

國立交通大學

資訊科學與工程研究所

碩士論文

一個高速端對端訊務偵測與防範系統之設計與雛型
製作

Design and Prototyping of a High Performance P2P Traffic Detection
and Filtering System

研究生：蔡宗勳

指導教授：陳耀宗 教授

中華民國 九十五年 八月

一個高速端對端訊務偵測與防範系統之設計與雛型製作
Design and Prototyping of a High Performance P2P Traffic
Detection and Filtering System

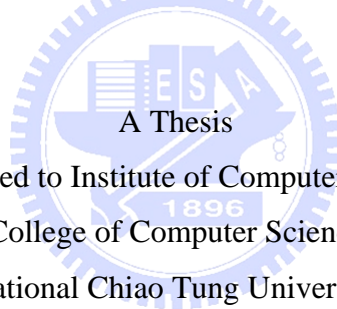
研究生：蔡宗勳

Student : Tzong-Shiun Tsai

指導教授：陳耀宗

Advisor : Yaw-Chung Chen

國立交通大學
資訊科學與工程研究所
碩士論文



A Thesis
Submitted to Institute of Computer Science
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science

August 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年八月

一個高速端對端訊務偵測與防範系統之設計與雛型製作

學生：蔡宗勳

指導教授：陳耀宗 博士

國立交通大學資訊工程與科學研究所

摘 要

隨著網際網路的普及，接踵而來的是各種多元而性質不同的服務應用，例如，全球資訊網(World Wide Web)、檔案傳輸協定(File Transfer Protocol)、電子郵件(E-Mail)、網路電話(Voice over IP) 等等。而在最近十年裡，發展了另一種的網路服務行為：同儕網路(Peer-to-Peer Network)，簡稱為 P2P 網路。

P2P 網路日益普及，已經佔了一般企業與學術網路流量的百分之八十以上，而如此高的使用率，在有限的網路頻寬下，容易將一般網路資源耗盡；也因為 P2P 網路採用端對端的架構，沒有像傳統用戶伺服器(Client-Server)模式容易將連線紀錄保存下來，所以一旦有非法使用者將企業或私人私密資料藉由 P2P 網路洩漏給第三者，也不易由傳統檢查紀錄檔的方式查詢；也因此，除了公司企業的洩密危機外，常常會存在著網路間流傳非授權軟體或檔案的威脅。基於上述原因，一般的 ISP 業者、企業主管機關與學術研究相關單位皆為此而困擾，所以對 P2P 網路的偵測與防範就有其必要性。

本篇論文針對 P2P 訊務發展一套有效的偵測與防範系統，著重如何正確的偵測出 P2P 網路的封包，將其加以攔截。我們對每一個進入系統封包的資料部份進行檢查是否含有代表 P2P 訊務的特徵字串，來做為判斷的依據。如判斷為 P2P 訊務，則系統再將以攔截。我們利用管線的概念來設計系統並實作於 Intel 的網路處理器上，且利用了 IXP2400 的硬體功能加強檢查及過濾封包的速度，以達成高效能的目的。

在本篇論文中，我們介紹 P2P 訊務的特性，以及系統用來判別 P2P 訊務的詳細演算法，最後經由 Intel 所提供的模擬器實際模擬後，我們的系統在平均來說每秒可以處理數百萬個封包，比市場上之最先進產品有更好之價格性能比，在文中我們也描述了模擬

的結果。



Design and Prototyping of a High Performance P2P Traffic

Detection and Filtering System

Student : Tzong-Shiun Tsai

Advisor : Dr. Yaw-Chung Chen

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

With Internet technology becoming more and more popular, various network applications have been proposed, for examples, World Wide Web (WWW), File Transfer Protocol (FTP), E-Mail, and Voice over IP (VoIP). In the last decade, another network service has been developed, that is Peer-to-Peer (P2P) Network.

While P2P network spreading rapidly, P2P traffic has occupied up to 80% of corporations and schools' network traffic. Under limited network resources, such high occupancy consumes most of network bandwidth. Since to there is no log file to keep connection record with the P2P network. Such network behavior leads to blabbing personal secrets illegally and spread files without authorization. This problems lead to the motivation of this investigation to detect and filter P2P traffic.

In this thesis, we develop an effective detecting and filtering system for P2P traffic. We check whether an incoming packet payload containing P2P traffic signature or not. Once identifying a P2P packet, our system drops the packet. We also take advantage of content pipeline programming concept and special-purpose network processor hardware to build up a high-performance system.

According to simulation results, it shows that our system can handle several million packets per second, which is better than state-of-the art commercial products; further our

implementation is more cost effective than those products.

Keyword: P2P detection and filtering, content pipeline, network processor.

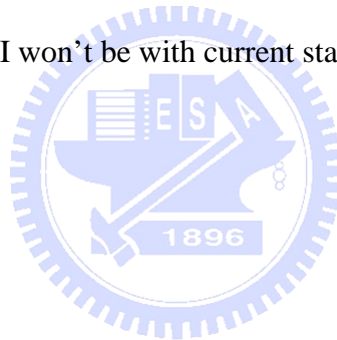


Acknowledgement

First of all, I would like to express my sincerity to Prof. Yaw-Chung Chen for providing valuable suggestions and ideas during my graduate study, and guiding me with deep technical sight. During the preparation of this thesis, he also provides me excellent research facilities.

Besides, I appreciate very much members of Multimedia Communications Laboratory in Computer Science Department, including REX, Onlyjack, Chunyung, Dinlie, Ashraf Milhim, Alexander, Elson, and Wei-Min, especially to Wei-Min for their cheers so that I have had a great time in my graduate school life.

Finally, I would like to express my appreciation to my family and my beloved girlfriend, Ariel, without their encouraging, I won't be with current status.



Contents

摘要	i
Abstract.....	iii
Acknowledgement.....	v
Contents.....	vi
Table List.....	viii
Figure List	ix
Chapter 1 Introduction.....	1
1.1 Overview	1
1.2 Motivation and Purpose.....	2
1.3 Organization of the Thesis.....	4
Chapter 2 Related Works.....	5
2.1 Peer-to-Peer Network	5
2.1.1 Introduction and Definition of Peer-to-Peer Network.....	5
2.1.2 How Peer-to-Peer Network Works.....	7
2.1.3 The Gnutella Protocol.....	9
2.1.4 The BitTorrent Protocol	10
2.1.5 Peer-to-Peer Network Properties.....	13
2.1.6 Peer-to-Peer Network in the Present Day.....	14
2.2 Current Peer-to-Peer Protocol Detection Methodologies.....	16
2.2.1 Port number based Peer-to-Peer Detection Method	17
2.2.2 Signature-based Peer-to-Peer Detection Method	17
2.3 Network Processor Overview	20
2.3.1 What Is Network Processors	21
Chapter 3 Proposed Approaches	26
3.1 Design Principles.....	26
3.1.1 Transparent Property	27
3.1.2 High Performance Property	28
3.1.3 Flexibility Property.....	28
3.2 Limitations.....	29
3.3 Design Patterns.....	31
3.4 Software Viewpoint.....	32
3.4.1 The Traditional Programming Model.....	32
3.4.2 The Content Pipeline Programming Model.....	33
3.4.3 The Proposed System Model.....	34
3.5 Hardware Viewpoint	36
3.5.1 General PC Review	37

3.5.2 Why Network Processor.....	38
3.6 Integrated Hardware with Software.....	39
Chapter 4 Proposed Schemes and Implementation	40
4.1 Overview of System Architecture	40
4.2 Introduction of System Basic Configuration.....	42
4.3 P2P Protocol Detecting and Filtering Algorithm	44
4.4 Signature Table.....	48
4.4.1 Signature Table Implementation	50
4.4.2 Dynamically Configured Signature Table.....	52
5.1 Introduction of Simulator	54
5.2 Simulation Results.....	55
5.3 Comparison and Discussion	61
Reference:.....	67



Table List

Table 2.1 Signatures of common P2P applications	20
Table 2.2 differences between different memory types in IXP2400.....	25



Figure List

Figure 2.1 Peers can communicate without considering underlying network infrastructures....	7
Figure 2.2 The BitTorrent Protocol network architecture.	11
Figure 2.3 P2P traffic flow during 9/25/2001 to 9/25/2004.	14
Figure 2.4 Network traffic distributions from 1993 to 2004 by CacheLogic company.	15
Figure 2.5 P2P market shared by country.....	16
Figure 2.6 Intel IXP2400 network processor chip.....	21
Figure 2.7 Intel IXDP2400 system components deployment architecture.....	23
Figure 2.8 Intel IXP2400 chip hardware architecture.	23
Figure 2.9 Intel microengine chip hardware architecture.....	24
Figure 3.1 P2P filter system network topology.	28
Figure 3.2 The work flow of traditional programming model.	33
Figure 3.3 Work flow of content pipeline programming model.....	34
Figure 3.4 P2P filter system model.	35
Figure 3.5 P2P filter system with content pipeline and multi-threading.	36
Figure 4.1 P2P filter system procedure.	41
Figure 4.2 Three basic functionalities of network application.	42
Figure 4.3 Packets queue array relationship in IXP2400.	43
Figure 4.4 IP datagram.	44
Figure 4.5 TCP datagram.	44
Figure 4.6 P2P detection and filtering flow chart.....	48
Figure 4.7 Signature table entry format.....	50
Figure 4.8 A signature table example.....	52
Figure 5.1 Processing cycles per packet with 5 P2P signatures.	56
Figure 5.2 Number of mega packets per second with 5 P2P signatures.....	57
Figure 5.3 Processing cycles per packet with 60 P2P signatures in average case.....	58
Figure 5.4 Number of mega packets per second with 60 P2P signatures in average case.	59
Figure 5.5 Processing cycles per packet with 60 P2P signatures in worst case.	60
Figure 5.6 Number of mega packets per second with 60 P2P signatures in worst case.....	61

Chapter 1 Introduction

1.1 Overview

Since computers were developed in 1940s, there were often related new technologies being invented. As time passed, many service models are also developed to provide users more selections of services, for examples, World Wide Web (WWW), File Transfer Protocol (FTP), E-Mail, Voice over IP (VoIP), etc. Those network services allow users do various activities, including communications with others.

In the past decade, another network service model, called Peer-to-Peer Network (P2P network for short), is created. It is proposed to act contrarily to the traditional way and connects with each other for sharing resource or contents. In traditional network service architecture, stations act as either a client or a server, but not both. While in P2P network, stations act as both a client and a server. The traditional Client-Server network service architecture faces some problems such as Server is limited in resources owned by itself, e.g. CPU performance, hard disk storage capacity, network bandwidth, etc, and need to pay additional attentions on the management of a server. Also, machine that play the role of server may attract hacker to attack, finally, it is not easy to invite extra machines to play the role of server for a specific service immediately.

Nowadays, personal computer runs much faster and technologies not only make hard disk storage much larger, but also allow Ethernet bandwidth to reach 1Gbps or even 10 Gbps. A client already has sufficient ability to act as a server. In the meantime, P2P network grows. P2P network using Client-Client architecture solves problems resulted from traditional architecture. It can share resources owned by itself, share CPU computing power, hard disk storage, and files. And it is also easy to join an existing P2P network as well as to exit from

the network.

P2P network has plenty of advantages to draw people to jump in, e.g. users can share their files with others, find files which they don't have from others, and exchange files like they meet with each other. Based on various attractive properties of P2P network, many PC users start to use P2P network to find what they want and feedback to other users. As a result of this situation, P2P network brings other problems.

Problem resulted from P2P network can be categorized into three types. First, P2P network traffic consumes most of bandwidth of the intranet. To an enterprise, it may want to ensure good performance for some critical application; while to an ISP corporation, it may be charged by other upstream ISPs due to P2P traffic. Second, files spread between users in P2P network may violate copyright easily; files broadcasted in P2P network can be any kind of contents which came from different sources. Due to the architecture of P2P network, there is no log to verify who provides the files so that illegal copies of files may be spread everywhere easily, for example, an un-authorized movie. Third, secret documents of corporations or governments may be easily blabbed out without permission because it is unable to record flows in P2P network.

1.2 Motivation and Purpose

With the growing population of P2P networks, the P2P network traffic can occupy up to 90% bandwidth of corporations, schools, and ISP ADSL backbone for home use. Under limited network bandwidth, network manager need to assure that certain service's quality can be guaranteed. Network manager does not like to see that most of network bandwidth is occupied by P2P network, which causes them a headache. And for a school network manager, he/she concerns whether flows include illegal or un-authorized contents inside in addition to

large P2P flow rate, because downloading such contents will be law suited by copyright held organization. In the same manner, for a corporation network manager, he even concerns whether the business secret is revealed to someone else.

So P2P network traffic does affect existing network systems and brings network managers of corporations or school a troublesome. In addition, few business applications must rely on P2P network, so it is necessary to detect and filter P2P network traffic under certain conditions.

To solve problems caused by P2P network traffic, in this thesis, we propose a system that can recognize and filter P2P network traffic packets efficiently. We expect that it can inspect Internet traffic to find P2P traffic and then filter it out without slow down the traffic flow rate. We also expect that our system will not affect users when they use the network so that users will not be aware the existence of a filter that inspects the traffic.

Owing to the requirement of transparency, we need a high performance inspection algorithm and a machine with sufficient computing power. So we choose to use pipeline technique and realize our algorithm using a network processor. A network processor is dedicated to process related network traffic, the whole hardware design is devoted to processing packets flowing in the Internet. So using network processor is a better choice to deal with packet processing than using a general purpose personal computer.

At present, there are some products related to P2P traffic reorganization and filtering. But they are quite expensive and hard to be configured by network manager. So if we can take advantage of network processor to achieve same performance then we can achieve a cost-effective design.

In the rest part of this thesis we will introduce the algorithm we used, regarding how and why network processor does better than a regular personal computer with a fast NIC (Network Interface Card), so that we choose it to put our system on.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. In chapter 2, we will introduce related works and describe both their advantages and disadvantages. In chapter 3, we will describe the whole scenario about our system and discuss the algorithm and hardware architecture in depth. In chapter 4, we will present the implementation of our system. In chapter 5, we will present simulation result. Finally we will give some conclusion and point out some future work in chapter 6.



Chapter 2 Related Works

In this chapter, we will address what is Peer-to-Peer network and how a peer works with others, and issues brought with the trend of using Peer-to-Peer network.

In the following, we will introduce two detecting and filtering approaches for Peer-to-Peer network. We will also describe what is the network processor and its specific functionality of processing network packets.

2.1 Peer-to-Peer Network

In this section we describe the definition of Peer-to-Peer network and its properties. In the following sub-sections we take two P2P network protocols with different properties as example with brief introduction. And after describing different P2P protocols we discuss their common properties and the condition of P2P traffic in the current days.

2.1.1 Introduction and Definition of Peer-to-Peer Network

Peer-to-Peer network (P2P network for short) started from Gnutella in 2000, and others such as Seti@Home and OceanStore are totally based on distributed computing. In general, P2P network shares computer resources directly, including CPU cycles, storage, and contents [1]. When examining whether network architecture is belonging to P2P network or not, we consider whether there is a server involved to provide members with resources or contents. In traditional network architecture, there is a machine playing the role of server. It provides a channel to allow other machines which play the role of client to get content from it. In the

Client-Server architecture server needs to concern lots of things and we describe in following section.

Servers in client-server architecture must be sufficient in the aspect of computing power, storage size, management control of connection, and even bus bandwidth inside the server to achieve better performance, and even consider whether the server is overloaded so that it will not be the bottleneck in the system. A server is so important in traditional Client-Server architecture. On the other hand, there are only few limitations to restrict a client to connect to a server. The limitations to a client are relative low comparing to that of a server, thus in the Client-Server architecture, a client is always dumb. It does not need good equipments for connecting to servers.

But in a P2P network, on the contrary, there is no server but only client instead. Thanks to modern technique regular personal, computers in the present day already have enough ability to provide content to another, or called sharing directly. Peers in P2P network have equal capabilities, in other words, client in P2P network is not only a client but also a server. Such changes also lead network service behavior to change.

P2P network avoids traditional centralized management and control owing to the lack of a server. Moreover, P2P network organizes itself to form a group instead of through a centralized mechanism. In a P2P network peers do management, administration, and maintenance by themselves. P2P network employs distributed and decentralized properties so that it does not need a detailed infrastructure to build up the network, and peers can connect with others. For example, as Figure 2.1 shows, when end system A connects to end system B through P2P network, it does not need to concern the underlying network topology and can communicate with each other. In other words, P2P network runs on application layer (layer 7 in ISO 7 layers) but does not concerns underlying TCP/IP layers. There is no need to pay additional attention on management and no access fee with P2P network either, users can find files they want through P2P network free instead of through traditional server that users may

be charged for service.

