CS 591 : Software Architecture & Software Ecosystems

Portfolio

Team Members :
Sandeep N Kundalwal (T22051)
Mahima Gupta (T22055)
Ashutosh Litoriya (T22056)

Open Source Software : **WireShark**

1. Introduction

Wireshark is a tool used for network troubleshooting and network analysis. It is used to capture packets flowing from the network adaptor of the system. Wireshark can read and write capture files in its natural file format, pcapng and pcap, and various other file formats.

It is mostly used by individuals for analyzing network traffic. It is also used by network administrators for network monitoring. OpenSource developers can contribute to the application by developing dissector plugins for newer protocols

1.1 Types of Users

Following are the types of users for Wireshark application[2] -

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- QA engineers use it to verify network applications
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals

1.2 CrossPlatform Compatibility

It is a platform independent program, though it requires a few libraries (Qt, libcap and others) for cross-platform compatibility. The Qt library is used to build the UI for Wireshark and is used to provide a platform independent UI. Libpcap and Npcap provide the packet capture capabilities that are central to Wireshark's core functionality [1]

1.3 Components

Major Components of Wireshark:

1. Packet Dissection (Dissector Plugins - dissecting the captured packets)
2. Capturing File I/O (Wiretap library - reading and writing capture files)
3. Capture (interface for capture engine)
4. User Interface (QtWidgets - cross platform application development framework)
5. Utilities (miscellaneous helper code)
6. Help (external web browser and text output)

Components of Wireshark have been discussed in detail in Section 2 - Architecture.

1.4 Tool EcoSystem

Wireshark is developed by *application developers* as well as *library developers*. Along with the developers, OSS Contributors can also submit patches, which can be added to the release after reviews by the developers.

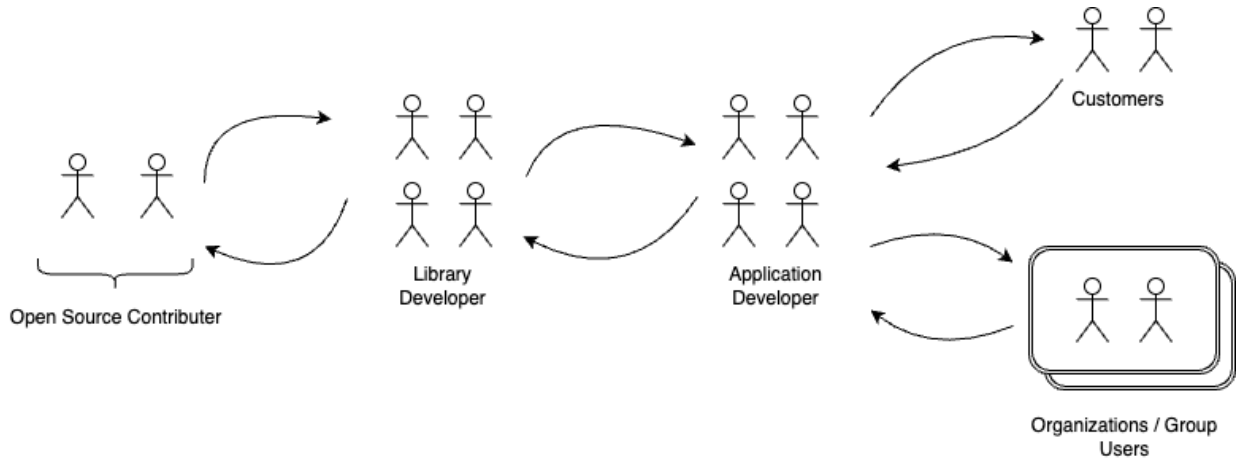

Figure : EcoSystem Diagram for WireShark Application

Responsibilities of Application Developers have been listen below -

- Develop & Maintain the User Interface using QtWidgets for Wireshark application.
- Integrate the User Interface with the libraries for smooth user experience.
- Develop & Maintain the interface for the capture engine.
- Develop required web applications for supporting help features.
- Bug Fixing

Responsibilities of Library Developers have been listed below -

- Write dissector plugins for dissecting packets.
- Develop & Maintain Wiretap Libraries which read and write capture files in specified file formats.

- ❖ Develop and Maintain utilities required for application.

Scope of actions for Open source contributors has been listed below -

1. Write new dissector plugins/libraries to support newer protocols or updated versions of previous protocols.
2. Add Support for different file formats.
3. Bug Fixing

1.4 Conferences for Wireshark Community

Developers along with users are a part of the OSS Community. **Sharkfest** is a series of annual educational conferences staged in various parts of the globe and focused on sharing knowledge, experience and best practices among Wireshark developers and user communities. It was launched in 2008. [26] There are mainly two shark fests in a year, SharkFest US and SharkFest Europe.

