks* (*WANs*).

Secure control and configuration of the *FPX* system are needed to enable deployment of *FPX* systems throughout the public Internet. Run time reconfiguration allows dynamic hardware plugins to be sent over networks and to update the functions on the *FPX*. It is envisioned that the *FPX* system will be distributed over large geographic areas and operate over public networks, making on-site configuration and management infeasible. The security of the remote control and configuration is a serious concern. Robust security mechanisms are needed to protect the reconfigurable network nodes from unauthorized access and to ensure the integrity of reconfiguration when being reprogrammed over the network.

1.1 Background

1.1.1 FPX Platform

The logic of the *FPX* is implemented within two *FPGAs*: the *Reconfigurable Application Device* (*RAD*) which is used to prototype new networking functions and protocols, and the *Network Interface Device* (*NID*) which is used to interface between the line card and *RAD*.

The *FPX* platform can act as a network monitor to screen and analyze the traffic passing through the Internet. It can also actively process the traffic using predefined policies. *Global Velocity* has found one such application for copyright-protection that detects and blocks peer-to-peer transfer of copyright-protected content

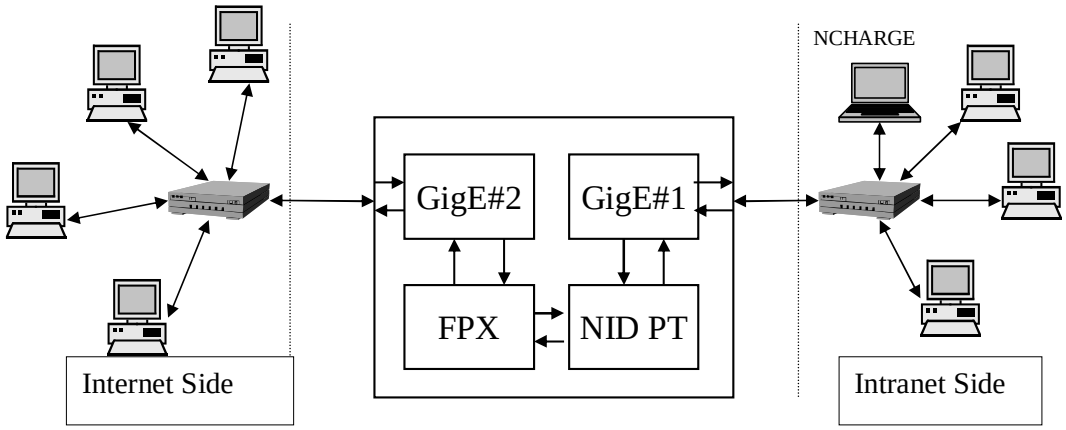

Figure 1.1: FPX in a box

on a network. The *FPX* can also be used to scan and block viruses and Internet worms.

Two *FPX* cards and two line cards are used in a *FPX* standalone platform as is shown in Figure 1.1. The side on which the control console is attached is defined as the *Intranet* side. Notice that it is unnecessary for the control console to be directly connected to the box. Since the box is Internet addressable, control packets can be issued remotely. On the *Intranet* side, the *NID* on the *FPX* card is configured to pass through traffic (*NID-PT*). It is responsible for extracting and encapsulating packets that pass through the network. The other side of the *FPX* platform is called the *Internet* side. On this side, there sits a regular *FPX* card which monitors or processes the packets. For historical reasons, the basic unit processed in *FPX* is an *ATM* cell. Internet packets are encapsulated into *AAL5* frames. A suite of protocol wrappers were designed to process data at different protocol layers.

The *FPX* platform works at gigabit-per-second link speed. The *Gigabit Ethernet* line card interface allows the *FPX* to sit in an *Ethernet LAN* and work in both

passive and active modes. The line card has a Gigabit Ethernet Controller Application Specific Integrated Circuit, the *PM3386 S/UNI-2xGE* [44]. The *PM3386* is a monolithic *ASIC* that implements full-duplex 1000 Mbps *Ethernet MAC* transport function. The *PM3386* provides connectivity to an on-chip *SERialize/DESerialize* (*SERDES*) and *Gigabit Media Independent Interface GMII* functions. It also provides a data transport interface to the up stream device via an industry standard *POS-PHY Level 3* interface. The system's backplane connector sends and receives data formatted in *ATM* cells. In order to translate between *ATM* cells and *Ethernet MAC* frames, an extra circuit is needed. A Xilinx *FPGA XC2V1000* [45] is arranged on the line card to implement those functionalities. The *XC2V1000* belongs to the Xilinx *VIRTEX-II* family. It has what is called the equivalent of 1M system gates and also has 160 Kbits of block *RAM*. These resources are enough to implement packet processing functions needed for the circuit.

The initial version of the *FPX* platform did not use any encrypted mechanism to control and configure the hardware. There is an urgent requirement to add this functionality in order to make the *FPX* reliable and secure in a public network.

1.1.2 Gigabit Ethernet Processing

Ethernet has been the dominant *LAN* technology since the early 1970s. *Gigabit Ethernet* is built on top of the *Ethernet* protocol, but increases speed tenfold over *Fast Ethernet* to 1 Gbps. This protocol, which was standardized in June 1998, promises to be a dominant technology for high-speed local area network backbones and server connectivity. Reference [39] gives a detailed technology review of *Gigabit Ethernet*.

Ethernet uses a 48-bit physical address to identify each node. Each frame has a 14-byte *MAC* header which includes a 6-byte destination address, a 6-byte source

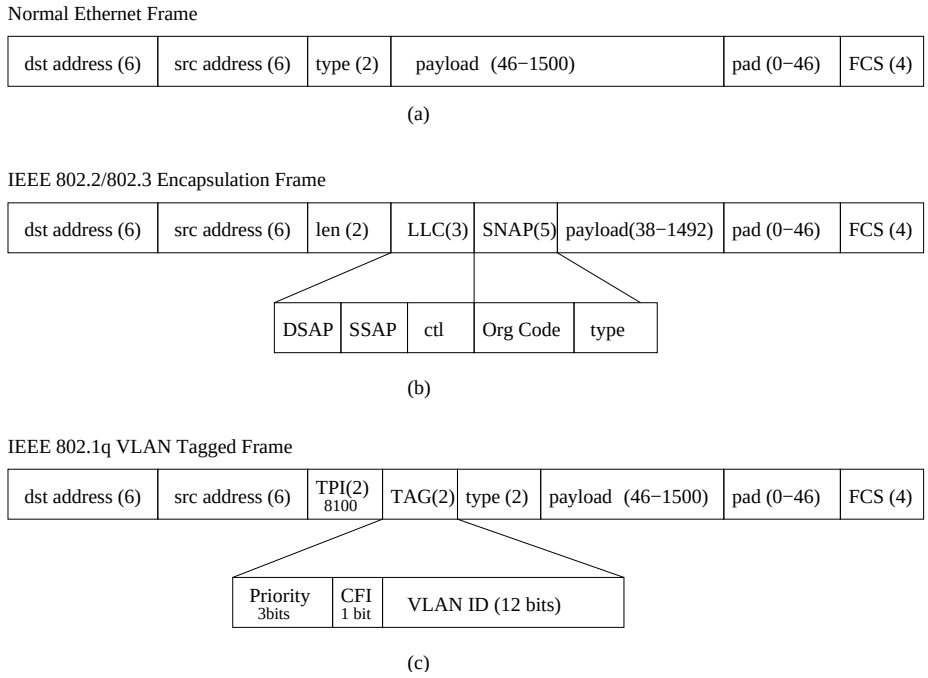

Figure 1.2: Ethernet Frame Format

address and 2 bytes payload type field (In *Ethernet V2*, this field is also defined as payload length field if the value is greater than 1500). Figure 1.2(a) shows a typical *Ethernet* frame as defined in *RFC894*. *IEEE 802.2* also defines a *LLC/SNAP* field for the *Ethernet* header, as shown in Figure 1.2(b). Figure 1.2(c) shows another type of an *Ethernet* frame which is defined in *IEEE 802.1q* known as *Virtual LAN* (*VLAN*) field. A *VLAN* can be viewed as a group of devices on different physical *LAN* segments which can communicate with each other as if they were all on the same physical *LAN* segment. Networks can have differential classes of service (*COS*) functions based on the value of the *VLAN ID* and the service priority. *VLAN* technology enables flexible network segmentation by assigning different *VLAN IDs* to the different subnetworks. It improves network management by managing the logical *LAN* instead of the physical *LAN* and increases the performance by isolating the broadcast domains and enforcing the *COS*. It also enhances the network security by limiting the extent to which packets are broadcast.

In our application, we use *Ethernet* to carry *IP* packets. One key protocol is the Address Resolution Protocol (*ARP*)[35]. Whenever an *IP* packet is ready to be forwarded in an *Ethernet* frame over the network, the directly connected host or gateway router's physical address must be resolved using the destination *IP* address. To resolve a forwarding *IP* address to its *MAC* address, *ARP* sends out a broadcast frame called an *ARP Request* on the shared media. Any host or router which has the requested *IP* address or is responsible for that address will send back to the sender an *ARP Reply* which contains the *MAC* address corresponding to the desired forwarding *IP* address.

An *ARP* cache is maintained in order to keep the number of broadcast *ARP Request* frames to a minimum. Recently resolved *IP* addresses and their corresponding *MAC* addresses are stored in a table. The *ARP* cache is checked first before sending an *ARP Request* frame. Only when there is not a matched entry will an *ARP Request* packet be sent. *ARP* cache entries can either be dynamic (based on *ARP Replies*) or static (if configured manually). Static *ARP* cache entries are used to prevent *ARP Requests* from being broadcast for commonly-used local IP addresses, such as routers and servers. The problem with static *ARP* entries is that they have to be manually updated when network interface equipment changes. Dynamic *ARP* cache entries have a time-out value associated with them so that entries are removed form the cache after a specified period of time. The *ARP* packet format is shown in Figure 1.3.

1.1.3 Network Security and Cryptography

Although we enjoy the abundant information available on open networks, when we use the Internet, we also face the threat of being infected with a virus, or having

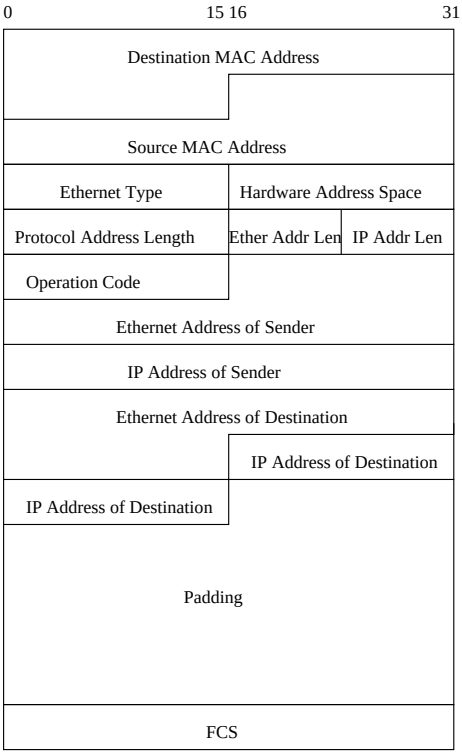

Figure 1.3: Address Resolution Protocol Frame Format

our machine hacked. The growth of the Internet requires us to protect both our infrastructure and information from malicious attacks.

In general, attacks can be classified into two major categories: *Denial-of-Service* (*DOS*) and *Unauthorized Access*. For a *DoS* attack, the attacking host(s) sends more requests to a target machine than it can handle. Usually the source of the attack is hard to trace. *Unauthorized Access* includes eavesdropping, transmission of fake data, replay of previously sent messages and data destruction. Cryptography can provide protection against many types of attacks at a reasonable cost. Many encryption and authentication algorithms have been developed and are widely used in network security as well as in other fields.

There are two common types of encryption algorithms. The first type uses private-keys (also called symmetric-keys) which only have one unique key for both

encryption and decryption; the other type uses public and private key-pairs (also called asymmetric-keys). Basic attributes of encryption technologies are *confusion* and *diffusion* [41]. *Confusion* intends to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible. *Diffusion* seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible. For an ideal algorithm, the security depends only on the secret key.

The *Data Encryption Standard* (*DES*) [30] was developed by an *IBM* team and adopted as a national standard in 1977. Despite the growing concerns about its vulnerability, *DES* is still widely used to protect sensitive online applications. *3DES* is a variation of this standard that can be billions of times more secure if used properly. The procedure of *3DES* encryption is exactly the same as regular *DES*, except that *DES* is repeated three times. In 2000, *National Institute of Standards and Technology* (*NIST*) selected the *Rijndael* algorithm as the proposed *Advanced Encryption Standard* (*AES*) [31] for protecting data through encryption. In addition to the increased security that comes with larger key sizes, *AES* can encrypt data much faster than *3DES*.

Most of the private-key algorithms are perfectly suited for hardware implementation due to their regular structure and bit-wise operations used to encrypt data. Many cipher modes can be applied to provide extra security for some kinds of attacks or to protect data integrity. Some classic modes are *Electronic Codebook Mode* (*ECB*), *Cipher Block Chain Mode* (*CBC*), *Output Feedback Mode* (*OFB*), *Cipher Feedback Mode* (*CFB*) and *Counter Mode* (*CTR*). All of these modes have their own advantages and disadvantages for different applications. Some novel modes provide interesting and more secure features [1][32].

Public-key or asymmetric-key encryption algorithms use two different keys for encryption and decryption. A key pair is generated and a public key is used to encrypt messages and the ciphertext can only be decrypted with a private key. For a good encryption algorithm, it is very difficult to determine the private key given the public key. This type of algorithms can also be used for digital signatures, authentication and key distribution. One popular algorithm in this category is the *Rivest-Shamir-Adleman* (*RSA*), named after the researchers who developed it [41]. Since asymmetric algorithms are relatively slow, they are seldom used for encryption of large amounts of data.