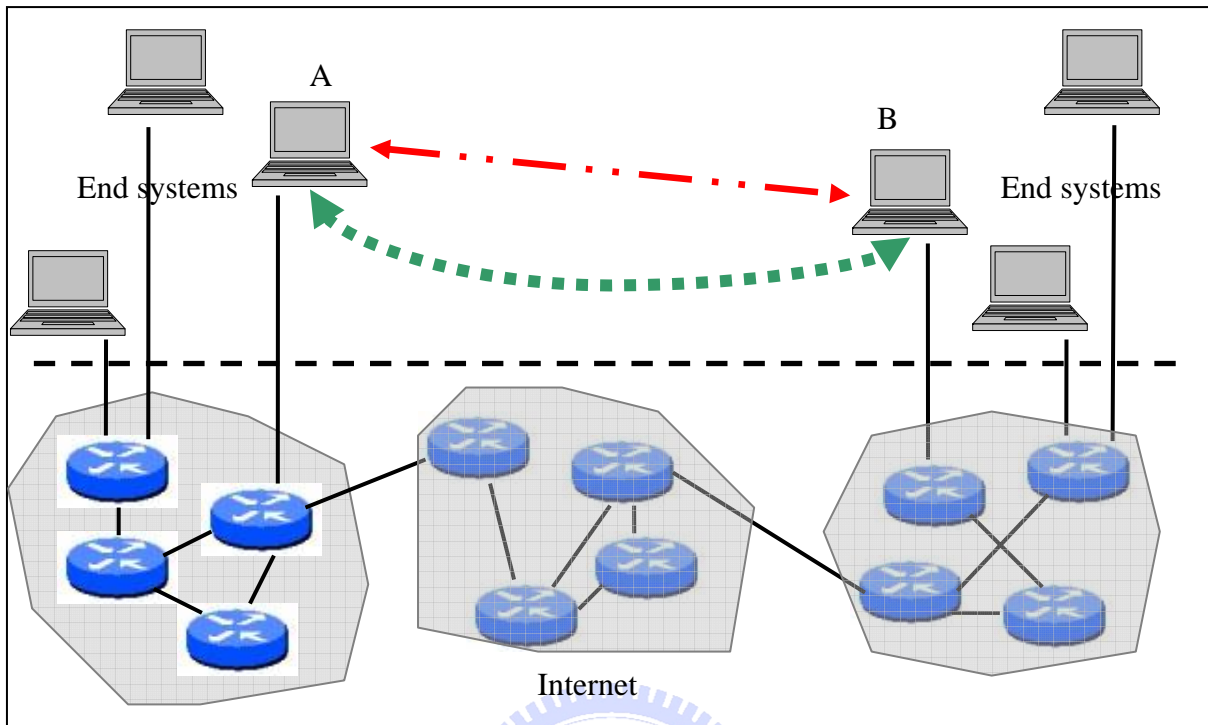


Figure 2.1 Peers can communicate without considering underlying network infrastructures.

Because it is easy to join or to leave a P2P network group, this means that there is little restriction being a P2P network group member. It is quite easy to find P2P application software from Internet and users just need to download P2P application software from Internet and install it. Then users can participate in P2P network. For those reasons, the number of P2P users, also called peers, is growing quickly.

2.1.2 How Peer-to-Peer Network Works

Clients in P2P network are unlike clients in traditional network service architecture any more. It downloads contents from others and provides contents at the same time. Contents are distributed to peers but not centralized in a certain station. Because there is neither a server inside a P2P network group or centralized database and centralized management naturally.

For a peer in a P2P network group, how to discover other peers location is the first issue that needs to be solved.

To solve the aforementioned issue, peers position and perform interaction between each other; peers need to know where they are and even where neighbors are. Finally, peers can connect to each other directly or indirectly. Besides, peers may leave when a P2P network is operating or new peers join in during the operation.

So there must be some peers which maintain certain other peers' location like the way servers did in traditional client-server architecture. Based on P2P network's centralization we can categorize P2P network into three types [1]:

a. **Purely Decentralized Architectures:**

Every peer has the same functionality in the P2P network. They act as client as well as server. And there is no central coordination. Those peers in this kind of architecture are called "servent", means "SERVer + cliENT".

b. **Partially Centralized Architectures:**

The basis of partially centralized architecture is same as purely decentralized architecture. However, some peers in partially centralized architecture are assumed to play a more important role than others, called "supernode". They store index for local files shared by local peers. The way to decide which peer acts as a "supernode" changes with different P2P network dynamically. If a supernode fails, there will be another node chosen to replace the original one.

c. **Hybrid Decentralized Architectures:**

In hybrid decentralized architecture, there is a central server that facilitates interaction between peers by maintaining directories of meta-data describing the shared files stored by the peer node. Although the end-to-end interaction and file exchanges may take place directly between two peer nodes, the central servers facilitate this interaction by performing the lookups and identifying the nodes storing

the files.

In the next sub-section we take two P2P network protocols, Gnutella and BitTorrent, as examples to explain how they work.

2.1.3 The Gnutella Protocol

We take the P2P network protocol, Gnutella [3], as an example first because Gnutella protocol is one of the first generation P2P network protocols so it is worthy to describe Gnutella protocol as an example. Gnutella network is categorized into purely decentralized architecture and is devised by Justin Frankel at 2000. In Gnutella network architecture, file and directory are all distributed in peers. So under Gnutella network if any node (or peer) desires to download contents then the first thing needed to do is to join an existing Gnutella network then use it to download or provide contents and that are two stages in Gnutella network.

Generally speaking, in Gnutella network there is a well known node called “Anchor” that uses public IP address so that when a new peer wants to join a Gnutella network, it connects to the anchor first. To understand how a Gnutella network works, we need to know four base jargons. These four base jargons [4] are explained as follows:

1. Ping

A Ping command is used to announce that here is a new peer joining and probes for other existing peers in this network group.

2. Pong

A Pong command is used to reply a ping command. A pong command packet includes IP address and port number of the responding server. A pong command packet also includes number of files shared and number of kilobytes shared by the responding server in this network group.

3. Query

A Query command packet is used to request downloading contents. It includes some necessary information of desired contents in a query command packet.

4. QueryHit

A QueryHit command packet is used to reply a query command. It includes IP address and port number of responding server and numbers of result and result set.

So in the first stage in Gnutella network is to join the network, when a new peer connects to an existing Gnutella network through an anchor, it will send a “Ping” command to the member of the network it tries to connect to. The anchor or other members will flood this “Ping” message to help the newly joining member discover the network group members. And a “Pong” message will be replied from servers to help build up connection with it.

The second stage in Gnutella network is finding desired contents. A peer that wants to download files will broadcast a “Query” message to the network group first. And this “Query” message will be flooded to the deeper structure of network group several times. If there are some servers which can provide desired contents then they will reply a “QueryHit” message back to the peer that wants to download. Finally, this peer can download files directly based on “QueryHit” messages from those servers who can provide the requested content.

2.1.4 The BitTorrent Protocol

We take BitTorrent protocol [7], [8] as an example because it has become one of the most popular P2P network protocols [5] today. The BitTorrent protocol is categorized into hybrid decentralized architecture and is devised by Bram Cohen at 2001.

Figure 2.2 shows the BitTorrent protocol working architecture. Inside the BitTorrent architecture there is a machine acting like a server in traditional network but differs from the

server in providing peers' locations instead of contents. After peers get locations of other peers through the server, they can download from or upload to each other files directly. In other words, all peers need to get other peers' location first before downloading content, so it is necessary to have a server to provide "tracker" which is responsible for providing peers' location to allow peers to share contents.

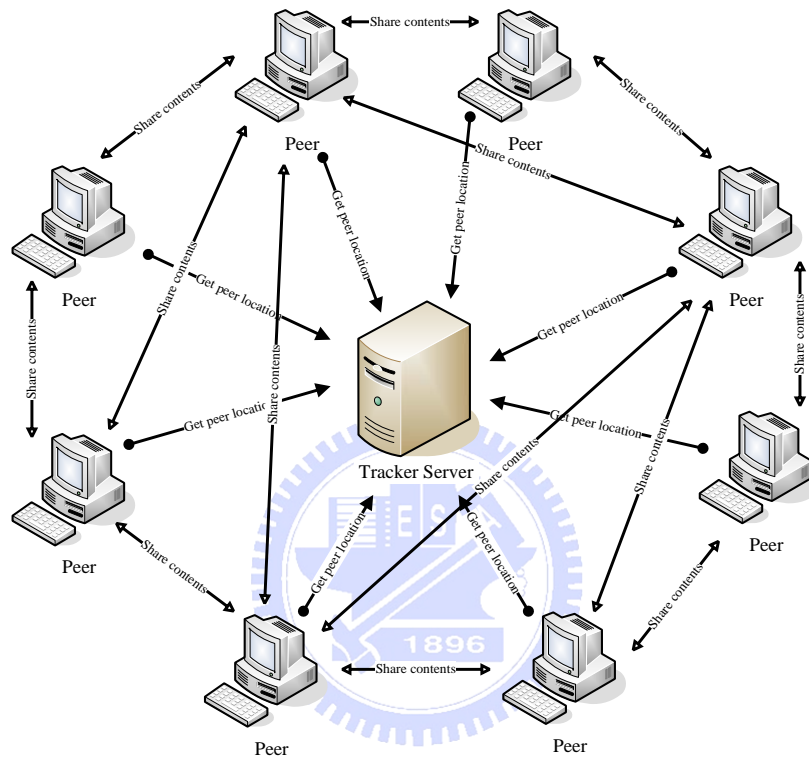


Figure 2.2 The BitTorrent Protocol network architecture.

Generally speaking, the BitTorrent working model is while users are using BitTorrent protocol to download contents they are also uploading pieces of file to each other simultaneously. The main purpose of it is making peers' download rate be proportional to their upload rate to achieve fairness [6].

To understand how BitTorrent protocol works, there are some jargons needed to know. The following shows the four base jargons [6].

a. **.torrent file:**

A .torrent file consists of information about the file including the file name, the file

length, the hashing information and the URL of a tracker.

b. Tracker:

A tracker helps those peers who desire to download contents to find each other. Once a peer (or node) needs to download a file, it sends information regarding the desired file, which port it should listen for downloading and other information to tracker by HTTP. Then tracker responds with a list of contact information for peers which are downloading the same file. Through a tracker is the only way for peers to find each other.

c. Seed:

A seed is to make content available. A peer who has already downloaded a complete file and be willing to share with other peers is called a seed.

d. Pieces of file in fixed size:

In order to keep tracking of what files stored in a peer, BitTorrent protocol cuts files into pieces of fixed size, typically a quarter megabyte. Each peer reports to all of its peers what pieces it has. To verify the file correctness, the SHA 1 hashes of all the pieces are included in the .torrent file, and a peer reports what pieces it has after checking the hashed value.

To start using BitTorrent protocol, users (or peers) who desire to download a file need to download a “.torrent” file from an ordinary website. Then, using a BitTorrent client software, for example, BitTorrent, same with the protocol name, to read the downloaded .torrent file and connect to trackers to get other peers who is downloading from the same file location.

To start to download file from peers, since there is no content related to the desired file, peer should download pieces of file from others in the beginning and then start to upload file to other a short time later.

If a peer wants to provide file so that other peers can download from it, the peer must make a .torrent with required information inside and publishes it to a website. Therefore,

peers who desire to download that file can get the .torrent file from the website to know the location of uploader.

So far, it is obvious that BitTorrent works well, and it uses two servers, one for playing as a tracker and another for providing .torrent files.

2.1.5 Peer-to-Peer Network Properties

Even though there are three types of P2P network architecture, one behavior is never changed. That is how to find desired contents and start to download contents from peers. Typically two phases are involved if a peer desires to download content, as shown in the following [2].

1. Signaling

During the signaling phase a client searches for the content and determines which peers are able and willing to provide the desired content. In many P2P protocols this phase does not involve any direct communication with the peer that will eventually provide the content.

2. Download

In this phase the peer (or node) who wants to download files contacts one or multiple peers directly to download the desired content.

In summary, peers that want to download content must find a peer which can provide the content first and then start to download from it.

During signaling phase it is not necessary to contact with peer directly, so the majority of P2P traffic happened in download phase. Also in the download phase, we discover the significant “signatures” that represents P2P protocols or P2P applications; this will be addressed in latter chapters.

2.1.6 Peer-to-Peer Network in the Present Day

P2P protocol was developed about a decade before and has already grown to be a stable member in network traffic nowadays.

As shown in Figure 2.3 [9], this was compiled from netflow data [10] available for the Abilene network [11]. In Figure 2.3, we can find that during 9/25/2002 to 3/25/2003 time period known P2P traffic had big decreasing and did not show an obvious increasing later. It can be explained the traffic had been restricted by copyright related laws but it may be not the only reason to explain the figure's shape.

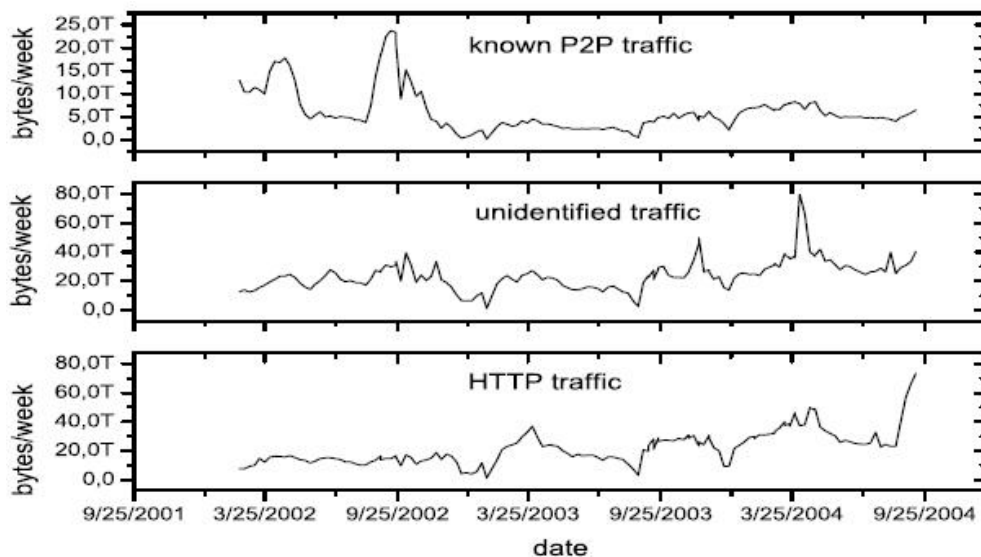


Figure 2.3 P2P traffic flow during 9/25/2001 to 9/25/2004.

Past P2P protocol like the first generation P2P protocol, such as Gnutella protocol, uses some standard port to transfer files. For instance, Gnutella protocol uses ports 6346 and 6347 for TCP and UDP traffic as default specification settings. They are easily detected and filtered by layer 4 infrastructures such as firewall. For P2P protocol developers, to avoid being detected easily they start using more sophisticated method to camouflage P2P network traffic so that less and less P2P protocol applications can be clearly recognized by either TCP or UDP port numbers.

Though laws related to Intellectual Property and penalty fines influence partial P2P network traffic but according to Telefonica, Spanish and South American telecom or ISP corporation reports over 50% of IP traffic in their network being P2P network traffic. Also according to Sprint's IP Monitoring Project, in August 2002, for the majority of the monitored links between New York and San Jose, P2P traffic is approximately 20% of the total volume and in April 2003, 20-40% of total bytes corresponds to P2P traffic. They all show P2P network traffic is a significant fraction.

But over the same time interval, the other TCP traffic category in Sprint's Project also increased. And these unknown TCP traffic can be recognized by known P2P protocol port numbers and this implies P2P protocol is shifting from using standard port numbers to arbitrary port numbers [5].

According to CacheLogic company's [12] research report announced in 2005, it is shown that P2P traffic activity is growing, as Figure 2.4 shows.

