CSED353: Computer Network Term Project

권민재, 20190084

July 3, 2020

Contents

1 개요

본 프로젝트는 single board computer와 여러 소프트웨어를 활용하여 unique WiFi AP를 제작하는 프로젝트이다. 이 프로젝트에서는 single board computer로 Raspberry Pi3 B 모델을 사용하였으며, OpenWrt를 운영체제로 사용하였다. 이 WiFi AP는 AP의 서브넷과의 동일한 환경을 제공하기 위한 VPN을 제공하며, caching을 지원하는 proxy server의 역할도 수행한다. 그리고 더 나아가, 포스텍 내 HTTP 사이트에 대한 캐싱도 이 AP가 수행할 수 있도록 기능을 추가하였다. 이러한 기능을 추가하기 위해 OpenVPN, Squid, Nginx 등의 소프트웨어가 사용되었으며, 이들의 효율적인 관리를 위해 luci 또한 이용하였다.

2 구조

2.1 OpenWrt

OpenWrt는 wireless router를 위한 리눅스 기반의 오픈 소스 운영체제로, 이 프로젝트에서 Raspberry Pi의 운영체제로 사용했다. 많은 리눅스 배포판들 중에서 이 운영체제를 선택한 이유로는, 우선 이 운영체제는 '라우터'를 위한 운영체제라는 점을 꼽을 수 있다. Raspberry Pi와 같은 임베디드 보드에서 라우터 기능을 수행하기에는 성능적인 제약이 많기 때문에, 꼭 필요한 기능만 설치되어 있고, 가벼운 운영체제가 적합하기 때문이다. 게다가, OpenWRT로 라우터를 만드는 사람들이 많이 존재하기 때문에, 인터넷을 통해 OpenWrt를 이용하여 AP를 제작하는데 있어서 많은 정보를 쉽게 접할 수 있다는 장점 또한 존재한다. 이러한 장점이 있기 때문에 이번 프로젝트의 운영체제로 OpenWRT를 선택하였다.

2.2 OpenVPN

OpenVPN은 VPN을 제공하는 프로토콜과 프로그램을 일컫는다. VPN을 naive하게 직접 구현하는 것은 상당한 시간이 걸리고, 보안상 믿을 수 없는 시스템이 만들어질 수 있기 때문에, 대중화된 프로그램을 사용하는 것으로 결정하였다. 여러 대중화된 프로그램들 중에서, OpenVPN이 OpenSSL을 이용하여 통신을 암호화하기 때문에 안전하고, 많은 플랫폼에서 사용할 수 있다는 장점이 있어서

OpenVPN을 이용하여 VPN Server를 WiFi AP에 탑재하는 것으로 결정했다.

2.3 Squid

Squid는 대표적인 proxy 프로그램으로, caching 기능을 지원하여 웹 서버의 반복적인 request를 막고, 트래픽을 줄여줄 수 있도록 도와주는 프로그램이다. Squid를 이용한 proxy를 WiFi AP에 탑재함으로써 프록시에 연결함으로써 트래픽 부하를 줄이고, 캐시를 이용하여 더 빠르게 인터넷을 할 수 있도록 하기 위해 해당 기능을 WiFi AP에 추가하기로 결정하였다.

2.4 Nginx

Nginx는 최근 많이 사용되는 웹 서버 소프트웨어로, 가벼우면서 높은 성능에 강점을 가진다. Nginx은 웹 서버 프로그램으로서 여러 기능을 수행하며, Squid와 같이 리버스 프록시나 캐싱과 같은 역할도 할 수 있다. 그래서 이 점을 이용하여 Nginx를 WiFi AP에 탑재해서 WiFi AP의 클라이언트들에게 *.postech.ac.kr에 대한 웹 캐시를 제공하고자 한다.

3 기능

모든 파일의 내용은 설정 파일들 중 중요하다고 생각되는 일부 부분들만 발췌한 것이다.

