nDPI - Quick Start Guide

Open and Extensible LGPLv3 Deep Packet Inspection Library

Version 1.0
February 2014

nDPI web	http://www.ntop.org/products/ndpi/
ntop site	www.ntop.org
nDPI License	LGPLv3

Table of Contents

1. Introduction

nDPI is a DPI library based on OpenDPI and currently maintained by ntop.

In addition to Unix, we also support Windows, in order to provide you a cross-platform DPI experience. Furthermore, we have modified nDPI to be more suitable for traffic monitoring applications, by disabling specific features that slow down the DPI engine while making them un-necessary for network traffic monitoring.

nDPI allows application-layer detection of protocols, regardless of the port being used. This means that it is possible to both detect known protocols on non-standard ports (e.g. detect http non ports other than 80), and also the opposite (e.g. detect Skype traffic on port 80). This is because nowadays the concept of port=application no longer holds.

Over the past few months we have added several features including:

- Enhancement of the demo pcapReader application both in terms of speed/features and encapsulations supported (for instance you can now analyse GTP tunnelled traffic).
- Ability to compile nDPI inside the Linux kernel so that you can use it for developing efficient kernel-based modules.
- Various speed enhancements so that nDPI is now faster than its predecessor.
- Added many protocols (to date we support ~170 protocols) ranging from "business" protocols such as SAP and Citrix, as well as "desktop" protocols such as Dropbox and Spotify.
- Ability to define port (and port range)-based protocol detection, so that you can complement protocol detection with classic port-based detection.
- In order to let nDPI support encrypted connections, we have added a decoder for SSL (both client and server) certificates, thus we can figure out the protocol using the encryption certificate. This allows us to identify protocols such as Citrix Online and Apple iCloud that otherwise would be undetected.
- Ability to support sub-protocols using string-based matching

1.1 Download Source

nDPI is automatically downloaded when you build ntop and nProbe. However nothing prevents you from using it as a standalone DPI library. The source code can be downloaded from the https://svn.ntop.org/svn/ntop/trunk/nDPI/.

2. nDPI Library

2.1 Compiling nDPI Source Code

Start using nDPI Library is very simple. In order to compile this library you must meet certain prerequisites such as:

```
GNU autotools/libtool
gawk
gcc
```

To do this you need to install them using the following commands:

Fedora	`yum groupinstall Development tools`
	`yum install automake libpcap-devel gcc-c++ libtool`
Debian	`apt-get install build-essential libpcap-dev`
Mac OSX	`port install XXX (Please install macports)`

Once done that, you can compile the nDPI source code as follows:

```
cd <nDPI source code directory>
./configure
make
```

2.2 Compiling the demo pcapReader Source Code

Starting using pcapReader demo is also simple. In order to compile this you must use the following command:

```
cd <nDPI source code directory>/example
make
```

2.3 pcapReader Command Line Options

The demo pcapReader application can be used both in terms of speed/features analysis and encapsulations support. In particular, it is possible to specify a lot of command line options.
The available options and a minimal explanation of each option are listed below:

```
./pcapReader -h
pcapReader -i <file|device> [-f <filter>][-s <duration>]
           [-p <protos>][-l <loops>[-d][-h][-t][-v <level>]

Usage:
  -i <file.pcap|device>     | Specify a pcap file to read packets from or a device
                            | for live capture
  -f <BPF filter>           | Specify a BPF filter for filtering selected traffic
  -s <duration>             | Maximum capture duration in seconds (live traffic
                            | capture only)
  -p <file>.protos          | Specify a protocol file (eg. protos.txt)
  -l <num loops>            | Number of detection loops (test only)
  -d                        | Disable protocol guess and use only DPI
  -t                        | Dissect GTP tunnels
  -h                        | This help
  -v <1|2>                  | Verbose 'unknown protocol' packet print. 1=verbose,
                            | 2=very verbose
```

-i <file.pcap|device>

This specifies a pcap file to read packets from or a device for live capture. Only one of these two can be specified.

-f <BPF filter>

It specifies a BPF filter for filtering selected traffic. It allows nDPI to take only those packets that match the filter (if specified).