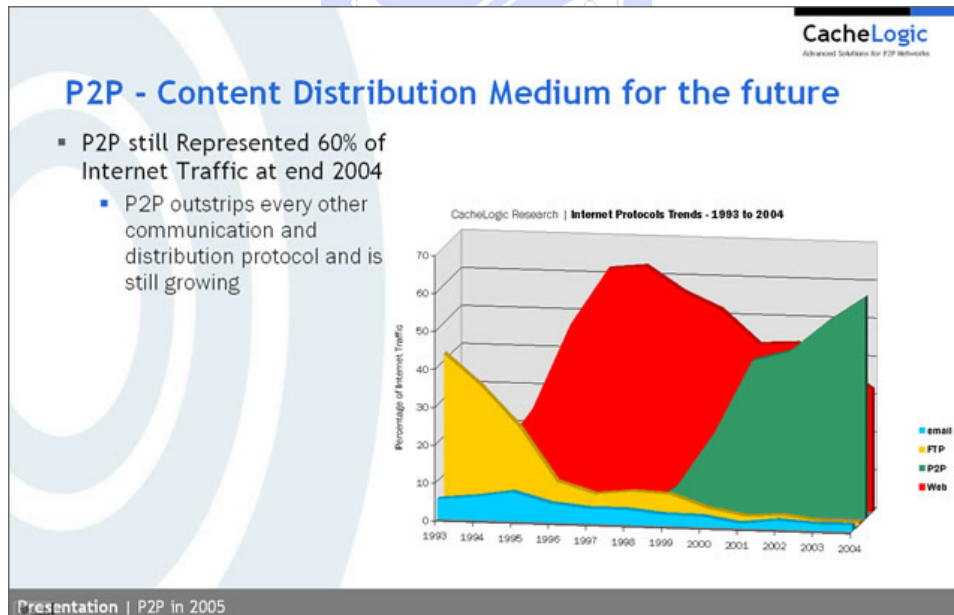


Figure 2.4 Network traffic distributions from 1993 to 2004 by CacheLogic company.

Also shown in CacheLogic's report, P2P protocol used in various countries also varies, as Figure 2.5 shows.

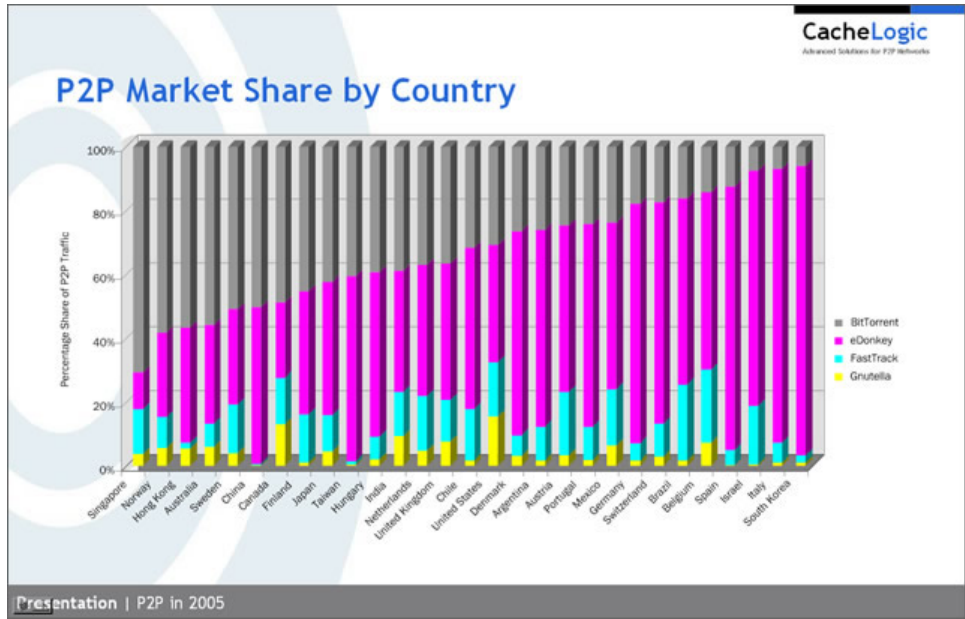


Figure 2.5 P2P market shared by country.

In Figure 2.5, we can see that Taiwan, the most popular P2P protocols are BitTorrent and eDonkey.

So it is necessary to filter P2P traffic if it has affected current network bandwidth or involved lawsuit related copyright owned by others.

2.2 Current Peer-to-Peer Protocol Detection Methodologies

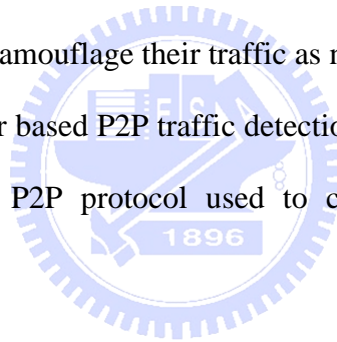
There are some methods which can be used to filter P2P traffic, for example, port number based method and signature-based method. In this section we will address these two different mechanisms briefly.

2.2.1 Port number based Peer-to-Peer Detection Method

Methods based on port number are efficient in the past. Since once a TCP or UDP port number used by P2P protocols or applications is detected, port-number based method can be used to filter them easily, also implementation of port-number based method is easy. It can be built upon a regular firewall system based on PC or commercial firewall products but also other related Layer-4 network devices.

Port-number based P2P traffic detection method can be easily structured by network manager and supported by any layer 4 related products, but it can not be applied nowadays to detect all P2P protocols. To avoid being detected by port-number based method, P2P applications start to change TCP or UDP port number, and use arbitrary port number, including port 80 for WWW, to camouflage their traffic as normal network traffic.

So in summary, port-number based P2P traffic detection method is no longer effective to P2P traffic nowadays, because P2P protocol used to camouflage their traffic to avoid detecting.



2.2.2 Signature-based Peer-to-Peer Detection Method

Another effective P2P traffic detection method has been proposed. After gathering network traffic flowing through network backbone and calculating statistics on collected data, we could find that in the TCP or UDP traffic there are some “signatures” inside the packet payload used by P2P protocols or applications. P2P protocols or P2P applications use signatures to distinguish their own packets from other packets. That is the way they communicate between nodes and nodes. So with different P2P protocols or P2P applications there are different signatures.

Because signature exists within packet payloads; therefore, signature-based P2P traffic

detection method is based on packet payload examination. Such method is more accurate than port-based method mentioned above but also need more powerful computing capability.

Because P2P protocols are not standardized as TCP/IP, they are devised by private organizations and are evolving day by day. The only way to know what signatures they used is by reverse engineering technique [2], [9], [18].

The following is the introduction of signatures of some popular P2P protocols or applications from their packet payloads.

a. **Gnutella protocol**

There are some applications in Microsoft Windows, for example, Morpheus [13], Limewire [14], BearShare [15], and XoloX [16]. Files downloaded in Gnutella use http requests. Files requests are initiated with “GET /uri-res/” command. Inside a TCP packet used by Gnutella, it is started with “GNUTELLA” or “GIV” strings; while inside an UDP packet; it is started with “GND” string. Though there are string inside Gnutella packets, Gnutella application usually use standard port number to transfer file. So it can be detected either by signature-based method or by port number based method.

b. **DirectConnect protocol**

Application for DirectConnect protocol in Microsoft Windows is, for example, DC++ [17]. All packets used in DirectConnect protocol started with “\$”, and such condition makes it easy to recognize DirectConnect traffic by just examining few bytes from start of packet payload. Moreover, all packets in DirectConnect started with “\$” are also ended with “!”.

c. **FastTrack protocol**

Applications in Microsoft Windows for FastTrack protocol, for example, KaZaa [19], Grokster [20], and iMesh [21]. FastTrack protocol has been the most popular P2P protocol for a long time [5], [9]. TCP packets with files downloaded in FastTrack protocol are started with a string “GET /.hash=”, and other with a “GIVE” command. Also in UDP packets used in FastTrack protocol there are some signatures inside. From the beginning of those UDP

packets, they include strings “0x028”, “0x290000002901”, “0x280000002900”, “0x270000002980”. And for a popular application, KaZaa, of FastTrack protocol uses additional string to communicate that is “X-Kazaa” string started from the packet inside KaZaa application.

d. eDonkey protocol

Applications supporting eDonkey protocol in Microsoft Windows are, for example, eDonkey2000 [22], and eMule [23]. Packets used in eDonkey protocol are started with “0xe3” or “0xc5”.

e. BitTorrent protocol

Applications in Microsoft Windows for BitTorrent protocol are, for example, BitTorrent [7], and BitComet [24]. The communication in BitTorrent protocol starts with a handshake packet which includes string “0x13BitTorrent protocol”. And for BitTorrent applications, BitComet, packets in UDP also include string “d1:ad2:id20” or “d1:rd2:id20”.

Table 2.1 shows some possible signatures of popular P2P protocols.

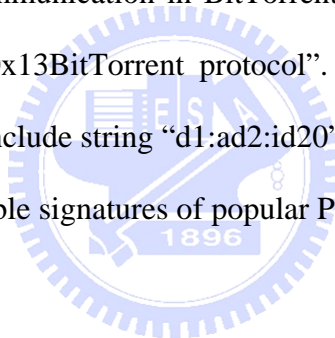


Table 2.1 Signatures of common P2P applications

<i>P2P Protocol</i>	<i>String</i>	<i>Transfer protocol</i>	<i>Default ports</i>
eDonkey2000	0xe3, 0xc5 "Get /.hash", "GIVE"	TCP/UDP	4661-4665
Fasttrack	0x270000002980, 0x280000002900, 0x29000000, 0xc028 0xc1 (5 bytes), 0x2a(3 bytes)	TCP UDP UDP	1214
BitTorrent	"GET /announce?info_hash", "GET /torrents/" "GET TrackPak", "0x13BitTorrent" 0x00000005, 0x0000000d, 0x00004009	TCP TCP TCP	6881-6889
OpenNap/Napster (starting 3rd byte first 2 bytes is packet length)	0x31 (1 byte), 0x0000ca00 (4 bytes) 0xc800, 0xc900, 0xcb00, 0xcc00, 0x0000, 0xd600 0x0200, 0x0300, 0x0600, "SEND", "GET" (3 bytes)	TCP TCP TCP	6699
Gnutella	"GNUTELLA", "GIV", "GET /uri-res/", "GET /get/" "X-Versio", "X-Dynami", "X-Query", "X-Ultrap", "X-Max", "X-Quess", "X-Try", "X-Ext", "X-Degree", "X-Gnutel" "GND"	TCP TCP TCP UDP	6346-6347
MP2P	GO!!, MD5, SIZ0x20 , STR0x20 0x000000 (starting at 3rd byte)	TCP UDP	10240-20480 41170, 22321
Direct Connect	"\$Send", "\$Search", "\$Connect", "\$Get", "\$MyInfo" "\$MyNick", "\$Direction", "\$Hello", "\$Quit", "\$Lock", "\$Key" "\$SR", "\$Pin"	TCP TCP UDP	411-412
Soulseek (starting at 3rd byte first 2 bytes is packet length)	0x00000100, 0x00000300, 0x00000700, 0x00001200, 0x00001a00 0x00000900, 0x00002800, 0x00002900, 0x00002a00, 0x000003310000, 0x0000(4 bytes)	TCP TCP TCP	2234, 5534
Ares	"GET hash:", "PUSH "	TCP	
EarthStation 5	"GET /\$\$\$\$\$\$\$\$/" 0xcf64 (starting at 3rd byte)	TCP UDP	

In summary, signature-based P2P traffic detection method is more accurate than port-number based detection method but also more adaptive for the current time. Since packet payload examination is required, it also needs higher computing power to handle such huge quantity of packets in the Internet.

2.3 Network Processor Overview

For the purpose of accelerate whole system performance, we decide to implement our system on network processor instead of general PC processors and utilize content pipeline programming model instead of traditional model.

During various network processors manufacturers, we choose and utilize IXP2400 series network processor from Intel Corporation to be our implementation platform. We also take advantage of different hardware properties of IXP2400 to raise system utilization so in this

section we address the properties of IXP2400 from the hardware point of view. We introduce what role the network processor plays in the network and hardware components inside IXP2400.

Details about our system design pattern will be addressed in the following chapters.

2.3.1 What Is Network Processors

In various network processor products, we choose Intel IXP2400 series, Figure 2.6, to help design the filter mechanism.



Figure 2.6 Intel IXP2400 network processor chip.

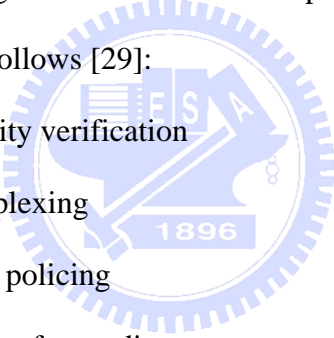
Intel Network Processor is dedicated for network access and edge applications that require high level of processing performance to support value-added network services at OC-48/2.5 Gbps line rate.

As Intel Network Processor IXP2400 product brief [26] mentioned, it is designed for a wide range of access and edge applications including multiservice switches, routers, broadband access devices, and infrastructure wireless systems. The IXP2400 is a fully programmable network processor that implements a high-performance parallel processing architecture on a single chip for processing complex algorithms, deep packet inspection, traffic management, and forwarding at wire speed.

Inside an IXP2400 network processor chip, there are an Intel XScale core processor and eight independent multi-thread microengine processors [33]. The core can be programmed with common tools, standard language, and can run standard operating systems, e.g. Monta Vista [29] Linux. And the microengine is special-purpose RISC processor with eight threads of execution.

In this thesis, we use Intel IXDP2400 Advanced Development Platforms to develop our system. The whole architecture of IXDP2400 [30] equipped with dual IXP2400 network processors and other media cards is shown in Figure 2.7. In Figure 2.7 we can see that there are two IXP2400 to form a core processor center. And media cards as I/O port allow packets in and out. With IXDP2400 we can develop various applications with high performance.

Generally speaking, the ingress IXP2400 has responsibility to handle all incoming packets; they are summarized as follows [29]:

- 
- a. Error detection and security verification
 - b. Classification or demultiplexing
 - c. Traffic measurement and policing
 - d. Address lookup and packet forwarding
 - e. Header modification and transport splicing
 - f. Reassembly or flow termination
 - g. Forwarding, queuing, and scheduling

The egress IXP2400 processor has responsibility to handle all outgoing packets; they are summarized as follows [29]:

- a. Addition of error detection codes
- b. Address lookup and packet forwarding
- c. Segmentation or fragmentation
- d. Traffic shaping
- e. Timing and scheduling

- f. Queuing and buffering
- g. Header modification
- h. Output security processing

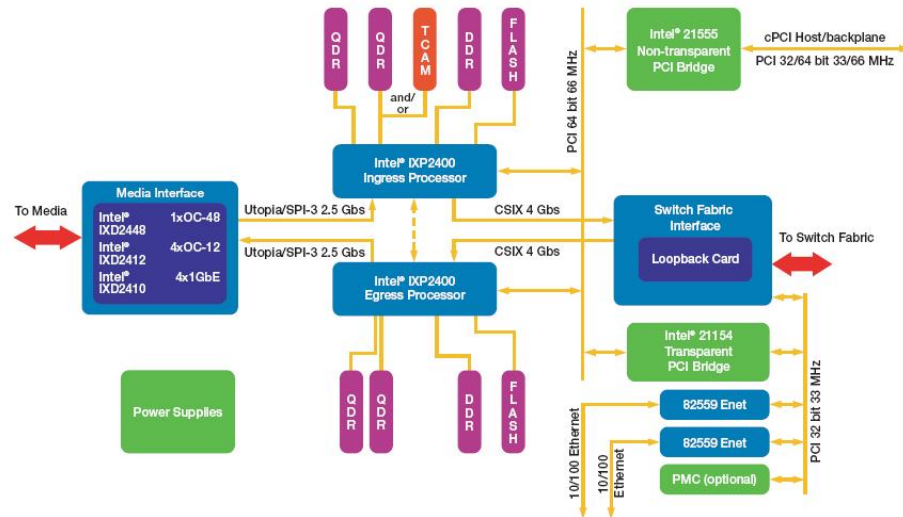
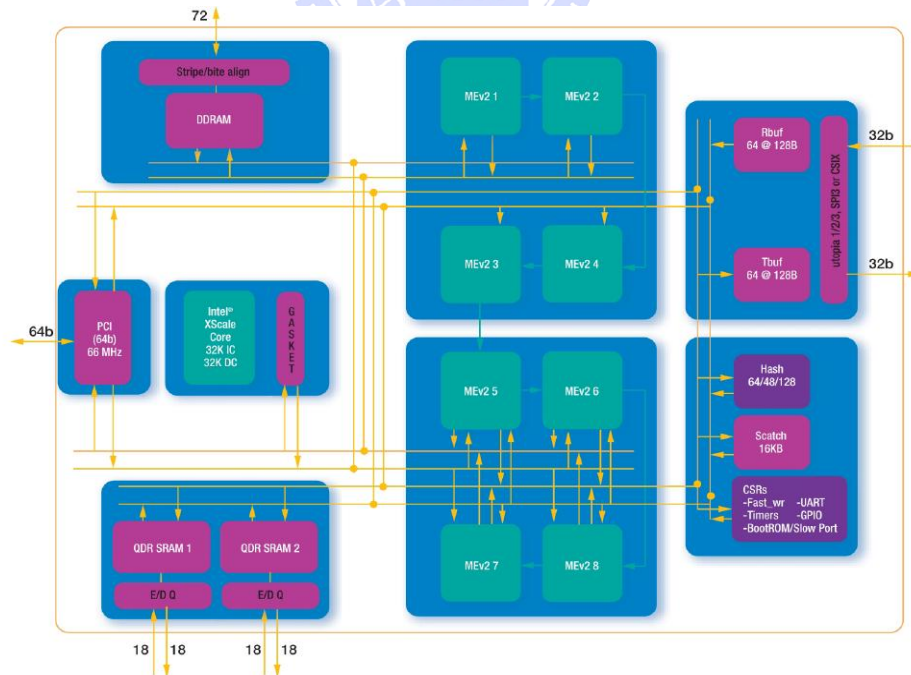


Figure 2.7 Intel IXDP2400 system components deployment architecture.

