Santa Clara University COEN 233 Computer Networks

Deep Packet Inspection Performance Optimization

Team 2:
Zhuangzhi (George) Duo
Hari Khanal
Geetha Srigiriraju

Professor Ming-Hua Wang

Preface

With global network traffic (e.g., voice, mobile data, IPTV, etc.) predicted to increase substantially through this decade, network operators, network equipment providers, and embedded suppliers all share a great interest in ensuring optimal performance of network equipment before its deployment in an active network.
Sharing this common interest has spawned in recent years the development of tools to optimize performance and improve security. One such solution is Deep Packet Inspection (DPI), which has emerged as a technology that can help mitigate network risks and improve operational performance.
The purpose of this paper is to summarize how we have optimized DPI using techniques that enhance the operational performance of Regular Expressions and introduce a SNORT in Network Intrusion Detection mode to monitor the network traffic and analyze it against a defined rule set.

Acknowledgements

We would like to extend our deep and sincere gratitude to our program instructor, Professor Dr. Ming –Hwa Wang, Department of Computer Engineering. His profound knowledge combined with his continued guidance and inspirations have helped shape our attitude towards this project.
We would also like to extend our gratitude to Santa Clara University to provide us with a challenging atmosphere to continue our education.
Our gratitude also extends to our family members whose support and help throughout our project phase is most indebted.

Table of Contents

DEEP PACKET INSPECTION PERFORMANCE OPTIMIZATION

COEN 233 – Computer Networks

By

Zhuangzhi Duo

Hari Khanal

Geetha Srigiriraju

1 Abstract

Deep packet Inspection (DPI) is one the key component of a Network Intrusion Detection System (NIDS) and it compares packet content against a set of rules written in Regular Expressions. The techniques and processing costs involved in Deep Packet Inspection are extremely expensive.

In this paper, we are using HMAC-SHA-1 algorithm to process payload before passing it into regular expressions. State-of-the-art system, Snort is being used to compare packet content to a set of rules. This will save a lot of states, thereby improving the processing costs and performance of the DPI system.

2 Introduction

DPI combines the functionality of an Intrusion Detection System (IDS) and an Intrusion Prevention System (IPS) with a traditional stateful firewall. This combination makes it possible to detect certain attacks that neither IDS/IPS nor the stateful firewall can catch on their own. Previously port-based traffic classification was employed which is now deemed imprecise due to the large amount of applications using non-standard ports in order to escape network limitations. Due to the dynamic nature of Internet applications these days, the most used technology for traffic classification is DPI. In case the application can tolerate some compromises in terms of accuracy (such as many measurement-based tasks) and in presence of normal traffic, the processing cost can be greatly reduced while improving the classification

precision, making DPI suitable also for high-speed networks Deep Packet Inspection (DPI) systems have been increasingly performed on dedicated hardware, as an attempt to speed up the packet processing for high speed links. This is mainly caused by the current demand for CPU-intensive processing required by regular expression functions, which investigate the packet payload trying to match patterns of application signatures. The DPI is a core component for many systems plugged in the network including proxies, packet filters, sniffers, IDS, and IPS. Network components use DPI as an essential inspector where it is applied in different layers of the OSI model. Unlike the early beginnings of using DPI where it was applied in only one layer depending on the header (e.g., proxies and firewalls etc.), nowadays, layer-independent attacks force us to inspect attacks in all the layers. According on the intrusion detection literature, efforts to obtain fast implementations can be categorized into two main categories: (1) design of an efficient data structure with optimized memory access rate, and (2) design of high throughput algorithm to process intruder signature. To support increasingly complex services, regular expression (regex) and DFA (Deterministic Finite Automata) have been used to replace string by these systems due to its higher expressiveness and flexibility.

In our project, we are proposing a Fast Deep Packet Inspection system to accelerate signature matching. One theoretical approach is that instead of directly dealing with regular expression interpreter, we find out a way to increase the performance exponentially by using Thompson NFA. And then we can convert the NFA to DFA for final pattern matching by using algorithm from Aho Ullman. Another approach in our practice is to use the snapshot and hashing algorithm to shorten the signature patterning duration time, which can be quantified in our system more straightforward.

Since most of commercial IDS product is based off Snort, our proposal is to use Snort system as the baseline to compare the performance difference by using our fast DPI approach. By adding the enhancement mentioned above, we are looking forward to the performance optimization.

Section- 3 and Section-4 of our paper explains the related work we have researched and the DPI improvements suggestions we have proposed.

3 Project Overview

DPI is massively used in firewalls, IDS and other security related applications, which are known to have scalability issues because of their processing requirements. Most researched papers study regex matching as a topic in automata theory [6]. String matching techniques such as Wu-Manber [8] algorithms have been the foundation for many signature-based detection engines. Regular expressions increase the capability and maintainability of threat detection systems by increasing the flexibility of threat definitions. Schaelicke et al. [9] characterized the performance of the Snort software on general-purpose processors. They showed that their highest performance test system could only simultaneously handle 217 payload rules on a network running at a 100 Mbps rate. Cascarano et al [2] avoided optimizing solutions for corner cases that might be important for security applications. They failed to recognize encrypted /tunneled traffic and had extreme sensitiveness to the signature dataset. Wang et al [1] used a Length based matching (LBM) for accelerating regex matching. LBM had its limitations where probability of having to execute accurate matching was assumed to be low. Sailesh et al [4] introduce the concept of Content Addressed Delayed Input DFA (CD2FA) which provides compact representation of regular expressions.

4 Solutions

We are going to propose DPI performance improvement in terms of architecture and DPI engine optimization

4.1 Architecture Solution

Regarding architecture, we inherit snapshot-based classification concept and enhance this approach with the hashing solution. The idea is when the large file is transferred in the network, instead of buffering the whole file and parsing the

contents into DPI engine, we take first N bytes snapshot from file, and do CRC-32 bits checksum, combining with first 8 significant leading bytes and 8 trailer bytes which is the length of hash data to generate the 20 bytes HMAC key.