-s <duration>

It defines the capture duration in seconds, only for live traffic capture.

-p <file>.protos

It specifies a protocol file (e.g. protos.txt) to expand the support of sub-protocols and port-based protocol detection. Be careful, the protocol defined to protos file overwrites the existing protocol.

-l <num loops>

Number of detection loops (test only).

-d

This flag disables the nDPI protocol guess and uses only DPI.

-t

It dissects GTP tunnels

-h

It prints the pcapReader help.

-v <1|2>

Using this flag, pcapReader generates verbose output that can be used to tune its performance. Number one is the lowest level that displays the packets with 'unknown protocol', number two is more verbose.

```
UDP 62.101.93.101:53 > 192.168.1.132:56130 [proto: 5/DNS][2 pkts/260 bytes][chat.stackoverflow.com]
TCP 192.168.1.132:59323 > 62.161.94.220:80 [proto: 7/HTTP][8 pkts/1925 bytes][]
UDP 62.101.93.101:53 > 192.168.1.132:56682 [proto: 5/DNS][2 pkts/258 bytes][diy.stackexchange.com]
UDP 62.101.93.101:53 > 192.168.1.132:56916 [proto: 5/DNS][2 pkts/524 bytes][conjugator.reverso.net]
.....................................................................................
.....................................................................................
TCP 192.168.1.132:59323 > 62.161.94.220:80 [proto: 7/HTTP][8 pkts/1925 bytes][]

Undetected flows:

TCP 192.168.1.132:57995 > 157.55.133.142:12350 [proto: 0/Unknown][3 pkts/208 bytes][]
TCP 65.55.223.47:33033 > 192.168.1.132:57997 [proto: 0/Unknown][8 pkts/547 bytes][]
TCP 127.23.238.168:16384 > 240.199.103.219:0 [proto: 0/Unknown][11 pkts/1611 bytes][]
TCP 127.23.238.168:0 > 240.199.103.219:16384 [proto: 0/Unknown][11 pkts/2734 bytes][]
```

2.4 Protocol File

nDPI has the ability to support sub-protocols using string-based matching. This is because many new sub-protocols such as Apple iCloud/iMessage, WhatsApp and many others use HTTP(S) that can be detected by decoding the SSL certificate host or the HTTP "Host:". Thus we have decided to embed in nDPI an efficient string-matching library based on the

popular Aho-Corasick algorithm for matching hundred of thousand sub-strings efficiently (i.e. fast enough to sustain 10 Gbit traffic on commodity hardware).

You can specify sub-protocols at runtime using a protocol file with the following format:

```
# Subprotocols
# Format:
# host:"<value>",host:"<value>",.....@<subproto>
host:"googlesyndacation.com"@Google
host:"venere.com"@Veneer
```

in addition you can specify a port-based protocol detection using the following format:

```
#  Format:
#  <tcp|udp>:,<tcp|udp>:,.....@
tcp:81,tcp:8181@HTTP
udp:5061-5062@SIP
tcp:860,udp:860,tcp:3260,udp:3260@iSCSI
tcp:3000@ntop
```

You can test your custom configuration using the pcapReader (use -p option) application or enhance your application using the ndpi_load_protocols_file() nDPI API call.

3. Examples

In this section we show some pcapReader use cases.

3.1 Live Capture Mode

The following example shows the pcapReader live capture mode by using the parameter -i to specify the device and the parameter -s to specify the live capture duration.

```
$ ./pcapReader -i eth0 -s 20

-----------------------------------------------------------
* NOTE: This is demo app to show *some* nDPI features.
* In this demo we have implemented only some basic features
* just to show you what you can do with the library. Feel
* free to extend it and send us the patches for inclusion
-----------------------------------------------------------

Using nDPI nDPI ($Revision: #### $)
Capturing live traffic from device eth0...
Capturing traffic up to 20 seconds

pcap file contains
      IP packets:   2390          of 2391 packets total
      IP bytes:     1775743
      Unique flows: 78
      nDPI throughout: 122.30 pps / 709.92 Kb/sec
      Guessed flow protocols: 0


Detected protocols:
      DNS                 packets: 57            bytes: 7904          flows: 28
      SSL_No_Cert         packets: 483           bytes: 229203        flows: 6
      FaceBook            packets: 136           bytes: 74702         flows: 4
      DropBox             packets: 9             bytes: 668           flows: 3
      Skype               packets: 5             bytes: 339           flows: 3
      Google              packets: 1700          bytes: 619135        flows: 34
```