Though encryption itself can provide authentication to some extent, a keyed hash function is preferred for that function. Algorithms such as *Message Digest Version 5* (*MD5*) or *Secure Hash Algorithm* (*SHA-1*) can generate the digest of a message. *Hashed Message Authentication Code* (*HMAC*) embeds these hash functions into what is called a "black box" to have the last digest code only depend on a secret key [22]. Since it was published in *RFC2104*, *HMAC* has become the most popular method of data authentication.

Firewall technologies and network intrusion detection devices are now widely deployed in organizations and corporations. They audit and classify packets to detect and prevent harmful attacks. In order to keep up with the explosive growth of the network bandwidth, encryption and decryption tasks must be performed in hardware. Reconfiguable hardware is especially well-suited for the implementation of such network processing functions.

1.2 Contributions of This Work

The logic that controls a *Gigabit Ethernet* line card has been implemented on an *FPGA* and verified to work with the *FPX* platform. Another version was designed for the *Multi-Service Router* (*MSR*) project, where the line card is mainly used in the *Washington University Gigabit Switch* (*WUGS*).

A Secure Control Packet Processor (*SCPP*) for dynamic configuration of the *FPX* has been designed and verified. The module implements several *IPSec* standards including *AES*, *3DES* and *HMAC* using either *MD5* or *SHA-1*. A secure acknowledgement protocol was designed so that the *FPX* platform can be securely configured remotely via the Internet.

Based on this infrastructure, the security of the *FPX* platform was enhanced by introducing a secure protocol with a secure communication channel between a control console and an *FPX* platform. This solution is unique in that it runs in hardware on the end system, so that it has both higher performance and safety as compared to protocol stacks implemented in software atop an operating system . This secure control scheme can also prevent replay attacks.

1.3 Related Work

An *Internet Protocol* (*IP*) stack implemented in reconfigurable hardware is used to process packets [6]. It includes a set of layered wrappers which process *ATM* cells, *AAL5* frames, and *IP* and *UDP* packets. An *ARP* engine was implemented so that the system can work in *LAN* environment. Other projects implemented *ARP* as part of *IP* stack. The protocol stack described in [33] is limited to 10Mb/s *Ethernet*

operation. It implements the lower layers of a protocol stack and only supports *IP* and *ARP* functions.

Symmetric-key encryption algorithms and message digest hash functions are widely used for network security. Many of them are well-suited for hardware implementation. There are many papers that discuss *FPGA* implementations of *AES*, *DES*, *HMAC* and other security algorithms [16][9][14][17]. These kinds of hardware designs are all tradeoff between the throughput and the circuit area. Usually, iteration unrolling and pipeline technologies are used to improve the throughput. To optimize the area, iteration rolling is used at the expense of throughput degradation.

For secure remote control, *S. Gultchev et. al.* presented a secured *Reconfigurable Management Architecture* (*RMA*) to enforce robust security mechanisms on mobile *Software Radio Terminals* [37]. Due to the heavy computation cost, this scheme is inefficient to implement in a pure hardware environment. The low data throughput makes it unsuitable for high-speed data communication. *R. Chakravorty et. al.* presented a *Smart Box Management* (*SBM*) - an end-to-end remote management framework for Internet enabled devices [7]. Though this work is focused on a software framework, it provides a good framework for remote device management. While many of the network management technologies today only monitor nodes, the *MIDAS* project by *S.N.Bhatti et. al.* offers management capability for a large distributed system [4]. Other work done by *J. Forne, et. al.* presents a solution providing secure communications over an extended *Ethernet LAN* [27]. Most of the works listed above focus on development of a software-based security framework and involve a huge computational effort, which can cause unbearable overhead and bottleneck performance for applications such as programmable network routers, sensors, and firewalls.

1.4 Outline

The thesis is organized as follows. Chapter 2 presents the design, verification and performance evaluation of the logic for a *Gigabit Ethernet* line card implemented in an *FPGA*. *ARP* processing and packet transformation are discussed in detail. Chapter 3 discusses the design of some cryptographic modules and their application in the *SCPP* structure for secure remote control of a *FPX*-based firewall, including a review of the implementation of the *AES*, *3DES*, *HMAC* and a specific mode, *Error Propagation Block Chaining* (*EPBC*). In chapter 4, a more secure configuration protocol is proposed which enables the *FPX* platform to provide a secure control and configuration interface for general application. Finally, Chapter 5 concludes with a summary and a discussion of future work.

Chapter 2

Gigabit Ethernet Line Card FPGA

2.1 Protocol Stack

The protocol stack used in the *FPX* platform is shown in Figure 2.1. On the line side, *IP* packets are transmitted over *Gigabit Ethernet*; On the system side, *IP* packets are formatted into *AAL5* frames and split into *ATM* cells.

The *ATM* cell has a fixed and short length that makes it suitable for fast switching and processing functions. One *ATM* cell has a 5-byte header and a 48-byte payload. The routing information is indicated by *Virtual Path Identifier* (*VPI*) and *Virtual Circuit Identifier* (*VCI*) fields in *ATM* header. *Header Error Control* (*HEC*), the last byte of the header, is used to control the header's correctness. *AAL5* is designed for packet transmission and is widely used to transport Internet Protocol data. In *AAL5*, a frame with an arbitrary length is put into a *Protocol Data Unit* (*PDU*). In this design, we implemented classical *IP* over *ATM*. A *PDU*'s length is a multiple of 48 octets. One bit in the *Payload Type Identifier* (*PTI*) field of the *ATM* header is used to indicate whether a cell is the last one of a *PDU*. The last 8 octets of the *PDU* are used as a trailer, which contain the information about the

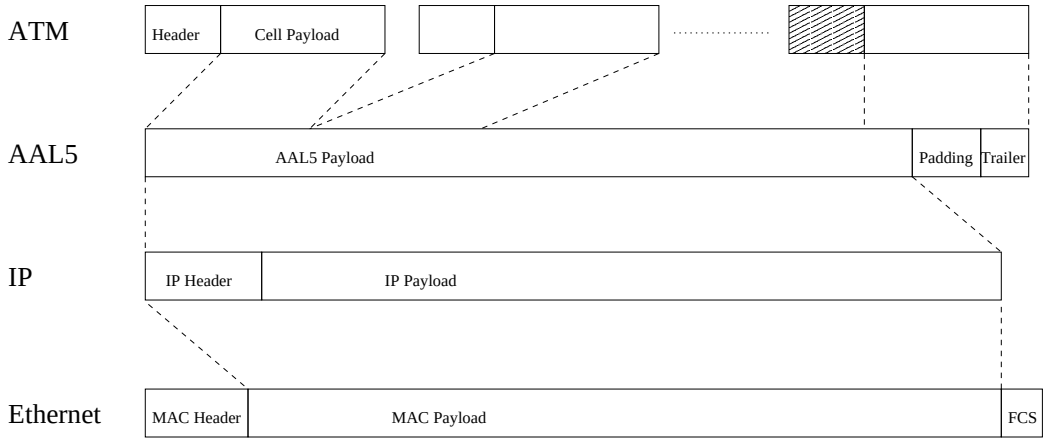

Figure 2.1: FPX in a box Protocol Stack

actual length of the payload data and a 32-bit *CRC* to ensure data integrity. Any gap between the frame and the trailer is filled with padding. Since *PDU*s are multiples of 48 octets, the trailer always ends at a cell's boundary and can therefore be located. The segmentation of frames with *AAL5* and *ATM* cell's format are illustrated in Figure 2.2.

2.2 Architecture

The two major tasks of the *Gigabit Ethernet* (*GigE*) *FPGA* are to translate protocols and to perform *ARP*. The hardware logic must act as a bridge between *ATM* and *Ethernet* networks, handle both *ARP* request and reply, and maintain an *ARP* table. Further, *IEEE802.1q VLANs* are also supported to make it flexible in different application environments. Figure 2.3 shows the overall structure of the *FPGA* circuit.

2.2.1 Cell and Frame Wrapper

The cell and frame wrapper module is part of "*Layered Protocol Wrappers*"[6][5], that are a collection of *VHDL* components processing high-level Internet protocols directly

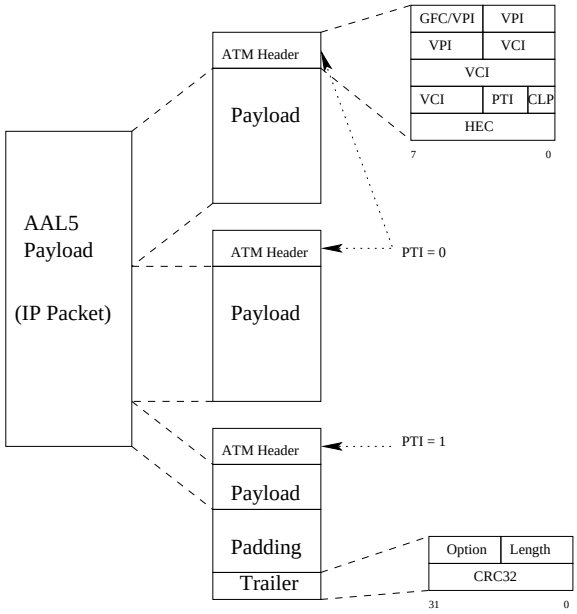

Figure 2.2: AAL5 to ATM mapping

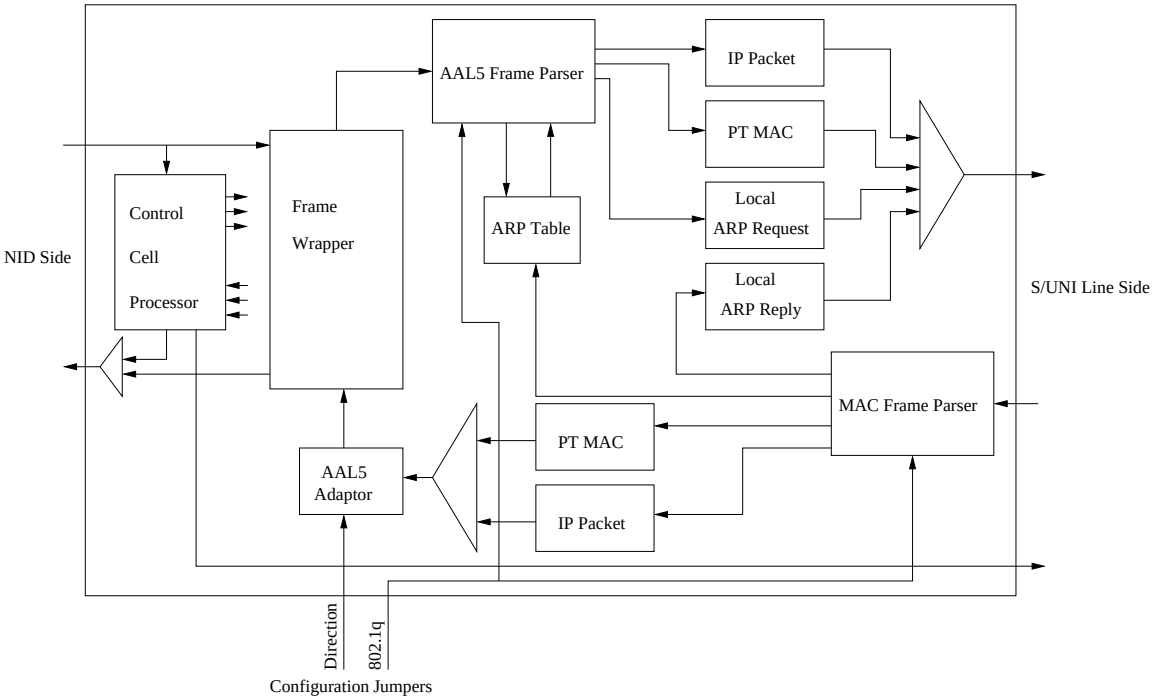

Figure 2.3: GigE FPGA Block Diagram

in hardware. The hardware library has four components that are used together to transmit and receive: fixed-length cells (*ATM*), variable-length *AAL5* frames, *IP* packets and *User Datagram Protocol* (*UDP*) frames. They are primarily designed for the *FPX*, but could be also used in any hardware design. The *ATM* cell wrapper and the *AAL5* frame wrappers are used in the implementation of the *Gigabit Ethernet* circuit design.

Incoming *ATM* cells from the *FPX* system side uses a signal *Start Of Cell* (*SOC*) to delimit the cells. The *HEC* is calculated and compared for every cell to check the cell's validity. If the check fails, the cell is dropped. Only valid cells are passed to the *AAL5 Frame Wrapper*. The *Frame Wrapper* extracts the *AAL5* frames from the incoming cell stream, strips off any padding data, calculates *AAL5 Frame Checksum* (*FCS*) to verify the frame integrity and marks error frames. The *Frame Wrapper* replaces the *SOC* signal with three signals, namely *Start of Frame* (*SOF*), *End Of Frame* (*EOF*) and *Data Enable* (*DataEN*). The *AAL5* frame data is sent to the *Egress MAC* module for processing.

An incoming *AAL5* frame from the *Ingress MAC* module is segmented to a series of 48-byte data chunks in order to fit them into *ATM* cells. The remainder is padded with some zeros to make the size fit into one cell obeying the frame length information provided by *Ingress MAC* module. The *Frame Wrapper* calculates the frame *FCS* and then appends it as the *AAL5* frame trailer. The *ATM* cells are passed to *Cell Wrapper* from *Frame Wrapper*. The *Cell Wrapper* calculates *HEC*, fills it in *Header* for each cell and then forwards the cells to system side.

2.2.2 Control Cell Processor and Line Card Self-Configuration

A *Control Cell Processor* (*CCP*) block is used to receive control cells from the system side and extract the control commands to configure both *FPGA* and *GigE* controller *ASIC*. The configurations include the *GigE* line card's *MAC* address, *IP* address and mask, *ARP* table and initialization of the *GigE ASIC*. Through *CCP* a control host can also read back all the statistics counters and any *ARP* table entry.

To enable the system to bootstrap itself, a set of initialization control cells are stored in an on-chip *Read Only Memory* (*ROM*). When the system boots up, the cells are automatically injected into the *CCP*. Thus, the whole line card is configured and the system can immediately operate without any outside control.