There is also the hardware architecture of IXP2400 as shown in Figure 2.8.



Intel® IXP2400 Network Processor Block Diagram

Figure 2.8 Intel IXP2400 chip hardware architecture.

IXP2400 is an embedded RISC processor. Inside an IXP2400, there are eight independent microengines. The microengine has an instruction set specifically tuned for processing network data [27]. Each microengine has an independent instruction store and code store, the Intel XScale core initializes this instruction store before the microengine begins running. The instructions are executed in a six-stage pipeline and on average take one clock cycle to execute when pipeline is full while a microengine is running. Except memory access or branch happened in instructions or instructions aborted in pipeline, it won't take more than one clock cycle to execute.

IXP2400 equips one channel DDR DRAM and two channels QDR SRAM. Usually, DRAM is used to store packets which needed to be processed and SRAM is used to store pointers which point to packets' actual location in DRAM. Figure 2.9 shows the microengine's architecture.

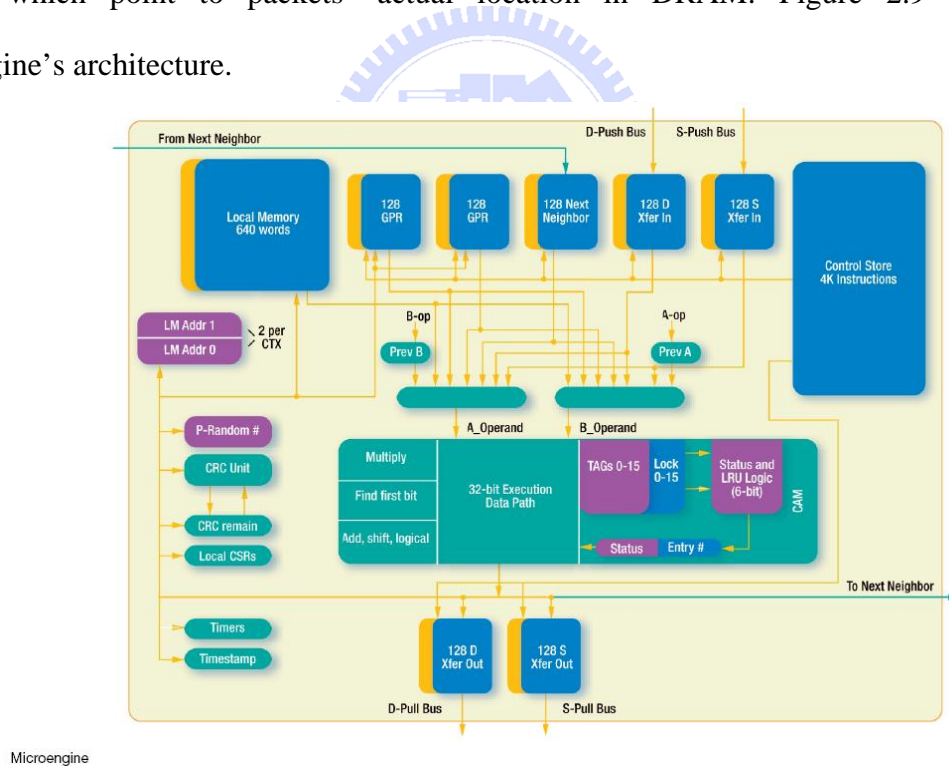


Figure 2.9 Intel microengine chip hardware architecture.

As Figure 2.8 shows, each microengine has its own Local Memory with least memory access latency and registers pool to use. And there is next-neighbor register in each

microengine to provide microengine with sequence-number to communicate. Each microengine has eight hardware-assisted, i.e. zero-overhead context switches, threads for execution. The term zero-overhead means microengine duplicates states, e.g. registers and program counters, for each thread so that execution switch between one thread to other threads can be done quickly [27]. All threads in a particular microengine execute same code from the single instruction store in the microengine.

In Figure 2.8 and Figure 2.9, it shows clearly that there are DRAM, SRAM, and Scratchpad for IXP2400 and Local Memory for each microengine. Each one of this memory hierarchy has different memory access latency and access unit, as shown in Table 2.2.

Table 2.2 differences between different memory types in IXP2400

Memory	Logical width in bytes	Size in bytes	Latency in clock cycles
Local Memory	4	2560	3
Scratchpad	4	16K	60
SRAM (QDR)	4	16 MB per-channel	90
DRAM (DDR)	8	1 GB per-channel	120

So it is indicated that Local Memory has least memory access latency, because it is built on-chip of each microengine, and the next are Scratchpad and SRAM for an IXP2400, and DRAM for an IXP2400 has longest memory access latency.

Chapter 3 Proposed Approaches

In this chapter we describe our proposed schemes principles for P2P traffic detection and filtering which are suitable for network traffic flow nowadays. The main objective of our proposed schemes is detecting and filtering P2P traffic packets in time without decreasing throughput for network users, in other words, it is transparent to network users.

In the following sections we will describe our design principles and design pattern of our proposed schemes. We address the issue from both software and hardware points of view.

3.1 Design Principles

In this section, we discuss our design principles of the whole system. There are two main design requirements we followed in the thesis and we will address in the following sections. These two requirements are:

- a. Transparent property.
- b. Cost-effective property.

To play a successful role as a network device, we also concern reliability. It is necessary to have sufficient computing power to avoid consuming too much network bandwidth. We also concern flexibility; we consider whether our system can be updated by system administrators easily. And in this chapter we will explain how we achieve those requirements.

As we mentioned in Section 2.2, Current Peer-to-Peer protocol Detection Methodologies, there are two most popular P2P traffic detection methodologies we have known. But with newer P2P protocols or P2P applications being proposed, there are more and more different ways which are devised to camouflage their traffic instead of being detected. The ways through which P2P protocols or applications used to camouflage their traffic have already

evolved from regular fixed port-number to arbitrary port-number and we believe there will be newer methods to be proposed in the future.

For certain newer P2P protocols or applications, they even have to examine network flows to detect signatures of a certain P2P protocol or application. Those use separate packets in a flow to stand for a P2P protocol or application. In other words, various packets in a flow have different characteristics, for example, packets in flows with different packet size then have different characteristics.

In our thesis, we focus on signature-based methodology which is the most popularly way used by P2P protocols or applications to distinguish their own packets from the others. So in the following of this thesis our full attention will be put on how to devise a signature-based mechanism with good performance and it is adaptive nowadays.

3.1.1 Transparent Property

Under the pressure of enormous number of packets in the Internet, a system with capability to detect and filter P2P traffic in time is needed. To devise a system prototype that adapts to current network traffic rate, some principles are as follows.

“The system must not affect the network availability of users. That is, the feature of the system development must not decrease network bandwidth so that users under the system can work as normal as before. “

The whole system topology is shown as Figure 3.1. In Figure 3.1 we can see packets belonging to P2P traffic and non-P2P traffic are delivered by users and after being filtered by our system only non-P2P traffic can pass into Internet.

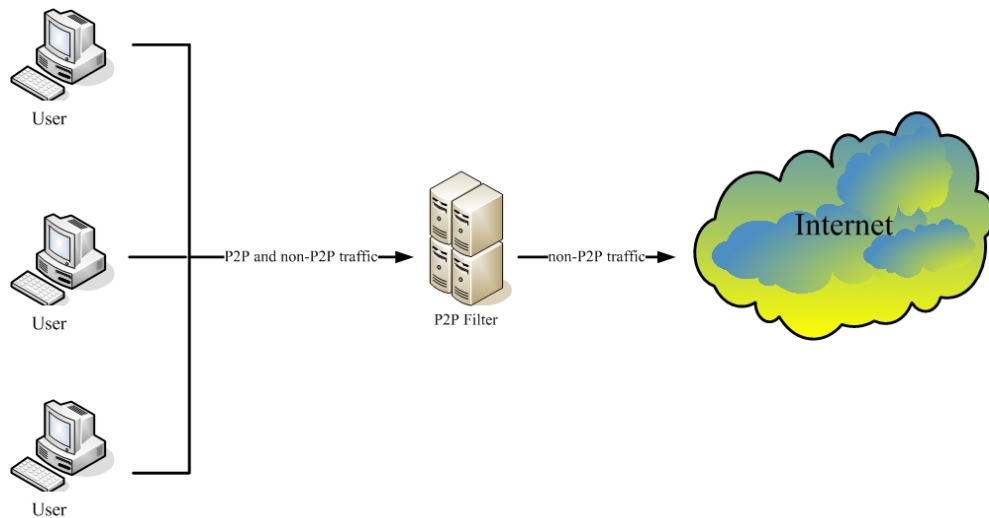


Figure 3.1 P2P filter system network topology.

So the issue we considered is dealing with each incoming packet without delaying packets transmission. Therefore, our detecting and filtering procedure must have good detecting and filtering algorithm, specific-designed programming paradigm.

3.1.2 High Performance Property

In our thesis, first, we focus on how to distinguish P2P traffic packets accurately from vast Internet traffic flows. Then, to develop a system without influencing network operation to achieve transparent property, we focus on how to design an adaptive P2P traffic detecting and filtering system from both viewpoints of hardware and software. It is necessary to make both two aspects meet our expectation because only emphasis on one single aspect would not achieve satisfactory system performance. In the following sections we will discuss these two different aspects.

3.1.3 Flexibility Property

After considering how to raise the whole system performance, we concern the flexibility

property of system.

Under different circumstances, there are different distributions of P2P protocols or applications. For instance, as shown by Figure 2.5, in the different countries, the usages of P2P protocols are not with the identical distribution. So to system administrators with different environments, they may have different requirements on blocking P2P traffic.

In order to meet those various requirements, we devise a dynamically-configured signature table inside the system, so system administrator can configure the signature table with priority according to the requirement. They can put the most popular P2P signatures in the beginning of signature table to filter out the corresponding P2P traffic. It also means that system can filter those most used P2P traffic with least computing cycles and highest system utilization because it only checks with the first few rows and can detect and filter those P2P traffic.

Such design provides system flexibility to system managers. They can configure a proper signature table with different needs to allow system dealing with different P2P protocols or applications with different priorities.

Details about implementations of signature table are addressed in Chapter 4.

3.2 Limitations

There are some limitations that we must take into account while designing P2P traffic detection and filtering system prototype. Those limitations not only affect the system performance but also influence system design approaches. We should consider all limitations to let our system be adaptive to various traffic conditions. The limitations are as follows:

P2P protocols and message exchange mechanism of P2P applications

To detect and filter P2P traffic packets accurately, the most important issue is to

distinguish packets in flows which are categorized as P2P protocols or P2P applications. Most P2P protocols and P2P applications are designed or developed by person or private organization, so the detailed messages exchange mechanism are not open to public. Due to the nature of P2P protocols and P2P applications, the way to discover the properties of those message exchange mechanisms is only by reverse engineering technique.

After gathering and analyzing a large number of network traffic flows from Internet, a certain signature in packet payload to represent a certain P2P protocol or P2P applications can be found. P2P protocol or P2P application uses the signature to make their packets differentiated with others in Internet. Packets with the signatures inside the packet payloads can be thought as P2P traffic packet.

Network traffic properties

Packets flowing in the Internet have several properties which affect system performance. They are, (1). Packets in Internet have variable packet size or TCP/IP header size, (2). Packets flowing in Internet are in an enormous quantity, (3). Packets flowing in Internet are in a high transmission rate that results from larger and larger network bandwidth, e.g. 10 Mbps to 100 Mbps or 1 Gbps. To accommodate those network traffic properties, a system with high computing power is required.

Varieties of P2P protocols and P2P applications

Since there is more than one type of P2P protocol or P2P application, only detection based on signatures discovered from the above method can not accommodate all P2P protocols and P2P applications. Once a new P2P protocol or P2P application appears, it needs to do the analysis analyze and find the properties again for the detection when of new P2P traffic.

Encryption

In order to avoid being detected by network devices, more and more P2P application developers take advantage of encryption to camouflage their traffic flows. Packets with

encryption can not be distinguished so that they can pass through the devices successfully. Therefore, to detect P2P traffic with encryption, it needs more effort on the analysis of Internet traffic flows. By categorizing flows operating models, some new behaviors of P2P protocols or P2P applications may be found.

3.3 Design Patterns

While developing an adaptive system, we consider how to optimize system computing capability regardless using software or hardware solutions.

From the point of view of software, we should construct a new programming paradigm which differs from the general ones. In order to improve overall system performance, in our thesis we make good use of “content pipeline” concept. With ordinary programming model, programmers use straight-through programming style in which whole system operation components are centralized on a processor. And with content pipeline programming model, we divide system into several components, and deploy different processors with different program components. We take advantage of content pipeline programming model to improve system performance in software.

From the point of view of hardware, in order to process packets as quick as possible it needs hardware architecture equipped with special-purposed design and parallel-processing functionality. There are two different type machines we can choose to build up our system. The first one is the most popular PC; another is special-purposed hardware for network processing, like network processor with multi-processing capability.

In the following sections we will address how we make decisions between two different programming models and why we choose to take network processors as our system platform instead of general PCs.

3.4 Software Viewpoint

From the point of view of software, we consider what programming model fits our requirement. We compare ordinary programming model and the content pipeline concept, the programming model we used in our thesis, then describe why we use content pipeline programming architecture.

3.4.1 The Traditional Programming Model

With traditional programming model, all system components are centrally executed on a single processor. That is, we develop all functions we need, for a network device, packets-receiving, packets-processing, and packets-transmitting, into programs, then after compiling we let processor executes the object code.

So for the processor, its responsibilities are: first, the processor triggers related device for receiving packets from Internet and stores packets into storage; second, the processor does processing on packets stored, e.g. P2P traffic detection and drops P2P traffic packets, and finally, processor transmits packets into a proper port to destination, as shown in Figure 3.2.

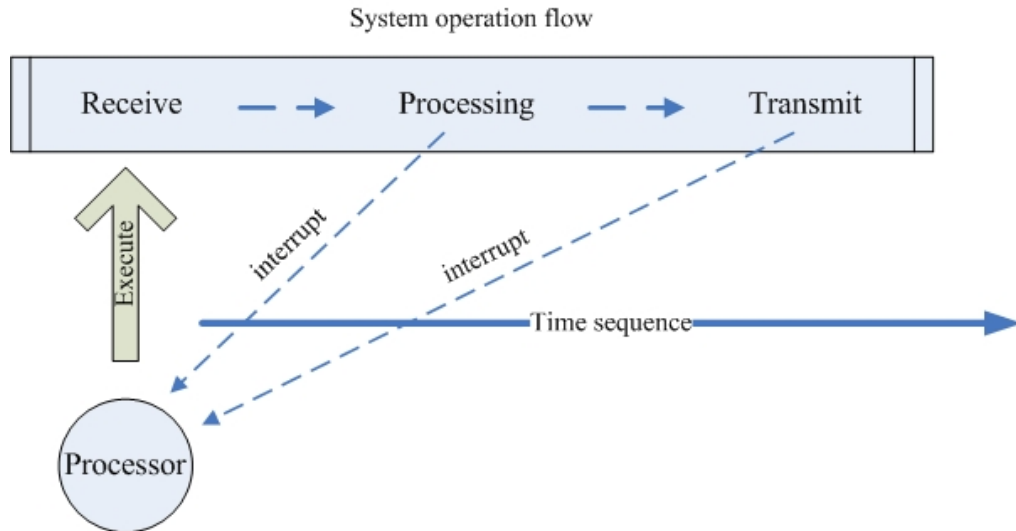


Figure 3.2 The work flow of traditional programming model.