Fast DPI Key = Significant Leading Bytes + CRC-32 Checksum + Hash Size Trailer

This HMAC fast DPI key will be used for two purposes.

The first purpose is to feed this key into the HMAC-SHA1 function to generate the 160 bits unique signature message digest (MD). This MD will become the input for DPI engine for fast IDS pattern match by computation of the MD for real traffic and direct comparison.

MD = HMAC-SHA1(Hash Data, Fast DPI Key)

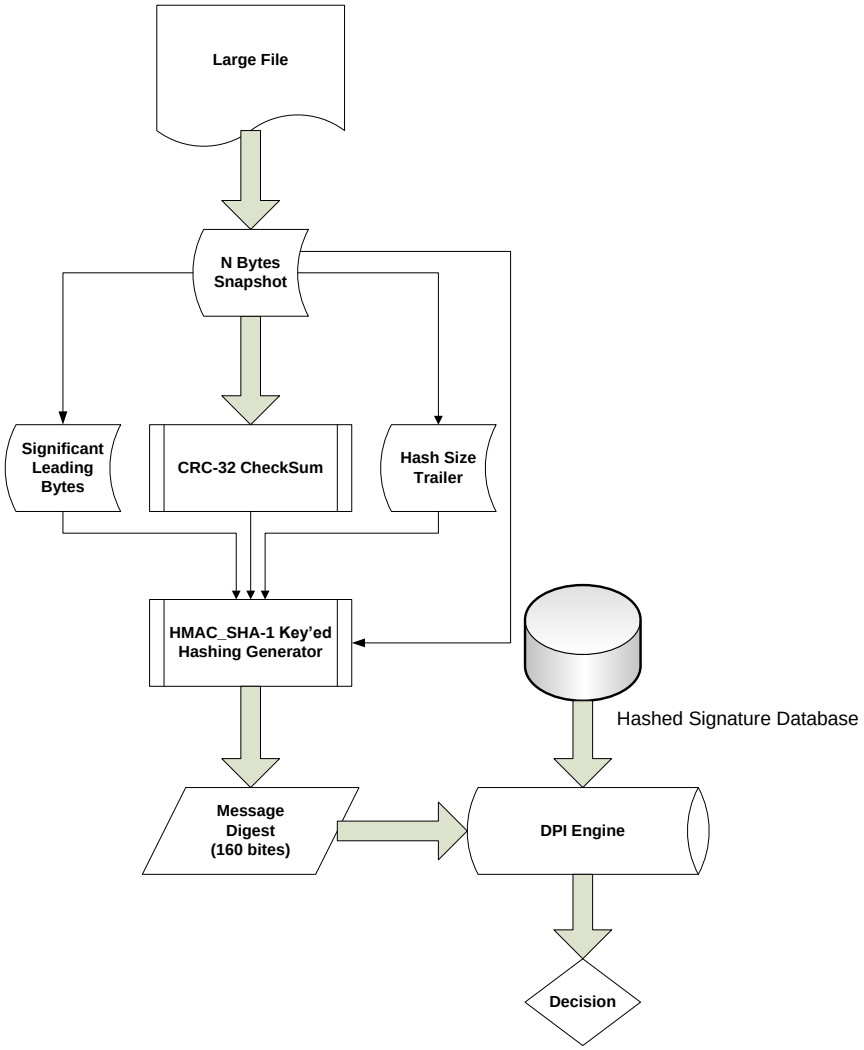

Fig1. File Snapshot and Hashing for architecture optimization

The second purpose is when direct hashed signature matching does not succeed; the fast DPI key will be used as the hash index to accelerate the signature matching table lookup process.

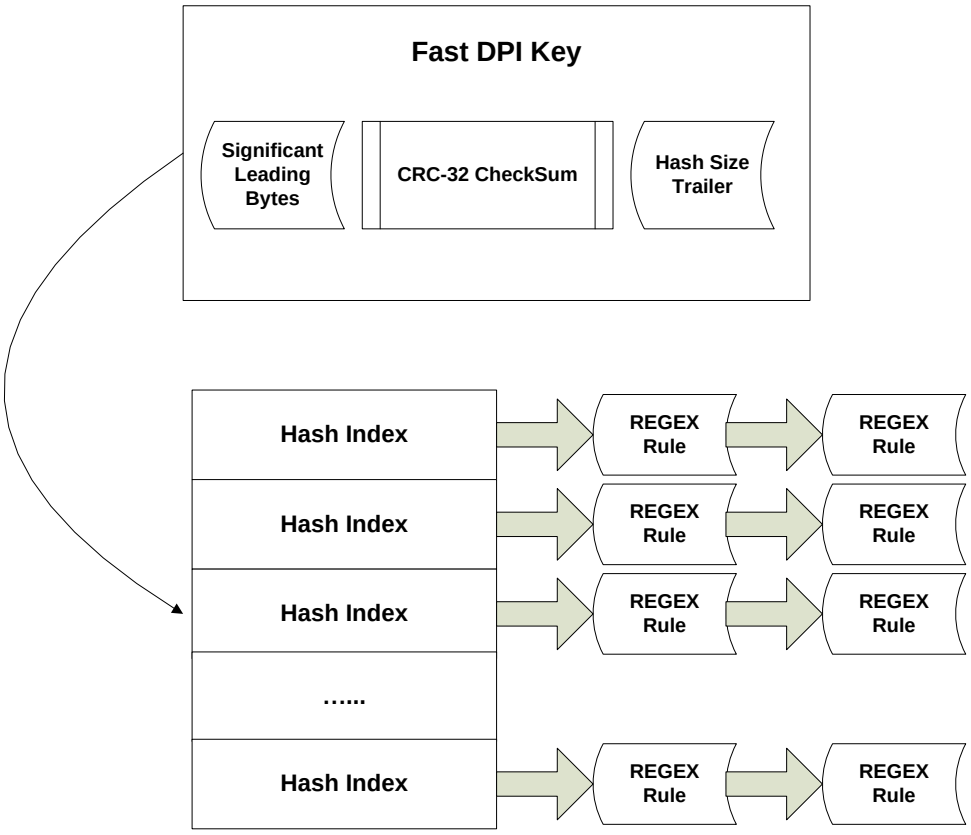

Fig2. Fast DPI Key Hashing for architecture optimization

Typically the IDS system signature is saved in the database in linear list as depicted in fig 3 below, the traversal time is proportional to the number of REGEX rules in the database.