2. Architecture

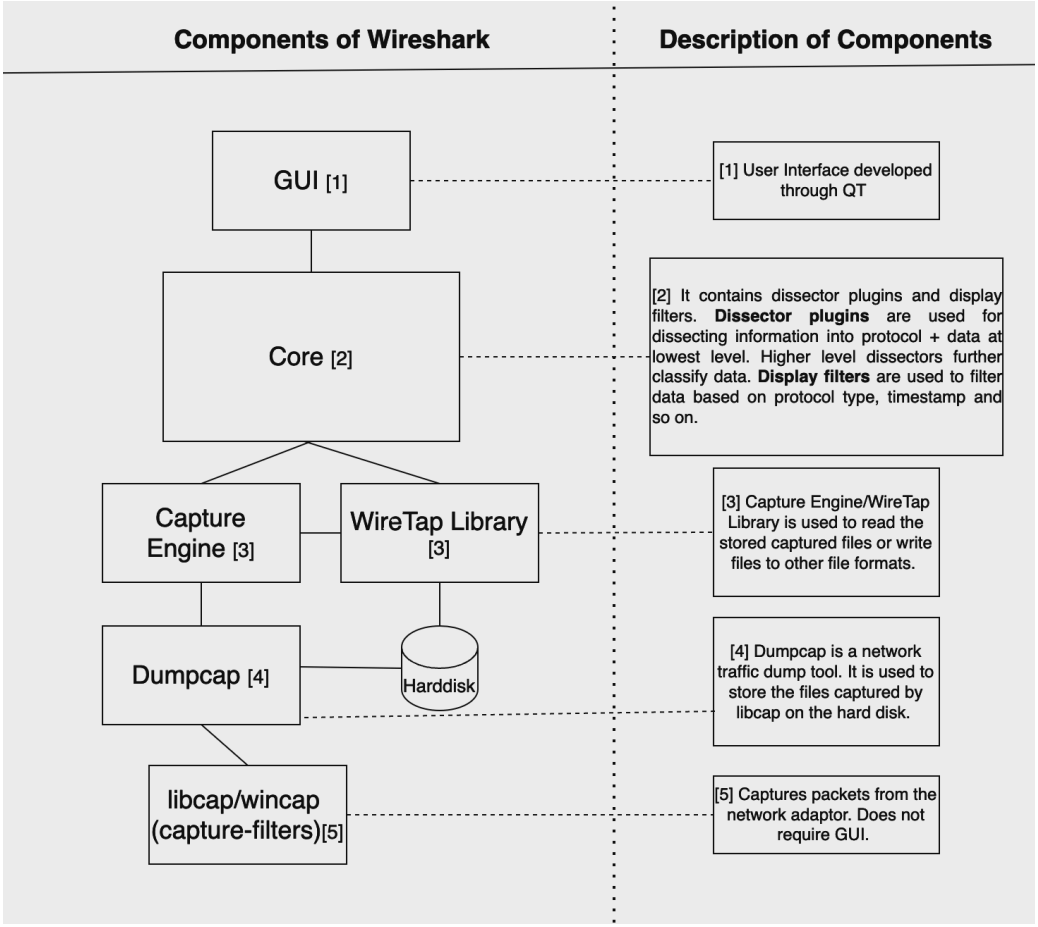

Figure : Components of Wireshark Architecture with description

Detailed Description of components [1] -

1. ***GUI****:* Handling of all user input/output (all windows, dialogs and such).
   Development framework : Qt - A cross-platform application development framework providing the foundation for Wireshark's graphical user interface (GUI) and core functionalities. Qt offers portability across various operating systems like Windows, macOS, and Linux.

2. ***Core***: Core is the main source code of the application. EPAN directory includes all the wireshark libraries that handle the captured data. It reads & manipulates the data stored by the Capture engine and displays on the User Interface. For customization, display filters are coded in the *epan* for filtering data based on user requirements.
   a. ***EPAN****:* Enhanced Packet ANalyzer (The packet analyzing engine, the heart of Wireshark) — the packet analyzing engine. Epan provides the following sub directories:
      i. *Protocol Tree*: EPAN dissectors capture network packets and build a hierarchical tree structure representing the different protocol layers involved in the communication. Each layer displays relevant information specific to that protocol, allowing you to visualize the data flow and identify potential issues.
      ii. *Dissectors Plugins*: Translate raw packet data into human-readable information.
      iii. *Display Filters*: EPAN empowers users to filter captured traffic based on specific criteria. This module handles this functionality. You can construct complex filter expressions using various operators, keywords, and protocol fields to isolate relevant packets from large datasets. This allows you to focus on specific aspects of network communication, like identifying specific devices, protocols, or error messages.

3. ***Capture***: The capture module acts as the bridge between the user and the capture engine, handling:.
   a. *Communication with capture engine*: Translating user requests into instructions for the underlying engine and receiving captured packets.
   b. *Capture engine interaction*: Utilizing platform-specific libraries (WinPcap/libpcap) to access network interfaces and manage capture operations by utilizing platform-specific libraries (WinPcap/libpcap) to access network interfaces
   c. *Capture filtering*: Applying user-defined filters to selectively capture packets based on specific criteria.
   d. *Capture file management*: Creating and managing capture files for storing captured data.

4. ***Wiretap***: The wiretap library is used to read and write capture files in libpcap, pcapng, and many other file formats.

5. ***Dumpcap***: The capture engine itself. This is the only part that executes with elevated privileges. Here's a breakdown:
    a. Wireshark captures network packets flowing through a chosen network adapter.
    b. These packets are saved to a file on your hard disk for further analysis.
    c. To ensure the security and stability of Wireshark, the functionalities requiring elevated privileges are isolated into a separate program called *dumpcap.*
    d. This separation allows the rest of the Wireshark code (dissectors, user interface, etc.) to run with normal user privileges.

6. ***Libpcap/Npcap***: These are external libraries that provide the packet capture capabilities that are central to Wireshark's core functionality. The filtering in Npcap and Libpcap works at a much lower level than Wireshark's display filters and uses a significantly different mechanism. That's why there are different display and capture filter syntaxes.

3. Customisation

Following features can be added/modified in Wireshark application-

- Add support for a new protocol (i.e., add a new dissector)
- Change or extend an existing dissector
- Add support for other file formats
- Adding a Tap Handler (to get event driven notification on packets matching certain protocols and/or filters)

3.1 Dissector Plugins

To Dissect **-** To analyze anything in a minute detail. Dissector plugins peel all the layers of a packet and dissect the packet based on certain criterias which helps the user analyze the network traffic.