3.2 pcap Capture Mode

The most simple way to create a pcap file is to use tcpdump command as in the following example:

```
ntop$ tcpdump -ni eth0 -s0 -w /var/tmp/capture.pcap -v
tcpdump: listening on en1, link-type EN10MB (Ethernet), capture size 65535 bytes
Got 0
Got 64
Got 75
Got 76
^C122 packets captured
122 packets received by filter
0 packets dropped by kernel
```

Once the pcap file has been created you will be able to launch the demo pcapReader with the parameter -i:

```
$ ./pcapReader -i /var/tmp/capture.pcap

-----------------------------------------------------------
* NOTE: This is demo app to show *some* nDPI features.
* In this demo we have implemented only some basic features
* just to show you what you can do with the library. Feel
* free to extend it and send us the patches for inclusion
-----------------------------------------------------------
```

```
Using nDPI nDPI ($Revision: #### $)
Reading packets from pcap file /var/tmp/capture.pcap...

pcap file contains
      IP packets:   4911          of 4911 packets total
      IP bytes:     3321544
      Unique flows: 145
      nDPI throughout: 612.80 K pps / 3.09 Gb/sec
      Guessed flow protocols: 11


Detected protocols:
      Unknown             packets: 6             bytes: 764           flows: 1
      DNS                 packets: 50            bytes: 6158          flows: 25
      HTTP                packets: 2537          bytes: 841638        flows: 73
      SSL_No_Cert         packets: 1522          bytes: 303380        flows: 10
      SSL                 packets: 24            bytes: 1648          flows: 8
      FaceBook            packets: 644           bytes: 201216        flows: 17
      Skype               packets: 11            bytes: 872           flows: 6
```

3.3 Protocol File

In order to clarify the features of the protocol file we are now going to explain how you can identify the flow of ntop.org.

For instance it is possible to do it by editing protos.txt.

```
ntop$ echo 'host:"ntop.org"@nTop'> protos.txt
```

Once the protocol file has been modified you will be able to launch the demo pcapReader with the parameter -p:

```
$ ./pcapReader -i en1 -s 30 -p protos.txt

------------------------------------------------------------
* NOTE: This is demo app to show *some* nDPI features.
* In this demo we have implemented only some basic features
* just to show you what you can do with the library. Feel
* free to extend it and send us the patches for inclusion
------------------------------------------------------------

Using nDPI nDPI ($Revision: #### $)
Capturing live traffic from device en1...
Capturing traffic up to 30 seconds


WARNING: only IPv4/IPv6 packets are supported in this demo (nDPI supports both IPv4
and IPv6), all other packets will be discarded


pcap file contains
      IP packets:   4755          of 4757 packets total
      IP bytes:     1766370
      Unique flows: 245
      Guessed flow protocols: 16

Detected protocols:
      Unknown             packets: 1             bytes: 94            flows: 1
      DNS                 packets: 38            bytes: 5160          flows: 19
      HTTP                packets: 265           bytes: 59831         flows: 20
      SSDP                packets: 20            bytes: 9564          flows: 14
      SSL                 packets: 33            bytes: 2572          flows: 13
      DropBox             packets: 17            bytes: 2481          flows: 6
      Skype               packets: 12            bytes: 944           flows: 2
      Google              packets: 2544          bytes: 612765        flows: 94
      nTop                packets: 407           bytes: 66765         flows: 32
```

4. API nDPI

In this section the nDPI API is highlighted.

The demo pcapReader will be now taken has a basic example to show how to initialize the library. It is required to have a compiled library and a properly configured Makefile (i.e the demo Makefile).

