Control 1
Redes (2023-1)
Pauta

José M. Piquer

1. P1: sockets

En los códigos de ejemplos de proxies que están en el material del curso, hay uno: *proxy-threads.py* que contiene un proxy simple, con dos threads (una para cada dirección del flujo) y un proceso pesado por cliente, de modo de soportar múltiples clientes simultáneos. Les pedimos ahora modificar ese código para obtener un proxy UDP, es decir que ambos sockets sean UDP en vez de TCP.

Por supuesto, no es tan simple como cambiar todas las partes en que dice tcp por udp.

Entregue el código de un proxy UDP que funcione en forma equivalente al del ejemplo. Si hiceron la T1, pueden usarla para probar el proxy, poniéndolo entre su cliente y el servidor de anakena. Expliquen todas las modificaciones que hiceron (en el mismo código), tanto el por qué como la idea de su solución.

HINT: el problema de pasar de TCP a UDP es que en UDP no hay conexiones, sólo paquetes. Pueden usar el truco de REUSEPORT que aparece en otros servidores UDP del curso.

```
#!/usr/bin/python3
# proxy UDP
# Usando procesos para multi-clientes y threads dentros de cada proxy
import os, signal
import sys
import socket, jsockets
import threading
import struct

def childdeath(signum, frame):
```

```
    os.waitpid(-1, os.WNOHANG)

def copy_sock(data, conn1, conn2):
    if data:
        conn2.send(data)

    while True:
        try:
            data = conn1.recv(20000)
        except:
            data = None
        if not data: break
        conn2.send(data)
    conn2.close()

# Este el servidor de un socket ya conectado
# y el cliente del verdadero servidor (host, portout)
def proxy(addr, data, conn, host, portout):

    # Ahora este socket sólo recibirá paquetes de este único cliente
    conn.connect(addr) # ahí lo "conecto"
    # timeout de 40s, para que muera sin tráfico
    conn.setsockopt(socket.SOL_SOCKET, socket.SO_RCVTIMEO,
                    struct.pack("LL", 40, 0))

    conn2 = jsockets.socket_udp_connect(host, portout) # ahí lo "conecto"
    if conn2 is None:
        print('conexión rechazada por '+host+', '+portout)
        sys.exit(1)

    print('Cliente conectado')

# copy_sock() corre en ambos threads, uno de conn->conn2 y otro de conn2->conn
    newthread1 = threading.Thread(target=copy_sock, daemon=True,
                          args=(data, conn,conn2)) # este recibe primer paquete
    # el flag daemon es para que muera si muere el otro
    newthread1.start()
    copy_sock([], conn2, conn) # este no recibe primer paquete
    print('Cliente desconectado')
```

```
# Main
if len(sys.argv) != 4:
    print('Use: '+sys.argv[0]+' port-in host port-out')
    sys.exit(1)

portin = sys.argv[1]
host = sys.argv[2]
portout = sys.argv[3]

signal.signal(signal.SIGCHLD, childdeath)

s = jsockets.socket_udp_bind(portin)
if s is None:
    print('bind falló')
    sys.exit(1)

while True:
    # Esta es la magia del REUSEPORT: espero un primer paquete
    data, addr = s.recvfrom(20000) # primer paquete ahora debo guardarlo
    if not data:
        continue
    print('recibi cliente')
    # Para este cliente, voy a crear otro socket en el mismo port
    conn = jsockets.socket_udp_bind(portin)
    if conn is None:
        print('could not open 2nd socket')
        sys.exit(1)
    # Voy a crear un thread para que se conecte con ese cliente en el nuevo
    # socket!

    pid = os.fork()
    if pid == 0: # Este es el hijo
        s.close() # Cierro el socket que no voy a usar
        proxy(addr, data, conn, host, portout) # va con el primer paquete
        sys.exit(0)
    else:
        conn.close() # Cierro el socket que no voy a usar
```

2. P2: Aplicaciones

En el protocolo NTP, existe un problema difícil: el cliente pregunta la hora del servidor, quién responde con la hora actual. Pero, hay un retraso en la red entre que un paquete es emitido y luego es recibido. Ese tiempo se conoce como *delay*. Un problema en Internet es que ese *delay* no lo conozco a priori, e incluso a veces no es simétrico: es decir el *delay* desde el cliente al servidor no siempre es igual al *delay* desde el servidor al cliente. Al recibir un paquete con la hora del servidor, debo sumarle ese em delay al ajustar mi reloj local. El paquete NTP incluye muchos tiempos, justamente para arreglar ese problema. Revise los campos y sus significados y proponga un algoritmo que pueda usar un cliente que pregunta la hora a un solo servidor para estimar ese *delay* aprovechando estos campos. Discuta la precisión que tendría este ajuste y si se podría mejorar.

