NwSec HW2 Report

Shan Zhu, Nairong Zhang, Ziming Chen

***Abstract*—This paper outlines our project to explore the application of the BitTorrent protocol for peer-to-peer file sharing(P2P). qBittorrent application is chosen as the BitTorrent client running on both server and client. The client is running in the Kali Docker container created on the Kali Virtual Environment while the server is running on the Ubuntu Docker container on the Raspberry Pi. Throughout the paper, we will first introduce our project as a whole as well as explaining how P2P architecture and BitTorrent protocol are implemented. Then we will describe in detail how we developed the corresponding building and runtime environment for the client and server to run the application. Moreover, we will describe how we successfully ran the qBittorrent application and several challenges we met as well as how we solved them. We will also use Wireshark to capture the packets transferred between the client and server to justify the feasibility of our application and its environment. Finally, we will review the whole process and conclude what we have learned from the project as well as stating a few difficulties.**

I. Introduction

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts.

To implement a simple but fully-functioning P2P architecture, we decided to build out a baseline environment for the BitTorrent application on two Debian-based systems. a Kali environment in the VMWare as the client and an Ubuntu ARM environment on the Raspberry Pi as the server. For the BitTorrent application, a host can be either a server or a client. The BitTorrent trackers specified in the torrent provide a list of files available on other peers' computers and allow the client to find those peers also known as seeders. To realize the BitTorrent functions, we install qBittorrent, a BitTorrent client, on both our client and server environment. After installation, we initiated the BitTorrent client to perform the P2P communication.

In this project, we started by installing Kali and Ubuntu on the Virtual Environment and Raspberry Pi, configured the network, and installed Docker in both environments to enable the flexibility and portability of the application. Then we developed the building and runtime environments for both the client and server for the qBittorrent application. Due to the computational limitation of RPI, we also attempted to perform cross-compile in Kali Virtual Environment to find the possibility to accelerate the compiling process. Through qBittorrent, we initialized communication between any arbitrary client or server on our P2P architecture.

II. Background

A. Brief Description of BitTorrent Protocol

BitTorrent (BT) is a peer-to-peer communication protocol for file sharing, with a group of hosts downloading and uploading the same torrent and transferring data between each other within a particular group called a "swarm". A computer uses a BitTorrent client to upload or download files. When a computer loads a .torrent file into its BitTorrent client, it joins a "swarm". The BitTorrent client will contact "BitTorrent trackers" specified in the .torrent file in order to find relevant peers. Those peers will then upload the file data to the client.

The most important feature of BitTorrent is that, every client downloading the files can also be a server with the capability to upload the files they have downloaded. This decentralized mechanism greatly accelerates the downloading speed as well as reducing the bandwidth compared with traditional downloading mechanisms. Also, a tracker only needs to keep track of the clients connected to the swarm, but not the content of the files being transferred, thus makes a relatively small tracker bandwidth already sufficient to support large numbers of users.

B. Protocol Implementation

BitTorrent protocol is implemented at Application Layer and uses both TCP and UDP at the Transport Layer. Typically, TCP is used by BitTorrent applications as P2P communication requires reliable data transfer. But UDP can also be used when DHT (distributed sloppy hash table) technology is applied. DHT protocol is a UDP-based peer-to-peer tracker protocol. When a BitTorrent client enables DHT to add a torrent and try to download a file, the DHT node will contact nearby nodes, who then contact other nodes until they finally locate the information about the file inside the torrent. In this setting, each node becomes a tracker, and there is not a traditional tracker anymore who serves as a "centralized" server to some extent and manages the "swarm", and BitTorrent is now a completely decentralized tracking peer-to-peer file-sharing system. In our application, however, both tracker and DHT are enabled to best support the successful communication between the BitTorrent client and server if the tracker fails.

C. qBittorrent

The BitTorrent client we decided to use is an application called qBittorrent, which supports multiple platforms such as Debian and ubuntu. One special feature and advantage of qBittorrent is that it has a feature-rich Web UI, allowing users to remotely control the qBittorrent application. This feature makes it ideal to be the BitTorrent client we choose to use in

our system since we use the Ubuntu server, a headless system without graphical user interface.

More specifically, we will use qBittorrent-nox as the BitTorrent client on our ubuntu server. The main difference is that qBittorrent has a Graphical user interface while qbittorrent-nox is terminal-based and has only a web-interface.

III. Project Goal

This project is divided into four major parts, which are basic environment setup, building environment development, runtime environment development and the running of qBittorrent application. The basic environment setup mainly prepares the Linux environments and tools for the client running in the VMWare and server running on the Raspberry Pi. Building environment development intends to create two Docker containers that successfully installed the qBittorrent application for both the client and the server and generate the executable binary file that could be used in the runtime environment. To accelerate the compilation for the RPI, we attempted to run cross-compile on Kali in the VMware to generate the binary file that could run on RPI. As for the runtime environments, their main purpose is to provide the environments to run the executable binary files for the qBittorrent application for both the client and the server. After successfully run the application, we will perform the BitTorrent application by tranferring Finally, We will be justifying our success of this P2P architecture by showing the results from our Wireshark captured packets.