To start to using the API of nDPI within your application - in addition to your includes - you must also add the following include file:

```
#include "ndpi_main.h"
```

The library can be initialized as follows:

1. Declare the protocol bitmask and initialise the detection module

```
NDPI_PROTOCOL_BITMASK all;
ndpi_struct = ndpi_init_detection_module(
       detection_ticks_resolution,
       malloc_wrapper,
       free_wrapper,
       debug_printf);
```

This function will allow you to initialise the detection module. The fields have the following meanings:

- u_int32_t ticks_per_second;
  The timestamp resolution per second (like 1000 for millisecond resolution).
- void* (*__ndpi_malloc)(unsigned long size);
  Function pointer to a memory allocator.
- void* (*__ndpi_free)(void* prt);
  Function pointer to a debug output function, use NULL in productive environments.

2. Enable all protocols (note that you can enable a subset of the protocols if you) via the appropriate macro and set them within the detection module.

```
// enable all protocols
NDPI_BITMASK_SET_ALL(all);
ndpi_set_protocol_detection_bitmask2(ndpi_struct, &all);
```

This function will allow you to set the protocol bitmask already defined within the detection module.

3. In order to load an existing protocol file you must use the following function:

```
dpi_load_protocols_file(ndpi_struct, _protoFilePath);
```

4. Once captured the flows from your pcap file or ingress device, they can be analyzed by using the following function:

```
protocol = (const u_int32_t)ndpi_detection_process_packet(
```

```
ndpi_struct,
ndpi_flow,iph ? (uint8_t *)iph : (uint8_t *)if,
ipsize,
time,
src,
dst);
```

The fields have the following meanings:

- struct ndpi_detection_module_struct *ndpi_struct;
  The detection module.
- struct ndpi_flow_struct *flow;
  Flow void pointer to the connection state machine.
- const unsigned char *packet;
  The packet as unsigned char pointer with the length of packetlen. The pointer must point to the Layer 3 (IP header).
- const unsigned short packetlen;
  Packetlen the length of the packet.
- const u_int32_t current_tickt;
  The current timestamp for the packet.
- struct ndpi_id_struct *src;
  Void pointer to the source subscriber state machine.
- struct ndpi_id_struct *dst;
  Void pointer to the destination subscriber state machine.

5. Once the flows have been analysed, it is necessary to destroy the detection module via the use of the following function:

```
ndpi_exit_detection_module(ndpi_struct, free_wrapper);
```

The fields have the following meanings:

- struct ndpi_detection_module_struct* ndpi_struct;
  The detection module to be cleared.
- void (*ndpi_free) (void *ptr);
  Function pointer to a memory free function.

For further information we suggest to read the files

nDPI/example/pcapReader.c,
nDPI/src/include/ndpi_structs.h,
nDPI/src/include/ndpi_public_functions.h
nDPI/src/ndpi_main.c.

The protocol dissector files are contained in the nDPI/src/protocols directory.

5. Developing nDPI custom protocol

In this section we show the way to include your protocol inside nDPI.

5.1 Introduction

Each nDPI protocol is implemented as an entry function to be used at runtime by nDPI. The nDPI comes with several protocols that can be used as example for this activity. Below, we list the main concepts you need to know if you plan to develop an nDPI protocol.

5.2 Creating new protocol

Each protocol has to have a corresponding #define inside the following include file

```
<nDPI source code directory>/src/include/ndpi_protocols_osdpi.h
```

as follow

```
#define NDPI_PROTOCOL_MY_PROTOCOL       171
```

where `NDPI_PROTOCOL_MY_PROTOCOL` is the protocol "name" and `171` is the protocol ID that must be unique.