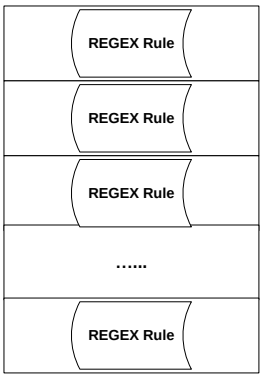

Fig3. Original Signature Organization Diagram

By using the Fast DPI Key hashing index, we can dramatically reduce the traversal time from O(N) to O(1) if hashing collision does not occur.

Theoretically, the N bytes payload requires $O(2^N)$ processing time in the worst case. By reducing the input sequence stream for DPI engine from N bytes to 160 bites (20 bytes), we can save tremendous CPU cycles and resources such as RAM. The efficiency can be increased 2^{N-20} times by only considering the input sequence length factor seemingly.

If HMAC-SHA-1 MD signature can be matched directly from signature database, the performance can be increased N times. If MD signature cannot be matched directly, the Fast DPI Key Hashing index method will help performance increase by M times, where M is the number of buckets in signature hash table.

4.2 DPI Engine Optimization Solution

Regular expressions are a notation for describing sets of character strings. When a particular string is in the set described by a regular expression, we often say that the regular expression *matches* the string.

Another way to describe sets of character strings is with finite automata. Finite automata are also known as state machines, and we will use "automaton" and "machine" interchangeably. The machine is called a *deterministic* finite automaton (DFA), if in any state each possible input character leads to at most one new state. The machine is not deterministic because it has multiple choices for the next state. Since the machine cannot peek ahead to see the rest of the string, it has no way to know which is the correct decision. In this situation, it turns out to be interesting to let the machine *always guess correctly*. Such machines are called non-deterministic finite automata (NFAs or NDFAs).

Generally people will directly deal with the regular expression interpreter to parse the regular expression. However standard interpreters which have been implemented by many languages and widely spread seem very performance costly. In the other words, they have become the bottleneck of IDS system implementation.

For example, Perl is a very popular interpreter, which is also adopted by Snort PCRE (Perl Compatible Regular Expressions). When it is used to match a 29-character string, it requires 60 seconds to process on the average using dual core 2.33GHz CPU. [10]This low efficiency drives us to find out the better practical way to do this

job. In our proposal, we have identified labeled Thompson NFA [11]which only requires 20 microseconds to match the same string.

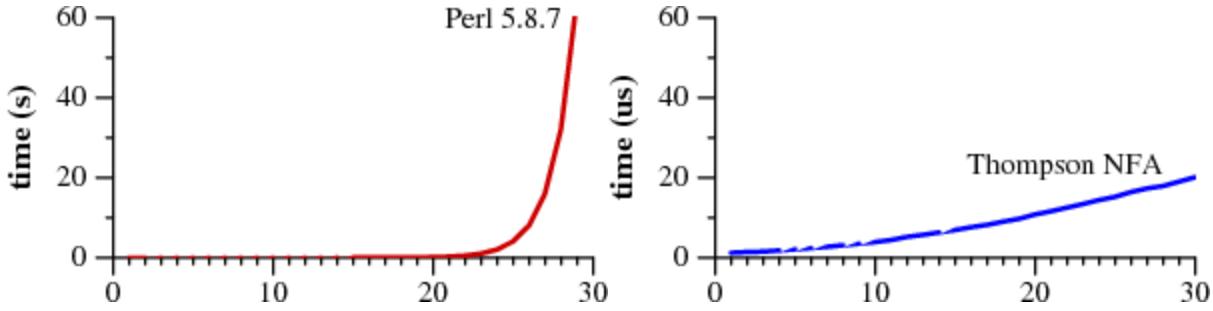

Fig4. Perl RE interpreter vs Thompson NFA

The Thompson NFA implementation is a million times faster than Perl when running on a miniscule 29-character string. The trends will continue when string length grows. The Thompson NFA handles a 100-character string in under 200 microseconds, while Perl would require over 10^{15} years!

Regular expressions and NFAs turn out to be exactly equivalent in power: every regular expression has an equivalent NFA (they match the same strings) and vice versa. There are multiple ways to translate regular expressions into NFAs. The method we are going to adopt was first described by Thompson in his 1968 CACM paper.

The NFA for a regular expression is built up from partial NFAs for each sub expression, with a different construction for each operator. The partial NFAs have no matching states: instead they have one or more dangling arrows, pointing to nothing. The construction process will finish by connecting these arrows to a matching state.

Consider the regular expression which uses recursive backtracking approach, it requires $O(2^n)$ time. In contrast, Thompson's algorithm maintains state lists of length approximately n and processes the string, also of length n, for a total of $O(n^2)$ time.

Once we get the regular expression converted to NFA, we can go further to convert the NFA to DFA by using "subset construction" algorithm [12]. We have noticed that DFAs are just a special case of NFAs. On the other hand the subset construction introduced above shows that for every NFA we can find a DFA which recognizes the same language. DFAs are more efficient to execute than NFAs, because DFAs are only ever in one state at a time: they never have a choice of multiple next states. Any

NFA can be converted into an equivalent DFA in which each DFA state corresponds to a list of NFA states.