In this programming model, processors need to deal with any possible conditions happened in network, including receiving, processing, and transmission. Besides, processors also need to handle different interrupts or requirements from underlying devices. For instance, if other network device sends an interrupt to the processor while it is receiving another incoming packet and proceeding with packet-processing portion, the processor needs to suspend current job to handle the interrupt. On the contrary, a processor can not deal with receiving packet successively if it needs to take care about packet-processing.

Because all things should be handled within a processor, so even with high clock rate CPU packet processing in, this programming model may still be the bottleneck of the whole system.

3.4.2 The Content Pipeline Programming Model

To avoid above problem, we utilize content pipeline programming model instead of the traditional programming model.

To construct a network application, e.g. router, switch, or P2P filter, it needs many

detailed components. We utilize content pipeline concept to divide whole system into several independent tasks and deploying those tasks into different processors properly, as shown in Figure 3.3.

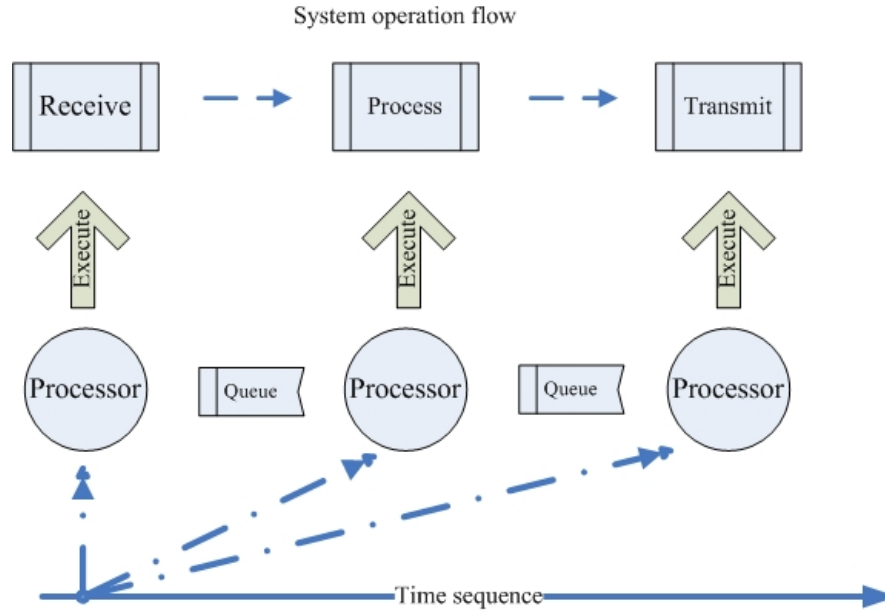


Figure 3.3 Work flow of content pipeline programming model.

In Figure 3.3, it clearly shows that tasks can be executed by different processors at the same time period, and for communicating between pipeline stages or different processors, we use queues to store temporary data from last stage to next stage. Therefore, with this concept we can deploy more complicated tasks with more processors and normal tasks with fewer processors to reach the best performance.

Such programming concept is adaptive to any applications as long as an application can be divided into several independent tasks. With hardware equipped multi-processing, it will amply utilize system for optimized performance.

3.4.3 The Proposed System Model

With content pipeline programming model, we can divide our system into several tasks.

Figure 3.4 shows the model of our system.

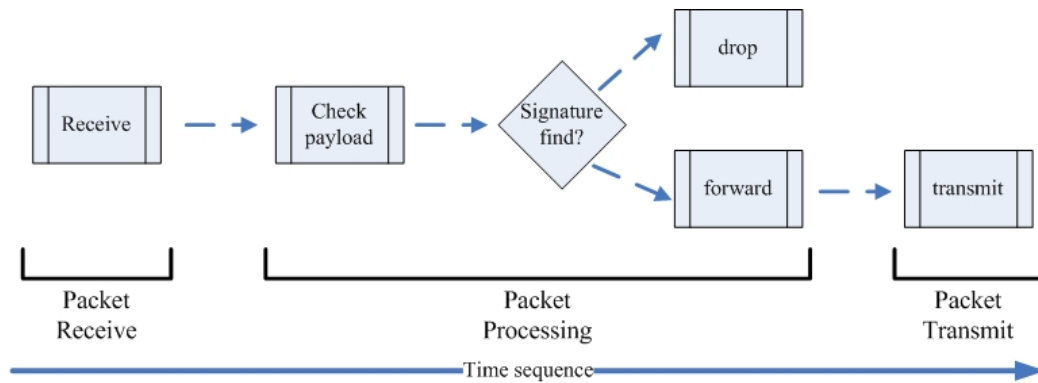


Figure 3.4 P2P filter system model.

In Figure 3.4, it clearly shows that during packet-processing portion, we divide the process into 4 tasks. In the packet-processing stage, we concentrate on P2P traffic detection and filtering. So those 4 tasks are:

- a. Examine packet payload to check whether P2P signature exists.
- b. Examine whether signature is valid or not.
- c. If it conforms to P2P traffic which to be blocked then drop it.
- d. If it belongs to normal network traffic then forward it as normal packet.

After receiving packet from I/O device, we check each packet payload to find whether any signature stands for P2P traffic inside. If there is a signature found in a packet then we mark it as P2P traffic packet, if not then we don't mark it. In addition, packets marked as P2P traffic packet will be dropped in the packet-processing stage, and those as normal packets will be forwarded to a proper port for transmission.

With content pipeline concept, we also utilize multi-threading technique for improving system computing performance. We deploy more than one processor to execute packet-processing because it needs more computing capability, and deploy less number of processors to execute packet-receiving and packet-transmitting tasks. After applying multi-threading processors, our system can be represented as that in Figure 3.5.

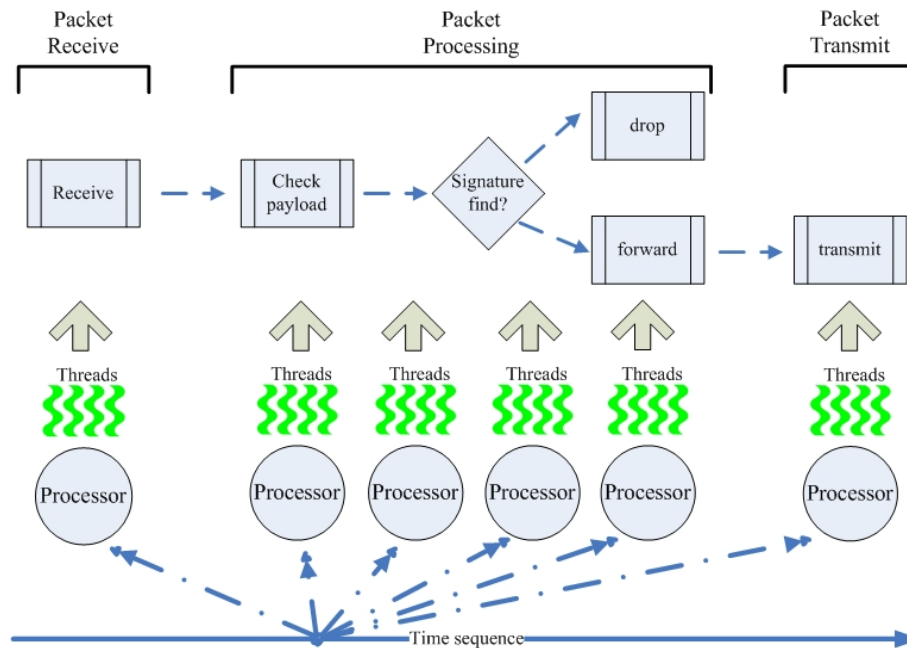


Figure 3.5 P2P filter system with content pipeline and multi-threading.

As Figure 3.5 shows, after initiating pipeline processing, the system can execute packet receive, packet processing, and packet transmit tasks at the same time period. Current pipeline stage will put processed data into queues for next pipeline stages.

In summary, from the software viewpoint, utilizing content pipeline concept to divide our system into several tasks can improve packet processing speed. In following section we discuss how to improve system performance from hardware.

3.5 Hardware Viewpoint

From the aspect of hardware, we consider CPU clock rate, memory access latency, bus bandwidth, and whether it supports multi-processing or not.

For the main purpose of our system, we need a machine with high bus bandwidth, parallel-processing capability and low memory access latency. With general PCs, although the current technique makes PC equipped with dual-core processors, e.g. Intel® Core™ Duo

Processors [38], and PCI-Express bus but they are still not sufficient to be a network packet-processing device.

In the following sections, we will address why general PCs are not proper to be network devices and we also address our approaches to meet high performance purpose.

3.5.1 General PC Review

Owing to hardware design in PC there are some reasons that make PC not suitable for our requirements.

We discuss those reasons from bus bandwidth, CPU computing capability with multi-processing support, and memory access latency:

Bus bandwidth:

Even PCI-Express can provide 1 GB/s bandwidth, it is still not sufficient to be a network device since nowadays network interface card can reach gigabit per second performance. It may lead to a bottleneck due to the limited bus bandwidth when network I/O device communicates with CPU through the bus. So for a network core device such capacity is still not enough.

CPU computing capability with multi-processing support:

In addition, although computer CPU nowadays is equipped with capability of dual-core processing but if there exists other kind of machines that provide more than two parallel processing units, then for achieving the best performance we still think that general PC is not good enough.

Memory access latency:

Finally, from the viewpoint of memory storage we consider memory access latency problems. To achieve the best system utilization we need to provide low memory access latency environment to store significant data. Here data is stored in PC with SRAM so if we

can find a way to store significant amount data with cache-level memory then the overall system performance can be improved significantly.

On the other hand, even if a PC with high performance CPU is equipped with latest PCI-Express bus and gigabit level network interface card, the system utilization may not reach the expected result. This is because the operating system architecture on PC, once there is a packet incoming, the network interface card sends an interrupt to the operating system, and such interrupt may be queued when system is busy with other process on going. That leads to degraded performance. It clearly shows that a totally different architecture and more efficient hardware are needed.

3.5.2 Why Network Processor

For the above reasons, we decide to use a network processor which is dedicated for packet-processing rather than a general-purpose PC to build up our system prototype. And during various network processor manufactures we choose Intel IXP2400 for the sake of convenience.

Inside IXP2400 there are eight microengines which perform different processing tasks independently. It is more powerful than general dual-core processor because they are all equipped with their own instruction sets and registers which are designed for packet-processing and on-chip memory with very low memory access latency. Those eight microengines can work together on a packet-processing project but focus on different tasks.

Here, we make those microengines responsible for packets-processing work on P2P traffic detection and filtering. After examining P2P traffic, those microengines pass the packets to other microengines and keep examining the next incoming packets repeatedly.

Furthermore, each microengine is equipped with independent Local Memory that has lowest memory access latency resulted from its on-chip property no matter it uses SRAM or

DRAM. It is useful to develop a system with significant volume of data kept inside the Local Memory, because the more memory access latency we saved the higher system performance we earned.

In this thesis we put our signature table (discussed in next chapter) for detecting P2P traffic packets into Local Memory because the number of signatures is not very large, so those microengines can use the signature table inside the corresponding Local Memory without accessing SRAM or even DRAM.

3.6 Integrated Hardware with Software

After describing our programming model and hardware requirement, we can implement our system with content pipeline concept on network processor. With eight microengines inside the processor, it is more proper to implement the content pipeline programming model. We deploy some microengines to process packet-receiving, some microengines to process packet-processing, and others to process packet-transmitting.

Furthermore, with network processor, it implements context switching with low penalty and low memory access latency with Local memory, those properties cause network processor suitable for handling packet-processing task. With a network processor, we can control the detail components directly without interfering from operation system. And this property is useful for our system.

We develop our system with consideration of software and hardware viewpoint; also we make content pipeline concept work well with a network processor. In the following chapter we will focus on the packet-processing task, and discuss how to develop a mechanism to distinguish P2P traffic within microengines.

Chapter 4 Proposed Schemes and Implementation

In this chapter we discuss the more detailed schemes of our approach implementation in this thesis. First we address our system architecture and then we address our system's algorithm of P2P traffic detection and filtering. Finally we describe our implementation of system.

4.1 Overview of System Architecture

Since there is more than one signature for different P2P protocols, we construct a signature table to store various signatures. To achieve P2P traffic filtering purpose, once an incoming packet is categorized to P2P traffic then it will be dropped by the system. So the main filtering procedures of our system are as follows:

- a. Get a packet from storage which is stored.
- b. Extract packet payload and check whether a signature is included in the incoming packet payload in the signature table.
- c. If there is a certain signature included in the packet payload then categorizes this packet to P2P traffic by marking it as P2P_traffic with system register.
- d. On the contrary, packets without P2P signatures are forwarded as normal without marking anything.
- e. Then those packets marked as P2P_traffic will be dropped by the system.

In addition, we choose to use the pipeline architecture, and we select the network processor Intel IXP2400 from various network processor manufactures to improve system performance. The network processor IXP2400, which has special-designed hardware

architecture for processing network packets not only has special hardware architecture designed for peripheral but also has been specially designed in itself that differs from regular processors.

Figure 4.1 outlines the algorithm of our system.



Figure 4.1 P2P filter system procedure.

As Figure 2.8 in Chapter 2 shows, inside an IXP2400 there are a general-purpose Intel XScale core processor and eight special-purpose microengine processors and inside each microengine there are eight threads. Such hardware architecture design can attain parallel execution. In this thesis we focus on microengines arrangement and utilize eight microengines. Figure 2.9 shows microengine's hardware architecture.

In this thesis, we modify Intel 4Gb Ethernet IPv4 Forwarding Application for Intel IXDP2400 Advanced Development Platform to fit in with our requirement. In the original application (4Gb Ethernet IPv4 Forwarding Application), it does not equip detecting and filtering functionality of P2P traffic but just check routing table inside the system for each incoming packet then decides which ports to deliver packets. To meet our requirement, we put another module that allows the application to have capability to detect and filter P2P traffic

packets. We discuss our detection and filtering algorithm detailed as the following sections.

4.2 Introduction of System Basic Configuration

In this section we describe basic system behaviors including basic building blocks used and packets storing mechanism before describing filtering algorithm.

We have described that we build up our system on Intel IXP2400 Advanced Development Platform equipped with dual IXP2400 Network Processors. One of dual IXP2400 Network Processor plays the role of ingress processing and another plays the role of egress processing. Ingress processor and egress processor have different responsibilities.

When developing programs to act as ingress processing or egress processing, we exploit Intel libraries and building blocks from Intel Internet Exchange Architecture (IXA) Software Development Kit 4.1 to achieve this objective.

Figure 4.1 shows main procedure of packet-processing. All applications about packet-processing can be prototyped into this procedure. In Figure 4.2 it indicates that the left block acts as ingress component for receiving packets and the right block acts as egress component for transmitting packets. While the middle block has responsibility to process incoming packets, e.g. detecting and filtering P2P traffic, or forwarding.

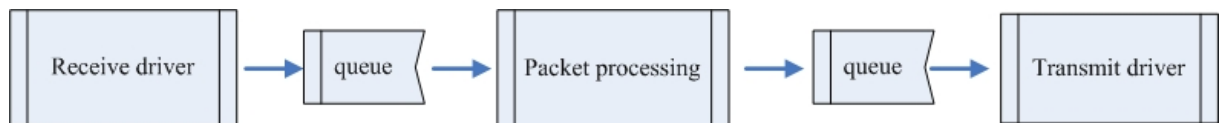


Figure 4.2 Three basic functionalities of network application.

Once a packet coming in, it will be buffered in DRAM of IXP2400 with First Input First Output (FIFO) queue management mechanism because DRAM has larger storage size than others memory categories. And SRAM keeps the pointers which point to the actual packets storage addresses in DRAM. In other words, though DRAM keeps the buffer for packets but

the FIFO mechanism is associated with SRAM. When the receiving block in Figure 4.2 receives packets it allocates buffer and insert related information into SRAM in the Queue Array table. On the other hand, after transmitting a block in Figure 4.2 it de-allocates buffer and removes related information from SRAM. In Figure 4.3 [29], it shows the relationship for packets buffering between SRAM and DRAM.

