Trabajo Práctico

Programación de Sockets de Internet con Python

Arellano E. Nahuel
nahuel.arellano@gmail.com

1. Utilizando el módulo socket escriba un programa servidor que implemente el protocolo echo con SOCK_STREAM y otro con SOCK_DGRAM. También implemente los respectivos clientes para consultar dicho servicio. Realice pruebas con cliente y servidor en su sistema local (localhost) y luego en uno remoto. En todos los casos realice la captura y explique el resultado.
Modifique el programa anterior para soportar múltiples clientes simultáneos. ¿El comportamiento del servidor basado en SOCK_DGRAM es igual cuando se pasa a SOCK_STREAM?

Se ejecuta los scripts server_udp_echo.py y client_udp_echo.py para realizar pruebas sobre udp utilizando la siguiente sintaxis:

```
server_udp_echo.py [-h] [--host HOST] [--port PORT]
client_udp_echo.py [-h] --data DATA [--host HOST] [--port PORT]
```

Se ejecuta los scripts server_tcp_echo.py y client_tcp_echo.py para realizar pruebas sobre tcp utilizando la siguiente sintaxis:

```
server_tcp_echo.py [-h] [--host HOST] [--port PORT]
client_tcp_echo.py [-h] --data DATA [--host HOST] [--port PORT]
```

ej.:

```
python server_udp_echo.py
python client_udp_echo.py --data "hola mundo"

python server_tcp_echo.py
python client_tcp_echo.py --data "hola mundo"
```

Analizando las capturas del echo se puede contemplar las diferencias entre los protocolos. En TCP se puede visualizar el Three-way handshake (negociación en tres pasos) y las confirmaciones de las tramas enviadas, mientras que en UDP solo se envia las tramas con mensaje sin confirmación. En ambos casos el mensaje viaja en texto plano.

2. Implemente el protocolo DayTime (http://www.ietf.org/rfc/rfc867.txt), cliente y servidor, con UDP. Escriba un protocolo de aplicación propio que le permita especificar la fecha y hora de un determinado huso horario y en función de eso responda. El servidor recibe como parámetro la configuración regional del lugar donde opera y debe sincronizarse con algún servidor de hora.
Agregue al inicio de su programa la especificación de su protocolo (la estructura de PDU debe ser de longitud fija).

Se ejecuta los scripts server_ntp.py y cliente_ntp.py utilizando la siguiente sintaxis:

```
server_ntp.py [-h] [--host HOST] [--port PORT]
cliente_ntp.py [-h] [--format FORMAT] [--host HOST] [--port PORT]

ej.:
python server_ntp.py
python cliente_ntp.py  --format %c
```

3. Escriba un programa que implemente un cliente http utilizando el módulo socket (similar a wget). Agregue opciones para que el programa use un proxy y guarde en un archivo de log todos los header HTTP recibidos. Además, considere evaluar el header de redirección.

Se ejecuta el script ejer3.py utilizando la siguiente sintaxis:

```
ejer3.py [-h] --url URL [--proxy PROXY] [--port PORT]

ej.:
python ejer3.py --url http://www.unlu.edu.ar/
```

4. Se requiere una aplicación para calcular el RTT en la red (consulte http://www.ietf.org/rfc/rfc2681.txt). La misma debe permitir el ingreso de una lista de direcciones IP y un intervalo de tiempo (el mismo para todas) contra las cuales medir el RTT. Evalue los resultados con SOCK_STREAM y con SOCK_DGRAM.

Se ejecuta el script ejer4.py utilizando la siguiente sintaxis:

```
ejer4.py [-h] [--segundos SEGUNDOS] [--udp] [--tcp]

ej.:

python ejer4.py --udp
python ejer4.py --tcp
```

Se desarrollo el programa eje4.py para calcular el RTT para los protocolos UDP y TCP.
Para su correcta ejecución deben cargarse los archivos IPsUDP.txt y IpsTCP.txt con la lista de ip separadas por “;” que se desea probar.

