OpenDPI.org

OpenDPI Integration Manual

Version: 1.0
Date: 4 September 2009

Table of Contents

Introduction

About

OpenDPI is a software library designed to classify Internet traffic according to network protocols. For this purpose mainly deep packet inspection (DPI) is used.

OpenDPI is derived from PACE, the traffic classification engine of ipoque, a provider of carrier-grade DPI and bandwidth management solutions. In contrast to ipoque's PACE engine, OpenDPI does not support the detection of encrypted protocols and it does not use any heuristic and behavioral analysis for classification.

Supported Protocols

The following protocols are supported by OpenDPI. The complete list of protocols supported by PACE is available at http://www.ipoque.com/products/protocol-support.

P2P File Sharing

BitTorrent, eDonkey , KaZaa/Fasttrack, Gnutella, WinMX, DirectConnect, AppleJuice, Soulseek, XDCC, Filetopia, Manolito, iMesh, Pando

Voice over IP

SIP, IAX, RTP

Instant Messaging

Yahoo, Oscar, IRC, unencrypted Jabber, Gadu-Gadu, MSN

Streaming Protocols

ORB, RTSP, Flash, MMS, MPEG, Quicktime, Joost, WindowsMedia, RealMedia, TVAnts, SOPCast, TVUPlayer, PPStream, PPLive, QQLive, Zattoo, VeohTV, AVI, Feidian, Ececast, Kontiki, Move, RTSP, SCTP, SHOUTcast

Tunnel Protocols

IPsec,GRE, SSL, SSH, IP in IP

Standard Protocols

HTTP, Direct download links (1-click file hosters), POP, SMTP, IMAP, FTP, BGP, DHCP, DNS, EGP, ICMP, IGMP, MySQL, NFS, NTP, OSPF, pcAnywhere, PostgresSQL, RDP, SMB, SNMP, SSDP, STUN, Telnet, Usenet, VNC, IPP, MDNS, NETBIOS, XDMCP, RADIUS, SYSLOG, LDAP

Gaming Protocols

World of Warcraft, Halflife, Steam, Xbox, Quake, Second Life

False Positive and False Negative rate

OpenDPI is designed to minimize the misdetection (false positive) rate. Most detections have a misdetection rate of $< 2^{-64}$ ($= 5 * 10^{-18}$ %) per flow[1]. A random flow is defined as a packet stream with random payload and random-length packets between 0 and 1460 bytes payload for each packet.

OpenDPI is designed for traffic management, which requires a very low false negative rate.

Differences between OpenDPI and PACE

PACE, the commercially available DPI engine from ipoque includes the following extensions:

- behavioral detection: full detection of encrypted protocols like Bittorrent, eDonkey and Skype
- asymmetric detection: protocol detection even when the up- or downlink is missing
- IPv6 support
- performance optimizations: up to 5x faster
- memory optimizations: smaller memory footprint per flow
- optimized hash table for connection tracking allows up to 1 million inserts per second per CPU with automatic timeouts

1 A **flow** is a bidirectional communication channel between two network applications. It is identified by the 5-tuple (SRC IP, DST IP, SRC PORT, DST PORT, IP PROTOCOL) in combination with an idle time-out. This definition applies for both TCP and UDP. A flow is sometimes also referred to as a connection or session.

OpenDPI Demonstration Package

Restrictions of the OpenDPI Demonstration Package

OpenDPI does not contain any facilities for flow-tracking or subscriber ID-tracking. In the OpenDPI demonstration application a very simple flow and ID-tracking mechanism is implemented which does not perform high data rates.

Installation of Required System Packages

OpenDPI comes with a simple makefile and a demonstration application for Linux and has been tested on 32-bit and 64-bit x86 Linux systems.