Each dissector decodes its part of the protocol and then hands off decoding to subsequent dissectors for an encapsulated protocol. *Every dissection starts with the Frame dissector which dissects the details of the capture file itself* (e.g. timestamps). From there it passes the data on to the lowest-level data dissector, e.g. the Ethernet dissector for the Ethernet header. The payload is then passed on to the next dissector (e.g. IP) and so on. **At each stage, details of the packet are decoded and displayed** [5].
Wireshark dissects packets in what it calls *'two-pass*' dissection [4]. Wireshark performs a first pass of dissecting all packets as they are loaded from the file. All packets are dissected sequentially and this information is used to **populate Wireshark's packet list pane** and to build

state and other information needed when displaying the packet. Wireshark later performs 'second pass' ad-hoc dissections on the packets that it needs data from. This enables Wireshark to fill in fields that require **future knowledge,** like the 'response in frame number' fields, and correctly calculate reassembly frame dependencies[4].

3.2 Coding Standard for Dissector Plugins

Even though customising dissector plugins is available for the open source contributor, it is recommended to follow the specified formats/prototypes mentioned in the documentation for avoiding compatibility issues.

For writing a new dissector, following steps need to be followed -

1. Construct a protocol tree
2. Protocol Registration
3. Field Registration
4. Adding Items and Values to the Protocol Tree

The protocol tree, or proto_tree, is a GNode, the N-way tree structure available within GLIB. Add routines which allow dissectors to add items and new branches to the tree. Many of the columns are handled by code outside individual dissectors; most dissectors need only specify the value to put in the "Protocol" and "Info" columns. [3]

Pyreshark - A Wireshark plugin providing a simple interface for writing dissectors in Python.

3.3 Usability Guidelines for Plugin Developers

To ensure the portability and robustness of Wireshark, guidelines have been provided in the ReadMe files [6]. Even though numerous customisation options are available for the Open Source contributor, it is recommended to follow the structure mentioned in the documentation. For every capture file that Wireshark reads, the capture follows a structure called "*cfile*".

Similarly, while creating a dissector plugin, following steps need to be followed -

1. The plugin should be placed in a new plugins/epan/diss_name directory which should contain at least the following files -
    a. ReadMe
    b. CMakeLists.txt
2. To add the custom plugin dissector to the build as a custom extension, contributor has to pass the custom plugin dir on CMake generation step command line or edit CUSTOM_PLUGIN_SRC_DIR to set the relative path of your plugin in set() method.
3. Set up a working root for wireshark to successfully add your plugin to wireshark.
4. Register elements in your plugin for the toolbar along with a callback for activation of an item.

Unless the above mentioned steps are followed, Wireshark will not be able to use the custom plugin.

4. Configurations

Majorly, there are three settings which can be configured in Wireshark application-

- Capture Filters
- Display Filters
- Coloring Packet Displays

4.1 Capture Filters

Capture Filters are used to capture the data of specified protocols. Capture filters are set before starting a packet capture and cannot be modified during the capture[7]. It can be used to block remote session traffic or capture only TCP/UDP traffic to or from the specified port and many more actions.

It can be observed from the below figure that the user gets the option to set capture filters before starting the capturing of data.

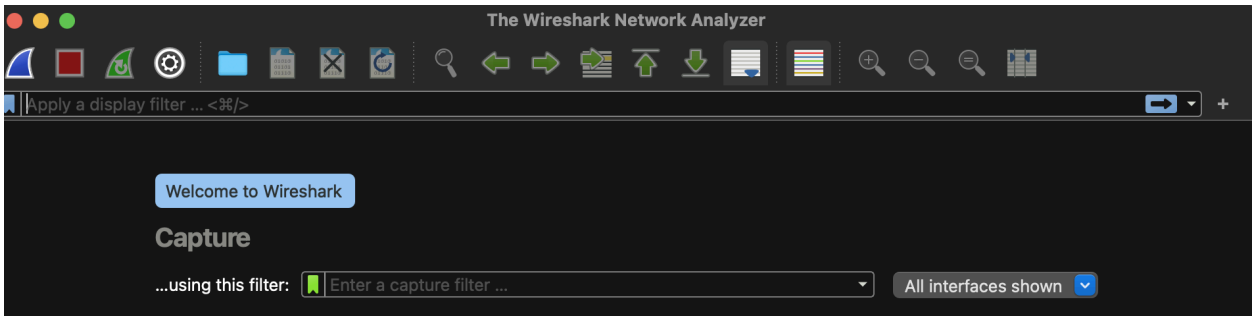

Figure : Wireshark Application Main Window before capturing packets

4.1.1 Working of Capture Filters

Capturing is done using a two task model: the currently running (***parent***) process will spawn a ***child*** process to do the real capture work, namely controlling libpcap. When a capture is started, the parent builds a "command line" and creates a new child process with it. A pipe from the child to the parent is created which is used to transfer control messages[8]. The child will initiate libpcap and send the parent a "*new capture file is used*" control message through the pipe. The child cyclically takes the packet data from libpcap and saves it to disk. From time to time it will send the parent a "*new packets*" control message. If the parent process receives this "*new packets*" message and the option "*Update list of packets in real time*" is used, it will read the packet data from the file, dissect and display it[8].

While this child process is running cyclically, it will call capture_pcap_cb() for every packet captured. Inside this, the packet data is converted into wtap (wiretap) format and saved to file. Beside saving, it is trying to do some basic dissecting (for the statistic window), by calling the appropriate capture function. In the capture parent reads data from the pipe and shows it on the main screen.

