COMP3210 Coursework report
Network traffic analysing software

Ziqian Qin (zq4g21@soton.ac.uk)
8th May 2024

1. Introduction

Home networking has become familiar with the development of the Internet. A typical home network setup usually involves just a router provided by the Internet Service Provider. More devices can be added to the network via Wi-Fi or ethernet cables. These routers' network monitor features are simple, such as showing the number of connected devices and IP addresses. They failed to showcase the network's utilisation: which node sends more packets than the others and which services were heavily used by connected devices. This project will demonstrate software that shows insights about a network analysed from captured network traffic.

1.1. Project goals

The project aims to build an application that can import a network traffic dump, analyse it and show insights about the network traffic in a user-friendly way. The information shown should include the following:

- The IP and MAC address of the device that sends the most packets on the network.
- The IP and MAC address of the device that consumes the most bandwidth on the network.
- Top 10 visited IP addresses by bandwidth.
- Top 10 visited domain names by bandwidth.

The criteria mentioned above should be sufficient to understand the usage of a network. Moreover, as IPv6 is actively deployed by content hosts and Internet Service Providers, about 50% of devices in the UK are reachable via IPv6 [1]. The proposed software should also be capable of showing the percentage of IPv4 packets against that of IPv6 packets. To understand to what extent IPv6 is used in a network.

2. Design and implementation

The project aims to develop an analysis program with a graphical user interface. It will be implemented in Java with JavaFX. The program needs a parser to parse the pcap file. Java has many pcap libraries to choose from, including jpcap, jNetPcap. While both are wrappers of the libpcap library, jNetPcap employs a more native approach. A general parser for binary content

named Kaitai Struct is also available outside these libraries. According to [2], parsing a pcap file in Kaitai Struct has a considerable performance advantage over wrappers of libpcap. A pcap file can be huge and contain many packets so that a slower parser will result in a longer processing time. Hence, this project will utilise Kaitai Struct as the parser. A definition of the pcap file format is available with Kaitai Struct. It will be used to generate Java code to parse pcap files. Kaitai Struct's format specification follows a modular design. The pcap definition allows parsing to the level of each layer two entries. Further definition, in this case, IPv4/IPv6 packet definition, is needed to parse layer two entries into IP packets.

The program also needs a mechanism to detect the domain name of an IP address. The design for this goal will utilise SSL certificates and reverse DNS (rDNS) lookup. Java's built-in InetAddress class supports obtaining rDNS records from an IP address. However, an rDNS record of an IP address may not reflect the website hosted by that server. It could be an entry set up by the Internet Service Provider or just empty. An SSL certificate will be a more consistent way of getting the domain name hosted by the server. When a socket connects to a server with SSL enabled, the server should return its SSL certificate to the client. Hence, the server's domain name can be found on the SSL certificate. When a Java SSL socket connects to a server, the certificate will be validated during the handshake process. Since the program will connect to the server using an IP address, the SSL socket will check if the certificate provided by the server is valid for that IP address. This is not true for many cases, resulting in closed connections and failure to obtain the certificate. To address this issue, a custom trust manager proposed by [3] is used for that SSL socket. Its code is shown in Figure 1. The custom trust manager addresses the problem by trusting any certificate it has seen. It should not be a security problem as the program will not engage with the server further.

```
private static class SniTrustManager implements X509TrustManager {  1 usage  dz-paji

    @Override  3 usages  dz-paji
    public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {

    }

    @Override  5 usages  dz-paji
    public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {

    }

    @Override  6 usages  dz-paji
    public X509Certificate[] getAcceptedIssuers() { return null; }
}
```

Figure 1 Implemented custom trust manager.

Querying the domain name can be slow, as many connected servers may not be standard HTTPS servers, leaving the connection to timeout. The connection timeout is set to 1 second. For each packet not from a standard HTTPS server, the processing time is at least 1 second. Given the number of packets and distinct IP addresses, the processing time for a whole pcap file can take long. To optimise the performance, the following strategies were adopted:

- For a distinguished IP address, only query it once by saving its result for later.
- Only query for public IP addresses.

- Use multi-threading when parsing packets.

Thread-safe objects were used for concurrent operations, as shown in Figure 2.

```
private final ConcurrentHashMap<String, Integer> localTalkers = new ConcurrentHashMap<>(); // IP: Packet count.  18 usages
private final ConcurrentHashMap<String, Long> localTalkersData = new ConcurrentHashMap<>(); // IP: data(bytes)  17 usages
private final ConcurrentHashMap<String, String> addressResolution = new ConcurrentHashMap<>(); // IP: MAC  6 usages
private final ConcurrentHashMap<String, Long> dataCount = new ConcurrentHashMap<>(); // dstIP: Data(bytes)  21 usages
private final ConcurrentHashMap<String, ArrayList> sniRecords = new ConcurrentHashMap<>(); // dstIP: Arraylist(SNI(domain name))
private final ConcurrentHashMap<String, String> rDNSRecords = new ConcurrentHashMap<>(); // IP: rDNS  9 usages
private final ConcurrentHashMap<String, Long> sniDataCount = new ConcurrentHashMap<>(); // SNI: Data(bytes)  21 usages
private final AtomicInteger ipv4Counts = new AtomicInteger( initialValue: 0);  2 usages
private final AtomicInteger ipv6Counts = new AtomicInteger( initialValue: 0);  2 usages
private final Logger logger = LogManager.getLogger(PacketParser.class);  10 usages
private AtomicBoolean doSNI, dorDNS = new AtomicBoolean( initialValue: false);  11 usages
```

Figure 2 Thread-safe objects used by the parser.