HINT: Hay una recomendación en Internet de cómo hacerlo, pero no están obligados a seguirla ni a descubrirla, pueden sugerir sus propias ideas, defendiendo bien su solución.

El paquete NTP tiene los siguientes campos útiles para esto:

- *Reference Timestamp: sirve para conocer la hora a la que ajustamos el reloj local la última vez*
- *Origin Timestamp: hora de envío de la pregunta*
- *Receive Timestamp: hora a la que el servidor recibió la pregunta*
- *Transmit Timestamp: hora a la que el servidor respondió la pregunta*

La idea es anotar la hora de envío en el paquete desde el servidor, de modo que el cliente pueda saber cuánto demora en llegar, al compararlo con la hora de recepción. Sin embargo, el problema es que justamente estamos comparando tiempos entre relojes que pueden estar derivando y ¡queremos ponerlos al día! Por eso, simplemente mirar la diferencia entre la hora de envío del servidor y la hora de recepción del cliente no es una buena idea. Puedo entonces usar la hora de envío de la pregunta (está con el mismo reloj que recibe la respuesta) y suponer que el delay *es simétrico, y aproximar a la mitad del tiempo entre preguntar y recibir la respuesta. Después de eso, actualizo mi reloj a la hora del servidor más el* delay *calculado. La limitación de este esquema es que no funcionaría bien si la conexión no es simétrica en* delays.

3. P3: DNS

Por muchos años, China ha bloqueado el acceso de sus redes a sitios externos que consideran 'subversivos', incluidos facebook, Twitter, YouTube, etc. Para ello, usa varias estrategias, y una de las principales de modificando las respuestas del DNS. Esta técnica se conoce como *DNS hijacking* o *DNS Poisoning* (también se usa para ciertos ataques de seguridad). Averigüe cómo funciona y cómo se implementa y explíquelo. Luego, busque cómo este mecanismo (al parecer sin querer) generó errores en Chile, donde durante unos días algunos usuarios no podían navegar en facebook y youtube debido a ese error. Explique lo que ocurrió.

Finalmente, discuta la eficacia de este método: ¿qué tan fácil es, para un usuario en China, accesar facebook a pesar de esto? ¿Cómo podría hacerlo? ¿Nosotros podríamos ayudarlo desde Chile?

La idea es responder la petición de traducción de nombres de dominio a direcciones IP con IPs falsas. Para eso, debemos tener servidores locales (resolvers) que respondan en vez de los servidores oficiales (es decir que se hagan pasar por primarios de los dominios objetivo) o simplemente alterar las respuestas cuando pasen por un firewall 'estatal'. Esto permite que, incluso si uso resolvers fuera de China (como 8.8.8.8), las respuestas igual llegarán falladas y bloquearé el acceso a esos dominios.

El error en Chile se debió a que algunas redes chilenas comenzaron a usar un servidor raíz en China como el principal. Ese servidor raíz era 'oficial' del mundo, por lo que China no debía alterar sus respuestas. Probablemente debido a un error de configuración (no es trivial definir qué paquetes debo alterar y cuáles no, significa saber si la IP de origen está en China o no y si pertenece a alguna organización a la que no hay que bloquear) las respuestas de este servidor raíz comenzaron a ser alteradas y los usuarios en Chile se vieron desconectados de estos servicios por varias horas.

Una alternativa sería no usar nombres de dominio y accesar las direcciones IP directamente. Esto no resultaría muy útil, por que estos nombres están dentro de la aplicación misma, y en links HTTP en sus propias páginas. Otra forma sería tener un resolver propio que conociera y respondiera por estos dominios, pero son muy dinámicos y no es fácil mantener un sistema así al día. La mejor forma es usar VPNs, que permiten enviar tráfico encriptado fuera de China, pero estas son ilegales allá.

Nosotros en Chile, podríamos instalar una especie de proxy DNS, que camufle el tráfico para que no parezca DNS, y toda pregunta que recibamos la hacemos nosotros en Internet, y sólo transferimos la respuesta. No debemos usar el puerto oficial de DNS ni el formato normal de los paquetes para que

no nos detecte el firewall.