For compilation, the following development resources are required:

- gcc
- make
- libpcap-dev

To install these packages on Debian or derivative systems like Ubuntu, call:

```
sudo apt-get install gcc make libpcap-dev
```

For OpenSuse 10.3 and newer, use:

```
sudo zypper in gcc make libpcap-devel
```

Extraction of the OpenDPI Demonstration Package

Extract the file with the following command:

```
user@pc:~$ tar -xzf OpenDPI.tar.gz
```

Compilation of the OpenDPI Demonstration Package

To compile the OpenDPI demonstration application, change into the directory created by the previous step and use 'make' to compile the OpenDPI library and the demonstration application.

```
user@pc:~$ cd OpenDPI/
user@pc:OpenDPI$ make
...
user@pc:utils$ ls -l
total 2144
drwxr-xr-x 2 user user    4096 2009-08-31 12:58 include
drwxr-xr-x 3 user user    4096 2009-08-31 12:59 lib
-rw-r--r-- 1 user user 1427432 2009-08-31 12:59 libOpenDPI.a
-rw-r--r-- 1 user user     858 2009-08-31 12:58 Makefile
-rwxr-xr-x 1 user user  723249 2009-08-31 12:59 OpenDPI_demo
-rw-r--r-- 1 user user   14624 2009-08-31 12:58 OpenDPI_demo.c
-rw-r--r-- 1 user user     877 2009-08-31 12:58 Readme.txt
```

After the compilation, the demonstration application 'OpenDPI_demo' is ready to use.

Testing the OpenDPI Demonstration Application with a Sample PCAP File.

Install Wireshark

To test OpenDPI, please record a pcap file, for example using Wireshark. Wireshark can be installed under

Debian and derivates like Ubuntu with:

```
apt-get install Wireshark
```

With Opensuse, use

```
sudo zypper in Wireshark
```

Other OS versions of Wireshark can be found at http://www.Wireshark.org/

Recording a Trace with Wireshark

Please use the following steps to record a clean pcap file.

– stop all applications which use the network access (browsers, e-mail clients, chat programs)
– start Wireshark
– start a live capture on the active network interface
– start the test application (browser, e-mail client, chat program, p2p client, ...)
– use the test application to generate traffic (download a p2p file, place a voice call, ...)
– stop the test application
– stop the live record in Wireshark
– save the live record to a pcap file

Analyzing a Trace with the OpenDPI Demonstration Application

The OpenDPI demonstration application does a simple protocol accounting listing the number of packets, bytesm and flows per protocol. To analyze a trace with the OpenDPI demonstration application use the option '`-f`' with the path to the pcap file. For this demo, an HTTP sample has been recorded to the home folder as http.pcap.

```
user@pc:OpenDPI$ ./OpenDPI_demo -f ~/http.pcap
```

The output will look somewhat like this:

```
pcap file contains
        ip packets:   2932          of 2958 packets total
        ip bytes:     2743485
        unique ids:   28
        unique flows: 39

detected protocols:
        unknown              packets: 70             bytes: 4556          flows: 26
        HTTP                 packets: 2829           bytes: 2734479       flows: 2
        NETBIOS              packets: 27             bytes: 3692          flows: 5
        ICMP                 packets: 6              bytes: 758           flows: 6
```

This output displays that 70 packets have been identified as “unknown”, 2829 packets as “HTTP”, 27 packets as NETBIOS and 6 packets as ICMP in this trace.

OpenDPI Integration

Usage of include file

Only a single header file from the include directory must be included:

```
#include "ipq_api.h"
```

All other header files from the include directory must not be included.

Initialization

OpenDPI uses just one global structure which stores all settings and temporary variables for the packet processing. It can be defined with:

```
static struct ipoque_detection_module_struct *ipoque_struct = NULL;
```

OpenDPI is designed to use custom optimized allocation and free function. This example uses two simple malloc and free wrappers for both functions.

```
static void *malloc_wrapper(unsigned long size)
{
	return malloc(size);
}

static void free_wrapper(void *freeable)
{
	free(freeable);
}
```

As different systems have different timestamp resolutions, OpenDPI works with resolutions ranging from one second to one millisecond. The recommended resolution is 1ms or 10ms.