Opciones de ejecución:

- para udp: python ejer4.py --udp [--segundos]
- para tcp: python ejer4.py --tcp [--segundos]
- para ambos: python ejer4.py --udp --tcp [--segundos]

UDP

****
Estadisticas IP: 195.46.39.40
Paquetes: enviados=13, Recibido=12, Perdido=1
Promedio=239 ms
****
****
Estadisticas IP: 195.46.39.40
Paquetes: enviados=13, Recibido=12, Perdido=1
Promedio=239 ms
****
****
Estadisticas IP: 208.67.222.222
Paquetes: enviados=18, Recibido=17, Perdido=1
Promedio=169 ms
****
****
Estadisticas IP: 208.67.220.220
Paquetes: enviados=18, Recibido=18, Perdido=0
Promedio=169 ms
****

TCP

****
Estadisticas IP: www.google.com
Paquetes: enviados=50, Recibido=50, Perdido=0
Promedio=60 ms
****
****
Estadisticas IP: www.facebook.com
Paquetes: enviados=45, Recibido=45, Perdido=0
Promedio=66 ms
****
****
Estadisticas IP: www.unlu.edu.ar
Paquetes: enviados=59, Recibido=59, Perdido=0
Promedio=51 ms
****
****
Estadisticas IP: www.yahoo.com
Paquetes: enviados=11, Recibido=11, Perdido=0
Promedio=279 ms
****

Se puede observar que en promedio los tiempos de UDP son mayores que los de TCP, pero la mayor demora se produce contra el servidor web www.yahoo.com utilizando una conexión TCP, se puede deducir que el factor determinante en los retardos es la distancia.

5. Escriba un programa cliente y uno servidor que permitan ejecutar comandos Unix en una máquina remota (por ejemplo: pwd, ls, mv). Ejemplo:

#~$ python cliente.py
>>> pwd
>>> /home/usuario/Taller_II/

Sus programas debe tener una instancia de autenticación en la cual el nombre de usuario y la clave se encripten antes de enviarse por la red. ¿Cómo se puede resolver este problema? Revise cómo lo hace ssh. Incluya una opción para almacenar en un archivo indicado el log de toda la sesión.

Se ejecuta los scripts server_remote.py y client_remote.py utilizando la siguiente sintaxis:

```
server_remote.py [-h] [--host HOST] [--port PORT] [--log LOG]
client_remote.py [-h] [--host HOST] [--port PORT]

ej.:

python server_remote.py
python client_remote.py

user/pass: nahuel/1234
```

Se utilizo el algoritmo de cifrado AES para resolver la problemática de encriptacion de la autentificación que plantea el ejercicio.
Con AES se implemento una encriptación simétrica, donde los programas cliente y servidor acuerdan una clave pública para encriptar sus mensajes, en nuestro caso solo para la etapa de autentificación del sistema, el resto de los mensajes viajan en texto plano, a diferencia del protocolo SSH donde toda la comunicación entre el cliente y servidor es segura.

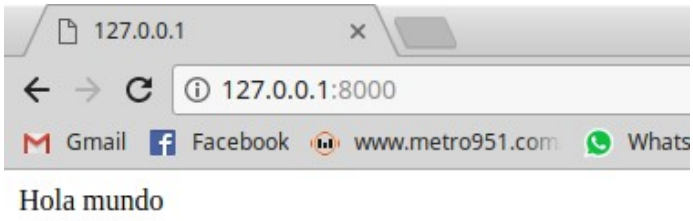
6. Desarrolle un servidor http básico que implemente las respuestas para los códigos HTTP de estado 200 y 404. El servidor retornará el código de estado correspondiente de acuerdo a si existe o no el objeto solicitado. Si el código es 404, debe retornar una página HTML pre-diseñada que contiene el mensaje explicando la situación. Realice pruebas con diferentes navegadores.

Se ejecuta el script ejer6.py utilizando la siguiente sintaxis:

```
ejer6.py [-h] [--host HOST] [--port PORT]

ej.:
python ejer6.py
```

Desde el explorador, estado 200:



Desde el explorador, estado 404:

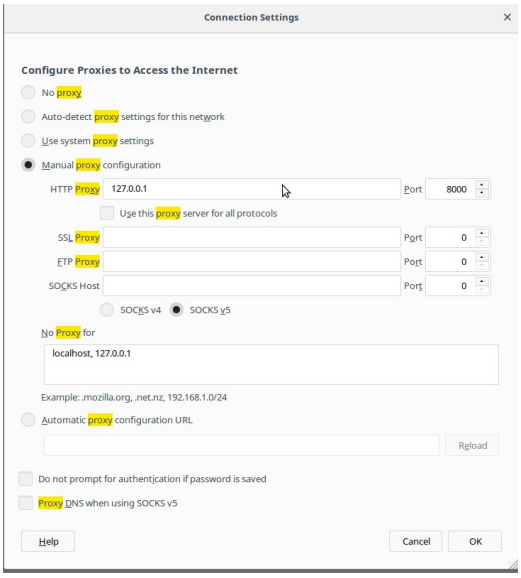
7. Desarrollo un servidor proxy http básico que permita además realizar cache de objetos de una página Web (imágenes, flash, applets, etc.).