4.1.2 Examples of Capture Filters

To set capture filters, the user must be familiar with linux commands for network monitoring. Some examples are given below -

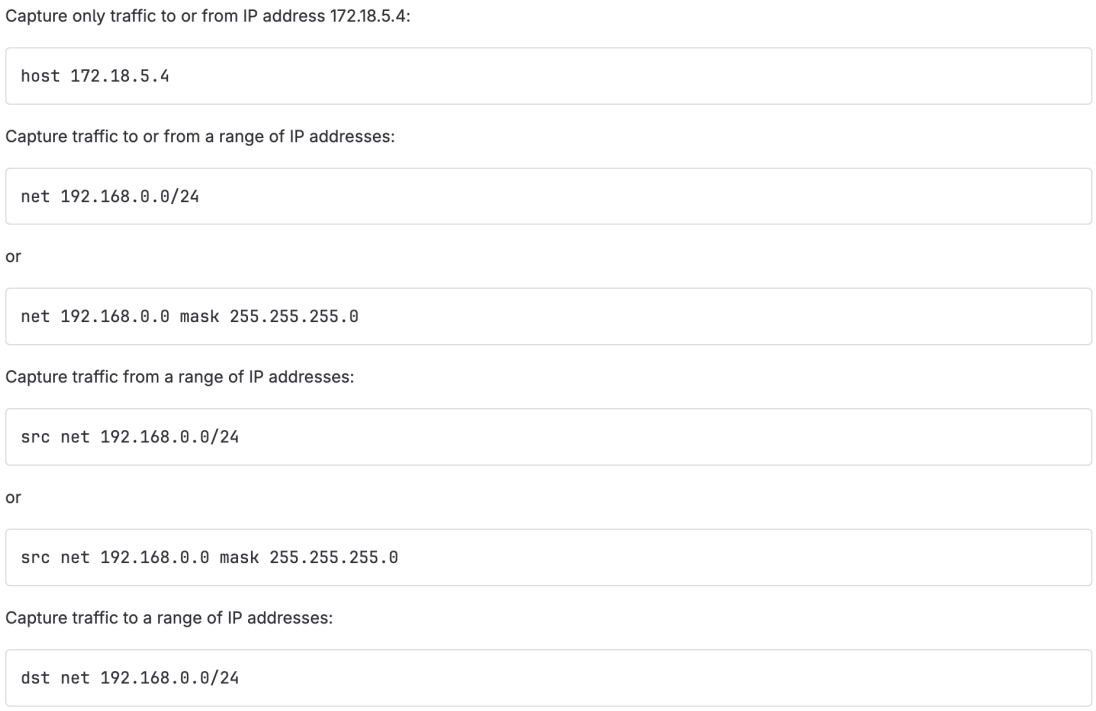

Figure : Examples for Capture filters [7]

4.2 Display Filters

Display filters are used for general packet filtering. The background code creates an equality check for the specified protocol and displays only the packets which exhibit that protocol. For detailed description of currently available protocols in Wireshark refer here (https://www.wireshark.org/docs/dfref/).

Display filters are not to be confused with Capture Filters. Capture Filters are used to capture traffic based on specified protocol whereas display filters are used to display traffic based on specified protocol. Suppose, the *user has set tcp as a capture filter*, then only tcp traffic will be captured and displayed. But if the user does not set any capture filter and *set tcp as display filter*, then it will display only tcp traffic and if we remove the filter, it would display packets with other protocols as well. **This wouldn't have been possible if tcp was set as a capture filter.**

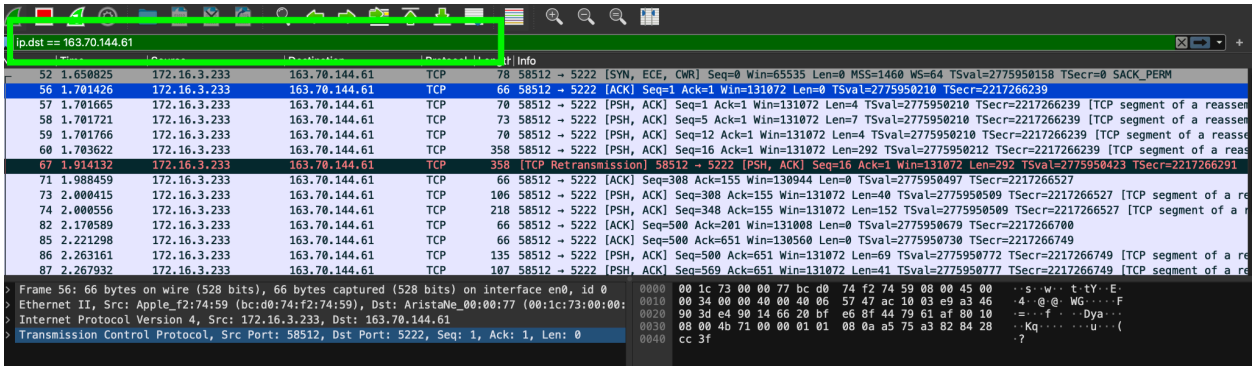

Figure : Display filter set to filter packets with a specified destination IP Address

4.2.1 Working of Display Filters

It works on the principle of coloring rules. Every proto_item in the proto_tree has an 'abbrev' field and a 'type' field, which tells the display filter engine the name of the field and its type (what values it can hold). The pointer to the proto_tree (in this case, 'protocol tree'), is passed to the top-level protocol dissector, and then to all subsequent protocol dissectors for that packet, and then the GUI tree is drawn via proto_tree_draw()[9].

4.2.2 Examples of Display Filters

Display filters are domain specific with boolean expressions. Some examples are listed below-

Show only SMTP (port 25) and ICMP traffic:

```
tcp.port eq 25 or icmp
```

Show only traffic in the LAN (192.168.x.x), between workstations and servers -- no Internet:

```
ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16
```

TCP buffer full -- *Source is instructing Destination to stop sending data*

```
tcp.window_size == 0 && tcp.flags.reset != 1
```

Filter on Windows -- *Filter out noise, while watching Windows Client - DC exchanges*

```
smb || nbns || dcerpc || nbss || dns
```

Figure : Examples of Display Filters [9]

4.3 Coloring Packet Displays

Coloring rules increase the readability of the output. Both display filters and coloring rules share the same syntax.