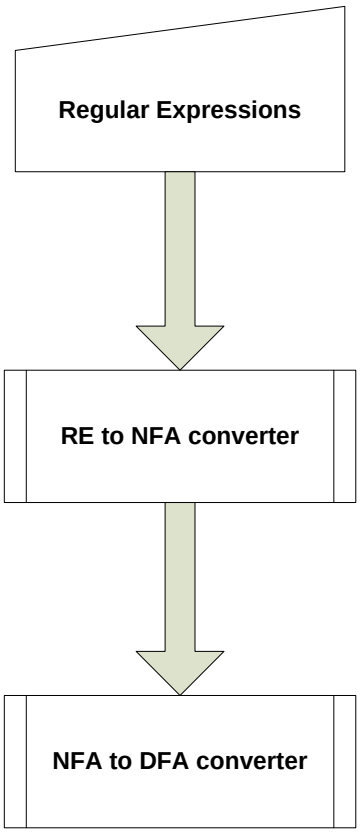

Fig5. DPI Engine construction

By building the DPI engine through the above conversions, we can increase the performance for general regular expression interpreter. Our experiment will focus on how much improvement we can obtain by using this optimization approach.

5 Hypothesis

Our proposal is based on two principles below:

- Our proposal only deals with the typical deployment scenario with recognized protocols. In the actual experimental system, we only inspect the HTTP traffic.
- Our proposal only deals with the typical traffic patterns and average cases. We are not evaluating the corner cases.

5.1 Positive Hypothesis

Our hypothesis is that we can increase the efficiency of DPI using Thompson Nondeterministic Finite-state Automata (NFA) and Aho Ullman Algorithm

5.2 Negative Hypothesis

DPI will not work on high-end network router that processes data rate at tera bits per second. So, DPI cannot work in very high speed backbone network.

6 Methodology

We are going to use the Snort to build the basic IDS system.

Snort® is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. Combining the benefits of signature, protocol, and anomaly-based inspection, Snort is the most widely deployed IDS/IPS technology worldwide. With millions of downloads and nearly 400,000 registered users, Snort has become the de facto standard for IPS.

Snort can be configured as Network Intrusion Detection System (NIDS) mode. With the most complex and configurable configuration, this mode allows Snort to analyze network traffic for matches against a user-defined rule set and performs several actions based upon what it sees.

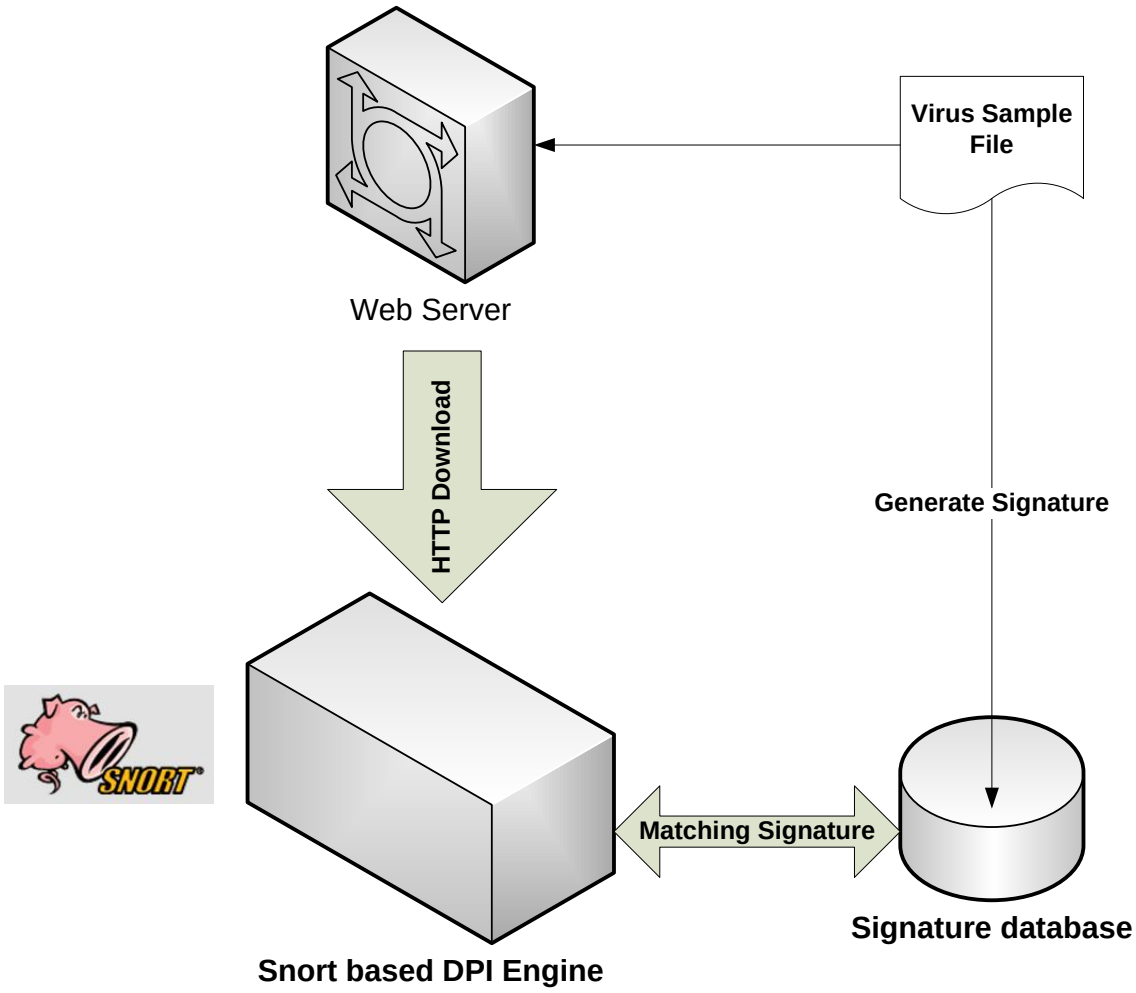

Fig6. Solution Methodology

The following steps are to be executed for verification.