3.1 Wifi Routed AP

우선 lan과 wifi, 두 개의 인터페이스를 만들고, wifi에게 서브넷을 부여하였다. 그 후, wifi의 패킷을 lan으로 포워딩시키고, lan에서 MASQUERADE를 통해 NAT를 구현함으로써 WiFi AP의 기능을 수행할 수 있도록 만들었다.

3.1.1 Interface

lan

```
config interface 'lan'
        option type 'bridge'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '141.223.175.220'
        option netmask '225.225.225.0'
        option gateway '141.223.175.99'
        list dns '8.8.8.8'
```

lan 인터페이스는 Raspberry Pi에 유선으로 연결된 인터넷에 연결되는 인터페이스로서, 외부 네트워크와 연결되는 역할을 한다. 학교 내에서 사용할 예정이기 때문에, 임시로 학교 내의 static ip를 할당하였다.

wifi

```
config interface 'wifi'
        option proto 'static'
        option ipaddr '192.168.2.1'
        option netmask '255.255.255.0'
```

wifi 인터페이스는 Raspberry Pi가 WiFi 무선 신호를 송수신하는 인터페이스를 말한다. 이 WiFi는 192.168.2.0/24의 내부 망 대역(서브넷)을 사용할 것이며, AP가 WiFi의 게이트웨이로서 192.168.2.1을 할당받을 수 있도록 설정하였다.

3.1.2 dhcp

```
config dhcp 'wan'
        option interface 'wan'
        option ignore '1'

config dhcp 'wifi'
        option interface 'wifi'
        option start '100'
        option limit '150'
        option leasetime '12h'
```

lan 인터페이스의 경우에는 static ip를 사용하고 있기 때문에, DHCP를 ignore 하도록 설정하였다. 반면에, wifi의 경우, routed WiFi AP라는 역할을 수행하기 위해서는 DHCP에 기반한 NAT가 필수적이기 때문에 위와 같이 DHCP 서버를 설정하였다.

3.1.3 firewall

```
config zone
        option network 'lan'
        option input 'ACCEPT'
        option forward 'REJECT'
        option name 'lan'
        option output 'ACCEPT'
        option masq '1'
        option mtu_fix '1'

config zone
        option network 'wifi'
        option input 'ACCEPT'
        option name 'wifi'
        option output 'ACCEPT'
        option forward 'ACCEPT'

config forwarding
        option dest 'lan'
        option src 'wifi'
```

기본적으로, wifi의 패킷을 lan으로 포워딩하도록 설정하여 wifi를 통해 무선 인터넷을 사용할 수 있도록 구축하였다. 그 후, lan에서는 wifi의 패킷들을 변환해서 보내는 NAT를 구현해야하기 때문에, option masq '1'을 통해 MASQUERADE를 활성화하여 NAT를 처리할 수 있도록 만들었다.

3.1.4 wireless

```
config wifi-iface 'default_radio0'
        option device 'radio0'
        option network 'wifi'
        option mode 'ap'
        option ssid 'beta'
        option key 'CENSORED'
        option encryption 'psk2'
```

wireless는 클라이언트가 어떻게 접속할지, 쉽게 말해서 클라이언트의 눈에 보이는 부분을 설정하는 곳이다. wireless에서 SSID와 암호화 방식을 설정해주었으며, 보안성을 갖추기 위해서 psk2 방식을 선택하였다. 'CENSORED'는 실제로 사용하고 있는 암호가 아닌, 임의의 암호이다.

3.2 VPN

VPN 시장에서 널리 사용되고 있는 OpenVPN을 활용하여 WiFi AP가 VPN Server의 역할 또한 수행할 수 있도록 만들었다.