4.3.1 How to Color Packets

To colorise a packet display based on filters, End user needs to go to View >> Coloring Rules and set background/foreground colors for filters. They can also import the coloring rules using "Import" or export their current rules for later use.

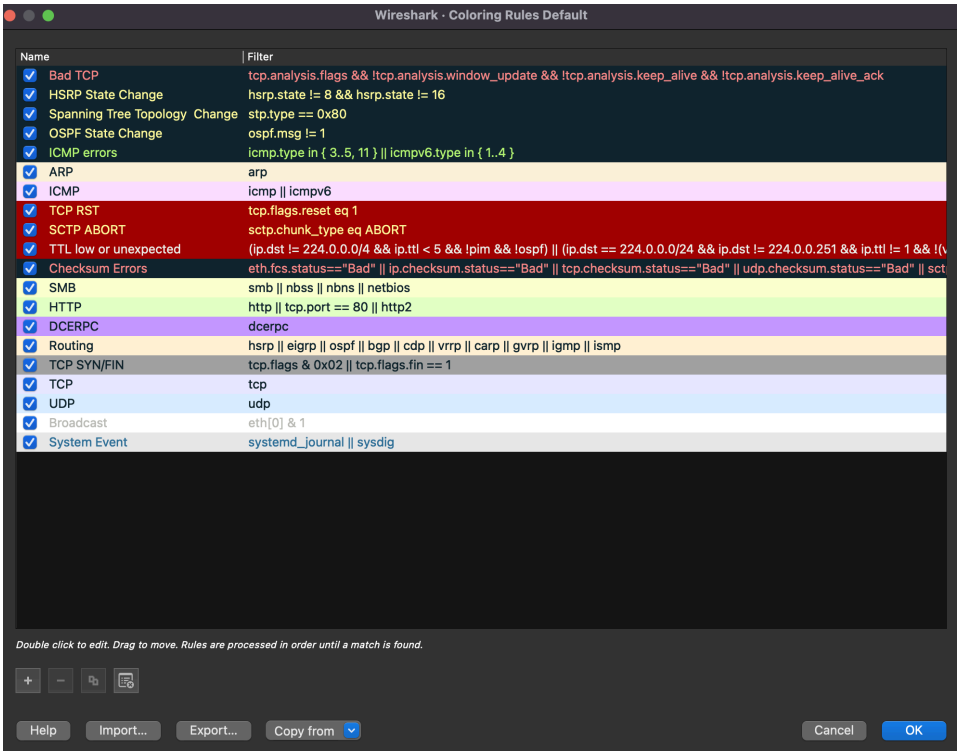

Figure : Interface for setting Coloring Rules

4.3.2 Purpose of Coloring Packets

Once the user starts the session, the wireshark application starts capturing packets and displays according to the set display filters. Here, the advantage of using coloring rules is that it is easier to filter the content. For instance, we set "*tcp*" as a display filter, then it will display all packets with TCP protocol. But using coloring rules, we can distinguish between tcp and bad tcp packets within the same output window. Please refer to the image, on the next page, for better understanding.

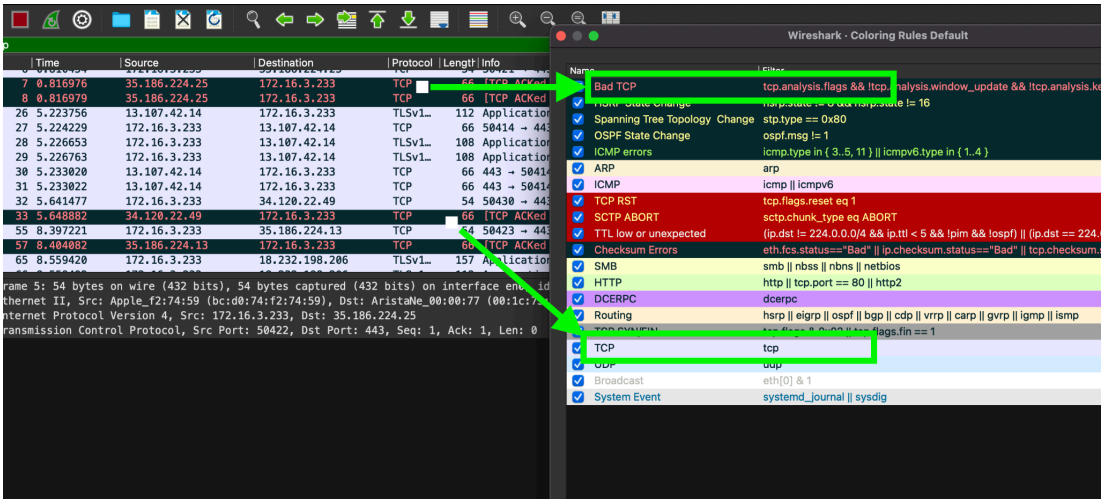

Figure : Using Coloring rules to distinguish packets within a single output window

4.4 End User Development

To use the wireshark application, it is expected that the end user has a basic understanding of network monitoring commands and network protocols. Keeping that in mind, we can say that end users have the possibility of configuring display filters and colorise packets. If an end user with expertise is using wireshark, they can also configure capture filters for meticulous analysis.

5. Development Cycle

5.1 Setting up Git Repository

Either *make* or *Ninja* can be used to build Wireshark; at least one of those must be installed. The following must be installed in order to build Wireshark:

- a C compiler and a C++ compiler;
- the Flex lexical analyzer;
- Python 3;
- CMake;

Following commands should be used to setup the repository and prepare the environment[10] -

```
$ git clone -o upstream git@gitlab.com:wireshark/wireshark.git
$ cd wireshark
$ git remote add downstream git@gitlab.com:USERNAME/wireshark.git
$ cp tools/pre-commit .git/hooks
$ chmod a+x .git/hooks/pre-commit
```

5.2 Merge Requests

After submitting the code, Create a merge request and ensure that "*Allow commits from members who can merge to the target branch*" is selected. Developers will review and changes are incorporated, if approved, in the next release[11].