2.2.3 ARP Lookup Table and Processing

An on-chip synchronous dual port *Random Access Memory* (*RAM*) was used to implement the *ARP* lookup table. The *Egress MAC* module performs read operations and *Ingress MAC* module performs write operations. The *CCP* can also access the table for debugging purposes or manual configuration.

In order to not interrupt the system's normal operation, the *CCP* has lower priority to access the table. When the *CCP* issues a write or read operation, if it happens that *MAC* modules are doing a similar operation, the operation from the *CCP* is paused and will not be processed until the operation from *MAC* modules is finished. Because two similar operation requests from the *MAC* modules have such long intervals from the point of view of the system, it is safe to perform the operation request from *CCP* after current operation from *MAC* modules. So at most 3 cycles after issuing the read request, the *CCP* can get the table output. Likewise, at most

3 cycles after issuing the write request, the data must have already been written into the table.

Ideally, an *ARP* lookup table should be able to handle any number of valid *IP* addresses and *MAC* addresses pair. But, in order to do this, an *ARP* lookup engine may need to scan a very long table to find the match. That is not practical for hardware implementation. In the implementation of the *Gigabit Ethernet* circuit, in order to limit the table to a size that could fit into the available space on the *FPGA*, a 10-bit index is used which restricts the table size to be 1024 entries. A 2-bit network identifier and *Least Significant Byte* (*LSB*) of the 32-bit *IP* address are combined to generate the lookup index. The device supports 3 different networks with network masks of at least 24 bits long. The network identifier is coded as 00, 01 and 10. That allows the host identifier to be at most 8 bits long and thus allows 256 hosts to be in one subnetwork.

The width of the data bus in the *RAM* is 49 bits. The lower 48-bit word is a *MAC* address and the 49th bit indicates validation of the the entry. "1" indicates a valid entry and "0" indicates an empty entry. The use of the extra bit simplifies the logic to check the entry validation and enables the software to control the *ARP* table aging.

2.2.4 Ingress MAC Processing

Figure 2.4 shows the block diagram of the *Ingress MAC* module. The *Ingress MAC* module receives and parses the *MAC* frame from the *GigE ASIC* through a *POS-PHY Level 3* interface. The destination *MAC* address is checked first. If it does not match this card's *MAC* address, the whole *MAC* frame will be passed through the *FPX* without any change. The frame is stored in a *FIFO* temporarily. If the frame

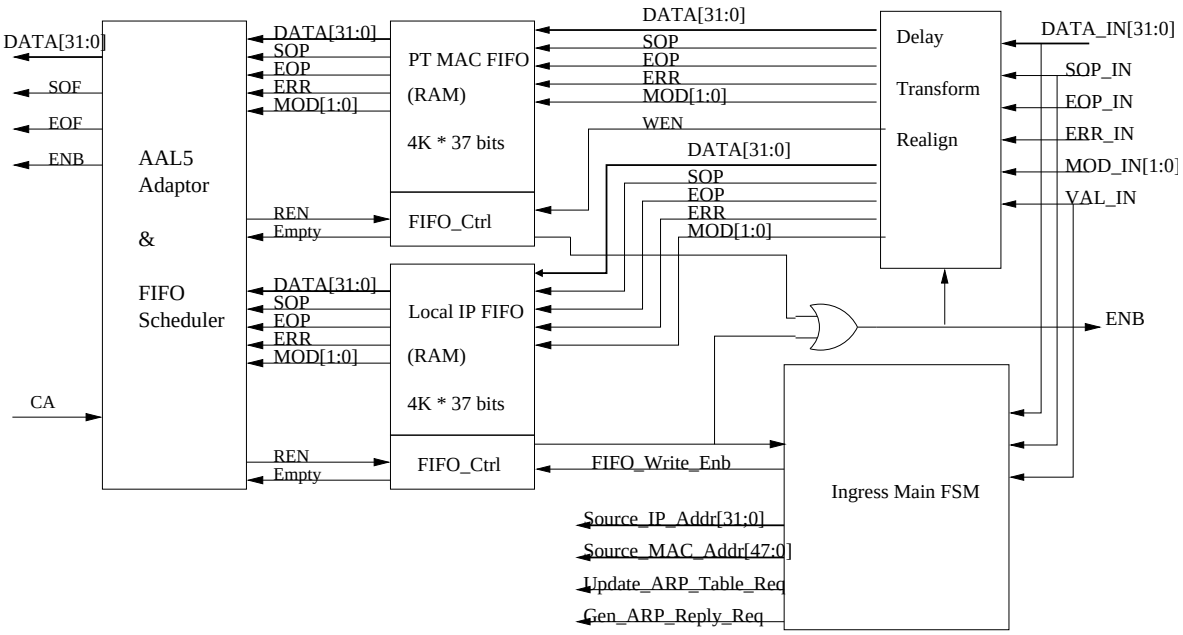

Figure 2.4: Ingress MAC Processing Block Diagram

type is either an *ARP* request or an *ARP* reply, the source *IP* address and the *MAC* address are fetched and the information is used to update the *ARP* lookup table. For a matched *ARP* request, an *ARP* reply is triggered on the egress side. If the frame has the matched destination *MAC* address and encapsulates an *IP* packet (usually this is a control packet), the *MAC* header is stripped off and the pure *IP* packet is stored in a FIFO temporarily. All other types of packets will be treated as unknown types and simply discarded.

A scheduler dispatches the pass-through traffic and local traffic to the *FPX* side in *round-robin* manner. The output *IP* packet or *MAC* frame is adapted into an *AAL5* frame in order to be fed to the *Frame Wrapper* module. All local *IP* packets are assigned *VCI* 50 and all pass-through frames are assigned *VCI* 51. The *Ingress MAC* module leaves the *FCS* field of the *AAL5* frame blank if the packet or frame is error free; on the contrary, if the packet or frame is marked as an errored one, a nonzero value is filled in the *AAL5 FCS* field. The *Frame Wrapper* is responsible for

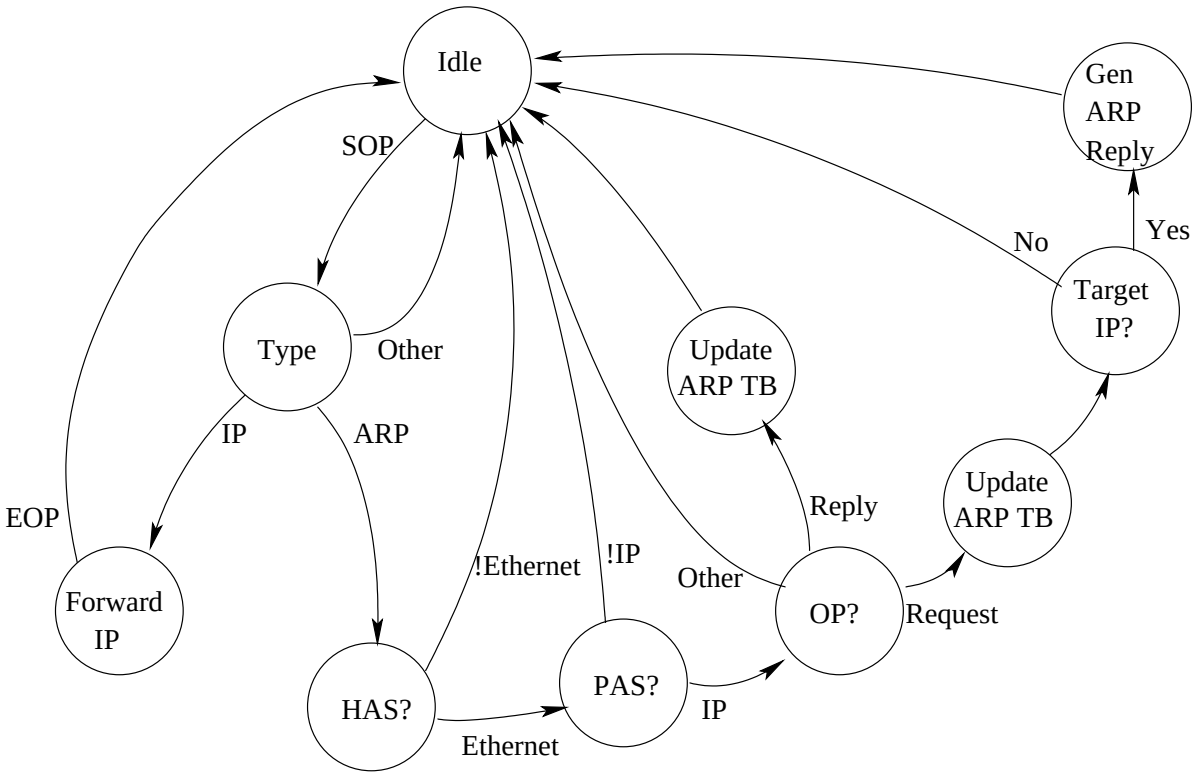

Figure 2.5: Ingress MAC Processing Main FSM

calculating the *FCS* based on the packet's correctness information. The main *Finite State Machine* (*FSM*) of the *Ingress MAC* module is shown in Figure 2.5.

2.2.5 Egress MAC Processing

As shown in Figure 2.6, the *Egress MAC* module receives an *AAL5* frame from the *Frame Wrapper* module. At first all data is buffered in a *FIFO*. This is because the back pressure signal from the *Egress MAC* module can not stop the traffic from the *Frame Wrapper* module immediately and that may cause data to be lost without a buffer. The *FIFO*'s full threshold serves as a back pressure signal to the *Frame Wrapper* module. The threshold must be set to a proper value so that enough room is left for incoming data before the *Frame Wrapper* module responds to this back pressure signal.

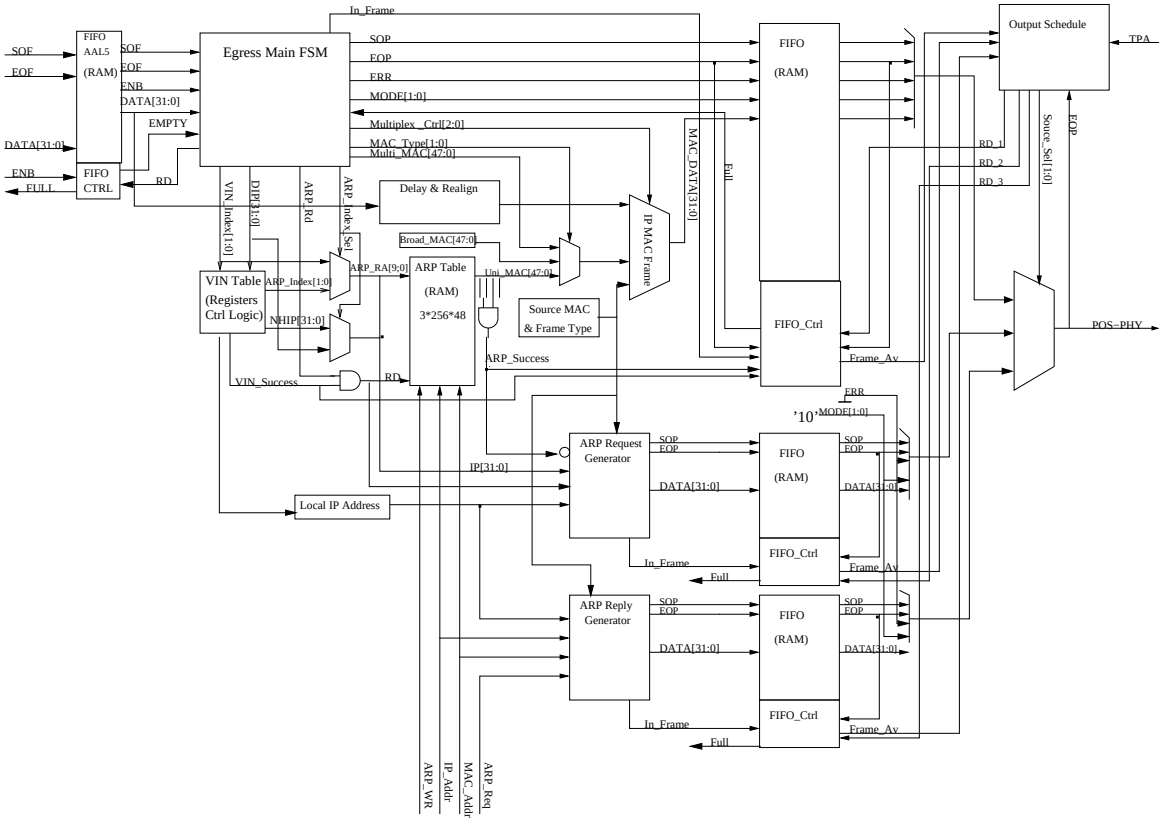

Figure 2.6: Egress MAC Processing Block Diagram

Once the *AAL5* frame *FIFO* is filled with a whole frame or reaches the predefined low threshold, the *Egress MAC* module begins reading data out of the *FIFO*. In the main *FSM*, the *AAL5* frame is classified based on the *VCI* value. *VCI* 51 means that a whole *MAC* frame (i.e. pass-through traffic) is encapsulated in a *AAL5* frame, so the *MAC* frame is extracted out and scheduled to be forwarded to the line side. *VCI* 50 means that the packet is a local *IP* packet which is generated in the *FPX*. Under this condition, the *FSM* must obtain the destination *IP* address from the *AAL5* frame. The *IP* address might be a broadcast address with format: {Network-Number, -1}. In this case, no *ARP* lookup is needed. Instead, the *Egress MAC* module maps the destination *MAC* address to the broadcasting *MAC* address *FF:FF:FF:FF:FF:FF* and forwards the *IP* packet. If the *IP* address belongs to a multicast type (i.e. class *D* address: *0xE0000000 + 28* bit group *ID*), there is no need to do the *ARP* table lookup either. The *IP* address is mapped to the multicast *MAC* address [19] and forward the packet. In case the *IP* address is a unicast one, the *FSM* uses the *IP* address to generate the index to search the *ARP* table. If a valid entry is retrieved, the retrieved *MAC* address is used as the destination *MAC* address, thus a whole *MAC* frame could be assembled to encapsulate the *IP* packet; if the queried entry is empty or invalid, the *IP* packet is simply discarded and an *ARP* request frame is generated and scheduled to be sent to the line side network.