Step 1.

Before doing the IDS scanning testing in NIDS mode, we will prepare for the virus sample files and host them in the web server for HTTP downloading. In IDS system, we create the virus signature using Snort rule definition.

Step 2.

Then we download the virus sample from the web server to IDS machine. The virus sample file should be scanned and analyzed by Snort NIDS. From generated log file and alert records, we can measure the performance for the basic Snort IDS system.

Step 3.

Then we feed the sample file into our DPI Engine. The snapshot of the sample file is used for HMAC-SHA-1 hashing to get the signature. Instead of hashing the whole file, we hash only the fixed first N bytes to compute hash much faster. The accuracy is still maintained in a certain level as well as perusing reducing hash computation time.

Step 4.

After receiving the result from Step 3, we can compare the performance data between original Snort system and our Fast DPI Engine to confirm if our proposal works or not and how much improvement we can gain from these solutions.

Step 5.

Optionally we can rebuild the IDS system using our DFA performance enhancement proposals. We can do snapshot and hashing for virus sample files to generate enhanced signature. And we can add Regular Expression to NFA conversion as well as NFA to DFA conversion into Snort modules.

Step 6.

Then we compare the signature against the signature stored in database server. For this project, we will use MySQL server running in Fedora 15 Linux machine. Furthermore, we will make it distributed system for scalability, and connect to MySQL server using IPv4 address.

Step 7.

After repeating the same testing using the same virus sample files, we can get another set of performance data from log file and alert records. By comparing the performance data with step 2, we can confirm if our proposal works or not and how much improvement we can gain from these solutions.

7 Implementation

In our project, we are implementing the packet inspection as follows.

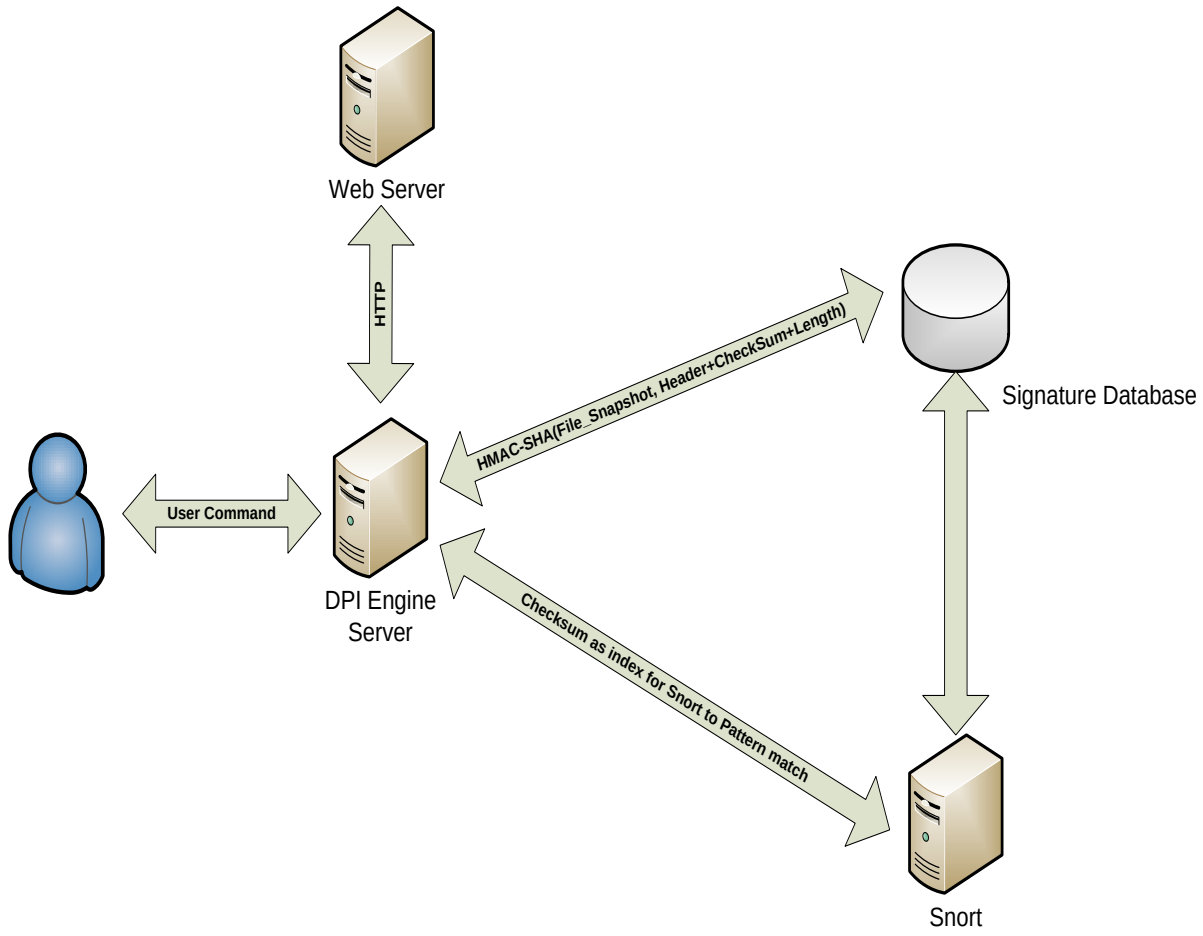

Fig7. Fast DPI Engine Architecture

A DPI server engine examines the packet payload as well as packet and frame headers. Using HMAC (Keyed Hash Message Authentication Code) and SHA-1 (Secure Hash Algorithm, a 160 bit MD signature of the payload is generated. After looking up the signature database, we can find out if the IDS pattern has been matched or not in the packet being processed.

7.1 CRC-32(Cyclic Redundancy Check) and Fast DPI Key

We use CRC-32 to generate the checksum for file snapshot. Then we take first 8 leading significant bytes from payload and hash size as 8 bytes trailer to concatenate all these three elements together to form the Fast DPI Key.