IV. Project Execution Highlights

A. Basic Environment Setup

For our application, the client is running in the Kali Docker container in VMWare and the server is running in the Ubuntu Docker container in the Raspberry Pi. To begin with, we installed the Kali system in the VMware and Ubuntu system in Raspberry Pi. Considering the flexibility and portability of our application, we then installed Docker in both environments. In this way, the applications running in Docker containers can be deployed easily to multiple different operating systems and hardware platforms. To better manipulate the Raspberry Pi, we also configure its network and installed SSH for remote control.

B. Building Environment

As mentioned, the qBittorrent application will be installed in both the client and server environment.

1) Client Building Environment: The client is running in the Kali Docker container built on the Kali Virtual Environment. Since qBittorrent is a native application written in C++, which uses Boost, Qt toolkit, and libtorrent-rasterbar libraries, we need to properly install these dependencies in the building environment. Specifically, the current version of qBittorrent-4.2.2 requires the libtorrent-rasterbar packages to be 1.1.0 version and above, while the default version by running "apt-get install" is 1.0.6. Thus, we need to manually install the libtorrent-rasterbar libraries from the source code to ensure the successful installation of the qBittorrent application.

2) Server Building Environment: The server is running in the Ubuntu Docker container running on the Ubuntu system in the Raspberry Pi. We initially tried to run cross-compile on the Kali Virtual Environment to speed up the compilation of the application but was challenged by the linking issues. Therefore, we chose to compile our application directly on the Raspberry Pi instead. Similar to the client building environment, we need to install the same dependencies for the qBittorrent application, i.e. Boost, Qt toolkit, and libtorrent-rasterbar libraries, but in the Ubuntu environment.

3) Cross-Compile: Due to the limited resources on the Raspberry Pi platform, we first intended to perform cross-compile in the Kali VM environment to generate an executable binary file for the target platform on RPI, by installing the closest GCC library to the one RPI has. We also utilized GNU Autotools packages (i.e. autoconf, automake, and libtool), with the notion of the host platform, indicating the platform where the compiler is actually compiled and the target platform, indicating what type of object code the package itself will produce.

We have successfully performed the compilation part, processing the source code files and generating the object files with the cross-compiler. However, when linking the qBittorrent application to create a single executable file from multiple object files, we encountered the Relocations Errors which will be explained in the Difficulty Section below. Thus, we chose to run the server building environment on the RPI directly.

C. Runtime Environment

We have successfully created the executable binary file and dependent libraries of the qBittorrent application for both client and server after running the building environment. To run the application, we need to provide runtime environments with all the dependencies it needed. Since the libtorrent-rasterbar package is manually installed and is essential for the application, it will be time-consuming if we install it again in the runtime environment. Thus, we directly copy the dynamically linked shared object libraries(.so file) from the building environment into the runtime environment that can be directly used by qBittorrent.

1) Client Runtime Environment: The client runtime environment will still be a Kali Docker container built on the Kali Virtual Environment. Since the .so file for libtorrent-rasterbar is directly copied into the container, we only need to install the Boost and Qt toolkit with other essential packages in the Kali container.

2) Server Runtime Environment: The server runtime environment will also be the Ubuntu Docker container running on the Ubuntu system in the Raspberry Pi. Similar to the client runtime environment, we only need to prepare the Boost and Qt toolkit and other essentials for the Ubuntu container and copy the .so file for the libtorrent-rasterbar from the server building environment.

D. qBittorrent Application

After creating the runtime environment for the client and server, we can directly enter the corresponding container to

start the application.

When running the qBittorrent application, one of the biggest challenges we have met is that we are always in some Local Area Networks no matter we are in our homes or in the JHU campus. To be more specific, we are always behind a NAT, which means our IP addresses are private and only have meaning to devices within the LAN. When we want to contact hosts beyond the LAN network to the global network, the router will use Network Address Translation to replace our addresses and port numbers, and represent us to communicate with hosts beyond the LAN. In our P2P application, everything goes well if we just want to download some files from other peers' computers, as NAT-enabled routers will handle this situation well. However, if we want to upload our own files to other peers, a critical problem arises as we cannot act as a server and accept TCP connections from outside.

To address this problem, a technique known as Universal Plug and Play (UPnP) is created, which is a protocol that allows a host to discover and configure an NAT nearby. For example, with UPnP enabled, the BitTorrent application running in my PC can request a NAT mapping between my computer's LAN address (10.0.0.196, 9090) and the WAN address of the NAT-enabled router (68.84.9.152, 22500). Now if the NAT accepts and creates the mapping, then hosts beyond the LAN can initiate TCP connections to (68.84.9.152, 22500), which is the WAN address of my router, who will then forward the requests to the application running on my computer with address (10.0.0.196, 9090).