Before submitting any patches, contributor has to perform the following steps on the patch -

- Perform Fuzz Testing (Stress Testing)
- Verify only approved functional calls are used
- Verify header files
- Verify display filter fields are named properly

5.3 Code Review

All the new features, improvements and prioritising tasks is done by a team including leadership along with technical members. Main members of the team are as follows -

1. Gerald Combs (*Owner*)
2. Guy Harris (*Maintainer*)
3. Peter Wu (*Maintainer*)
4. Roland Knall (*Owner*)
5. Tod Beardslay (*Reporter*)

In the documentation, It has not been specified whether there is another team of developers that work along with the leadership. However, after reading the issue board, it can be assumed that there is another team of developers supporting the leadership in maintaining the platform.

5.4 Deployment Pipeline

The Wireshark development team uses GitLab's continuous integration (CI) system to automatically build Wireshark for each Git merge request and commit. Automated builds provide cross-platform testing, ensuring that a developer on one platform doesn't break the build on other platforms. It also acts as a health indicator for the source code. It creates a source tarball, binary package, or installer, followed by Automated Regression Tests. GitLab's CI marks successful jobs with a green checkmark and failed jobs with a red "X". [12]

5.5 Release Details

The Wireshark download page lists three types of releases: *Stable*, *Old Stable*, and *Development*.

- The Stable release is the latest official version of Wireshark. In most cases this is the version you should use.

- The Old Stable release is an older official version of Wireshark which is still supported. You may be required to use one of these, for example if your organization has strict software approval policies.
- The Development version is used for testing new features

Official Wireshark release numbers are constructed as a three number tuple: **Major.Minor.Maintenance**. In some cases, such as the file version for Windows executables, a build number may be present: **Major.Minor.Maintenance.Build**.

The *Major release number* has reached **1** after numerous years of development. Only when the program undergoes a significant change, this number will be incremented. The change to Qt as primary GUI toolkit was such a change, resulting in the number to be set to **2**. The change to **3** came about when the GTK+ GUI toolkit was dropped and Npcap as a Windows capture library was introduced. The major release number was set to **4** when extending support to Qt6 as GUI toolkit, significant rework of the display filter engine and absorbing numerous other breaking changes. This number is likely to remain constant for a while [13].

The *Minor release number* basically follows an even/odd number scheme. For every official release an even number is assigned and the code is branched off for maintenance. The development continues on the next higher odd release number. Note that this development package is a moving target and meaningless without its commit ID.

The *Maintenance release number* gives you the stage of maintenance which that release is in. The *Build number* is the source code repository commit ID corresponding to that particular build.

6. Testing & Automated Test Suites

6.1 Automated Test Suite

The testing framework can run programs and check their stdout, stderr, and exit codes. It cannot interact with the Wireshark UI. Tests cover capture, command line options, decryption, file format support and conversion, Lua scripting, and other functionality[16].

The Wireshark sources include a collection of Python scripts that test the features of Wireshark, TShark, Dumpcap, and other programs that accompany Wireshark. The test suite uses pytest as a test runner**.** These are located in the test directory of the Wireshark source tree[14].

- Install two Python packages, pytest: `pip install pytest pytest-xdist`
- Build programs (“wireshark”, “tshark”, etc.): ninja
- Build additional programs for the “unittests” suite: ninja test-programs
- Run tests in the build directory: pytest

The test suite will attempt to test as much as possible and skip tests when its dependencies are not satisfied.

Custom Python test files can also be created outside of the Wireshark source tree. To include custom tests when running the Wireshark test suite, simply add the directory containing the test files to the pytest command line. *Note that file names must match the same conventions.* [15]

Pytest is a test framework which has full parallelization support (test-level instead of just suite-level), provides nice test reports, and allows modular fixtures. A fixture is a function decorated with @pytest.fixture and can either call pytest.skip("reason") to skip tests that depend on the fixture, or return/yield a value[16]. Test functions (and other fixture functions) can receive the fixture value by using the name of the fixture function as function parameters. In order for your tests to have access to the Wireshark test fixtures, add the following line in each test file:

```
from fixtures_ws import *
```

6.2 Tools for Fuzz Testing

6.2.1 Introduction to Fuzz Testing

Fuzz testing is the process of creating random or semi-random capture files, feeding them into Wireshark or TShark, and looking for crashes or other error conditions. The vast majority of Wireshark's code handles data that has been read directly from a live network or capture file; fuzz testing helps make sure that it handles this data safely. In particular, the packets created during fuzz testing are not necessarily well-formed, and Wireshark must be able to deal with these improper packets without crashing or doing anything else illegal. [17]

6.2.2 How To Fuzz Test

Wireshark's source comes with a script (**tools/fuzz-test.sh**) which can be used for fuzz testing. Simply run it from the root of the source tree and pass it the names of any capture files that are required to fuzz test. It works best if clang or gcc's AddressSanitizer is enabled. For eg.,

```
./tools/fuzz-test.sh myCapture1.pcap myCapture2.pcap ...
```

This script makes temporary copies of each file and introduces errors in them, then runs each of these edited captures through Tshark with many extra assertions and safety checks enabled. If an error is detected the script will produce as much information as it can and exit. It will leave behind the problematic edited capture file in the current directory, named something like fuzz-2013-01-01-1234.pcap. [17]

6.2.3 Additional Tools for Fuzz Testing

Apart from standard fuzz-test.sh script, Wireshark comes with two other tools for fuzz-testing:

- ☐ *randpkt* -- creates capture files with completely random data payloads
- ☐ *editcap* -- introduces errors into normal capture files

6.2.4 Randpkt (Random Packet Generator)

Randpkt is a small utility that creates a trace file full of random packets. By creating many randomized packets of a certain type, one can test packet sniffers to see how well they handle malformed packets. The sniffer can never trust the data that it sees in the packet because you can always sniff a very bad packet that conforms to no standard. randpkt produces very bad packets[18].