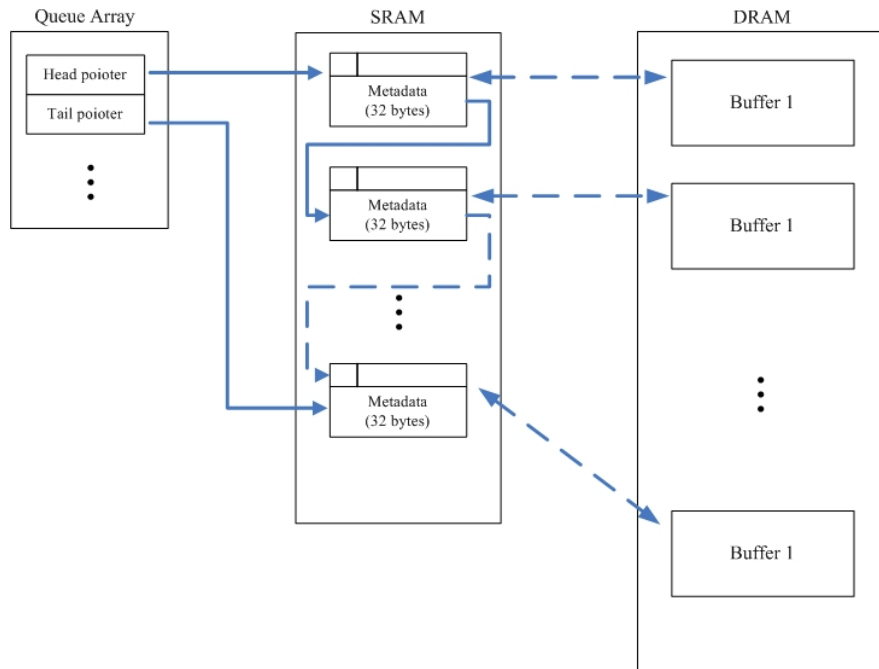


Figure 4.3 Packets queue array relationship in IXP2400.

While programming the left block of Figure 4.2, the ingress block, we use microblock called Receive (RX) provided by Intel Internet Exchange Architecture (IXA) Software Development Kit 4.1. And this microblock, RX, receive packets from media interfaces and copy them from Receive Buffer in Media Switch Fabric into a buffer allocated in memory. Then it passes the packet address to the next processing block. When developing the right block of Figure 4.2, the egress block, we use other microblock called Transmit (TX) in Intel SDK. The main functionality of the TX microblock is to send packets from memory to physical medial interface ports and de-allocates the buffer.

In the following we describe the middle block of Figure 4.2, the packet-processing block.

We will discuss how to build up a P2P traffic filtering system.

4.3 P2P Protocol Detecting and Filtering Algorithm

In this section, we begin to address detection and filtering algorithm of our system. Before comparison to find signature, first we must define packet payload offset for system to extract required information. When an incoming packet is received, system follows TCP/IP protocol standards to define IP header offset and TCP header offset. Figure 4.4 and Figure 4.5 show the IP header and TCP header.

IP header:

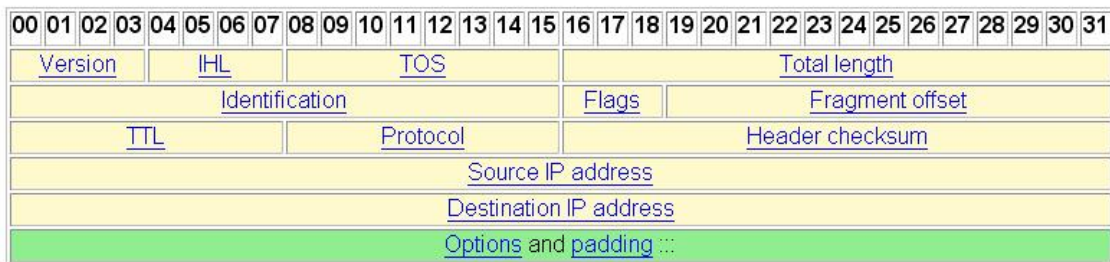


Figure 4.4 IP datagram.

TCP header:

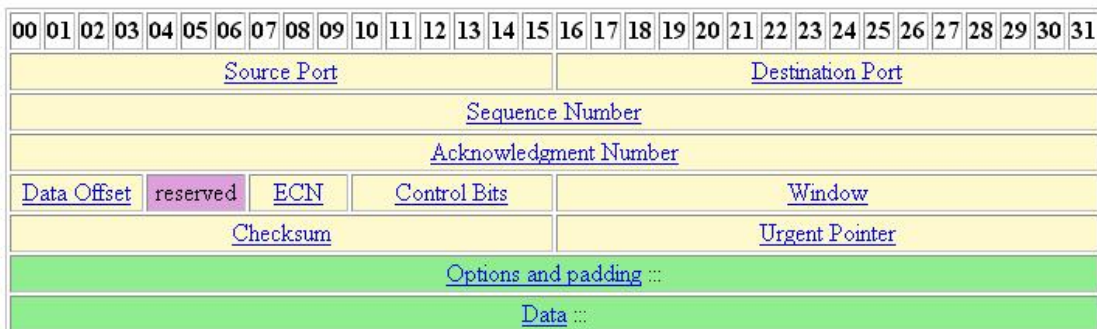


Figure 4.5 TCP datagram.

From Figure 4.4 it clearly shows that IHL (Internet Header Length) field specifies the length of IP header in 32-bit words, including any options. And it is 4-bit offset from the start

of IP header. From Figure 4.5 it clearly shows Data Offset field specifies where the TCP data begins, because the length of TCP header is variable, this is resulted from Option field value. So checking Data Offset field from TCP header is required, and it is 13-byte offset from the start of TCP header.

According to the above two fields, the system calculates packet payload offset from the starting of packet. From the literature proposed regarding signatures of P2P protocols or P2P applications, [2], [9], [18], [35], we know most of signatures appear at the beginning of packet payload because P2P protocols let peers communicate with each other through application layer.

Inside the packet payload, for most popular P2P protocols or applications nowadays, signature does not appear at all portion of payload but several bytes from the starting byte. It is not necessary to examine every byte of each incoming packet payload from start to end and we can detect most popular P2P traffic. So for the sake of system utilization we extract only 32 bytes from the starting of packet payload which is stored in DRAM. After identifying the first 32 bytes of packet payload, we copy them from DRAM to SRAM for the purpose of caching. According to Table 2, memory access latency of DRAM is larger than memory access latency of SRAM. After caching the packet payload, we can compare byte by byte between packet payload cache and entries in the signature table.

For detecting P2P traffic, we examine each byte in payload cache from the start to the end. With comparison, first we get an entry from signature table of corresponding microengine. Then the system extracts a byte in turn from start to end and stores in a register. After extracting a byte from entry of the signature table, the system then extracts a byte in turn from packet payload cache and stores it to another register as that in the above.

Before really detecting P2P traffic, system needs to check the flag field of an entry at first. If the value in flag field is equal to "0x96" then the system does comparison with this entry. If it is not, system jumps to the next entry in order and starts processing another entry.

Once the flag field is on, the system keeps value in signature length field. The value will be kept in a specific register for helping determine whether packets examined is belonging to P2P traffic or not. This value will be decreased to notice the progress of examination while examining packet payload and corresponding signature string in signature table. If the system does not keep this value then system will not be noticed after comparing all bytes of signature string in the signature table.

After preparing bytes for comparison, the system compares two registers from different sources. If two registers have the same value inside, the system will extract next byte from signature string in sequence and extract next byte from packet payload cache. At the same time, system checks whether the value extracted from signature length field is equal to zero or not.

If it is equal to zero then system determines that packet examined is belonging to P2P traffic and exits detection because an incoming packet only belongs to only one P2P protocol or P2P application at any time. If it is not equal to zero then system continues extracting next byte from signature string and next byte from packet payload cache. Finally system decreases the value from signature table by one and repeats the above process.

But if two registers have different values inside then it means that the packet with corresponding packet payload cache examined do not belong to P2P traffic with this signature string. Under this condition, the system will check the packet payload cache with the next entry of signature table. Unless values in the packet payload cache do not match any signature strings in signature table, the corresponding packet will be passed to the followed process and the system examines the next incoming packets.

In our system, there is a global register which can be used by any function in the system called “dl_next_block”. The purpose of this global register is to record which is the next processing stage for a certain packet. In normal condition, dl_next_block register for each incoming packet will be set to “ipv4” after the system received a packet from media interface

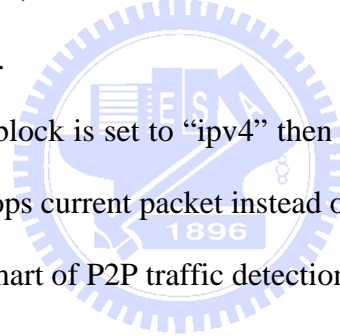
and confirmed that it belongs to IPv4 packet. In the original system setting, packets after receiving and corresponding dl_next_block register is set to “ipv4” will be forwarded by the system. To add a functionality to detect P2P traffic, we define a value, “P2P_packet”, additionally in the system to indicate what packets belong to P2P traffic.

In our system, if there is any kind of P2P protocols or P2P applications signatures is found inside an incoming packet, the corresponding dl_next_block register will be set to “P2P_packet” to warn the system this packet is a P2P packet. We detect P2P traffic by examining whether any signature is included by a packet, then followed by filtering process.

We have mentioned that a normal packet has value “ipv4” in corresponding dl_next_block register. This normal packet will be forwarded by system. In order to achieve the purpose of filtering P2P traffic, we add additional mechanism that checks the value in the dl_next_block before forwarding.

If value inside the dl_next_block is set to “ipv4” then system start to do preparations for forwarding, otherwise system drops current packet instead of forwarding.

Figure 4.6 shows the flow chart of P2P traffic detection and filtering.



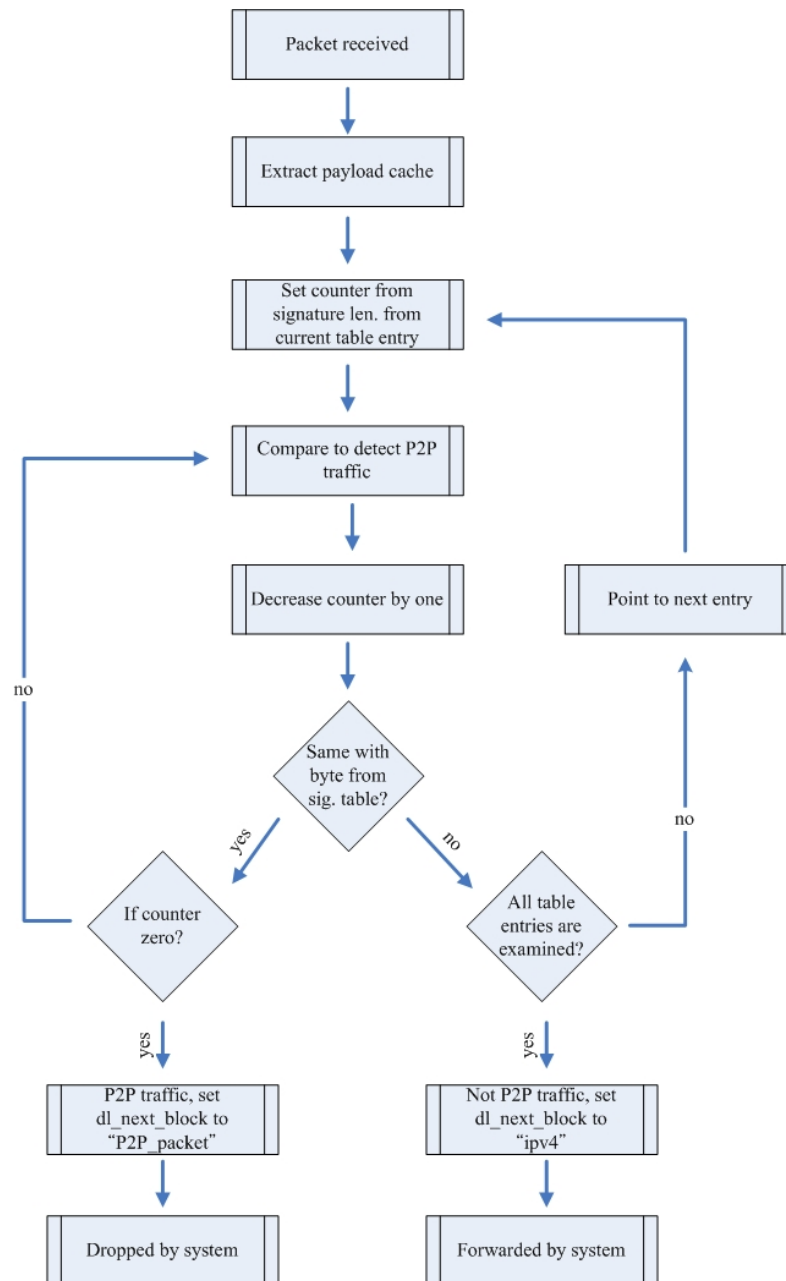


Figure 4.6 P2P detection and filtering flow chart.

4.4 Signature Table

After analyzing P2P network traffic flows in the Internet, we could find various signatures corresponding to different P2P protocols. For detecting and filtering P2P traffic accurately these signatures must be stored for examining each incoming packet.

Because signature table is used for providing signatures, the system can compare every signature resides in each incoming packet payload to detect P2P applications traffic. Unless it is needed to add a new signature for detecting a corresponding a new P2P protocol or P2P application, or a certain signature is no longer needed, the signature table won't be modified. Later in this chapter we will describe how to update entries in the signature table. Therefore, there is only read operation but no write operation to signature table under normal condition from a point of view of system. Because of this property we can consider to replicate signature table for each microengine to examine.

Besides, in order to achieve highest performance, it is important to decide where to store those signatures because it affects utilization of the whole system. Hence, we choose a memory location with least memory access latency to gain high performance. According to Table 2, it clearly shows that Local Memory on every microengine features least memory access latency.

Based on above two conditions:

- a. Since operation on signature table for every microengine can be done independently, it allows us to consider whether we can replicate signature table for each microengine instead of using a single central signature table in the system, this may reduce memory access time by avoiding microengines to access main memory, e.g. SRAM or DRAM, as well as preventing the memory access collision.
- b. Fortunately, except SRAM and DRAM which feature higher memory access latency there is Local Memory with least memory access latency on each microengine.

We decide to implement signature table in the Local Memory and replicate it for every microengine.

4.4.1 Signature Table Implementation

After deciding to put signature table into each Local Memory for multiple microengines to save memory access time and increase system performance, the implementation approach is, first, in our system program we develop a signature table, and use DevWorkbench to dispatch it to multiple microengines for executing the same program to reach this requirement.

In the system, there are four microengines, numbered 1, 2, 5, and 6 (as shown in Figure 2.8), are assigned to execute mission of P2P traffic detection and filtering. Other microengines have other responsibilities including receiving packets, queue management, task switching and so on.

To structure a signature table, we create a data structure format to represent each signature corresponding to certain P2P protocol or P2P application. Figure 4.7 represents an entry format in the signature table.

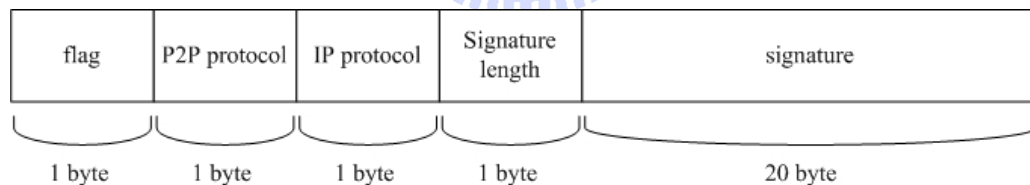


Figure 4.7 Signature table entry format.

Figure 4.7 shows, each entry of the signature table consists of a flag, P2P protocol, IP protocol, signature length, and signature fields. An entry takes totally 24 bytes for storing the required information. For designing the data structure to represent an entry, we consider that in Local Memory it fetches 4 bytes in a read operation. Consequently the size of each entry must be multiple of 4 bytes and finally we decide to assign 24 byte for each entry. The storage size for Local Memory on each microengine is 2560 bytes (refers to Table 2) so it can store

more than 100 different kinds of signatures and it is sufficient for nowadays P2P protocols.

Description for each field is as follows:

- a. **Flag:** It takes 1 byte to store the flag. This field is used to indicate whether the entry is meaningful or not. If the signature in a certain entry is correct or meaningful, the corresponding flag field will be set to “0x96”. So before comparing packet payload with the signature table, it is needed to examine the flag field. Then we could continue the following processing if the flag field is set, otherwise the system will deal with the next entry and ignore the following fields of the current entry.

Such configuration is convenient. Considering a condition that some signatures of P2P protocols or P2P applications have been filled into the signature table, under certain situation it is open to use these P2P protocols or P2P applications, and the system administrators just close the flag field instead of deleting the whole entry. Then in case sometime it is necessary to filter out these P2P, all a system administrator needs to do is turn on the flag fields instead of adding new entries.

- b. **P2P protocol:** It takes 1 byte to store the P2P protocol field. This field records what P2P protocol the following signature belongs to. For example, BitTorrent or eDonkey.
- c. **IP protocol:** It takes 1 byte to store the IP protocol field. This field records what packet's type when there is a signature inside. This field should be either TCP or UDP.
- d. **Signature length:** It takes also 1 byte to store the signature length field. This field records signature length of the current entry. In our system, signature length value helps us determine whether the signature string terminates or not.