The finalised program proceeds in the following steps:

1. Ask the user to import a pcap file with the option of enabling domain lookup and rDNS lookup.
2. Parse the pcap file and look up the domain name and rDNS if required.
3. Present the analysed results to the user.

3. Conclusion and discussion

An application was implemented to analyse a network traffic dump, as shown in Figure 3. It fulfils the goals defined by this project. It should allow the user to have a more comprehensive understanding of the utilisation of their home network.

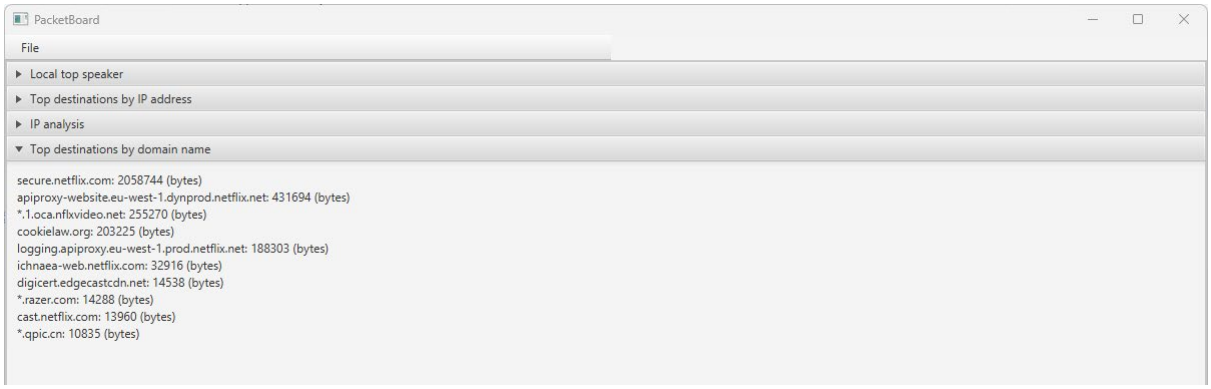

Figure 3 Implemented application.

Using the application requires the user to provide a pcap file. Only traffic destined for that node can be captured when capturing traffic on a node in the network. Therefore, capturing should be done on the router or a network tap to understand network usage. It is also worth mentioning that operating systems have slightly different pcap definitions. This project employed an H3C GR1108-P router to capture network traffic. It will write the magic number field in the pcap header as 0xa1b2c3d4 instead of the expected value of 0xd4c3b2a1 defined by Kaitai Struct.

Both values are valid but will change the ordering of other header values [4]. A possible solution to this problem is using Wireshark to save the dump in Wireshark pcapng format first, then convert the pcapng file back to pcap format.

The current design doesn't allow the identification of domain names for HTTP servers, as it does not employ SSL certificates. The analysis procedures do not cover details about the transport layer protocols used, which this project believes to be less significant for understanding network utilisation. Further work could be adding support to parse a packet to the application layer packet. Thus, the domain name of an HTTP server can be directly obtained from the HTTP header. While the change can provide more information about the network, such as the percentage of network bandwidth by each protocol, adding parsers for each application protocol can be bothersome. A short path can identify the application protocol by looking at the destination port of transport layer packets. And only include parsers of commonly seen application layer protocols, such as HTTP.

The domain name and rDNS lookup are done on the local machine. Therefore, the machine must have IPv6 access when analysing a pcap file containing IPv6 packets for the domain name and rDNS lookup.

Reference

[1] 'IPv6 test - Statistics for United Kingdom'. Accessed: Dec. 05, 2023. [Online]. Available: https://ipv6-test.com/stats/country/GB

[2] G. Alexandro, 'Java PCAP file parser library'. Accessed: May 08, 2024. [Online]. Available: https://stackoverflow.com/questions/26978618/java-pcap-file-parser-library

[3] Daniel, 'Trusting all certificates using HttpClient over HTTPS'. Accessed: May 08, 2024. [Online]. Available: https://stackoverflow.com/questions/2642777/trusting-all-certificates-using-httpclient-over-https

[4] Wireshark Wiki, 'Development/LibpcapFileFormat', Development/LibpcapFileFormat. Accessed: May 09, 2024. [Online]. Available: https://wiki.wireshark.org/Development/LibpcapFileFormat#file-format