The OpenDPI initialization requires the timestamp resolution as a parameter. The value is the timer resolution per second. A millisecond resolution is used in this example. The value is:

```
static u32 ipq_tick_resolution = 1000; // use millisecond resolution in OpenDPI
```

OpenDPI is initialized by two calls. The first call will allocate and initialize the global OpenDPI structure with the given data:

```
ipoque_struct = ipoque_init_detection_module(ipq_tick_resolution,
                                              malloc_wrapper, NULL);
```

The last parameter is for debugging only and must be set to NULL. The initialization is the only place where a memory allocation takes place. For performance and stability reasons, OpenDPI does not use any memory allocations during packet processing.

The second call will activate a number of protocols for detection. For this call, a bitmask is required. Each bit is used to activate (1) or deactivate (0) one protocol. To define a bitmask, use:

```
IPOQUE_PROTOCOL_BITMASK all;
```

The bitmask must be set to activate all protocols:

```
IPOQUE_BITMASK_SET_ALL(all);
```

This bitmask enables all protocols with the call:

```
ipoque_set_protocol_detection_bitmask2(ipoque_struct, &all);
```

Packet Processing

Generic description of the 4 input parameters

For packet processing and classification, 4 parameters are required.

The first type is the memory block from the initialization.

The second type is the packet itself.

The third type is a memory buffer of the connection of the packet to hold the state for this connection.

The fourth type is a similar memory buffer to hold the state for every subscriber.

Packet information

OpenDPI needs access to the packet. The required components are:

– pointer to IP header
– accessible length of the packet starting from the IP header
– packet timestamp

The accessible length is needed to avoid invalid reads when the detection would rely on the IP total length information, which could be wrong.

```
unsigned char *packet;  // pointer to the packet at Layer 3 (IP)
unsigned short int packet_length;   /* number of bytes which can be accessed
                                     *  from the packet pointer */
unsigned int timestamp; // arrival timestamp of the packet
```

Connection information

The state of every TCP and UDP flow is maintained along with internal values. The memory size of the state buffer is fixed and depends on the number of compiled protocols.

In the following code example, the variable 'flow' will be used for the connection tracking:

```
void *flow;       // pointer to the final state machine of the connection
```

The required size is the return value of a built-in function in OpenDPI:

```
unsigned int ipoque_detection_get_sizeof_ipoque_flow_struct(void);
```

The pointer must not be NULL for TCP/UDP traffic. For other traffic (where for example a connection tracking is impossible), NULL is accepted.

Subscriber information

A similar state buffer is maintained for every subscriber. In most situations, a subscriber is identified by its internal IP address. In this case, a memory buffer for every internal IP address must be maintained.

OpenDPI is designed to work in different network situations. There are two different situations for the subscriber mapping.

In an access gateway situation, every packet belongs to one subscriber. The subscriber is either the sender or the receiver of the packet. In this case, either the source or the destination subscriber is known.

In a non-access gateway, every packet could belong to up to 2 subscribers. This happens when one subscriber is the sender while the other is the receiver. In this case, both subscribers can be passed to OpenDPI.

In the following code example, the variable 'src' and 'dst' will be used for the sender and receiver subscriber.

```
void *src;  // pointer to the final state machine of the sender subscriber
void *dst;  // pointer to the final state machine of the dest subscriber
```

The required size for each element is the return value of a built-in function in OpenDPI:

```
unsigned int ipoque_detection_get_sizeof_ipoque_id_struct(void);
```

In an access gateway situation where only one subscriber is required, either 'src' or 'dst' can be NULL.

Packet Processing Integration Example

The packet processing is done with the 6 given parameters (packet, packet_length, timestamp, flow, src and dst) and the initialized global ipoque_struct:

```
unsigned int protocol;
protocol = ipoque_detection_process_packet(ipoque_struct, flow, packet,
                                          packet_length, timestamp, src, dst);
```

The returned protocol is one protocol from the protocol list in 'include/ipq_protocols_osdpi.h'.

As an example, the returned protocol can be printed to the console with:

```
static const char *protocol_long_str[] = { IPOQUE_PROTOCOL_LONG_STRING };
printf("Packet has protocol: %s\n", protocol_long_str[protocol]);
```

Termination

OpenDPI is terminated by a single call, which frees the memory of the detection structure:

```
ipoque_exit_detection_module(ipoque_struct, free_wrapper);
```