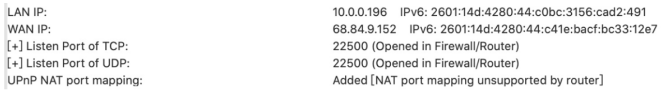

Fig. 1. UPnP enabled

Moreover, recall that we can use tracker information to keep track of connected peers. Therefore, we can make our BitTorrent application advertise to the tracker that it is available at (68.84.9.152, 22500), as shown in Fig.2. In this manner, a host running BitTorrent client is able to contact the tracker and learn that our server is currently running at (68.84.9.152, 22500). If this external host sends a TCP SYN packet to (68.84.9.152, 22500), our NAT will then receive the TCP SYN packet and modify the destination address to (10.0.0.1, 3345) according to the mapping entry set by UPnP, and forward the packet to my computer.

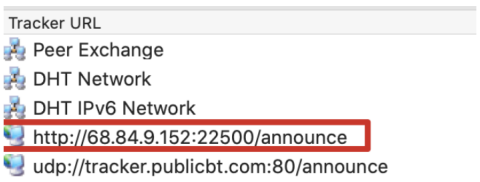

Fig. 2. BitTorrent Tracker information

Under some circumstances, UPnP won't always work as it requires both host and NAT be UPnP compatible. When we were testing our application, we indeed found that sometimes UPnP did not work as expected. So we tried an alternative way to make our router forward some incoming requests to our computers. Specifically, we manually set Port Forwarding in our router. Port Forwarding is similar to a wall, and the Wireless Gateway only opens ports for permitted or designated Internet traffic while blocking other risky requests with unopened ports. As shown in the picture below, we set the router to open port number 22500 and forward requests with this destination port number to my computer with LAN IP address 10.0.0.196. As for the BitTorrent application running on my computer, it is listening to this port number on the router, and is waiting to accept incoming requests. Note that UPnP automatically installs instances of port forwarding in NAT-enabled routers.

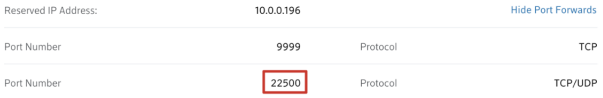

Fig. 3. Alternative Port Forwarding

After successfully setting all the configurations above, we can now create our own torrent files in order to act as a server and upload the files to others. Note that we need to add tracker information and also enable DHT to best enable the communication between the client and server. Since we do not have a Graphical User Interface for qBittorrent-nox, we need to get access to the application and control it remotely using its Web-UI feature.

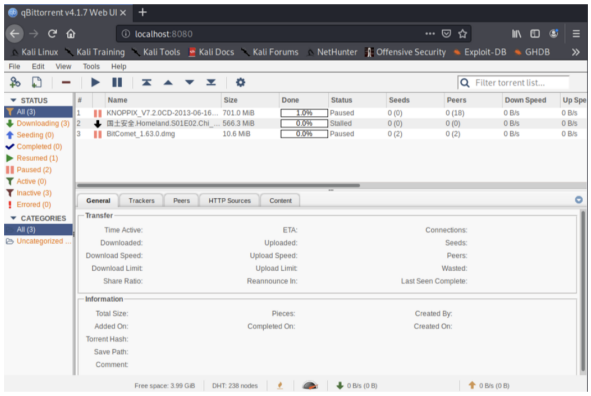

Fig. 4. qBittorrent WebUI

After successfully creating the torrent, we can distribute the torrent file to other hosts, so they can load the torrent into their BitTorrent application and download the corresponding files from our computer. In our BitTorrent application, either host on rpi or Kali VM can be a client or a server and can share files with each other. A Wireshark screenshot is shown below. We can notice that BitTorrent protocol (application-layer protocol)

or TCP protocols (transport-layer protocol) were used among the peer-to-peer communication.