Fast DPI key can be used as hash key for HMAC-SHA-1 computation as well as hashing index for signature lookup.

7.2 HMAC SHA-1(Keyed Hash Message Authentication Code)

We use HMAC-SHA-1 to generate the MD signature of a file snapshot so that we can compare it against the signature stored in database. This will speed up the inspection process tremendously.

Comparing 160-bit values can be done in a constant time. And only first N-byte of a file is used as a snapshot. We don't even need to process the whole file. Resources like CPU cycle and RAM can be saved a lot for large input file.

Let:

- Size of the input data be N bytes
- Snapshot if the input data be first K bytes instead of all N bytes
    - K bytes if (N > K)
    - Otherwise K = N bytes
- CRC-32 checksum for K bytes of snapshot
- Fast DPI Key = Leading Significant Bytes + CRC-32 + Hash Size
- HMAC-SHA-1 hash of K-byte data with Fast DPI Key

Then, HMAC-SHA-1 hash of K-byte input data is 160-bit (20 bytes).

Note: In this project, we use K as 4096 bytes

HMAC (Hash-based Message Authentication Code) is a specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret key. As with any MAC, it may be used to simultaneously verify both the data integrity and the authenticity of a message. Any cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA1 accordingly. The cryptographic strength of the HMAC depends upon the

cryptographic strength of the underlying hash function, the size of its hash output length in bits, and on the size and quality of the cryptographic key. An iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a compression function. For example, MD5 and SHA-1 operate on 512-bit blocks. The size of the output of HMAC is the same as that of the underlying hash function (128 or 160 bits in the case of MD5 or SHA-1, respectively), although it can be truncated if desired.

7.3 Perl and MySQL signatures:

In the final stage, MySQL database server is to be used to store signatures in a table. These signatures will be used to check if the input data is a virus. If it is not virus, snort will further process data for inspection.

We will use Perl script to generate 20-byte signature of K-byte, 4096 bytes or less, input data. Then, we connect to MySQL server using IPv4 address. If we find a match in database for the signature, the virus has been detected and no further processing is required. Otherwise, DPI engine will use Snort for pattern matching to inspect data further.

```
#!/usr/bin/perl -I

#PERL MODULE
use DBI;
use DBD::mysql;
use String::CRC32;
use LWP::Simple;
use Digest::SHA1  qw(sha1 sha1_hex sha1_base64);

my $nbytes = 4096;

my $filename = $ARGV[0];
my $file = get($filename) or die "Failed to fetch file";
my $filesize = length($file);
#my $filesize = -s $filename;
print "\nFile size: ",$filesize,"\n";

my $input_val;
if($filesize<nbytes){
        $input_val = $file;
}
else{
        $input_val = substr($file, 0, $nbytes);
}
```

```
my $input_size = length($input_val);
print "Input value size: ",$input_size,"\n";

my $checksum = crc32($input_val);
print "Checksum: ",$checksum,"\n";

my $digest = sha1($input_val);
print "SHA hash value: ",$digest,"\n";
#print "SHA hash length: ",length($digest);

#CONFIG VARIABLES
my $platform = "mysql";
my $host = "24.23.138.64";
my $port = "3306";
my $database = "COEN";
my $tablename = "coen233";
my $user = "root";
my $pw = "hcippetoluan";

#DATA SOURCE NAME
my $dsn = "dbi:mysql:$database:$host:3306";

#PERL MYSQL CONNECT
my $DBIconnect = DBI->connect($dsn, root, hcippetoluan) or die "Unable to connect to
mysql\n";

#DEFINE SEARCH QUERY
my $myquery = "SELECT signature from coen233 where signature='$digest'";
my $query_handle = $DBIconnect->prepare($myquery);
#EXECUTE QUERY
$query_handle->execute();

if($query_handle->fetchrow_array){
        print "Virus detected with signature: $digest \n";
}

else{
        #DEFINE MYSQL Query
        $myquery = "INSERT INTO $tablename (counter, signature, name, ext)
        VALUES (?, ?, ?, ?)";
        #VALUES (DEFAULT, $digest, DEFAULT, DEFAULT)";
        $query_handle = $DBIconnect->prepare($myquery);

        #EXECUTE QUERY
        $query_handle->execute(DEFAULT,"$digest",  DEFAULT, DEFAULT) or die "SQL
Error:$DBI::errstr\n";
}
```

Table 1. Perl script to connect to MySQL server for finding signature match

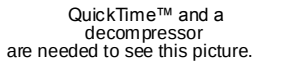

Fig8. SHA-1 HMAC Generation

7.4 SNORT

The SNORT system processes the traffic of packets on multi stages as illustrated in Figure 9.

SNORT rule may contain header and content fields where the header part checks the protocol, source and destination IP address and port, and the content part scans packets payload for one or more patterns. Rules with more than one pattern are called correlated rules. Furthermore, rules can also contain negation patterns, which mean negation of patterns stands for no occurrence of the pattern. The matching pattern may be in ASCII, HEX or mixed format. HEX parts are included between vertical bar symbols "|" as an example of a Snort rule is [14]:

alert tcp any any -> 198.165.200.24/32 111 (content: "idcj|3a3b|j"; msg: "mountd access";)