Se ejecuta el script ejer7.py utilizando la siguiente sintaxis:

```
ejer7.py [-h] [--host HOST] [--port PORT]
```

```
ej.:
python ejer7.py
```

Se configura el proxy del explorador:



8. Escriba un programa servidor http que de igual forma a lo visto en el práctico anterior que permita realizar un balanceo de carga. Este load balancer debe interactuar con el servidor http del ejercicio 6.

Se ejecuta el script ejer8.py utilizando la siguiente sintaxis:

```
ejer8.py [-h] [--host HOST] [--port PORT]
```

En el archivo de configuración config.ini se indicara los servidores entre lo que se balanceará la carga.

```
ej.:

python ejer6.py --port 8001
python ejer6.py --port 8002
python ejer8.py


config.ini
[nodo1]
host = 127.0.0.1
port = 8001

[nodo2]
host = 127.0.0.1
port = 8002
```

9. Implemente un servicio de chat punto-a-punto. Su programación debe permitir vincular a 2 usuarios y que puedan "dialogar". ¿Qué requisitos tiene el servicio? ¿Cómo se pueden implementar?

Se ejecuta los scripts server_chat.py y client_chat.py utilizando la siguiente sintaxis:

```
server_chat.py [-h] [--host HOST] [--port PORT]
client_chat.py [-h] [--host HOST] [--port PORT]

ej.:
```

Se ejecuta el servidor de chat:
```
python server_chat.py
```

Se ejecuta dos clientes de chat:
```
python client_chat.py
python client_chat.py
```

Para el desarrollo del chat un requisito que se presento, fue la necesidad de mantener en el servidor una lista dińamica de socket, con la finalidad de distribuir los mensajes y poder aceptar nuevas conexiones de los clientes, la solución fue implementada utilizando la llamada de sistema *select* para recuperar las conexiones disponibles para luego realizar el tratamiento correspondiente.

10. Escribir un programa servidor que retorne la hora en UTC y un cliente que ajuste su hora utilizando el Algoritmo de Cristian.

Se ejecuta los scripts server_utc.py y client_utc.py utilizando la siguiente sintaxis:

```
server_utc.py [-h] [--host HOST] [--port PORT]
client_utc.py [-h] [--host HOST] [--port PORT]

ej.:
```

```
python server_utc.py
python client_utc.py
```

11. En este ejercicio el desafío es implementar fiabilidad, ordenamiento de PDUs y control de congestión sobre UDP. Escriba un cliente y servidor de archivos con las ideas explicadas en esta página http://gafferongames.com/networking-for-game-programmers/reliability-and-flow-control/ y luego haga una comparación con respecto a TCP.

Se ejecuta los scripts servidor_udp.py y cliente_udp.py utilizando la siguiente sintaxis:

```
python servidor_udp.py [-h] [--host HOST] [--port PORT]
python cliente_udp.py  [-h] [--host HOST] [--port PORT]

ej.:

python servidor_udp.py
python cliente_udp.py
```

La pdu del protocolo se compone de la siguiente manera:
secuencia - int 4byte
ack - int 4byte
ack_bitfield - int 4byte
data - string

La principal diferencia del protocolo desarrollado con respecto a TCP es que permite un flujo constante de envio de paquetes, lo que lo convierte en una solución muy buena para los juegos multiplayer en tiempo real.
Uno de los inconvenientes de TCP es que ante la perdida de un paquete se detiene y espera el envio del paquete perdido, esto genera un retardo significativo para sistemas que requieren de tiempo real.