With any other *VCI* value, the *AAL5* frame is discarded and an unknown type of frame event is reported to the *CCP* module. The main *FSM* is shown in Figure 2.7.

After this processing, any local *IP* packet with a valid *MAC* destination address is encapsulated into a *MAC* frame and then is written into a *FIFO*. All pass-through *MAC* frames are written into this *FIFO* too. Those triggered *ARP* request frames are written into an *ARP Request FIFO*. The *Egress MAC* module also accepts the

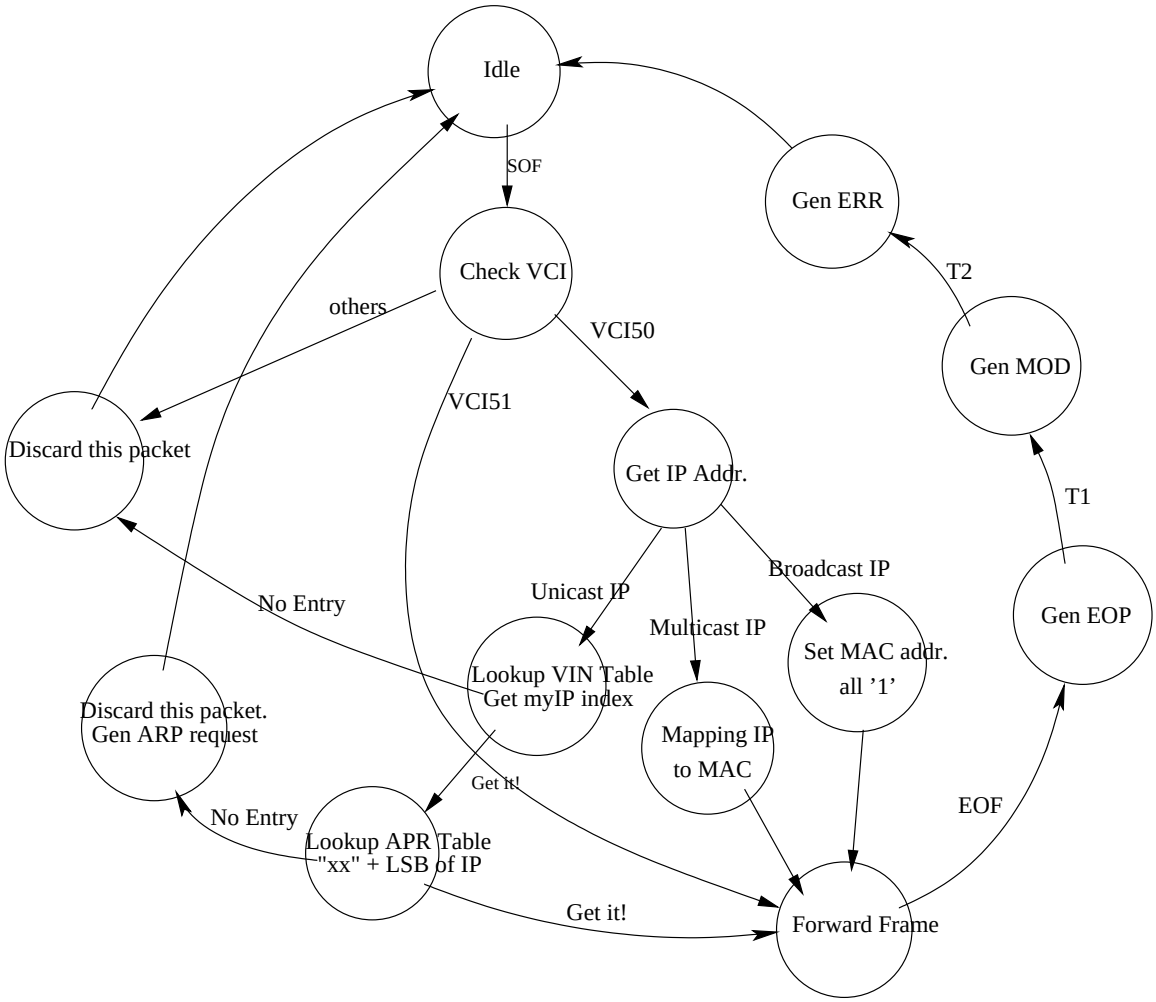

Figure 2.7: Egress MAC Processing Main FSM

requests from the *Ingress MAC* module to generate *ARP* reply frames and stores them into an *ARP Reply FIFO*. In a normal working environment, the *ARP* request and reply happen with low frequency, so the *FIFO* depth is set to 4 frames each.

At last, an output scheduler *FSM* works in round-robin manner to detect and dispatch data from three *FIFOs* alternately. If there is a whole frame in the *FIFO*, the frame is read out and transmitted through a *POS-PHY level 3* interface to the *GigE Controller ASIC*. At this point, all the *MAC* frames are not padded and the frame *CRC* is not calculated yet, so actually they are not complete *MAC* frames. The *GigE Controller ASIC* is responsible for handling them by correctly configuring the internal registers.

2.2.6 VLAN Support and Route Direction

In order to make the *FPX* system compatible in different application environments, *IEEE 802.1q VLANs* are supported in the *FPX* platform. A jumper on the card must be set before the system operation. For security reason, all local *IP* traffic must use *VLAN ID* 1. The pass-through traffic's *VLAN ID* remains unchanged.

To direct the *FPX*'s routing decision upon receiving cells from the *GigE* line card or system motherboard, a direction pin should be set through another jumper on the *GigE* line card. The setting actually controls one bit in the *VPI* field of the *ATM* cells which is used to decide where the cell should be forwarded: the *Internet* side or the *Intranet* side.

2.2.7 Interface

The *FPX* system side interface is simple and is described in [6]. The signal *SOC* indicates the first word of a cell. This word and the following 13 words belong to one

cell. A back-pressure signal is used to control the flow rate when needed. The line side uses a *POS-PHY Level 3* standard interface to connect the *ASIC PM3386*[44]. The *POS-PHY Level 3* interface is a 32-bit wide interface with a clock rate of up to 104 MHz (Though we only use 62.5 MHz clock in our system). *POS-PHY Level 3* was developed with the cooperation of the *SATURN Development Group* to cover all applications which bit rates are up to 3.2 Gbit/s. The *POS-PHY Level 3* specification [18] defines the requirements for interoperation between devices such as the multi-*PHY PM3386* and a single Link Layer device. Each direction within the *FPGA* logic contains a 4096-byte latency *FIFO*.

2.3 Test and Performance Evaluation

The *GigE* logic was synthesized using *Synplify Pro* and was placed and routed in a *Xilinx XC2V1000 FPGA*. Table 2.1 summarizes the *GigE FPGA* place and route results.

Table 2.1: Device Utilization and Timing Summary

IOBs	Block RAMs	SLICEs	Freqency
205	31	4202	
61%	77%	82%	82.2 MHz

Figure 1.1 shows an envisioned *FPX* system configuration. In order to debug the system, all the four cards (2 *GigE Line Cards*, 1 *NID-PT FPX* card and 1 regular *FPX* card) are put into a *WUGS* as shown in Figure 2.8. Two hosts with *GigE* interface connect two *GigE* cards respectively. 4 *GLink* cards are used for other line connections. A *TCP* flow is set up between the two hosts. By this means, we can monitor all internal interfaces by dumping the cells running through them.

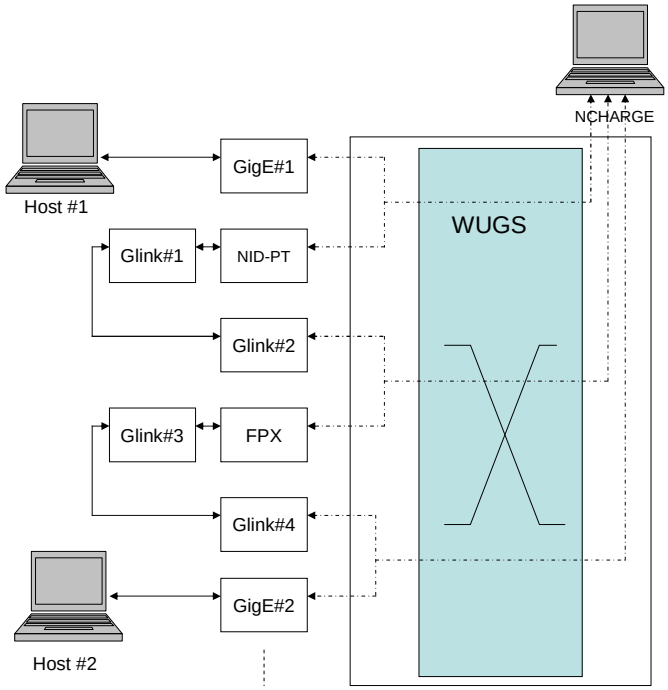

Figure 2.8: FPX System Configuration for Debugging

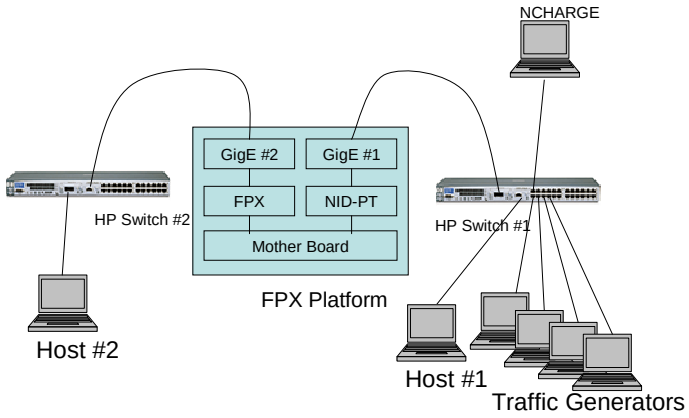

Figure 2.9: FPX System Configuration for Test

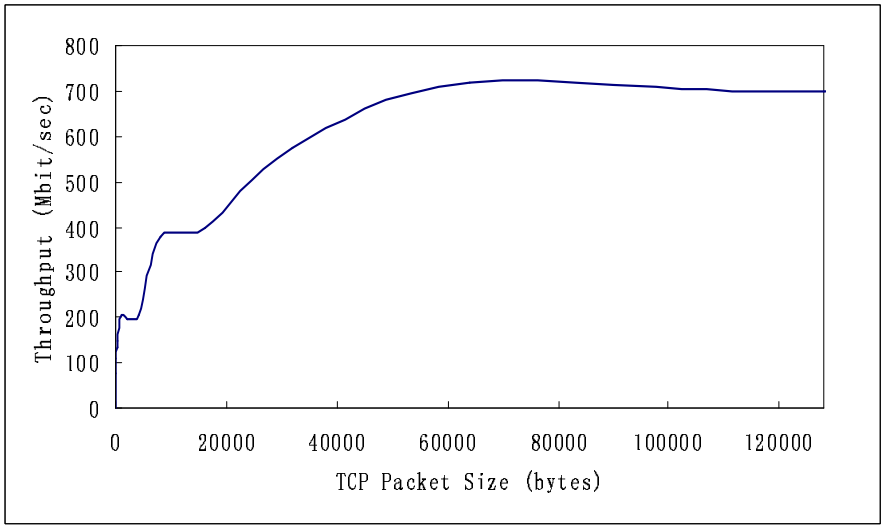

Figure 2.10: A Case of GigE Throughput Test

Once the system passed the test, we set up a standalone *FPX* system without the *WUGS* as shown in Figure 2.9. In this configuration, two *HP Procurve Switch 2524* [43] are used. Each switch has two *GigE* ports and 24 *10/100M Ethernet* ports. On the *Intranet* side, an *NCHARGE* sever is running to control and configure the *FPX* platform. A bunch of hosts can generate and send pass-through traffic to the *Internet* side.

The *host 1*'s *GigE NIC* uses a 64-bit/33-MHz *PCI* bus and *host 2*'s *GigE NIC* uses a 64-bit/66-MHz *PCI* bus. To test the system's throughput, *WSTTCP* [34] is used to transfer data through a *TCP* flow between the two hosts. Though in theory the *PCI* bus can inject data at a rate of at least 2 Gigabit/second, due to the software and operating system's limitations, actual throughput is far below that. Figure 2.10 shows the result of one of the experiments we have done. Note that the system only supports 1500-byte *MTU*.

The experiment shows that with one pair of hosts and one *TCP* flow, we can get at most 800 Mbit/second throughput. In order to make the system work under a

full gigabit rate, a bunch of hosts with *10/100M Ethernet* interfaces should be used to connect with the system through the switch. Each of them sets up a *TCP* connection with the *host 2* so we can easily get an aggregate gigabit throughput.

Chapter 3

Secure Control Packet Processor

To enable secure remote control of a *FPX*-based firewall, a *Secure Control Packet Processor* (*SCPP*) was designed and integrated into the *FPX* system. Some popular encryption and authentication algorithms were implemented in hardware and could be chosen from as the core components in *SCPP*.

3.1 Hardware implementation of Security Standard Cores

Confidentiality is achieved by encrypting the control packets before they are sent and decrypting them in the *FPX* platform. Among many block encryption algorithms, *3DES*[41] and *AES* (*Rijndael*)[31] were implemented because of their efficiency and straightforward hardware implementation.

We also need some form of acknowledgements from the platform to determine the control packet's status. At this point, the authentication of the message is more critical than confidentiality. *HMAC-MD5* and *HMAC-SHA1* were implemented for acknowledgement packet generating.