In our system, we use loop to implement comparison process between signature table and packet payloads. During the loop comparison, we use two conditions, one is whether every byte of signature is same as the sequent packet payload, and another

is the length of this signature. We will describe the comparison details later in this chapter.

- e. **Signature:** It takes 20 bytes to store the signature field. In this field, it records signature corresponding to certain P2P protocols or P2P applications. Since different signatures have different signature lengths and we decide to use 20 bytes for signature field so that it has enough room to accommodate various signatures.

Since it may have more than one type of signature for a P2P protocol or P2P application, it may happen that multiple entries are using to represent the same P2P name in P2P protocol fields.

Since not all P2P protocols or P2P application has same length of signatures, the rest of signature fields will be filled with “0xff”. Because P2P traffic signatures have some specific string to indicate P2P protocols or applications, so to set rest portion of signature field with “0xff” would not lead to ambiguity.

Figure 4.8 shows a signature table example.

```
.init signature_table 0x96080B0B 0x64313a61 0x64323a69 0x643230ff 0xffffffff 0xffffffff \
0x96080B0B 0x64313a72 0x64323a69 0x643230ff 0xffffffff 0xffffffff \
0x96080614 0x13426974 0x546f7272 0x656e7420 0x70726f74 0x6f636f6c \
0x96070601 0xe3ffffff 0xffffffff 0xffffffff 0xffffffff 0xffffffff \
0x96070601 0xc5ffffff 0xffffffff 0xffffffff 0xffffffff 0xffffffff
```

Figure 4.8 A signature table example.

4.4.2 Dynamically Configured Signature Table

We discuss dynamic configuration of signature table in this section. In many conditions it may be necessary to update the signature table. No matter it is adding additional signature strings for new P2P protocols or disabling certain signature checking to let the corresponding P2P protocols traffic to pass according to network management policies.

For the sake of simplifying managing signature table, we design a field called flag when

developing the data structure for signature table. The purpose of a flag field is to act like a guard. It tells the system whether the following signature string is legal or not. If the following signature string is legal and then the system will process it, otherwise the system will proceed to next entry and skip the current entry.

We fill flag field with value “0x96” when adding new signature strings for P2P protocols or P2P applications to indicate that the following signature string is legal. So, for the system, to examine whether it is necessary to check to following signature string for detecting P2P traffic, it just examines whether the flag field is equal to “0x96” first.

For a network manager, it is convenient because he just turns off certain flag fields instead of deleting all the entries when certain P2P protocols or P2P applications are enabled/disabled dynamically.

If there is a new signature string of a new P2P protocol or P2P application is found by reverse engineering or other trusted ways, a network manager can add this new signature string directly to the system by enter the corresponding ASII values of alphabets of the signature string into the system. This adding procedure can be done by network mangers without needing help from system developers.

On the other hand, if any signature string of P2P protocol or P2P application is out of date or no longer needed. Network manager can turn off the flag field of such P2P protocols or P2P applications, set others value except 0x96 to flag field to other values, or delete it directly. This deletion procedure also can be done by network managers without any help from system developers.

Therefore, the system we proposed can be configured dynamically by system administrators. Further it is also flexible to decide which P2P protocols or P2P applications will be filtered by the system.

Chapter 5 Numerical Results

Like many hardware products, before producing real product it is necessary to estimate its performance. In this chapter we describe how we perform the system simulation to get the numerical result and explain why we use simulation instead of field test, and introduce the simulator we used.

5.1 Introduction of Simulator

In this thesis we build our system with IXP2400 on IXDP2400. Because another SRAM memory card is not available and hard to get, so we proceed simulation instead of field test.

We use simulator accompanied with Intel Internet Exchange Architecture (IXA) Software Development Kit 4.1. With IDE, DevWorkbench, from Intel SDK 4.1, we can not only create a new task for Intel Network Processors IXP2XXX series but also take advantage of simulator accompanied with DevWorkbench to simulate the performance of the tasks.

Such design is convenient for a system developer, because using simulator accompanied with DevWorkbench the system developer can evaluate the task he created directly in the same programming environment without needing to prepare other simulator tool and extra effort in installation. Since the result from the simulator is generated from Intel Internet Exchange Architecture (IXA) Software Development Kit 4.1 so it is reliable and trustable for system developer to use because it comes with Intel Network Processors.

After we developed our system program for this thesis, we also use the simulator mentioned above to simulate the performance of our system. The following are the simulation results.

5.2 Simulation Results

In the simulation, the parameters we used are explained as follows:

a. **Packet size:**

1. For a normal (non P2P traffic) packet, in order to obtain system performance in the worst case, we set packet length with minimal size, 64 bytes. And inside the packet payload of normal packet there is no signature string included.
2. For P2P packets, we set a BitTorrent protocol packet to stand for P2P traffic. However, there is not only one P2P protocol as BitTorrent protocol in real world. But for testing system performance it does not matter for incoming packets to use only one P2P protocol type.

In order to record BitTorrent packet as real as we can, we use Ethereal [36] to capture packets transmitted in real world when download files through BitTorrent protocols. Also, for the popular BitTorrent protocol application on Microsoft Windows, BitComet, it uses not only signature string mentioned by [2], [9], [18], [35], but also another signature string devised by BitComet company. In order to detect the traffic produced by BitComet we refer to the signature used in ipp2p [37] module for iptables in Linux for detection purpose.

As a real BitTorrent packet transmitted in Internet, we set BT packet with length 122 bytes.

- b. **Processor speed:** for each microengine, the processing lock rate is 600MHz.
- c. **Input interface:** IXD2410 4Gbs interface card for Intel IXDP2400 Advanced Development Platform.
- d. **Receive buffer size:** 256 bytes.

e. **Transmit buffer size:** 256 bytes.

During simulation, we set various conditions to measure the system performance. First we construct 5 entries for the signature table. Within these 5 entries, we fill in with top 2 popular P2P protocols or P2P applications in Taiwan. According to Figure 2.5, it clearly indicates that the BitTorrent and eDonkey protocols are most popular P2P protocol in Taiwan and take most of P2P traffic volume. Therefore, in the first test environment we fill it with 5 related signature strings into signature table to filter out BitTorrent and eDonkey traffic.

In the second test environment, we construct our system to accommodate most popular P2P protocols in the world. For this purpose, we construct our system with 60 different signature strings for eDonkey protocol, BitTorrent protocol, FastTrack protocol, Gnutella protocol, MP2P protocol, DirectConnect protocol, Souseek protocol, Ares protocol, and EarthStation protocol. Those P2P protocols are the most popular on the rank. And to filter out those 9 different P2P protocols we set our signature table with 60 entries to cover 60 different signature strings.

The following are some figures to show the performance. Figure 5.1 shows the performance in the first test environment and it measures how many cycles needed to process a packet under different distribution of P2P traffic.

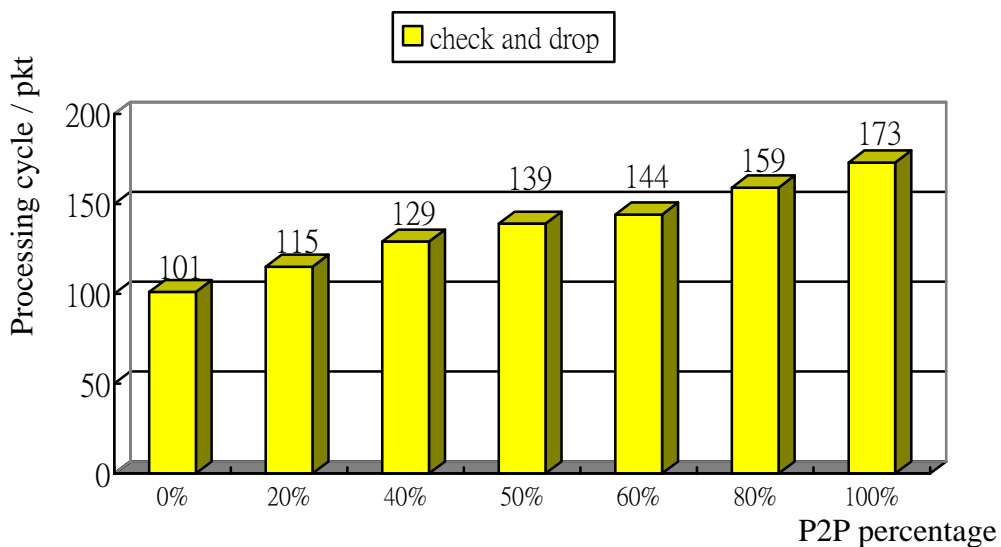


Figure 5.1 Processing cycles per packet with 5 P2P signatures.

In Figure 5.1, we can observe that under the first environment our system takes 101 clock cycles to process a packet with none of incoming packets being P2P traffic. In other words, all incoming packets are 64 bytes minimal size normal Ethernet packet. With 20% of incoming packets being P2P packets and other 80% being non-P2P traffic, our system takes 115 clock cycles to process an incoming packet. With 40% of incoming packets being P2P traffic and 60% being non-P2P traffic, our system takes 129 clock cycles to process an incoming packet. With 50% of incoming packets being P2P traffic and 50% are non-P2P traffic, our system takes 139 clock cycles to process an incoming packet. And our system takes 144 clock cycles, 159 clock cycles, and 173 clock cycles to 60% P2P traffic and 40 % non-P2P traffic, 80% P2P traffic and 20% non-P2P traffic, and 100% all P2P traffic, respectively.

Figure 5.2 shows the performance regarding the number of packets that our system can process per second.

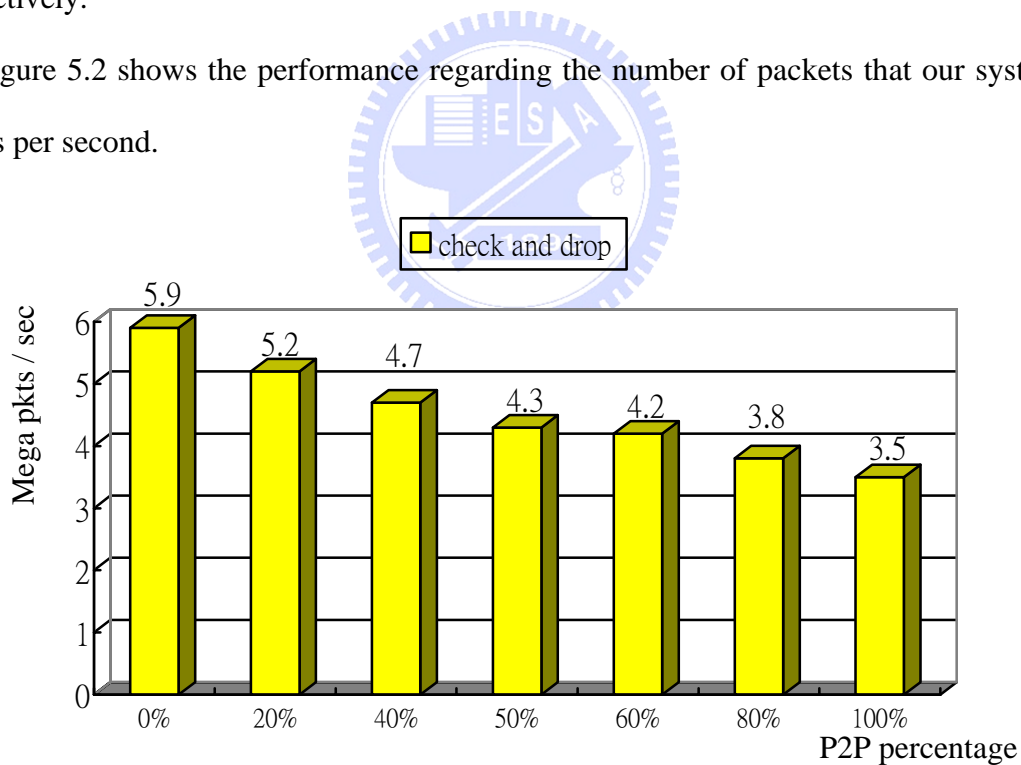


Figure 5.2 Number of mega packets per second with 5 P2P signatures.

We could observe the performance under the first test environment. Our system can process 5.9 mega packets per second if the incoming packets are all non-P2P traffic, and 5.2

mega packets per second with the 20% of incoming packet being P2P traffic and other 80% traffic being non-P2P packets, and 4.7 mega packets with 40% of incoming packets being P2P traffic and other 60% packets being non-P2P traffic, and 4.3 mega packets with 50% of incoming packets being P2P traffic and other 50% packets being non P2P traffic, and 4.2 mega packets, 3.8 mega packets, 3.5 mega packets to 60% of incoming packets being P2P traffic, 80% of incoming packets being P2P traffic, and 100% of incoming packets being P2P traffic, respectively.

In the second condition, because there are 60 entries in the signature table, the location of signature string found may affect the system’s performance. For average case, we set simulation environment with signature string found in the middle of signature table, the 30th entry. And for the worst case, we set simulation environment with signature string found at the last entry of signature table, the 60th entry.

In Figure 5.3 it shows the average case of the second test environment.

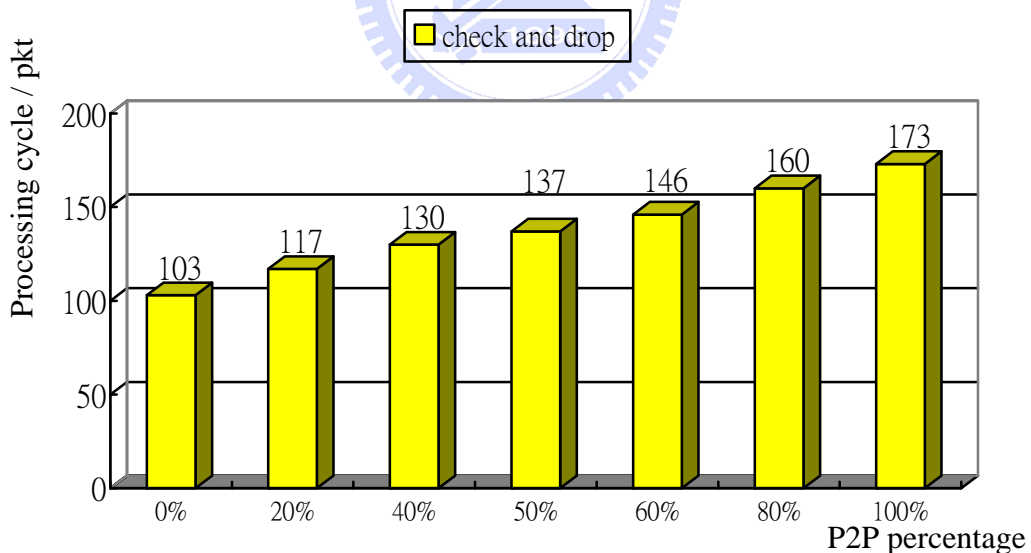


Figure 5.3 Processing cycles per packet with 60 P2P signatures in average case.

In Figure 5.3, we can find that our system takes 103 clock cycles to process a packet under the second test environment with all incoming packets being non-P2P traffic, and our

system takes 117 clock cycles to process a packet with 20% of incoming packets being P2P traffic, and our system takes 130 clock cycles to process a packet with 40% of incoming packets being P2P traffic, and our system takes 137 clock cycles to process a packet with 50% of incoming packets being P2P traffic, and takes 146 clock cycles, 160 clock cycles, and 173 clock cycles to process a packet with 60%, 80%, and 100% of incoming packets being P2P traffic.

Also, we can measure our system performance regarding how many packets per second our system can process, and we show the result in Figure 5.4.