12. Analice los requerimientos de un sistema de archivos distribuidos y escriba un programa que permita mantener sincronizado un directorio y todo su contenido replicado en varios equipos. Se deben contemplar los requerimientos de Transparencia, Replicación y Tolerancia a fallos. La funcionalidad del cliente debe ser provista a través de una terminal y debe poseer los comandos básicos de una terminal UNIX para listar, copiar, mover, etc.

Se ejecuta los scripts nodo_master.py y nodo_slave.py utilizando la siguiente sintaxis:
El HOST y PORT corresponden al socket donde va a escuchar el servidor.

```
python nodo_master.py    [-h] [--host HOST] [--port PORT]
python nodo_slave.py     [-h] [--host HOST] [--port PORT]
```

Se ejecuta el scripts client_remote.py utilizando la siguiente sintaxis:
El HOST y PORT corresponden al master.

```
python client_remote.py  [-h] [--host HOST] [--port PORT]
```

```
ej.:
python nodo_master.py

python nodo_slave.py --port 8001
python nodo_slave.py --port 8002

python client_remote.py --port 8000
```

El nodo master sincronizara la carpeta /SERVIDOR

La pdu del protocolo se compone de la siguiente manera:
id_nodo - int 4byte
accion - int 4byte
bytes_path - int 4byte
path - string
bytes - int 4byte
data - string

#NRO	ACCION	
0	ACCION_REGISTRAR	Esta acción permite al nodo master registrar en que ip y puerto brinda servicio los nodos slave. Tambien le asigna un número de identificación
1	ACCION_REMOTO	Permite resolver las peticiones del cliente remoto.
2	ACCION_COPIAR	Realiza la replicacion del directorio master al nodo slave
3	AGREGAR_ARCHIVO	Notifica ubicación y datos del archivo a agregar.
4	ELIMINAR_ARCHIVO	Notifica ubicación del archivo a eliminar
5	MODIFICAR_ARCHIVO	Notifica ubicación y datos del archivo a modificar.

13. Escriba un programa que implemente un anillo lógico como se muestra en la figura. Cada proceso en un host diferente tiene un ID (único) y cada uno puede transmitir cuando le llega el turno (secuencial).
Cuando se envía un mensaje se debe incluir a qué ID va dirigido y siempre se reenvía en anillo – por ejemplo – hacia el nodo de la izquierda. El mensaje deber circular hasta llegar nuevamente al nodo que lo originó quien dará lugar a que transmita el siguiente (similar al protocolo Token Ring). Cada nodo lee de un archivo la lista de mensajes que tiene que transmitir. Los programas finalizan su ejecución cuando el nodo con ID = 0 envía un comando de finalización. Usted debe definir el protocolo de aplicación, tanto la estructura de datos como el comportamiento ante cada situación. Trate de escribir su especificación en formato de RFC.

Se ejecuta el script nodo.py utilizando la siguiente sintaxis:

```
nodo.py [-h] [--id ID] [--id_dst ID_DST]
```

El nodo y el nodo destino deben estar configurados en el archivo `config.ini`

```
[nodoX]                    #Donde X es el Id del nodo
host = 127.0.0.1
port = 8000

ej.:
python nodo.py --id 0 --id_dst 1
python nodo.py --id 1 --id_dst 2
python nodo.py --id 2 --id_dst 3
python nodo.py --id 3 --id_dst 0
```