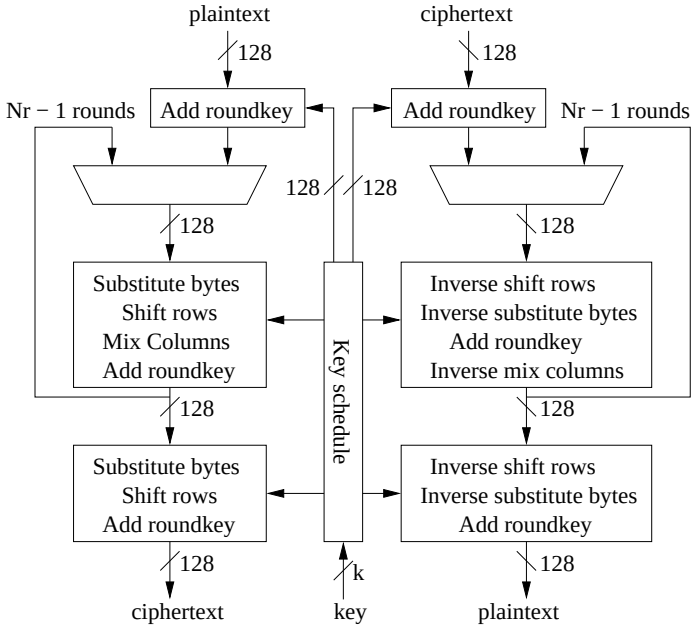

Figure 3.1: AES Algorithm Block Diagram

3.1.1 AES

The *AES* algorithm works by using cryptographic keys of 128, 192 or 256 bits to encrypt 128-bit data blocks. Figure 3.1 describes the AES algorithm. In the figure, k is the number of bits in the key, Nr is the number of iterations (or rounds) performed to complete the encryption or decryption of a single data block. Nr is a function of k and is 10, 12 or 14 for k of 128, 192 or 256 bits, respectively. The key schedule expands the original key to $Nr + 1$ roundkeys. The operations performed in each round are shown in the figure. Note that the last round for encryption does not include the mix columns step, while decryption is missing the inverse mix columns in its last round.

Although we only require *AES* decryption for the control packet processor, both encryption and decryption algorithms were implemented. This will make the module more useful in future applications. Figure 3.2 shows the block diagram of the *AES* implementation. Since the key and the number of rounds are decided at

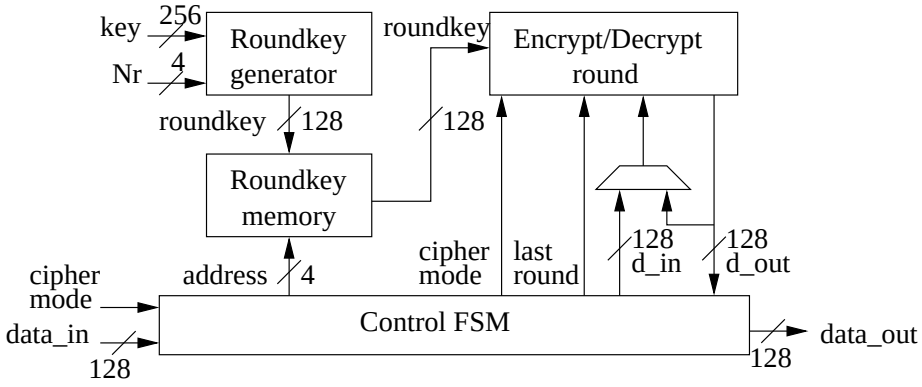

Figure 3.2: AES Hardware Implementation Block Diagram

compile time, a roundkey generator was implemented that computes the roundkeys upon system reset. These keys are stored in the roundkey memory (block *RAM*) and read as needed. Cipher mode controls the encryption and decryption selection.

AES uses 16 256-byte substitution boxes (*Sboxes*) for each encryption round and another 16 256-byte inverse *Sboxes* for each decryption round. If these *Sboxes* were implemented using registers, each round could be completed in one clock cycle. However, this would consume a great deal of our resources. Instead, we chose to implement these *Sboxes* as block *RAMs*. This increased each *AES* round time to two clock cycles but saved much of the chip resources for other modules. This sacrifice can be made without much concern as it is expected that control packets will arrive sparsely. The *EPBC* mode is used for data integrity purpose which requires feedback of the previous plaintext block for decryption or ciphertext block for encryption. This eliminates the possibility of using a loop-unrolling implementation of the encryption algorithm. Also, due to our limited *FPGA* resources, a loop-unrolling implementation makes the design impractical as it would take up most of the *FPGA* leaving no room for other modules.

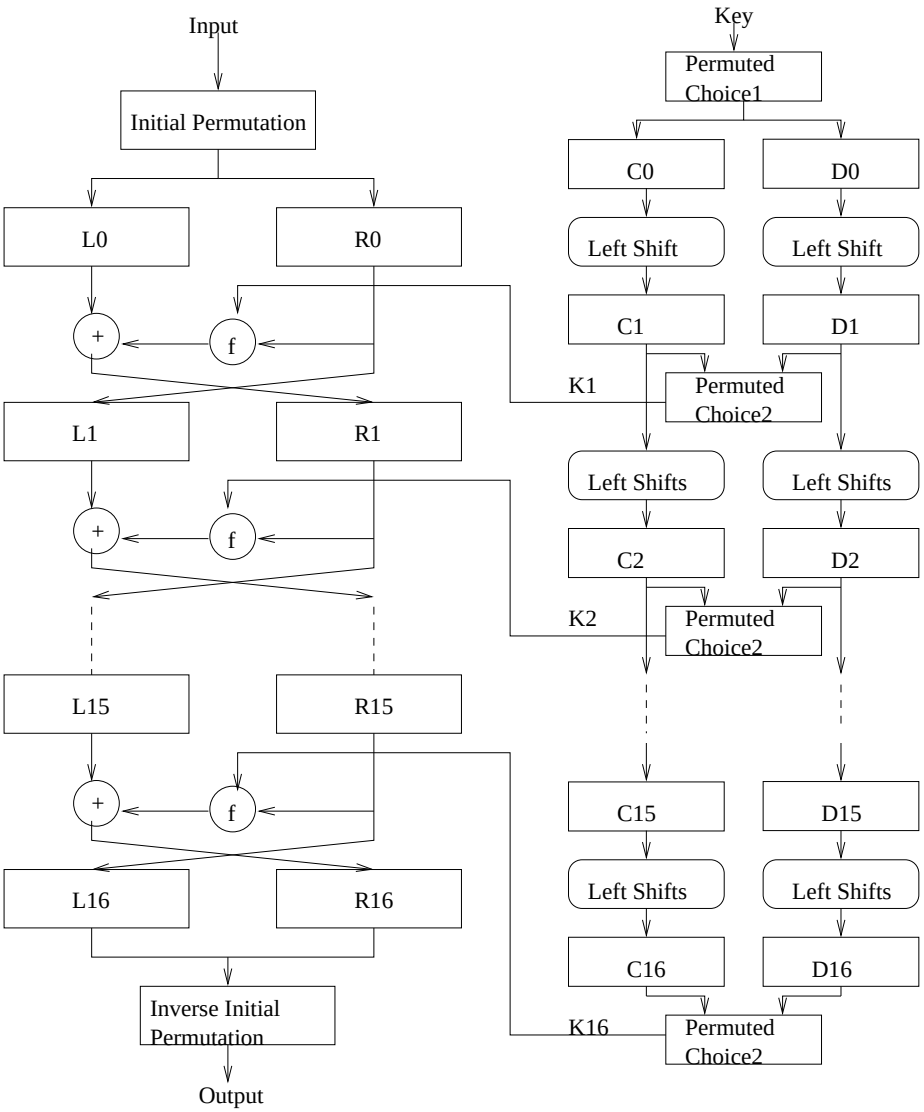

Figure 3.3: DES Implementation Block Diagram

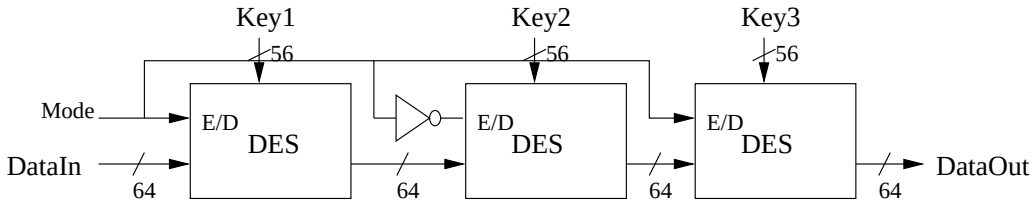

Figure 3.4: 3DES Hardware Implementation Block Diagram

3.1.2 Triple DES

DES is another widely used block encryption algorithm that uses 56-bit keys to encrypt data in 64-bit blocks. Given the vulnerability of *DES* to a brute-force attack, an alternative approach to get higher security is *3DES*. *3DES* uses three encryption stages of *DES* chained together and a unique key for each. The basic *DES* algorithm is described in Figure 3.3. Figure 3.4 shows the *3DES* encryption and decryption structures. A free *DES* core written in *VHDL* was found on the web and used to implement our *3DES*[15]. It contains both iterative and loop-unrolling versions of the algorithm. Again we chose to use the iterative version for the same reasons discussed in the *AES* section.

3.1.3 MD5 and SHA-1

MD5 and *SHA-1* are message digest algorithms specified for use in *Internet Protocol Security* (*IPSec*). Both algorithms take as input a message of arbitrary length and produce as output a message digest of 128 bits for *MD5* and of 160 bits for *SHA-1*. The input message is first padded and appended with message length to be a multiple of 512 bits. Then the message is processed in 512-bit blocks with an n-bit initial value, where n is 128 for *MD5* and 160 for *SHA-1*. Further details regarding *MD5* and *SHA-1* can be found in reference [36] and [13], respectively. The *MD5* and *SHA-1* cores were implemented using iterative architecture and have a latency of 197

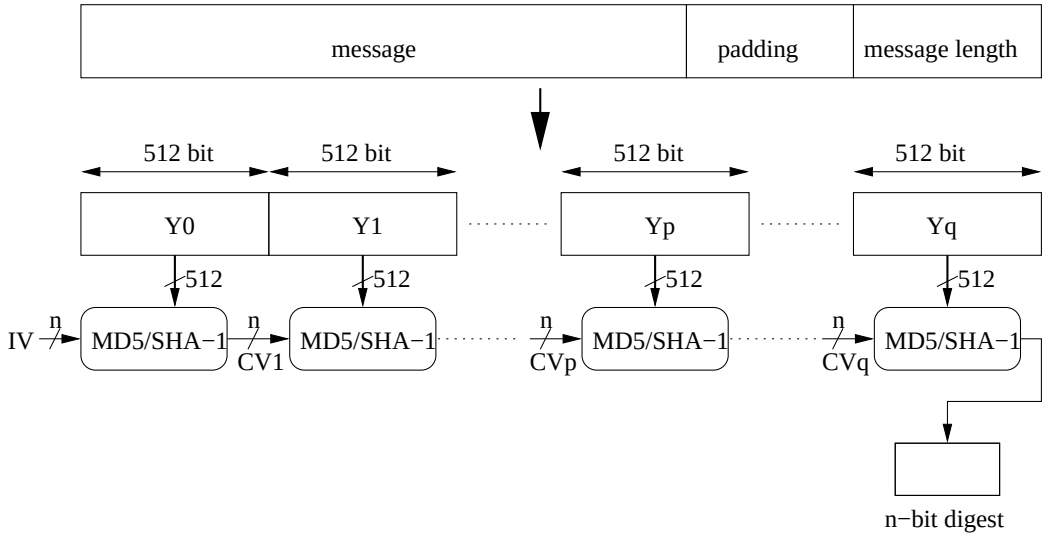

Figure 3.5: MD5 and SHA-1 Algorithm Architecture

and 245 clock cycles respectively to hash a 512-bit data block. These numbers are higher than other implementations of the same algorithms [12]. This is because the four 32-bit additions required by each step of the hashing algorithms were spread out over three clock cycles to increase clock speed. As mentioned earlier, control packets arrive sparsely so latency is not a high concern. The hardware architecture of *MD5* and *SHA-1* is shown in Figure 3.5.

3.1.4 HMAC

HMAC is used in conjunction with either *MD5* or *SHA-1*. It uses a secret key to validate the information being sent from the *SCPP* back to the administrating host. *HMAC* can be further described by the following equation:

$$HMAC_{text} = H(K \oplus opad, H(K \oplus ipad, text))$$

where K is a secret key (we use a 512-bit key), *ipad* is the byte $0x36$ repeated 64

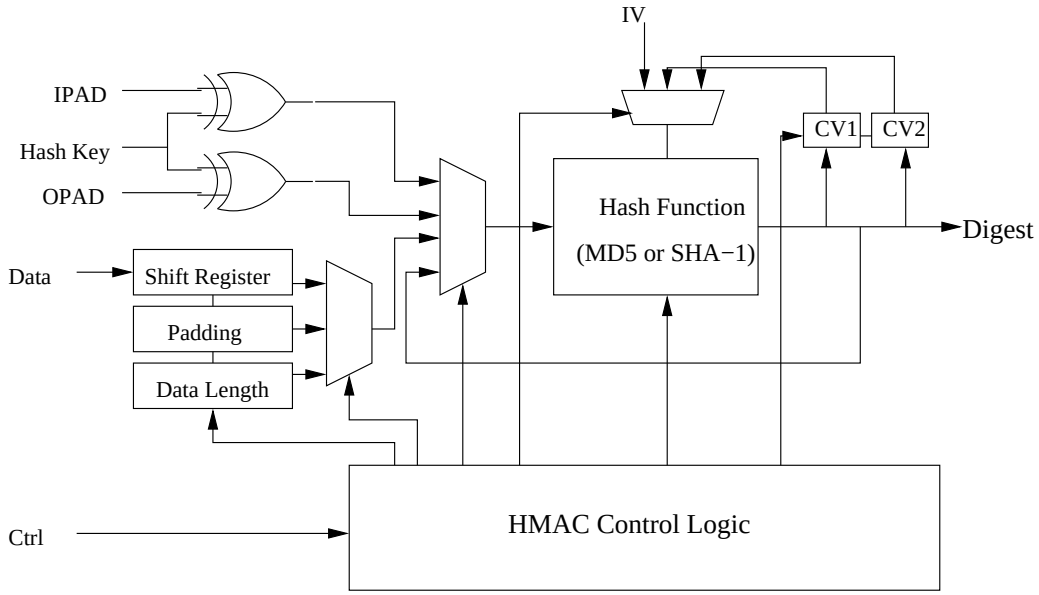

Figure 3.6: HMAC Hardware Implementation Block Diagram

times, *opad* is the byte $0x5C$ repeated 64 times, H is hashing function (*MD5* or *SHA-1*). The result is a 128-bit digest for *MD5* and a 160-bit digest for *SHA-1*. Figure 3.6 shows the hardware implementation of *HMAC*.