Once the protocol has been defined, you must create a new protocol source file like

```
<nDPI source code directory>/src/lib/protocols/my_protocol.c
```

with the following content

```
#include "ndpi_utils.h"

#ifdef NDPI_PROTOCOL_MY_PROTOCOLS

.....

#endif
```

where it will be necessary to define an entry function as a follow

```
void ndpi_search_my_protocol(
       struct ndpi_detection_module_struct *ndpi_struct,
       struct ndpi_flow_struct *flow)
{
  struct ndpi_packet_struct *packet = &flow->packet;

  NDPI_LOG(NDPI_PROTOCOL_MY_PROTOCOL, ndpi_struct, NDPI_LOG_DEBUG, "my
protocol detection...\n");

  /* skip marked packets by checking if the detection protocol stack */
```

```
  if (packet->detected_protocol_stack[0] != NDPI_PROTOCOL_MY_PROTOCOL) {
       ndpi_check_my_protocol(ndpi_struct, flow);
  }
}
```

and a detection core function to process a packet of a flow with the following content

```
static void ndpi_check_my_protocol(
struct ndpi_detection_module_struct *ndpi_struct,
struct ndpi_flow_struct *flow)
{
       struct ndpi_packet_struct *packet = &flow->packet;
       u_int32_t payload_len = packet->payload_packet_len;

       .....
       .....

       if("Found Protocol") {

              NDPI_LOG(NDPI_PROTOCOL_MY_PROTOCOL, ndpi_struct,
       NDPI_LOG_DEBUG, "Found     my protocol.\n");

              ndpi_int_my_protocol_add_connection(ndpi_struct, flow);

              return;
       }

       /*Exclude Protocol*/
       NDPI_LOG(NDPI_PROTOCOL_MY_PROTOCOL, ndpi_struct, NDPI_LOG_DEBUG,
                "exclude my protocol.\n");

       NDPI_ADD_PROTOCOL_TO_BITMASK(
              flow->excluded_protocol_bitmask,
              NDPI_PROTOCOL_MY_PROTOCOL);
       }

}
```

and a specific function to report the correct identification of the protocol as follow

```
static void ndpi_int_my_protocol_add_connection(
       struct ndpi_detection_module_struct *ndpi_struct,
       struct ndpi_flow_struct *flow,
       u_int8_t due_to_correlation)
{
  ndpi_int_add_connection(ndpi_struct, flow,
                   NDPI_PROTOCOL_MY_PROTOCOL,
                   /*Choose the type of your protocol*/
                   NDPI_CORRELATED_PROTOCOL or NDPI_REAL_PROTOCOL);
}
```

5.3 Add your protocol to nDPI

Once the protocol has been created, you must declare your entry function in the following include file

```
<nDPI source code directory>/src/include/ndpi_protocols.h
```

with the following content

```
/* my protocol entry */
void ndpi_search_my_protocol(
       struct ndpi_detection_module_struct *ndpi_struct,
       struct ndpi_flow_struct *flow);
```

Each protocol must be associated with a NDPI_SELECTION_BITMASK. The full list of NDPI_SELECTION_BITMASK is contained in the file

```
<nDPI source code directory>/src/include/ndpi_define.h
```

After choosing the selection bitmask for your protocol, you must inform nDPI of the new protocol by editing the file

```
<nDPI source code directory>/src/lib/ndpi_main.c
```

it is necessary to add your protocol to the the function as follow

```
void ndpi_set_protocol_detection_bitmask2(
             struct ndpi_detection_module_struct *ndpi_struct,
             const NDPI_PROTOCOL_BITMASK * dbm)
.....
.....
.....


#ifdef NDPI_PROTOCOL_MY_PROTOCOL
  ndpi_set_bitmask_protocol_detection(ndpi_struct,detection_bitmask,a,
                                NDPI_PROTOCOL_MY_PROTOCOL,
                                ndpi_search_my_protocol,
                                NDPI_SELECTION_BITMASK_MY_PROTOCOL,
                                SAVE_DETECTION_BITMASK_AS_UNKNOW,
                                ADD_TO_DETECTION_BITMASK);

  /* Update callback_buffer index */
  a++;
#endif

.....
.....
.....

ndpi_struct->callback_buffer_size = a;

  NDPI_LOG(NDPI_PROTOCOL_UNKNOWN, ndpi_struct, NDPI_LOG_DEBUG,
          "callback_buffer_size is %u\n", ndpi_struct-
>callback_buffer_size);
```