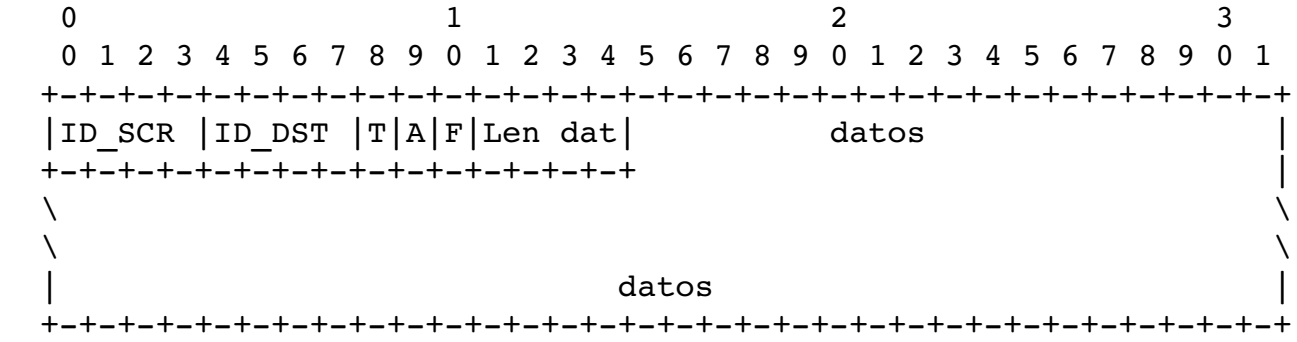
Para resolver el problema planteado en el ejercicio se definio el protocolo de aplicación Protocolo Anillo Lógico. Su especificación, en formato RFC, es la siguiente:

```
Protocolo Anillo Lógico

Introducción

El protocolo esta diseñado para el intercambio de mensajes dentro de
un anillo lógico.
Este protocolo utiliza como protocolo subyacente al protocolo de
control de transmisión (TCP).

Formato
```




```
Campos

ID_SCR: Id del nodo origen. Puede tomar valores del 0 al 4294967295
ID_DST: Id del nodo destino. Puede tomar valores del 0 al 4294967295
T: Flag de turno. Indica que el nodo puede o no enviar mensajes.
A: Flag de ACK. Indica que el nodo recibio el mensaje.
F: Flag de finalización. Indica que se termino la comunicación entre
los nodos.
Len dat: Longitud de datos del mensaje expresada en bytes.
```

14. Escriba un programa que implemente 3 tipos de nodos diferentes: Nodo master, Nodo de mapeo y Nodo de reducción. El nodo master, será el encargado de recibir un archivo de texto sin formato de al menos 5MB y deberá dividirlo a lo sumo en la mitad de nodos de mapeo que tiene la red y asignar una porción del archivo a cada nodo de mapeo. Una misma porción podrá ser asignada a más de un nodo de mapeo en simultáneo. Cada uno de estos nodos de mapeo deberá construir una estructura <termino: frecuencia> para esa porción de documento recibida. Al terminar la tarea el nodo anunciará la finalización de la tarea al nodo de reducción y si este se lo permite le enviará la estructura generada.
Por último, el nodo de reducción será el encargado de generar una única estructura con todos los índices recibidos.

Se realizo el desarrollo de los nodos master, mapper y reducer y se realizaron pruebas con el texto del Quijote de Cervantes.

Modo de Uso en consola:
Setear variables de configuracion en el archivo 'config.ini'

Ejecutar los mappers
Toma variable de conexion (host, port) por parametro, deben coincidir con las seteadas en el archivo de configuracion para poder establecer comunicacion con los otros nodos.

#nodos mappers
```
python nodo_mapper.py [-h] host port
```

Ejecutar reducer
Toma la configuracion desde archivo 'config.ini'.
Sin variables por parametros

#nodo reducer
```
python nodo_reducer.py
```

Ejecutar master
Toma la configuracion desde archivo 'config.ini'.
Debe ingresarse por parametro el archivo entrada a procesar y el archivo salida donde devolvera el resultado.

#nodo master
```
nodo_master.py [-h] in_file out_file
```

```
ej.:
python nodo_mapper.py 127.0.0.1 8001
python nodo_mapper.py 127.0.0.1 8002
python nodo_mapper.py 127.0.0.1 8003
python nodo_reducer.py
python nodo_master.py pg2000.txt salida.txt
```

15. Escribir un programa que permita mantener un índice distribuido de documentos. Cada nodo de la red debe ser capas de encontrar nodos conectados en el mismo segmento de red e intercambiar información acerca de los documentos que tiene disponibles. También se deberá poder emitir un listado que muestre para cada documento en qué nodo esta este disponible.

Se ejecuta los scripts nodo.py utilizando la siguiente sintaxis:

```
python nodo.py [-h] [--host HOST] [--port PORT] [--folder FOLDER]
```

El HOST y PORT corresponden al socket donde se va a brindar servicio y FOLDER a la carpeta que se quiere indexar.
Los puerto permitidos son el 8000, 8001, 8002, 8003, 8004 y 8005.
Para poder encontrar los nodos conectados en el mismo segmento de red se utilizo un socket SOCK_DGRAM configurado como SO_BROADCAST.

```
ej.:

python nodo.py --port 8000 --folder Folder/
python nodo.py --port 8001 --folder Folder2/
```