We implemented *HMAC* more efficiently by precomputing $H(K \oplus opad)$ and $H(K \oplus ipad)$ [41]. Because the padded message can fit into one block, this technique doubles the throughput of the circuit by eliminating two of the four hashes that need to be computed for each packet.

3.1.5 EPBC mode

To validate the integrity of control packets after they have been decrypted, *AES* and *3DES* were implemented in *Error-Propagating Block Chaining* (*EPBC*) mode. We use this mode instead of *HMAC* here because it is nearly free in regard to circuit overhead compared to the hash function. *EPBC* mode allows us to validate the integrity of the decrypted data by comparing the decrypted value of the last block to the predefined *integrity value*. If they are the same, it is reasonable to believe that

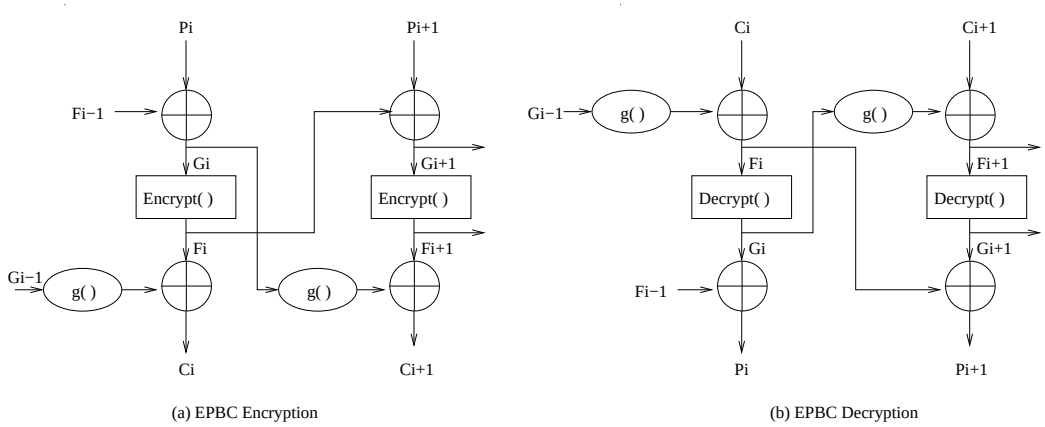

Figure 3.7: EPBC Mode Block Diagram

the control packet has not been tampered with. Otherwise the control packet should be discarded.

As shown in Figure 3.7, *EPBC* mode can be further described by the following equations where the initial values of F and G are two distinct initial vectors (*IVs*). The size of the *IVs* is the same as the block size for each algorithm (128 bits for *AES* and 64 bits for *3DES*). P and C represent the plaintext and ciphertext, respectively. The block number is denoted by i. And finally, E and D represent the encrypt and decrypt functions, respectively.

Encryption:	Decryption:
$G_i = P_i \oplus F_{i-1}$	$F_i = C_i \oplus g(G_{i-1})$
$F_i = E_{key}(G_i)$	$G_i = D_{key}(F_i)$
$C_i = F_i \oplus g(G_{i-1})$	$P_i = G_i \oplus F_{i-1}$

In the above equations, function $g()$ operates as follows:

$g(G) = \langle G_H + \overline{G}_L, G_H \cdot \overline{G}_L \rangle$

Where $G \equiv \langle G_H, G_L \rangle$, G_H and G_L are the high and low order halves of G respectively.

3.2 Secure Configuration and Acknowledgement

3.2.1 Configuration Console

A simple protocol was designed to ensure the reliable *FPX* control and configuration. The control packet is encrypted in the administrating host. For each received control packet, the *FPX* platform sends back a keyed authenticated acknowledgement packet to report the control packet's status (accepted or dropped). In the following cases, administrating host needs to resend the control packet:

- Timeout;
- Unrecognized acknowledgement packet;
- Acknowledgement packet reports the control packet is dropped.

If administrating host can't correctly configure the *FPX* platform for several iterations, it might indicate network failure or imply some attack is happening.

3.2.2 Encrypted Control Packet Format

The same control packet format is applied for both *3DES* and *AES* encryptions for uniform processing. The control packet uses 128-bit *IVs* since *AES* requires this. In the *3DES* application, only the first 64 bits of each *IV* are used for decryption and the remaining 64 bits are ignored. We need to pad the control packet body to make it a multiple of the block size. The padding for the control packet is dependent on the encryption algorithm chosen. The last data block in the control packet is an integrity check. This block uses a predefined value that is checked in the hardware when the packet has been completely decrypted. The layout of the control packet is shown in Figure 3.8a.

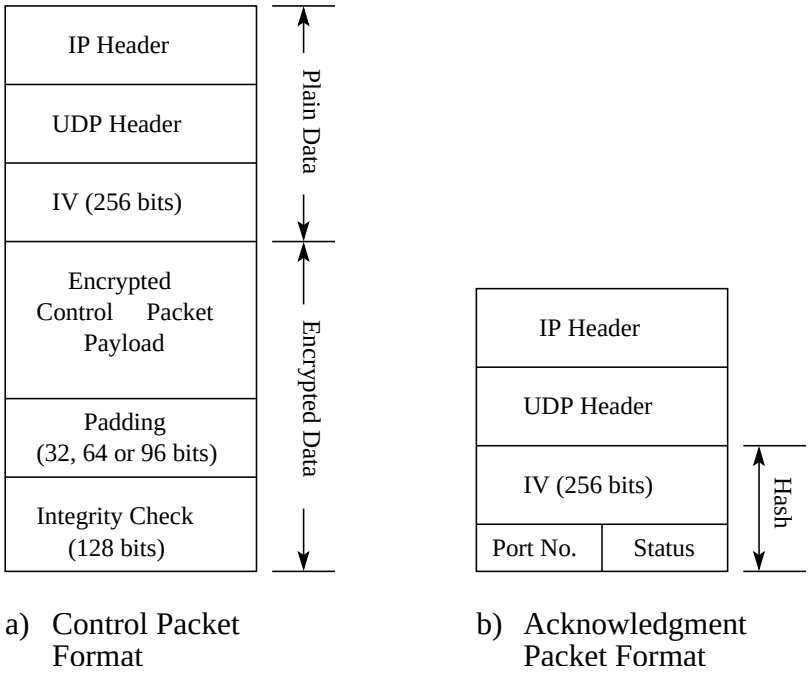

Figure 3.8: Control Packet Formats

3.2.3 Authentication Packet Format

In order to determine if a control packet that was sent to the *FPX* platform was actually accepted, the *FPX* platform should return an acknowledgement packet. This packet contains a hash of the *IVs* concatenated with the 16-bit destination port and a 16-bit status value. Since all *IVs* are randomly generated, each acknowledgement packet can be identified through its *IVs* without providing extra information. The layout of the authentication packet is shown in Figure 3.8b.

3.3 Infrastructure

Simply implementing the encryption and hashing algorithms is not nearly enough to provide security features for the *FPX* platform. Logic was required to identify control packets and pass these packets through *SCPP* while all other packets pass

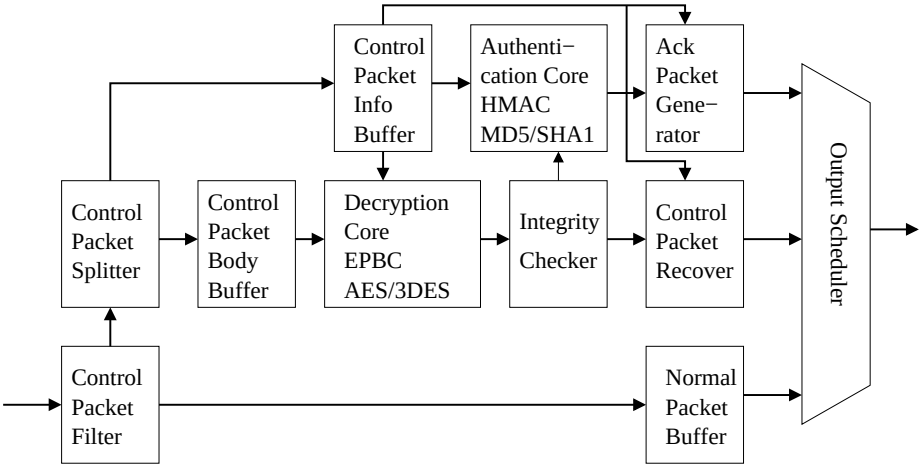

Figure 3.9: Control Packet Processor Architecture

through unmodified. Figure 3.9 illustrates the logic components in the secure control processor. A description of the major components' functionality follows.

Control Packet Filter

This block receives the data from the protocol wrappers and filters out the control packets based on the *FPX* platform's *IP* address and the *UDP* port number. These control packets are sent to the *control packet splitter*. Non-control packets are buffered in the *normal packet buffer* and scheduled to be forwarded to the downstream modules of the *FPX* platform.

Control Packet Splitter

This block further processes the control packets. It extracts the *IP* addresses, *UDP* ports and two *IVs* from the packet and buffers them to be used in the following blocks. Finally, it assembles the encrypted 32-bit words of the packet body into the larger data block necessary for each algorithm (64 bits for *3DES* and 128 bits for *AES*). These data blocks are buffered up to be sent to the *decryption core*.

Control Packet Body and Information Buffer

Due to the decryption latency, *FIFOs* are needed to buffer the data and all necessary information for decrypting, reassembling the control packet and creating authenticated acknowledgement packets.

Decryption Core

The decryption core instantiates either the *AES* or *3DES* core in *EPBC* mode. When the core is not busy, it dequeues a data block from the *control packet body buffer* for decryption. Once a block has been decrypted it is forward to the *integrity checker*.

Integrity Checker

The *integrity checker* is basically a 128-bit comparator. It compares the decrypted value of the last 128-bits of control packets against a predefined integrity check. If any of the cipher blocks was modified prior to decryption, this "error" will propagate to the last decrypted block and it will fail the integrity check. Exact matching means that the packet is the same as the original control packet so it is considered valid. Only valid control packets are forwarded to the downstream modules in the *FPX* platform. Invalid control packets are dropped.

Control Packet Recovery

When a decrypted control packet has successfully passed the integrity check, the recovery component rebuilds the control packet from the decrypted payload using the information previously stored in the *control packet info buffer*.

Authentication Core

The authentication core instantiates either *HMAC-MD5* or *HMAC-SHA1*. For each incoming control packet an authentication digest is computed. The message to be hashed consists of the two *IVs* for the control packet, the destination port, and a 16-bit field indicating the packet status. Once computed, the digest is forwarded to the *acknowledgement packet generator*.

Acknowledgement Packet Generator

For each incoming control packet, an acknowledgement packet is generated. The destination of this packet is simply the source of the corresponding control packet. This information is retrieved from the *control packet info buffer*. A *UDP* packet is then generated using the hash digest as the payload. Once an acknowledgement packet has been generated, the packet generator requests the output bus from the output scheduler.

Normal Packet Buffer

As decrypting control packets is a time consuming process, all other traffic is forwarded around the decryption engine. This buffer receives all normal traffic from the *control packet filter*. Whenever there is an outstanding packet in this buffer it requests the output bus from the output scheduler.

Output Scheduler

This block schedules the output packets in a basic round robin scheme. It services the *control packet recovery*, the *acknowledgement packet generator* and the *normal packet buffer*.

3.4 Results and Analysis

The architecture described has been synthesized using *Synplify Pro* and the *Xilinx* backend place and route tools to implement the design on a *Xilinx Virtex XCV2000E-6 FPGA*. Table 3.1 summarizes the results for the encryption cores, and Table 3.2 summarizes the results for the two authentication cores. The *Rate* column represents the throughput of control packets and acknowledgement packets through the *SCPP*.

From these results we can see that the *3DES* core is much more efficient with regards to resource usage. However, throughput of the *AES* core is several times higher than *3DES*. This is primarily due to two factors. The first is the fact that *AES* decrypts 128-bit blocks at a time, whereas *3DES* only decrypts 64-bit blocks. Secondly, *3DES* uses many more rounds than *AES*. With this in mind, it is clear that it would be more beneficial to chose *3DES* when space is a major concern. Otherwise, if throughput is a major concern, *AES* would be more suitable.

Table 3.3 shows the overall system performance of the *FPX* platform which implements a hardware based firewall with different algorithm combinations. *AES-128-HMAC-SHA1* gives the highest throughput.

Table 3.1: Synthesis Results: Encryption Cores

	LUTs		Block RAM		Timing	Rate
	#	%	#	%	MHz	Mbps
3DES	1223	3%	0	0%	55.2	98
AES-128	2503	5.8%	44	27%	85.6	547.8
AES-192	2610	6.0%	44	27%	86.7	462.4
AES-256	2677	6.2%	44	27%	92	420.6

Table 3.2: Synthesis Results: HMAC Modes

	LUTs		Block RAM		Timing	Rate
mode	#	%	#	%	MHz	Mbps
MD5	5284	13%	0	0%	58.5	152
SHA-1	5796	15%	0	0%	56.5	118

Table 3.3: Placed and Routed Firewall Design

	Slices		Block RAM		Timing	Rate
3DES-HMAC						
config.	#	%	#	%	MHz	Mbps
MD5	13239	68%	83	51%	49.2	1574
SHA1	13698	71%	83	51%	50.5	1616
AES-128-HMAC						
MD5	13543	70%	123	76%	45.8	1466
SHA1	14331	74%	123	76%	60.3	1930
AES-192-HMAC						
MD5	13775	71%	123	76%	48.8	1560
SHA1	14102	73%	123	76%	56.6	1811
AES-256-HMAC						
MD5	13698	71%	123	76%	50.6	1620
SHA1	14221	74%	123	76%	54.9	1758

We also compared the hardware implementations' performance with the software benchmarks found in [11]. The benchmark results are based on the C++ code running on an 850 MHz *Celeron* processor. From Figure 3.10, we find that the performance of *3DES* and *AES* is much better than the performance of the software implementation. For hash functions, software implementation is much faster than its hardware equivalent. It makes sense because hash algorithms were designed mainly for software implementation while most of the symmetric encryption algorithms were optimized for hardware implementation.