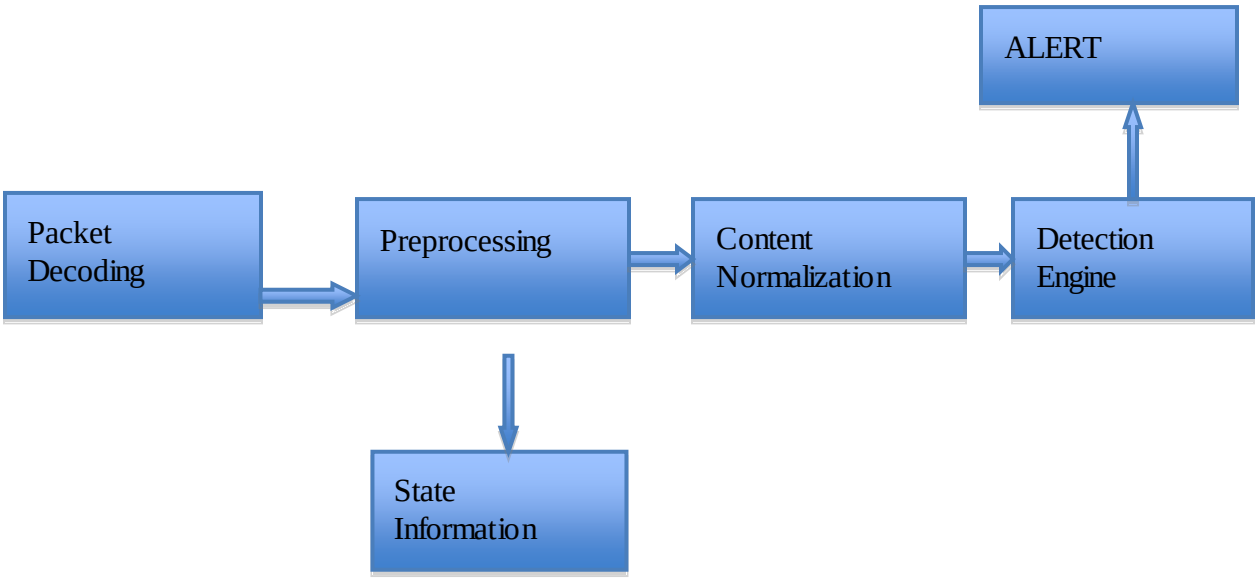

Fig9. SNORT Process stages

8 Data Analysis

8.1 Data Preparation

In our prototype Fast DPI system, we prepared for the virus sample file COEN233-virus.zip, which is about 22,738 KB. In the Snort System, a dedicated Snort IDS rule is created as below to catch this virus pattern from HTTP download.

alert tcp any 1025 -> any any (msg:"DPI Virus Pattern"; content:"\|63 68 61 70 74 65 72 37 2E 70 70 74 50 4B 01 02 14 20 14 20 02 20 08 20 66 60 2A 40 86 0D D5 93\|"; sid:1000888)

8.2 Snort System Setup

Snort 2.9.2 is installed in Linux Fedora on top of openSSL, libcap, libdnet, DAQ (Data Acquisition API) and PCRE (Perl Compatible Regular Expression).

Snort is launched in NIDS mode:
Snort –dev –A full –c dpi-snort.conf > output

Therefore we can capture all snort logs in output and raise alert to report the virus detection based on rule defined in dpi-snort.conf.

The alert can be monitored in the runtime by
Tail –f /var/log/snort/alert

8.3 HTTP server Setup

We use the webserver developed in P1 project to host the virus sample file on another host. Once the HTTP request comes from Snort machine, HTTP server will sends back this sample file for processing.

[gduo@merlot P1]$ ./webserver
Web Server is listening on merlot.sv.us.sonicwall.com, Port: 1025

8.4 IDS matching

From snort machine, we start the HTTP download by issuing
[gduo@tropicana project]$ wget http://merlot:1025/COEN233-virus.zip

Then HTTP server responds with

====> Received HTTP Request:
GET /COEN233-virus.zip HTTP/1.0
User-Agent: Wget/1.10.2 (Red Hat modified)
Accept: */*
Host: merlot:1025
Connection: Keep-Alive

====> Sent HTTP Response Header:
HTTP/1.1 200 OK
Connection: close
Date: Tue, 20 Mar 2012 00:47:54 GMT
Server: Zhuangzhi-Duo-webserver/1.0 (Linux)
Last-Modified: Mon, 19 Mar 2012 22:35:11 GMT
Content-Length: 23283380
Content-Type: text/html

In Snort alert, the corresponding pattern match message will be raised:

```
[**] [1:1000777:0] DPI Virus Pattern [**][Priority: 0]
03/19-16:12:10.041270 A4:BA:DB:35:44:5D -> 00:18:8B:4F:9E:1A type:0x800 len:0x5EA
10.202.2.77:1025 -> 10.202.2.209:56700 TCP TTL:64 TOS:0x0 ID:16134 IpLen:20 DgmLen:1500
DF
***A**** Seq: 0x44CC4D6B Ack: 0x2164B14A Win: 0x1920 TcpLen: 32
TCP Options (3) => NOP NOP TS: 1486451307 3437924032
```

From Snort log file output, we can trace down the packet activities

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

03/19-16:07:06.185321 00:18:8B:4F:9E:1A -> A4:BA:DB:35:44:5D type:0x800 len:0xC9
10.202.2.209:56699 -> 10.202.2.77:1025 TCP TTL:64 TOS:0x0 ID:54507 IpLen:20 DgmLen:187
DF
***AP*** Seq: 0x1E3154E2 Ack: 0x4189D36C Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 3437920265 1486447451
47 45 54 20 2F 43 4F 45 4E 32 33 33 2D 76 69 72  GET /COEN233-vir
75 73 2E 7A 69 70 20 48 54 54 50 2F 31 2E 30 0D  us.zip HTTP/1.0.
0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 57 67 65  .User-Agent: Wge
74 2F 31 2E 31 30 2E 32 20 28 52 65 64 20 48 61  t/1.10.2 (Red Ha
74 20 6D 6F 64 69 66 69 65 64 29 0D 0A 41 63 63  t modified)..Acc
65 70 74 3A 20 2A 2F 2A 0D 0A 48 6F 73 74 3A 20  ept: */*..Host:
6D 65 72 6C 6F 74 3A 31 30 32 35 0D 0A 43 6F 6E  merlot:1025..Con
6E 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D 41 6C  nection: Keep-Al
69 76 65 0D 0A 0D 0A                             ive....