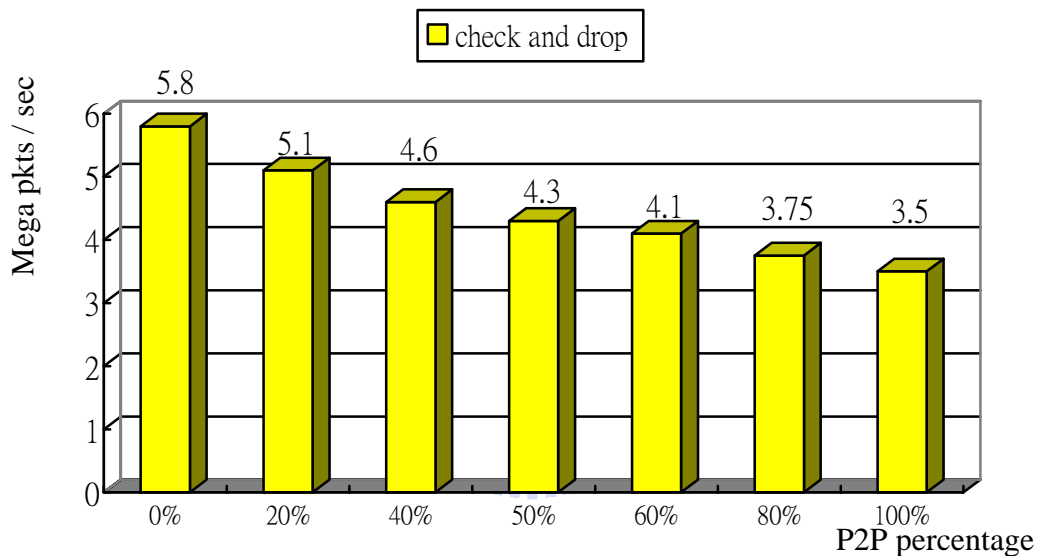


Figure 5.4 Number of mega packets per second with 60 P2P signatures in average case.

In Figure 5.4, we can observe that our system can process 5.8 mega packets per second with all incoming packets being non-P2P traffic, and 5.1 mega packets per second with 20% P2P traffic and 80% non-P2P traffic, and 4.6 mega packets per second with 40% P2P traffic and 60% non-P2P traffic, and 4.3 mega packets per second with 50% P2P traffic and 50% non-P2P traffic, and 4.1 mega packets per second, 3.75 mega packets per second, and 3.5 mega packets per second with 60% P2P traffic and 40% non-P2P traffic, 80% P2P traffic and 20% non-P2P traffic, and 100% all incoming packets being P2P traffic.

In the worst case of the second test environment, the simulation result is shown as Figure 5.5.

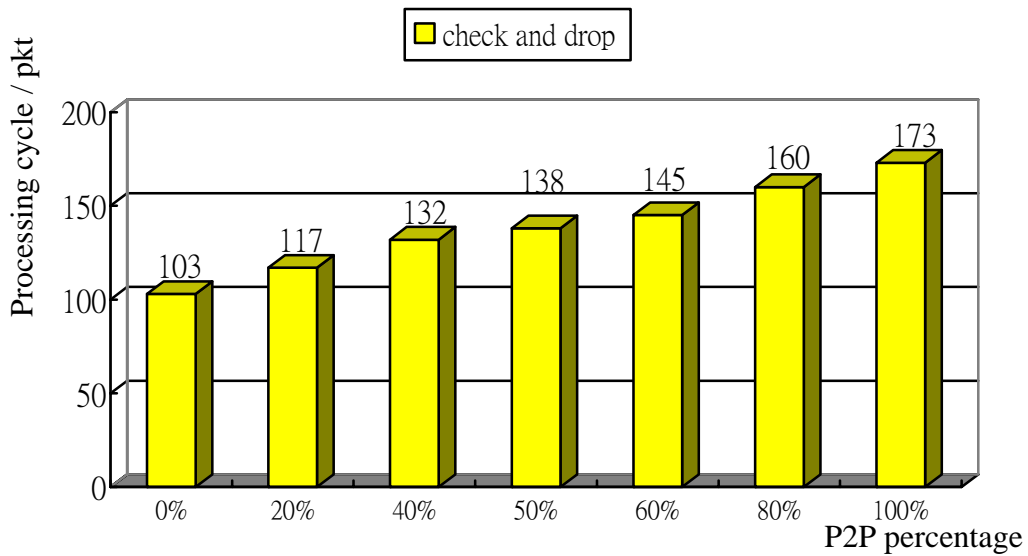


Figure 5.5 Processing cycles per packet with 60 P2P signatures in worst case.

As Figure 5.5 shows, our system takes 103 clock cycles to process a packet in the worst case with all incoming packets being non-P2P traffic, and takes 117 clock cycles to process a packet with 20% of incoming packets being P2P traffic and others being non-P2P traffic, and takes 132 clock cycles to process a packet with 40% of incoming packets being P2P traffic and others being non-P2P traffic, and takes 138 clock cycles to process a packets with 50% of incoming being P2P traffic and others being non-P2P traffic, and takes 145 clock cycles, 160 clock cycles , and 173 clock cycles to process a packet with 60%, 80%, and 100% of incoming packets being P2P traffic.

Finally, we measure our system regarding how many packets can be processed per second in the worst case under the second test environment, as shown in Figure 5.6.

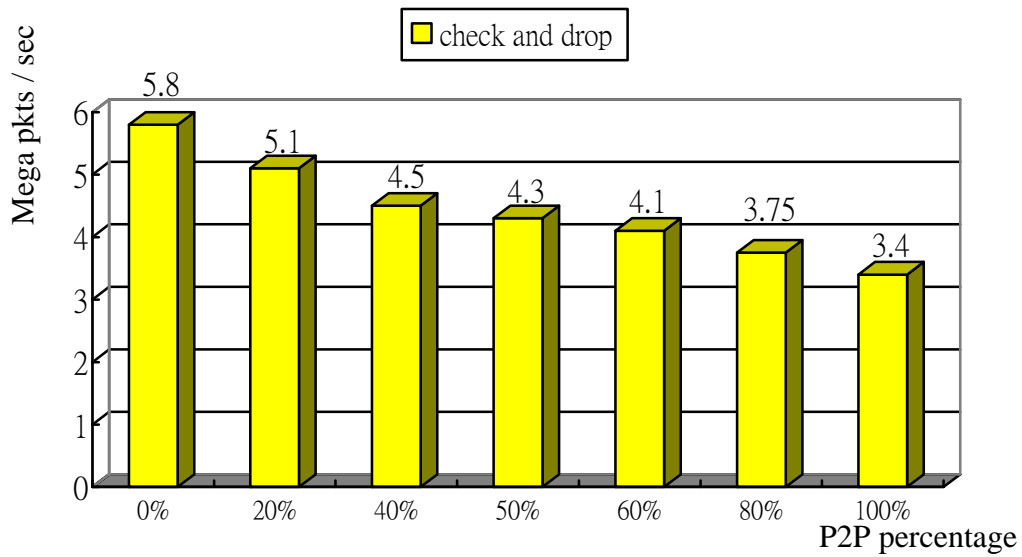


Figure 5.6 Number of mega packets per second with 60 P2P signatures in worst case.

As Figure 5.6 shows, our system can process 5.8 mega packets per second in the worst case of the second test environment with all incoming packets being non-P2P packets, and can process 5.1 mega packets per second with 20% of incoming packets being P2P packets and 80% of incoming packets being non-P2P packets, and 4.5 mega packets per second with 40% P2P packets and others being non-P2P packets, and 4.3 mega packets per second with 50% P2P packets and 50% non P2P packets, and 4.1 mega packets per second, 3.75 mega packets per second, and 3.4 mega packets per second with 60% P2P packets and 40% non P2P packets, 80% P2P packets and 20% non P2P packets, and 100% P2P packets.

5.3 Comparison and Discussion

From the data measured above, we can observe that our system has a stable performance under different environments.

Owing to the special-purpose hardware of pipelined architecture of network processor, our system can handle various P2P protocols or P2P applications filtering without losing

performance.

In our system, we take advantage of Local Memory which is an on-chip memory of each microengine. With this inborn superiority, it saves lots of memory access latency while comparing packet payload inside incoming packets with those signature strings in the signature table. For a system, data accessed from cache has less access latency than data accessed from main memory, therefore, we consider Local Memory cache and it did work well.

Since entries of signature table are not as many as that in a regular routing table, we can put all entries of signature table into Local Memory of each microengine though the memory size of Local Memory is limited. Another reason that supports us to do this is signature lookup for each incoming packet being considered as an independent process. In other words, signature lookup for each incoming packet does not influence other examinations in other microengines. Unless updating the signature table, there is only read operation for each microengine to access signature table in Local Memory.

Acting like cache, low memory access latency for Local Memory, independent processing for each microengine allow multiple microengines to compare packet payload with signature table, special-purpose designed hardware and content pipeline programming model, those advantages allows us to improve the whole system performance.

With macroscopic point of view, our system is developed by software with different programming type and accompanied with special-purpose designed hardware. In industries, there are also several similar products act with same responsibility. For example, the P-Cube Company [39], and the ASCENVISION Company [40], they provide some similar purpose products to handle P2P traffic packets. Since certain product devices have same functionality and even been equipped with other different functionalities, but these products are all quite expensive for customers.

Here we take products from industries for instance, the Ipoque Company [41], provides

related products to block P2P traffic packets. In their product lists, there are PRX-2000 including P2P-detected module at 12,390 Euros (€) and PRX-1000 including P2P-detected module at 8,990 Euros (€) and throughput are 500,000 packets/s and 30,000 packets/s, respectively.

So it shows clearly we develop a cost-effective and high performance system with assistance of Intel Network Processor from the simulation result shown above. In our thesis, we utilize microengines in IXP2400 and take advantage of specific programming model and our system performance can accommodate the detection and filtering of P2P traffic in enterprise as well as SOHO networks.



Chapter 6 Conclusion and Future Work

In this thesis, we proposed a high performance P2P packet filtering scheme based on pipe-lined architecture and multi-thread mechanism. We also implement the scheme using the state-of-the-art network processor.

P2P traffic occupies a significant portion of Internet traffic nowadays. It not only occupies a stable percentage of Internet traffic but also grows with the time. With flow rate of P2P traffic grows, certain problems appear, and this leads to the motivation of filtering out P2P traffic.

Owing to properties of P2P protocols, peers (or nodes) download files while they are uploading. So for an ADSL access link, its upload bandwidth may be occupied by P2P protocols' upload traffic and leads to other users unable to use network resource as usual. Such situation may happen in corporation or SOHO office if the network backbone in those organization is not large enough to accommodate many users using P2P protocol to download contents. For network managers in those organizations, they may want to ensure some critical applications can be guaranteed a bandwidth for certain performance.

On the other hand, P2P protocol acts differently from traditional network service architecture. In the traditional network service architecture there is a log file to keep the connection records, but there is no log file using P2P protocols to transmit contents. It results in another problem, blabbing secrets.

To an administrator of a company or a corporation, it is necessary to avoid being blabbed secrets from internal staffs. According to this reason, administrators may restrict the usage of P2P protocols in office environment.

Contents transmitted in Internet through P2P protocols are often invisible to others, because peers download a file from or upload file to others peers directly. As long as a peer is

willing to provide file so that other peers can find the required content easily. Therefore, file transmitted through P2P protocol may face the issue of copyright or intellectual property.

In order to avoiding tort, network managers may limit the usage of P2P protocols. According to three reasons mentioned above, sometimes it is necessary to detect and filter P2P traffic to ensure network quality, protect personal secrets, and avoid tort.

Consequently, in this thesis we take advantage of Intel Network Processor IXP2400 with Intel Internet Exchange Architecture (IXA) Software Development Kit 4.1 to develop a system that filters P2P protocols or applications specified by network managers, and it overcomes the performance needed to examine enormous number of packet in Internet, and the results demonstrated in Chapter 5 is satisfactory.

In our thesis, we implement system with signature-base detection algorithm. For more accurate judgment for P2P traffic, we can add extra flow-based algorithm into the system.

For flow-based detection algorithm, we consider to add five-tuples to represent a flow. The five-tuples consist of source IP address, destination IP address, source protocol port number, destination protocol port number, and the protocol used. When system discovers and decides a packet from Internet belonging to P2P traffic, the five-tuples of the packet are recorded into a table and given it a flow identification number. Sequentially, the following packet with this flow identification number matched will be considered as P2P traffic directly without check the signature table again.

Also considering false positive of system, we can add extra timer or dirty bit to avoid blocking wrong network flow, for example, blocking download by HTTP or SMTP.

In the future, we can add more accurate way to detect P2P traffic because newer P2P application may provide P2P users to transmit their traffic with data-encryption. Packet payloads with encryption can not be detected directly by signature-based system. To solve this issue, it is necessary to discover more accurate characteristics that represents P2P traffic but also performs additional analysis of network traffic to define P2P traffic behavior e.g.

flow analysis.

Besides our system, we can connect with database outside which maintain the latest P2P protocols or P2P applications signatures, and updating the signature table in Local Memory periodically. Such automatic update allows our system to detect and filter P2P traffic accurately and promptly. From the point of view of database outside, it can be maintained by expert system to extract the most precise P2P patterns and this may make the system more robust.



Reference:

- [1] <http://www.spinellis.gr/pubs/jrnl/2004-ACMCS-p2p/html/AS04.html>
- [2] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang, “Accurate, scalable in-network identification of p2p traffic using application signatures”, WWW2004, New York, New York, USA, May 17-22, 2004.
- [3] Gnutella, <http://www.gnutella.com/>
- [4] The Gnutella Protocol Specification v0.4, Clip2 <http://www.clip2.com>
protocols@clip2.com
- [5] Thomas Karagiannis, Andre Broido, Nevil Brownlee, kc claffy, and Michalis Faloutsos, “Is P2P dying or just hiding”, Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE, pp. 1532 - 1538 Vol.3, Nov.29 - Dec. 3, 2004.
- [6] Bram Cohen, “Incentives Build Robustness in BitTorrent”, May 22, 2003
- [7] BitTorrent, <http://www.bittorrent.com/>
- [8] <http://www.bittorrent.org/index.html>
- [9] Holger Bleul and Erwin P. Rathgeb, “A simple, efficient and flexible approach to measure multi-protocol peer-to-peer traffic”, ICN 2005, LNCS 3421, pp. 606–616, Springer-Verlag Berlin Heidelberg 2005.
- [10] Internet2 Netflow, Weekly Reports, data sets 02/18/2002 to 09/13/2004,
<http://netflow.internet2.edu/weekly/longit/long.dat>
- [11] The Abilene Network homepage, 2004, <http://abilene.internet2.edu>
- [12] CacheLogic, <http://www.cachelogic.com/>
- [13] Morpheus, <http://morpheus.com/index.asp>
- [14] Limewire, <http://www.limewire.com/english/content/home.shtml>
- [15] BearShare, <http://www.bearshare.com/>

- [16] XoloX, <http://www.xolox.nl/>
- [17] DC++, <http://dcplusplus.sourceforge.net/>
- [18] Thomas Karagiannis, Andre Broido, Nevil Brownlee, k claffy, and Michalis Faloutsos, “File-sharing in the Internet: A characterization of P2P traffic in the backbone”, November 2003.
- [19] KaZaa, <http://www.kazaa.com/us/index.htm>
- [20] Grokster, <http://www.grokster.com/>
- [21] iMesh, <http://www.imesh.com/>
- [22] eDonkey2000, <http://www.edonkey2000.com/>
- [23] eMule, <http://www.emule-project.net/home/perl/general.cgi?l=16>
- [24] BitComet, <http://www.bitcomet.com/>
- [25] Intel Network Processor IXP2400, <http://www.intel.com/design/network/products/npfamily/ixp2400.htm#diagram>
- [26] Intel IXP2400 Network Processor Product Brief, <http://www.intel.com/design/network/prodbrf/279053.htm>
- [27] Erik J. Johnson, Aaron R. Kunze, IXP2400/2800 Programming – The Complete Microengine Coding Guide, INTEL PRESS, April, 2003.
- [28] Monta Vista, <http://www.mvista.com/company/>
- [29] Douglas E. Comer, Network Systems Design using Network Processors, Pearson Prentice Hall, 2005.
- [30] Intel IXDP2400 and Intel(R) IXDP2401 Advanced Development Platforms Product Brief, <http://www.intel.com/design/network/prodbrf/253840.htm>
- [31] Intel IXP2400 and IXP2800 Network Processor Programmer’s Reference Manual, October, 2004.
- [32] Intel Internet Exchange Architecture Software Development Kit 4.2, <http://www.intel.com/design/network/products/npfamily/sdk.htm>

- [33] Intel IXP2400 Network Processor Hardware Reference Manual, October, 2004.
- [34] Intel IXP2XXX Product Line of Network Processors Development Tools User's Guide, November, 2004.
- [35] Angelo Spognardi, Alessandro Lucarelli, and Roberto Di Pietro, "A Methodology for P2P File-Sharing Traffic Detection", Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005, pp. 52 – 61, Second International Workshop on July 21, 2005.
- [36] Ethereal, <http://www.ethereal.com/>
- [37] ipp2p, <http://www.ipp2p.org/>
- [38] Intel Core Duo Processors, <http://www.intel.com/products/processor/coreduo/index.htm>
- [39] The P-Cube Company, <http://www.p-cube.com/index.shtml>
- [40] The ASCENVISION Company, <http://www.ascenvision.com/>
- [41] The Ipoque Company, <http://www.ipoque.com/>