3.4.1 Software Implementation

While *3DES*, *EPBC* mode and *HMAC* were original code, *AES*, *MD5* and *SHA1* implementations were obtained via the *National Institute of Standards and Technology* (*NIST*)[29].

On each execution, the program randomly generates *IVs* to be used for encrypting the control packet. The original payload is read from a file, encrypted, and

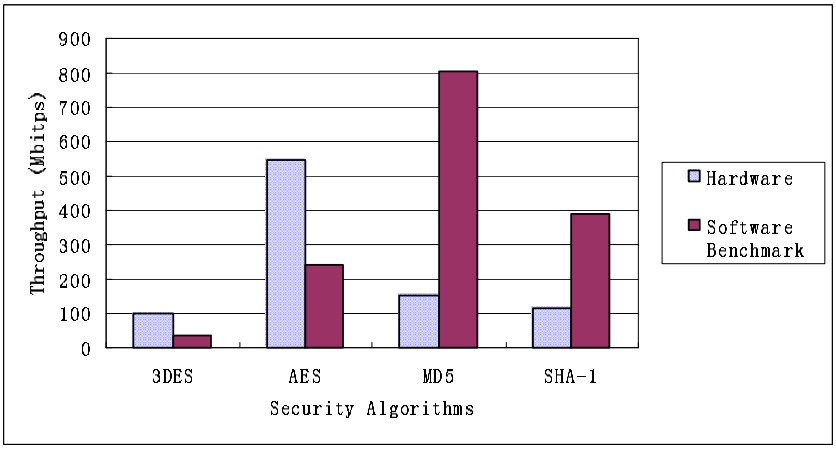

Figure 3.10: Hardware/Software Performance Comparison

appended to the *IVs* along with the integrity check to make up the encrypted payload for the control packet. At the same time, the selected *HMAC* algorithm is run on the *IVs* to create digests for comparison to feedback packets. A digest is created for a feedback packet signifying a successful configuration as well as one signifying a failed configuration (i.e. a dropped control packet).

3.4.2 Testing

Testing of *SCPP* was performed in a live network using the *FPX* platform (Figure 3.11) in a WUGS. A modified version of the *NCHARGE* [40] web interface was used to read our encrypted payloads and create the necessary packet and cell headers. The interface supplies a selection box for choosing both the encryption and the authentication algorithm. It also provides fields for other packet information including IP address, port number and non-encrypted payload data. Once the information for our test packet is entered, the web interface calls a *CGI* script to interpret the information, run it through the encryption software and send the packet to the hardware via a *Gigabit Link* connector between the *PC* and *WUGS*. *NCHARGE* then attempts

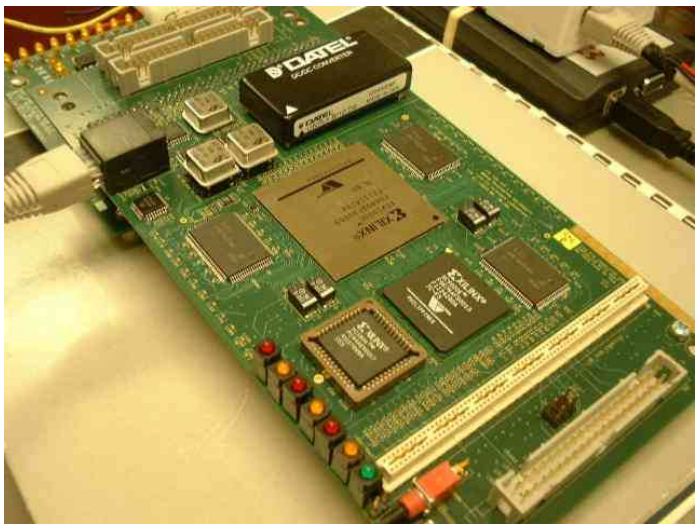

Figure 3.11: FPX Test Platform

to read any acknowledgement packets from the line. Any acknowledgement packet received is displayed and compared to the pre-computed digests. This comparison determines if the control packet was accepted by the *FPX* or not. If not, the control packet would need to be retransmitted until a successful configuration occurred.

AES-128, *AES-192*, *AES-256*, and *3DES* with both *HMAC-MD5* and *HMAC-SHA1* (for a total of eight different core modes) were all tested. Each core passed the tests by successfully configuring the *FPX* platform and returning expected acknowledgement packets. We also simulated different attacks to the *FPX* platform, such as fake control packets, message modification and denial of service. All attacks were detected by the *FPX* platform or the administrating host successfully.

Chapter 4

Secure Remote Control Protocol

4.1 Architecture

Some drawbacks exist in the *SCPP* structure. First, it is subject to a replay attack. That means an attacker can capture a control packet and send it to the target again at any time. The *FPX* cannot identify if it is a replay packet. Second, the communication protocol is stop-and-wait. It is too slow if we want to download a large bitfile to the *FPX* platform. In order to overcome these drawbacks, an improved protocol is presented. It can also be used in any environment in which we need to control a hardware based reconfigurable network node through the public Internet.

The secure configuration architecture provides the infrastructure for secure communications between the control console and *Reconfigurable Hardware Nodes* (*RHN*) over the Internet. Communications are conducted through encrypted and authenticated control packets. A functional module called the *Secure Control Manager* (*SCM*) performs all security related tasks, ensuring that only authorized access to the *RHN* is allowed. Figure 4.1 shows the security configuration architecture. A unique aspect of this architecture is that *SCM* is implemented in pure hardware.

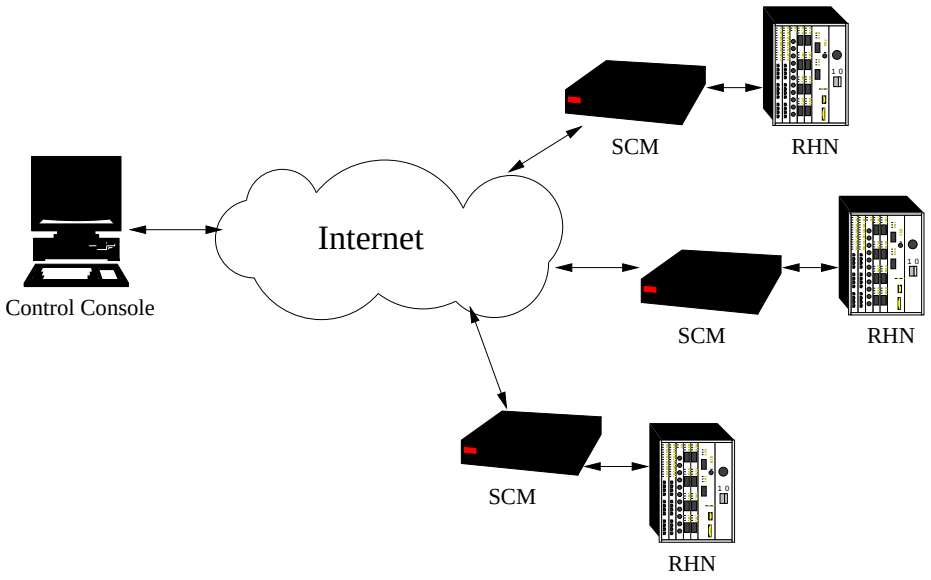

Figure 4.1: Secure Configuration Architecture

Avoiding software involvement in network processing is important because it prevents the nodes from being hacked using traditional vulnerabilities found in operating systems. *SCM* implements some encryption and authentication algorithms to guarantee data confidentiality and integrity. It is also responsible for establishment, maintenance and termination of the secure connection.

4.2 Protocol

A security protocol is needed to secure the communications between the control console and *RHNs*. The goal of this protocol is to provide a secure communication channel, so that attackers will not be able to damage or steal information from the *RHNs*. Even if the node is under attack or experiencing network failure, the protocol can let the control console be aware of the attack and take actions. The protocol includes a flow control and an error control scheme. It relies on positive acknowledgement and retransmission when the control console does not receive an acknowledgement

within a given timeout period. The protocol need to be implemented not only on the administration host in software, but also on the *SCM* in pure hardware which is hard to accommodate a complicated protocol. Fortunately, in most cases, the control console and the *RHN* act as master and slave. Control console is always take the initiative to establishment and termination of connections. This feature enables us to design a simple yet efficient protocol based on *Go-back-N ARQ* to support only end-to-end communication, with which at the same time configuration packets are issued only from one control console.

A sliding window mechanism is used to provide flow control. Similar to the mechanism in *HDLC*, an acknowledgement (*ACK*) of new incoming packets automatically moves the sliding window to grant permission to more packets. In our system, the size of the sliding window is decided mainly by the buffer size in the *SCM*. Error control is done by exerting encryption and authentication on each control packet. Retransmission of a packet is necessitated when a packet never arrives at the reconfigurable hardware node or an arriving packet is discarded by the *RHN* because of errors. Here we implement a batch retransmission strategy, particularly because of the requirement that the packets configure or reprogram the hardware nodes in an in-order fashion. It works by maintaining one retransmission timer for the entire sliding window. If no *ACK* is received before the timer expires, the control console must retransmit from the first packet and reset the timer; otherwise, the sliding window is adjusted and the timer is reset. A sequence number indicating the order of control packet is encrypted in the packet body. For each establishment of a new connection, a random sequence number is generated in the *RHN* and sent back to the control console. The sequence number is incremented during the maintenance of the connection.

Handling connection termination can be complicated in some protocols. However since only end-to-end communication is supported and the *SCM* does not preserve any information for the current connection except for the sequence number, the termination of the current connection is only meaningful for the control console. In this protocol, connection termination to the *RHN* is simply establishment of a new connection. As to the control console, an initiative termination is stopping sending new control packet after the *ACK* of the last control packet; and a passive termination is unsuccessful communication with a timeout period. In the second case, the control console will be aware of either the network failure or potential attacks to the system.

During the above discussion, we assume that before each connection the control console and the *RHN* shares a secret key, which is used to protect all transferred data. Security key management is extremely difficult for pure hardware implementation. Therefore, in our current protocol, we assume the secret key is hardcoded in the *RHN* and shared with each control console.

This protocol is strong enough to protect the configurable hardware nodes from the common attacks, which we talked about in the section one. First, eavesdropping and fake packet attacks are avoided by applying encryption and authentication to the control packets. Second, replay attack is prevented by using encrypted sequence number. By randomly generating sequence number *SN* from the control console side, even if malicious users capture all the packets sending from the host, re-sending them to the *RHN* will not be accepted due to the discontinuity of the sequence number. The administration host has a time-out mechanism, therefore, when *RHN* is under attack or network fails, the host can be aware of these problems and take action.

The communication between the administration host and the *RHN* is through *UDP* packets. We call it control packet. For uniform processing, all control packets

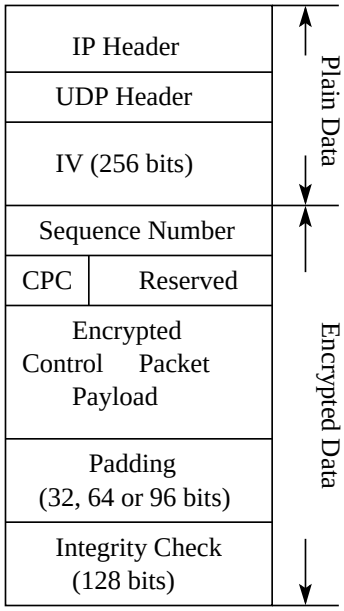

Figure 4.2: Secure Control Packet Format

have the same format. Control packets include session establishment packet, configuration packet, status inquiry packet, and acknowledgement packet. The destination IP address and *UDP* port number are used to identify the individual configurable module inside the *RHN*.

The first 4 32-bit words in the *UDP* payload is *IVs* required by *EPBC* mode. The next word is 32-bit *SN* for flow control. The first byte of the following word denote a unique *Control Packet Code* (*CPC*) for each type of the control packet. The administration host and the *RHN* can distinguish and process the packet by its *CPC*. All *RHN* control or configuration information follows. Control packet payload is padded to be multiple of the block size. The last 128-bit data block in the control packet is integrity check. This block uses a predefined value that is checked in the hardware when the packet has been completely decrypted. The layout of the control packet can be seen in Figure 4.2.

Communication is divided into two phases: the session establishment phase and configuration phase. To establish a communication session, the host sends a control packet with *CPC* equal to 0 during session establishment phase. Once the *RHN* receives and accepts the connection requirement, an acknowledgement control packet is sent back with *CPC* equal to 0. A randomly generated *SN* telling the administration host to start sending packets from the appointed *SN* is also encrypted in the acknowledgement packet body. If the administration host doesn't receive the *ACK* within a timeout period, it sends the initialization control packet again. If this fails after a predefined number of times, the administration host will know that it is experiencing network failure or attacks. If the administration host gets the *ACK* successfully, the communication goes into the configuration phase.

The administration host sends configuration packet ($CPC = 1$) or status request packet ($CPC = 2$) with successive *SN* within the bound of the current sliding window. The *RHN* receives a error-free packet, it sends back an *ACK* ($CPC = 1$) to tell the host the next expected *SN N'*. The effect of this *ACK* actually acknowledges all packets with *SN* prior to this one *N'*. If the *SN* of this error-free packet matches the expected *SN*, *RHN* will forward this packet to the rest of the *RHN* modules. If due to any error the *RHN* can not accept the current control packet, it silently discard the packet and whatever the following control packets till an acceptable one with the expected *SN*. If the host experiences a timeout without receiving any *ACK*, it will resend all the packets in the current window. On condition that *RHN* accepts a status inquiry packet, it sends back an *ACK* with the inquired information assembled in the packet body and *CPC* set to 2. This indicates it is not only an acknowledgement, but also a reply to an inquiry.