When creating packets of a certain type, randpkt uses a sample packet that is stored internally to randpkt. It uses this as the starting point for your random packets, and then adds extra random bytes to the end of this sample packet. For instance, To create random ARP packets, randpkt will create a packet which contains a predetermined Ethernet II header, with the Type field set to ARP. After the Ethernet II header, it will put a random number of bytes with random values.

Some of the configurable options for randpkt are -

- ➢ **-b <maxbytes>** : Default 5000. Defines the maximum number of bytes added to the sample packet.
- ➢ **-c <count>** : Default 1000. Defines the number of packets to generate.
- ➢ **-F <file format>** : Default pcapng. Sets the file format of the output capture file.
- ➢ **-t <type>** : Default Ethernet II frame. Defines the type of packet to generate. For eg:
  - Arp - Address Resolution Protocol
  - Bgp - Border Gateway Protocol
  - Bvlc - BACnet Virtual Link Control
  - Dns - Domain Name Service
  - Eth - Ethernet

6.2.5 Editcap (Edit Capture)

Editcap is a program that edits and/or translates the format of capture files. ***It is used to create modify/edit capture files which can be used for testing particular test scenarios.*** It reads some or all of the captured packets from the infile, optionally converts them in various ways and writes the resulting packets to the capture outfile (or outfiles). By default, it reads all packets from the infile and writes them to the outfile in pcapng file format.

Editcap can also be used to remove duplicate packets. Several different options (-d, -D and -w) are used to control the packet window or relative time window to be used for duplicate comparison. Editcap can be used to assign comment strings to frame numbers [21]. Editcap is able to detect, read and write the same capture files that are supported by Wireshark.

Some of the configurable options for editcap are -

- **-A <start time>** : Reads only the packets whose timestamp is on or after <start time>. The time may be given either in ISO 8601 format or in Unix epoch timestamp format.
- **-B <stop time>** : Reads only the packets whose timestamp is before <stop time>.
- **-C[offset:]<choplen> :** Sets the chop length to use when writing the packet data.. Positive values chop at the packet beginning while negative values chop at the packet end.
- **-D <dup window>** : Attempts to remove duplicate packets.
- **-E <error probability>** : Sets the probability that bytes in the output file are randomly changed. Editcap uses that probability (between 0.0 and 1.0 inclusive) to apply errors to each data byte in the file. For instance, a probability of 0.02 means that each byte has a 2% chance of having an error.This option is meant to be used for fuzz-testing protocol dissectors.
- **-F <file format>** : Sets the file format of the output capture file. The default is the pcapng format.
- **-L** : Adjust the original frame length accordingly when chopping and/or snapping (in addition to the captured length, which is always adjusted regardless of whether -L is specified or not). See also -C <choplen> and -s <snaplen>.
- **-o <change offset>** : When used in conjunction with -E, skip some bytes from the beginning of the packet from being changed. In this way some headers don't get changed, and the fuzzer is more focused on a smaller part of the packet. Keeping a part of the packet fixed the same dissector is triggered, that make the fuzzing more precise.
- **-s <snaplen>** : Sets the snapshot length to use when writing the data. If the -s flag is used, packets in the input file with more captured data than the specified snapshot length will have only the amount of data specified by the snapshot length written to the output file.
- **-t <time adjustment>** : Sets the time adjustment to use on selected packets. This feature is useful when synchronizing dumps collected on different machines where the time difference between the two machines is known or can be estimated**.**

6.3 Tools for Regression Testing

Happy Shark is the regression test framework for Wireshark consisting of a tool and a collection of capture files [22]. For running the tests, invoke the *make* file. The initial desired features were:

- Matching fields (the displayed text, byte offsets and length).
- Take a packet capture file and produce the expected "output".
- Have a filter that strips layers or just keeps a single layer.
- Allow preferences to be applied (SSL keys, port numbers, ...).
- Maybe check both single and second pass mode (tshark -2) to catch issues related to maintained state within a dissector.

6.3.1 Add External Tests

Add test/suite_external.py, which can dynamically generate tests from a configuration file. This is intended to make happy-shark useful, but it should make it easy to add simple TShark tests elsewhere. [15]

6.4 Other Tools

Apart from randpkt and editcap, there are few other internal tools which can be used to generate different types of capture files for testing purposes[20]. Other internal tools are listed below -

- Capinfos - It is a program that reads a saved capture file and returns any or all of several statistics about that file
- Mergecap - It merges multiple capture files into one
- Reordercap - It reorders input file by timestamp into output file
- Text2pcap - It generates a capture file from an ASCII hexdump of packets

6.5 Test Case Development

6.5.1 Code Coverage

Code coverage is a measurement method which can measure how much of your code was executed. It helps test case development. Be careful, however, because if the code reaches 100% code coverage with the test cases and all the test cases passed, it does not mean that the code is perfect. [23] The code coverage cannot detect if some checks have been left out. GCC(GNU Compiler Collection) along with libtool is required for code coverage measurement. GCOV is used for verifying code coverage analysis for Wireshark. Some of the known problems have also been specified in the documentation [23].

6.5.2 Secure Programming

Wireshark's code (especially the dissectors) is checked using a custom Perl script (tools/checkAPIs.pl) that checks whether various APIs are called. Unsafe APIs are 'prohibited' which raises errors on the buildbot if anyone checks in code using those APIs. Introducing API

routines to deal with TLVs (type, length, value) could eliminate a lot of hand-crafted dissector code with potential problems (like endless loops) [24].