3.2.1 openvpn

```
config openvpn openvpn_server

        option enabled 1
        option port 1194

        option proto tcp

        option dev tun0

        option ca /etc/openvpn/ca.crt
        option cert /etc/openvpn/server.crt
        option key /etc/openvpn/server.key
        option dh /etc/openvpn/dh.pem

        option server "10.8.0.0 255.255.255.0"

         list push "redirect-gateway def1"

        list push "dhcp-option DNS 10.8.0.1"

        option duplicate_cn 1

        option compress lzo
```

openvpn에서 주로 쓰는 포트인 1194번 포트에서 tcp 방식으로 vpn이 작동할 수 있도록 설정하였다. 각종 인증서와 키는 easyrsa를 통해 generate 했으며, 10.8.0.0/24 서브넷에서 vpn client들이 각자의 ip 주소를 할당받도록 설정하였다. 이를 위해 dhcp-option을 통해 dhcp가 작동되도록 하였고, 클라이언트들의

모든 트래픽을 받기 위해 "redirect-gateway def1" 옵션을 추가하였다. 테스트의 편의를 위해 같은 클라이언트 인증서로 여러 기기가 접속하는 것을 허용하기 위해 duplicate_cn을 활성화시켰으며, lzo를 통해 압축된 패킷을 주고받도록 하게 만들었다.

3.2.2 firewall.user

```
iptables -I INPUT 1 -p udp --dport 1194 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 1194 -j ACCEPT
iptables -I FORWARD 1 --source 10.8.0.0/24 -j ACCEPT
iptables -I FORWARD -i eth0 -o tun0 -j ACCEPT
iptables -I FORWARD -i tun0 -o eth0 -j ACCEPT
iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE
```

openvpn으로 통신되는 데이터들을 포워딩하기 위한 규칙을 설정하였다. 10.8.0.0/24 서브넷에서 위에서 정의한 tun0과 eth0(lan) 사이에 패킷이 오고 갈 수 있도록 설정하였고, 이때 NAT와 같은 기술이 필요하기 때문에 MASQUERADE도 활성화하였다.

3.3 Proxy with Caching

Squid를 이용하여 WiFi AP에 프록시 기능을 추가하였으며, 해당 프록시에서 Web caching을 할 수 있도록 설정하였다.

3.3.1 squid

```
debug_options ALL,1 33,2

acl localnet src ...
acl ssl_ports port 443
acl safe_ports port ...

http_access deny !safe_ports

http_access allow all

refresh_pattern ...

http_port 3128
```

```
cache_dir aufs /tmp/squid/cache 4096 16 512

cache_mem 8 MB
maximum_object_size_in_memory 100 KB
maximum_object_size 32 MB
```

우선 acl을 통해 통신을 허용할 포트와 로컬 서브넷을 지정해주었다. 이를 기반으로 특정 포트에서 프록시 접속을 차단하고, cache manager는 로컬 서브넷에서만 접속할 수 있도록 설정하여 기본적인 보안을 챙겼다. refresh_pattern을 통해 caching을 진행할 데이터의 형태를 지정하였으며, youtube도 캐싱이 가능하도록 설정하였다. /tmp/squid/cache에 총 4096MB의 캐시 메모리를 할당하였으며, 위의 규칙과 같이 캐시가 저장되도록 설정하였다.

3.4 Caching

dnsmasq.conf를 통해 *.postech.ac.kr로 가는 패킷을 192.168.2.1, 즉 WiFi AP로 우회시킨 다음에 nginx에서 캐시가 있다면 캐시를 돌려주고, 아닐 경우에 nginx에서 proxy path를 수행하게 하여 *.postech.ac.kr의 HTTP 페이지에 대한 웹 캐시를 구현하였다.