4.3 Application in FPX system

Considering the reality of the *FPX* system, we decided to apply the encryption in the *NID-PT* card and leave the data integrity and sequence number control logic to be processed in the regular *FPX* card. *AES* is exerted only to the control cell payload. Firstly, this arrangement conforms to the principle of layered protocol. Mapped into the seven layer network model, encryption functions could be in physical layer while data integrity and authentication check functions could be in link layer. Secondly, current *FPX* system and related software are all well defined and work well. The control cell's format inherited the data integrity check by *AAL0 CRC* and sequence number control by *SN* field. What's more, the length of the cell payload is 48 bytes, that are exactly 3 128-bit blocks for *AES* cipher. There is no important information that needs to hide in cell header so we don't need to encrypt it at all. All these considerations simplified the modifications and new designs that need to be added to the *FPX* system. The logic view of the *NID-PT FPX* card is shown in Figure 4.3

The *NID FPGA* on the *NID-PT* card has consumed more than 75% of the resource. The encryption and decryption module have to be placed in the *RAD FPGA*. Only control cells need to be routed to the *RAD*. Two modules shown in Figure 4.4 are used in *RAD* to process bidirectional control cells.

The incoming cell is splitted into two parts: the header and the payload. The cell header is stored in a *FIFO* and the cell payload is transformed to 3 128-bit data blocks to be buffered in another *FIFO*. An *AES* core retrieves the data block from the payload buffer to encrypt or decrypted it. After each 3 consecutive data blocks are processed, one cell header is read out from the header buffer and a whole *ATM* cell is reassembled and sent back to *NID-PT*. A back pressure signal is used to control the flow rate.

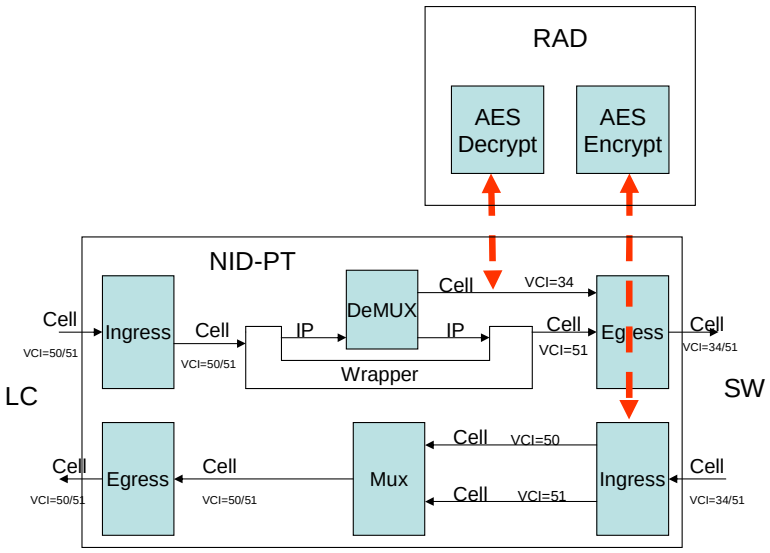

Figure 4.3: NID-PT FPX overview

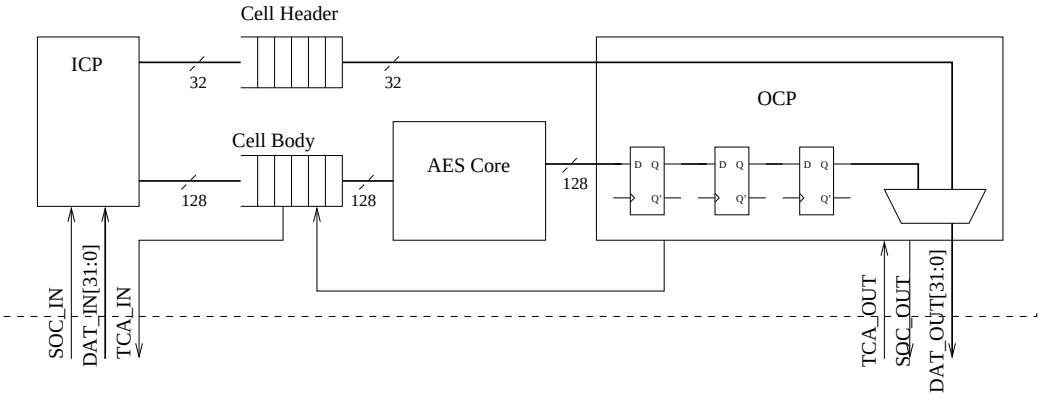

Figure 4.4: RAD AES Module Block Diagram

By restricting the key size to 128 bits, *AES* core need at least 11 clock cycles to process one 128-bit data block. The system will work under 62.5 MHz frequency, so roughly we can get the control cell throughput up to 727 Mbps.

Chapter 5

Conclusion

5.1 Remarks

The *FPX* platform can be applied in many network processing systems such as firewalls, packet classifiers, content scanners and copyright protection units.

A *GigE Card FPGA* was designed to provide a line interface for the *FPX* platform. The *FPGA* implements *ARP* and transforms packets between *ATM* and *Gigabit Ethernet*. The card is fully tested and can support 1 Gbit/sec throughput.

The *GigE* card *FPGA* implements *ARP*, but it is not fully compliant with the standard in that the *ARP* table does not support entry aging. That feature is expensive to implement in hardware. We can implement it in software by periodically polling the table entries. However, in the current *FPX* platform applications, this is not a critical requirement.

For historical reason, the basic unit processed in *FPX* is an *ATM* cell. It is not necessary in some of the *FPX* applications and it does introduce extra overhead. The evolving system will get rid of this step by step.

We also made efforts to solve the security problem when configuring the *FPX* platform through the public Internet. A suite of security algorithms were implemented in hardware and they were applied in *SCPP* to provide baseline protection to the control and configuration channel of the *FPX* platform.

Though the *SCPP* architecture provides strong protection to the *FPX* platform configuration channel, it is cumbersome and not very efficiency. The main achievement is we implemented a suite of hardware cores and verified them in real applications. They are the base blocks to build the future hardware based network security applications. Another problem is the *SCCP* can lack of defense against the replay attacks, so we proposed a more robust and more general protocol which intends to secure the control and configuration channel between central console and the hardware based network nodes. A practical module was designed to support this protocol in the *FPX* platform.

5.2 Future Work

As the network bandwidth continues growing explosively, the *FPX* platform may be applied in different and higher bandwidth environments, such as *SONET OC-192* and *10 Gigabit Ethernet*. More types of *Network Interface Card* (*NIC*) will be required. Still, logic are needed to provide the bridge between the line side and the system side. For robust application and fault tolerance, the system should support *Automatical Protection Switch* (*APS*). A backup *FPX* platform can provide protection when network failure occurs without interrupting the whole system.

For secure control and configuration, if we consider the scalability issue, it is actually a secure and reliable multicast problem. There already exist a lot of protocols and algorithms that try to solve this problem but no one is widely accepted

and deployed. This is still an open area for future research. The multicast model is source-specified, the problems include many aspects such as source authentication, access control, key distribution and network topology. The main constraint in our system is the limited hardware resource.

References

[1] Paulo Guedes Andre Zuquete. Efficient Error-Propagating Block Chaining. In *IMA International Conference*, pages 323–334, San Antonio, TX, USA, December 1997.

[2] WUSTL Applied Research Lab. MSR - A Scalable, High Performance MultiService Router Architecture. http://www.arl.wustl.edu/arl/projects/msr, 2002.

[3] WUSTL Applied Research Lab. The Field-programmable Port Extender. http://www.arl.wustl.edu/arl/projects/fpx, 2002.

[4] Saleem N. Bhatti, Graham Knight, D. Gurle, and P. Rodier. Secure remote management. In *Integrated Network Management*, pages 156–169, 1995.

[5] Florian Braun, John Lockwood, and Marcel Waldvogel. Reconfigurable Router Modules Using Network Protocol Wrappers. In *Proceedings of Field-Programmable Logic and Applications*, Belfast, Northern Ireland, August 2001.

[6] Florian Braun, John W. Lockwood, and Marcel Waldvogel. Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware. Technical Report WU-CS-01-10, Washington University in Saint Louis, Department of Computer Science, June 2001.

[7] Rajiv Chakravorty and Hans Ottevanger. Architecture and implementation of a remote management framework for dynamically reconfigurable devices. In *10th IEEE International Conference on Networks (IEEE ICON 2002)*, August 2002.

[8] Tom Chaney, J. Andrew Fingerhut, Margaret Flucke, and Jonathan S. Turner. Design of a Gigabit ATM Switch. Technical Report WU-CS-96-07, Washington University in Saint Louis, 1996.

[9] Pawel Chodowiec, Kris Gaj, Peter Bellows, and Brian Schott. Experimental testing of the Gigabit IPSec-compliant implementations of Rijndael and Triple DES using SLAAC-1V FPGA accelerator board. *Lecture Notes in Computer Science*, 2200, 2001.

[10] Sumi Choi, Jonathan S. Turner, and Tilman Wolf. Configuring sessions in programmable networks. In *INFOCOM*, 2001.

[11] Wei Dai. Crypto++ 4.0 benchmarks. Online:`http://www.eskimo.com/ weidai/benchmarks.html`, 2000.

[12] Janaka Deepakumara, Howard M. Heys, and R. Venkatesan. FPGA Implementation of MD5 Hash Algorithm. In *Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2001.

[13] FIPSP-180-1. Secure Hash Standard, April 1995.

[14] Viktor Fischer. Realization of the round 2 aes candidates using altera fpga, April 2000.

[15] free ip.com. Free-DES: Advanced Encryption Standard . Online:`http://www.free-ip.com/DES/`, March 2000.

[16] Kris Gaj and Pawel Chodowiec. Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays. *Lecture Notes in Computer Science*, 2020, 2001.

[17] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott. Comparative analysis of the hardware implementations of hash functions sha-1 and sha-512. In *Proc. Information Security Conference*, Sao Paulo, Brazil, September 2002.

[18] SATURN Development Group. Pmc-980495 saturn compatible interface for packet over sonet physical layer and link layer devices (level 3), 1998.

[19] IANA. Ethernet address assignment. http://www.iana.org/assignments/ethernet-numbers, 2001.

[20] IEEE. IEEE Std 802.1q-1998: Virtual Bridged Local Area Network.

[21] Xilinx Inc. Product Data Sheets. http://www.support.xilinx.com/partinfo/databook.htm.

[22] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Online `http://www.faqs.org/rfcs/rfc2104.html`, February 1997.

[23] John W Lockwood. Evolvable internet hardware platforms. In *The Third NASA/DoD Workshop on Evolvable Hardware (EH'2001)*, pages 271–279, July 2001.

[24] John W. Lockwood, Naji Naufel, Jon S. Turner, and David E. Taylor. Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX). In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, pages 87–93, Monterey, CA, USA, February 2001.

[25] John W. Lockwood, Jon S. Turner, and David E. Taylor. Field Programmable Port Extender (FPX) for Distributed Routing and Queuing. In *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, pages 137–144, Monterey, CA, USA, February 2000.

[26] John W. Lockwood, Christopher Zuver, Christopher Neely, James Moscola, and Sarang Dharmapurikar. An Extensible System-On-Chip Internet Firewall. Submitted to Design Automation Conference (DAC) 2003, December 2002.

[27] Forn Soriano Mels. Hardware implementation of a secure bridge in ethernet environments. In *Globecom'93*, 1993.

[28] James Moscola, John Lockwood, Ronald P. Loui, and Michael Pachos. Implementation of a content-scanning module for an internet firewall. In *FCCM03*, 2003.

[29] National Institure of Standards and Technology . Online:`http://www.nist.gov`, July 2000.

[30] NIST. Fips pub 46-3: Data encryption standard (des). http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, 1999.

[31] NIST. FIPS-197: Advanced Encryption Standard (AES) . Online:`http://csrc.nist.gov/encryption/aes/`, February 2001.

[32] NIST. Modes of operation for symmetric key block ciphers. http://csrc.nist.gov/CryptoToolkit/modes, 2001.

[33] School of Computer Science and Electrical Engineering. Vhdl ip stack. Online:`http://www.itee.uq.edu.au/ peters/xsvboard/stack/stack.pdf`, 2001.

[34] PCAUSA. Test tcp (ttcp) benchmarking tool for measuring tcp and udp performance. http://www.pcausa.com/Utilities/pcattcp.htm, 2003.

[35] David C. Plummer. RFC 826: An Ethernet Address Resolution Protocol, November 1982.

[36] R. Rivest. The MD5 Message-Digest Algorithm, RFC 1321. MIT LCS and RSA Data Security, Inc., April 1992.

[37] Gultchev S, Mitchell C, Moessner K, and Tafazolli R. Securing reconfigurable terminals – mechanisms and protocols. In *13th International Symposium on Personal, Indoor and Mobile Radio Communication (PIMRC'02)*, September 2002.

[38] P. Schaumont, H. Kuo, and I. Verbauwhede. Unlocking the Design Secrets of a 2.29 Gb/s Rijndael Processor. In *DAC 2002*, June 2002.

[39] Rich Seifert. *Gigabit Ethernet:technology and applications for high-speed LANs.* Addison-Wesley, April 1998.

[40] Todd Sproull, John W. Lockwood, and David E. Taylor. Control and Configuration Software for a Reconfigurable Networking Hardware Platform. In *submitted to Globecom 2001*, San Antonio, TX, USA, November 2001.

[41] William Stallings. *Cryptography and Network Security: Principles and Practices, 3rd edition.* Prentice Hall, 2003.

[42] David E. Taylor, Jonathan S. Turner, John W. Lockwood, and Edson L. Horta. Dynamic hardware plugins (DHP): Exploiting reconfigurable hardware for high-performance programmable routers, computer networks. *Computer Networks*, 38, February 2002.

[43] www.hp.com. HP Procurve Series 2500 Switches Management and Configuration Guide.

[44] www.pmc sierra.com. Product Details for PM3386 (S/UNI-2xGE) Dual Gigabit Ethernet Controller.

[45] www.xilinx.com. Virtex-II Platform FPGAs Advance Product Specification.