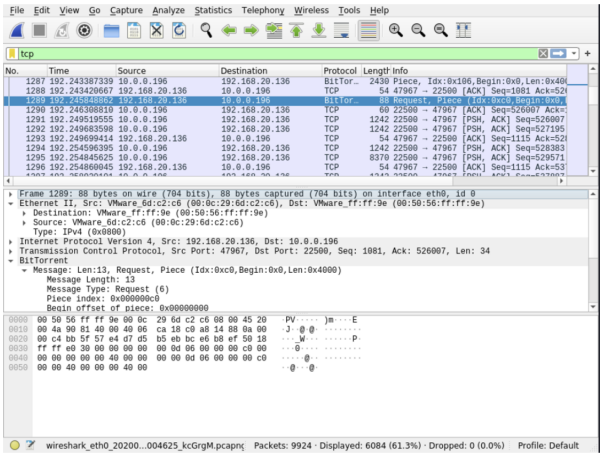

Fig. 5. Wireshark Capture

V. What was learned

Through this project, we have a deeper understanding of the principles of BitTorrent protocol and how files are transferred through P2P between client and server. By interacting with the Kali and Ubuntu with Raspberry Pi, we also familiarized ourselves with different Linux workspaces and corresponding commands to perform different tasks. By setting up RPI in the first place, we learned network setting and ssh-server and etc. By using Docker, we learned how to package an application(E.g., qBittorrent in our case) and its dependencies in a virtual container that can run on any Linux servers. Besides developing build and runtime Docker containers for client and server, we also gained experience on cross-compile when we tried to generate the executable Aarch64 binary file. What's more, we also achieved a large amount of practical experiences when exploring qBittorrent applications. We met the problem of NAT interfering with P2P applications and finally solved it by setting UPnP and manually setting Port forwarding in our BitTorrent application and the home router. We truly had a much deeper understanding of NAT. Also, we learned how to create our own torrent files, and had a better understanding of the important roles tracker and DHT play in the successful communication between client and server. Finally, we got our hands dirty by using Wireshark to capture the traffic between the client and the server.

VI. Difficulties

A. Cross-compile Linking Issues

As mentioned above, we first tried to use cross-compile to generate the Aarch64 executable binary file that can run on Raspberry Pi in the Kali VM. However, one of the libraries that the qBittorrent application depends on, libtorrent-rasterbar-dev needs to be more updated versions (1.1.0 and above), while the default version of this package when downloading with "apt-get install" is 1.0.6. Thus, we had to manually install this package. This introduced linking issues with the qBittorrent application during the cross-compilation, since the cross-compiler might not recognize the path where the dependent library locates. Though possible paths have been added to the compiler, the relocation problem still existed in the linking stage. Therefore, we choose to directly build the environment on the Raspberry Pi instead. The bash file (cross-compile.sh) for the cross-compile is still posted on the Github for future reference.

B. qBittorrent-nox Limitations

Since we don't have the Graphical User Interface on the Ubuntu server, we could only perform limited applications with the without-GUI version of qBittorrent, qBittorrent-nox. For example, we need to make a torrent of a local file, set the UPnP port forwarding rule in the application, and start uploading the file, and all of these have to be done in a terminal, which is actually impossible as qBittorrent-nox (even with a Web-UI enabled) does not support the function of creating torrent. Therefore, it is beyond our capabilities. Nevertheless, we recommend that you download qBittorrent with GUI to implement those functions, such as making torrent and set UPnP port forwarding. Currently, we successfully enabled the RPI to load torrent into its qBittorrent-nox application and download files from other peers.

VII. References

References

[1] "Peer-to-peer", En.wikipedia.org, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Peer-to-peer. [Accessed: 27- Mar- 2020].

[2] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 7th ed. Boston, MA: Pearson, 2016.

[3] Stallings, W. (2017). Network security essentials: applications and standards. Harlow, England: Pearson.

[4] DockerCon LIVE with theCUBE: Call for Papers is Open ... (n.d.). Retrieved from https://www.docker.com/blog/dockercon-live-with-thecube-call-for-papers-is-open/

[5] Leroux, S., & Leroux Engineer, S. (2019, May 9). The Kali Linux Review You Must Read Before You Start Using it. Retrieved from https://itsfoss.com/kali-linux-review/

[6] ARM Cross-Compilation: Kali Linux Documentation. (n.d.). Retrieved from https://www.kali.org/docs/development/arm-cross-compilation-environment/

[7] Kenlon, S. (2019, July 10). 32-bit life support: Cross-compiling with GCC. Retrieved from https://opensource.com/article/19/7/cross-compiling-gcc

[8] Preshing, J. (2014, November 19). How to Build a GCC Cross-Compiler. Retrieved from https://preshing.com/20141119/how-to-build-a-gcc-cross-compiler/

[9] Janetakis, N. (2018, December 25). Docker Tip 73: Connecting to a Remote Docker Daemon. Retrieved from https://nickjanetakis.com/blog/docker-tip-73-connecting-to-a-remote-docker-daemon

[10] Chaturvedi, V. (2019, May 22). What Is Docker & Docker Container: A Deep Dive Into Docker. Retrieved from https://www.edureka.co/blog/what-is-docker-container

[11] Cope, J. (2002, April 8). What's a Peer-to-Peer (P2P) Network? Retrieved from https://www.computerworld.com/article/2588287/networking-peer-to-peer-network.html

[12] Baydan, İ., & Kimble, D. (2017, August 9). How to install qbittorrent . Retrieved from https://www.poftut.com/install-use-qbittorrent-tutorial/