Wireshark will not manipulate things on the network, it will only “measure” things from it. Wireshark doesn’t send packets on the network or do other active things[2]. Wireshark is responsible for monitoring and analysing traffic. Since, wireshark directly captures packets from the network adapter, it may contain sensitive information. **No sensitive information or vulnerabilities should not be exposed**. To ensure these coding ethics, it is very important to do secure programming.

7. Reporting & Fixing Bugs

The Wireshark community collects bug reports in an issues database at *https://gitlab.com/wireshark/wireshark/-/issues*. Issues filed in the issue database are monitored by the developers and solved as time permits. It may be that additional information is required to find the cause of the problem. A comment will be added to the issue report requesting that information. These changes will be sent to the email address associated with the reporter’s GitLab account, so it is important to monitor that mailbox.[19]

When reporting problems with Wireshark, following information is requested to be added :

1. The version number of Wireshark and the dependent libraries linked with it, e.g. Qt, GLib, etc. To obtain this information, use command *wireshark -v.*
2. A detailed description of the problem along with information about the platform used by the reporter to run Wireshark. .
3. Copy the text of that error/warning message, if any, so others may find the build step where things go wrong.
4. Steps necessary to duplicate the problem.

It is also recommended to mention the kind of network from which data was being captured or any other capture file details. Any other specifics of the platform not covered by the default bug fields are also appreciated. [20]

Important Links

---

1. Open Issues Board : https://gitlab.com/wireshark/wireshark/-/issues
2. Progress Tracking Board : https://gitlab.com/wireshark/wireshark/-/boards
3. Milestones Board: https://gitlab.com/wireshark/wireshark/-/milestones?sort=due_date_desc&state=all
4. Wireshark Manual Pages - https://www.wireshark.org/docs/man-pages/

References

[1] ‘How Wireshark Works’. Wireshark.org. *https://www.wireshark.org/docs/wsdg_html_chunked/ChWorksOverview.html* (accessed March 2024)
[2] WireShark User Guide’. Wireshark.org. https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroPurposes (accessed March 2024)
[3] S.Bjorlykke, A.Broman. ‘ReadMe Dissector Plugin’. Github. *https://github.com/wireshark/wireshark/blob/master/doc/README.dissector* (accessed March 2024)
[4] ‘Dissect Packets’. Wireshark.org. *https://www.wireshark.org/docs/wsdg_html_chunked/ChWorksDissectPackets.html*(accessed March 2024)
[5] ‘Packet Dissection’. Wireshark.org. *https://www.wireshark.org/docs/wsdg_html_chunked/ChapterDissection.html#ChDissectWorks* (accessed March 2024)
[6] G.Combs. ‘Wireshark ReadMe for Developers’. Github. https://gitlab.com/wireshark/wireshark/-/raw/master/doc/README.developer (accessed March 2024)
[7] G.Combs. ‘Capture Filters’. Gitlab. *https://gitlab.com/wireshark/wireshark/-/wikis/CaptureFilters*(accessed March 2024)
[8] D.Perry. A. Broman. ‘ReadMe Capture’. Github https://github.com/wireshark/wireshark/blob/master/doc/README.capture(accessed March 2024)
[9] D.Perry. A. Broman. ‘ReadMe Display Filter’. Github *https://github.com/wireshark/wireshark/blob/master/doc/README.display_filter*(accessed March 2024)
[10] M. Kaplan. ‘Submitting Patches’. Gitlab. https://gitlab.com/wireshark/wireshark/-/wikis/Development/SubmittingPatches (accessed March 2024)
[11] ‘Working with Wireshark sources’. Wireshark.org. https://www.wireshark.org/docs/wsdg_html_chunked/ChSrcGitRepository.html (accessed March 2024)
[12] ‘Automated Builds’, Wireshark.org, https://www.wireshark.org/docs/wsdg_html_chunked/ChIntroAutomated.html (accessed March 2024)
[13] https://gitlab.com/wireshark/wireshark/-/wikis/Development/ReleaseNumbers(accessed March 2024)
[14] ‘Wireshark Tests’. Wireshark.org https://www.wireshark.org/docs/wsdg_html_chunked/ChapterTests.html (accessed March 2024)

[15] ‘External Tests’. Wireshark.org https://www.wireshark.org/docs/wsdg_html_chunked/ChTestsExternal.html (accessed March 2024)
[16] ‘Test Coverage’. Wireshark.org https://www.wireshark.org/docs/wsdg_html_chunked/ChTestsStructure.html#TestCoverage
[17] G.Combs. ‘Fuzz Testing’. Gitlab. https://gitlab.com/wireshark/wireshark/-/wikis/FuzzTesting (accessed March 2024)
[18] ‘Randpkt’. Wireshark. https://www.wireshark.org/docs/man-pages/randpkt.html (accessed March 2024)
[19] “Reporting Problems and Getting Help”. Wireshark.org “https://www.wireshark.org/docs/wsdg_html_chunked/ChIntroHelp.html#ChIntroBugDatabase (accessed March 2024)
[20] C.Maynard, ‘Tools’, Gitlab. https://gitlab.com/wireshark/wireshark/-/wikis/Tools (accessed March 2024)
[21] ‘Editcap’. Wireshark. https://www.wireshark.org/docs/man-pages/editcap.html (accessed March 2024)
[22] ‘Happy Shark’, Github. https://github.com/wireshark/happy-shark (accessed March 2024)
[23] Gitlab Migration. ‘Code Coverage’. Gitlab. https://gitlab.com/wireshark/wireshark/-/wikis/Development/CodeCoverage (accessed March 2024)
[24] Gitlab Migration. ‘Secure Programming’. Gitlab. https://gitlab.com/wireshark/wireshark/-/wikis/Development/SecureProgramming (accessed March 2024)
[25] ‘About SharkFest’. Wireshark. https://sharkfest.wireshark.org/about/(accessed March 2024)