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

8.5 Snort Performance Measurement

In Snort packets log, the HTTP packet arrives at ***03/19-16:07:06.185321.***
In Snort IDS alert, the virus is identified at ***03/19-16:12:10.041270***
Therefore the Snort takes more than 5 seconds in this scenario.

If we reconstruct the signature pattern to force Snort to parse the whole file, the result shows Snort takes about 1.5 minutes to complete the pattern matching.

8.7 DPI Signature Creation

DPI processor is used to generate the MD signature and save it into database before IDS processing.

DPI-Engine/dpiProcess COEN233-virus.zip 4096 dpi_sig
HMAC-SHA1 test succeeded
DPI snapshot: 4096 Bytes
result:
4528671b
DPI Hash Key Header:
0000: 50 4b 03 04 14 20 02 20 ## ## ## ## ## ## ## ## >PK.... ########<
DPI Hash Key CheckSum:
0000: 1b 67 28 45 ## ## ## ## ## ## ## ## ## ## ## ## >.g(E############<
DPI Hash Key Trailer:
0000: 00 10 00 00 00 00 00 00 ## ## ## ## ## ## ## ## >........########<
DPI Signature Message Digest:
0000: f6 a2 3f 4a f9 dd 72 02 6a ca 9f 59 b0 34 a9 bb >..?J..r.j..Y.4..<
0010: 4d 27 12 86 *## ## ## ## ## ## ## ## ## ## ## ## >M'..############<*
[gduo@tropicana project]$

8.8 DPI Engine Processing

We can feed the virus sample file into the DPI engine to do DPI processing to check the result.

[gduo@tropicana project]$ DPI-Engine/dpiEngine COEN233-virus.zip 4096 dpi_sig
HMAC-SHA1 test succeeded

[1332205944] Start processing [1:SIG:0] for DPI content

[1332205944] Complete processing [1:SIG:0] for DPI content

DPI Engine Signature by Computation:

0000: f6 a2 3f 4a f9 dd 72 02 6a ca 9f 59 b0 34 a9 bb >..?J..r.j..Y.4..<
0010: 4d 27 12 86 ## ## ## ## ## ## ## ## ## ## ## ## >M'..############<

DPI Engine Signature from DataBase:

0000: f6 a2 3f 4a f9 dd 72 02 6a ca 9f 59 b0 34 a9 bb >..?J..r.j..Y.4..<
0010: 4d 27 12 86 ## ## ## ## ## ## ## ## ## ## ## ## >M'..############<

DPI Alert: Signature MATCH in file COEN233-virus.zip
[gduo@tropicana project]$

8.9 Fast DPI Engine Performance Measurement

We find out the DPI processing time is almost zero based on timestamp being printed out from the console.

9 Conclusions

9.1 Summary

From the data we collected from the experiment, we can see the performance can be improved by using snapshot and hashing algorithm to optimize the DPI engine architecture.

Snapshot can definitely make CPU cycle and RAM consumption more efficient. MD signature can exponentially reduce the signature lookup duration if matches. Theoretically if MD signature cannot match in the first place, the Fast DPI Key hashing index will contribute the performance increase based on great advantage of Hash table.

On the other hand, the improvement can also be achieved from DPI engine optimization by converting regular expression into NFA, and then DFA in the theory.

9.2 Future Studies

In the future, we should complete the index hashing implementation in DPI engine to cover more scenarios.

And we need to do more research on Snort System to be able to add new module to do Regex / NFA / DFA in very efficient way.

10 Reference

[1] StriD2FA: Scalable Regular Expression Matching for Deep Packet Inspection. Xiaofei Wang, Junchen Jiang, Yi Tang, Yi Wang, Bin Liu, Xiaojun Wang, School of Electronic Engineering, Dublin City University, Dublin, Ireland. Department of Computer Science and Technology, Tsinghua University, Beijing, China,2011.

[2] Optimizing Deep Packet Inspection for High-Speed Traffic Analysis. Niccol` Cascarano, Luigi Ciminierao, Fulvio Risso,2009.

[3] Snort User Manual. SNORT R Users Manual 2.9.2 .The Snort Project December 7, 2011

[4] Advanced Algorithms for Fast and Scalable Deep Packet Inspection. Sailesh Kumar, Jonathan Turner, John Williams, Washington University

[5] Range Hash for regular expressions Pre-Filtering. Masanori Bando, N.Sertac Artan, Rihua Wei, Xiangyi Guo, H.Jonathan Chao. Polytechnic Institute of New York University.

[6] DPICO: A High Speed Deep Packet Inspection Engine Using Compact Finite Automata. Christopher L. Hayes and Yan Luo. Department of Electrical and Computer Engineering University of Massachusetts, Lowell.

[7] F. Anjum, D. Subhadrabandhu, and S. Sarkar. Signature-based intrusion detection for wireless ad-hoc networks: A comparative study of various routing protocols. In IEEE Vehicular Technology Conference, October 2003.

[8] S. Wu and U. Manber. Fast text searching: Allowing errors. Communications of the ACM, 35(10): 83–91,1992.

[9] L. Schaelicke, B. Moore T. Slabach, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection (RAID 2003), LNCS, Springer-Verlag, September 2003.

[10] Russ Cox. Regular Expression Matching Can be Simple and Fast in http://swtch,com , January 2007.

[11] Ken Thompson, "Regular expression search algorithm," Communications of the ACM 11(6) (June 1968), pp. 419–422. *http://doi.acm.org/10.1145/363347.363387* (PDF)

[12] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing, Reading Massachusetts, 1979. ISBN 0-201-02988-X. *(See chapter 2.)*

[13] Deep Packet Inspection for Intrusion Detection Systems: A Survey , Tamer AbuHmed, Abedelaziz Mohaisen, and DaeHun Nyang Information Security Research Laboratory, Inha University, Incheon 402-751,

[14] P. Wheeler and E. W. Fulp. A taxonomy of parallel techniques for intrusion detection. In ACM Southeast Regional Conference, pages 278–282, 2007.