3.4.1 nginx.conf

```
user nobody nogroup;
worker_processes 16;
pid /var/run/nginx.pid;
worker_rlimit_nofile 300000;

events {
    worker_connections 8192;
    multi_accept on;
    use epoll;
}

http {
    client_body_buffer_size     32k;
    client_header_buffer_size   8k;
    large_client_header_buffers 8 64k;
    include mime.types;
```

```
access_log /var/log/nginx/access.log;
error_log /var/log/nginx/error.log debug;

sendfile on;
tcp_nopush on;
tcp_nodelay on;
types_hash_max_size 2048;
keepalive_timeout 65;

resolver 141.223.1.2;
resolver_timeout 30s;

limit_rate_after 10m;
limit_rate 2048k;

proxy_cache_path  /nginx-cache/CS levels=1:2 keys_zone=CS:10m
                      inactive=72h max_size=1g;

proxy_cache_path  /nginx-cache/POSTECH levels=2:2 keys_zone=POSTECH:100m
                      inactive=72h max_size=1g;

proxy_cache_key "$scheme$host$request_uri$cookie_user";

server {
    listen 80;
    server_name *.postech.ac.kr;
    access_log /var/log/nginx/postech-access.log;
    error_log /var/log/nginx/postech-error.log;

    location / {
        proxy_next_upstream error timeout http_404;
        proxy_pass http://$host$uri;
        proxy_redirect off;

        # proxy_set_header Host $host;
        # proxy_set_header X-Real-IP $remote_addr;
        # proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        proxy_cache POSTECH;
        proxy_cache_valid 200 301 302 10m;
        proxy_cache_valid any 1m;
        proxy_cache_use_stale   error timeout invalid_header updating
                               http_500 http_502 http_503 http_504;

        add_header Host $host;
```

```
            # add_header X-Mirror-Upstream-Status $upstream_status;
            # add_header X-Mirror-Upstream-Response-Time $upstream_response_time;
            # add_header X-Mirror-Status $upstream_cache_status;
        }
    }
}
```

기본 resolver는 포스텍의 DNS로 설정하였으며, /nginx-cache/POSTECH/에 *.postech.ac.kr 페이지들의 캐시들이 저장되도록 설정하였다. WiFi AP의 80포트로 *.postech.ac.kr에 대한 요청이 들어왔을 경우, proxy_cache_valid 한 경우에 캐시를 클라이언트에게 돌려주고, 아닌 경우에는 원래 경로로 proxy_pass를 수행하여 클라이언트의 요청을 처리한다. 원래대로라면 웹 규약에 맞게 X-Forwarded-For 헤더를 추가하는 것이 적절하겠지만, 400 Bad Request (Request Header or Cookie too Large) 에러가 계속적으로 발생하는 원인으로 여겨져서 활성화시키지 않았다.

3.4.2 dnsmasq.conf

```
log-queries
log-facility=/etc/dnsmasq.log
address=/.postech.ac.kr/192.168.2.1
```

*.postech.ac.kr에 대해 WiFi router를 가리키도록 dns를 수정하여 *.postech.ac.kr를 향하는 패킷들이 라우터의 nginx로 들어오도록 만들었다. 다만, 현재 세팅으로는 무분별하게 패킷이 들어오기 때문에, 패킷이 WiFi AP 안에서 순환하다가 500 Internal Server Error를 일으킬 수 있으므로, 해당 문제가 발생하지 않도록 설정을 고도화할 필요가 있다.

4 토론 및 개선

- Squid를 이용하여, 프록시를 구현할 때 웹 캐시도 같이 구현했지만, 그 용량이 실 사용량에 비해 너무 작아서 효율이 떨어질 것으로 생각된다. 캐시를 저장할 외장 저장장치를 별도로 장착하여 해결할 수 있을 것이다.

- 프록시를 이용하여 WiFi AP에 접속한 후 *.postech.ac.kr 사이트를 이용한다면, Squid와 Nginx에서 같은 내용의 캐시를 동시에 쌓기 때문에 퍼포먼스에 손해가 있을 것으로 생각된다. nginx로 글로벌한 캐시를 마련하고, 프록시 환경에서의 웹 캐시를 삭제하는 것이 오히려 효율적일 것으로 예측된다.
- SSL 인증서와 관련하여 문제가 있어서 https 연결에 대해 nginx 캐시를 구현하지 못한 것이 아쉽다.

5 참고문헌

- OpenWrt Documentation (https://openwrt.org/docs/start)