Introducción a los Sistemas Distribuidos (75.43)
TP N°1: File Transfer

Esteban Carisimo y Juan Ignacio Lopez Pecora

Facultad de Ingeniería, Universidad de Buenos Aires

12 de abril de 2023

Resumen

El presente trabajo práctico tiene como objetivo la creación de una aplicación de red. Para tal finalidad, será necesario comprender cómo se comunican los procesos a través de la red, y cuál es el modelo de servicio que la capa de transporte le ofrece a la capa de aplicación. Además, para poder lograr el objetivo planteado, se aprenderá el uso de la interfaz de sockets y los principios básicos de la transferencia de datos confiable (del inglés Reliable Data Transfer, RDT).

Palabras clave— Socket, protocolo, tcp, udp

1. Propuesta de trabajo

Este trabajo práctico se plantea como objetivo la comprensión y la puesta en práctica de los conceptos y herramientas necesarias para la implementación de un protocolo RDT. Para lograr este objetivo, se deberá desarrollar una aplicación de arquitectura cliente-servidor que implemente la funcionalidad de transferencia de archivos mediante las siguientes operaciones:

- **UPLOAD**: Transferencia de un archivo del cliente hacia el servidor
- **DOWNLOAD**: Transferencia de un archivo del servidor hacia el cliente

Dada las diferentes operaciones que pueden realizarse entre el cliente y el servidor, se requiere del diseño e implementación de un protocolo de aplicación básico que especifique los mensajes intercambiados entre los distintos procesos.

2. Herramientas a utilizar y procedimientos

La implementación de las aplicaciones solicitadas deben cumplir los siguientes requisitos:

- La aplicaciones deben ser desarrolladas en lenguaje Python [1] utilizando la librería estandard de sockets [2].
- La comunicación entre los procesos se debe implementar utilizando UDP como protocolo de capa de transporte.
- Para lograr una transferencia confiable al utilizar el protocolo UDP, se pide implementar una versión utilizando el protocolo *Stop & Wait* y otra versión utilizando el protocolo *Selective Repeat*.
- El servidor debe ser capaz de procesar de manera concurrente la transferencia de archivos con múltiples clientes.

2.1. Interfaz del cliente

La funcionalidad del cliente se divide en dos aplicaciónes de línea de comandos: `upload` y `download`. El comando `upload` envía un archivo al servidor para ser guardado con el nombre asignado. El listado 1 especifica la interfaz de linea de comandos para la operación de upload:

```
> python upload -h
usage: upload [-h] [-v | -q] [-H ADDR] [-p PORT] [-s FILEPATH] [-n FILENAME]

<command description>

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
  -q, --quiet    decrease output verbosity
  -H, --host     server IP address
  -p, --port     server port
  -s, --src      source file path
  -n, --name     file name
```

Listing 1: Interfaz de linea de comandos de upload

El comando `download` descarga un archivo especificado desde el servidor. El listado 2 especifica la interfaz de linea de comandos para la operación de download:

```
> python download -h
usage: download [-h] [-v | -q] [-H ADDR] [-p PORT] [-d FILEPATH] [-n FILENAME]

<command description>

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
  -q, --quiet    decrease output verbosity
  -H, --host     server IP address
  -p, --port     server port
  -d, --dst      destination file path
  -n, --name     file name
```

Listing 2: Interfaz de linea de comandos de download

2.2. Interfaz del servidor

El servidor provee el servicio de almacenamiento y descarga de archivos. El listado 3 especifica la interfaz de linea de comandos para el inicio del servidor:

```
> python start-server -h
usage: start-server [-h] [-v | -q] [-H ADDR] [-p PORT] [-s DIRPATH]

<command description>

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
  -q, --quiet    decrease output verbosity
  -H, --host     service IP address
  -p, --port     service port
  -s, --storage  storage dir path
```

Listing 3: Interfaz de linea de comandos del servidor

2.3. Mininet

Se debe tener en cuenta que para ejecutar dicha arquitectura Cliente-Servidor se utilizará una herramienta conocida como Mininet.

Mininet es un conocido simulador de redes, cuya aparición se debe a la necesidad de contar con un simulador capaz de operar con distintos switches, entre ellos OpenFlow. El ingreso revolucionario de las SDN y la posibilidad de correr simulaciones, llevo a un gran crecimiento de mininet, por lo cual hoy es muy sencillo encontrar gran cantidad de información disponible. Para poder familiarizarnos nos mininet y su uso, hay un tutorial en el GitHub[1] oficial del proyecto. **A su vez, la cátedra presenta un breve tutorial de mininet en SDN, inspirado en el tutorial oficial.**

Mininet se utiliza de una forma radicalmente diferente al resto de los simuladores de red, ya que los dispositivos incluidos y su interconexión se definen a través de un script en Python.

- Abrir una nueva terminal e iniciar mininet con una topología simple, con 3 hosts conectados a un switch.

```
$ sudo mn --topo single,3 --mac
```

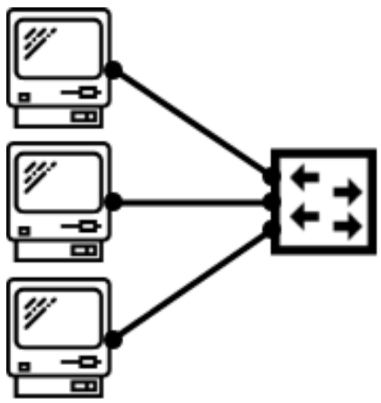

Figura 1: Topología Simple

- También se pueden crear topologías mas complejas, por ejemplo, se puede crear una topología propia denominada *FatTree* con el siguiente comando.

[1]Tutorial mininet con OpenFlow: `https://github.com/mininet/openflow-tutorial`

```
$ sudo mn --custom ~/mininet/custom/fattree.py --topo fattree --mac --arp --
   switch ovsk --controller remote
```

Para visualizar las topologías se puede usar la siguiente herramienta online

- `http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet`

Para poder validar que el protocolo desarrollado provee garantía de entrega es necesario forzar la pérdida de paquetes. Cuando se crea la topología es necesario indicar a los enlaces el porcentaje de pérdida de paquetes que queremos. Para esto utilizaremos un parámetro a la hora de ejecutar mininet.

3. Ejercicios

Los grupos deberán elaborar un código capaz de generar la topología indicadas sobre la cual se realizaran diferentes ensayos. El día de la entrega se deberá ejecutar la topología y la ejecucion del protocolo confiable en clase, mostrando su funcionamiento y dando evidencias de que se conoce el uso de las herramientas dadas.

Además, la ejecución del programa debe ser por consola y se especifique por parámetro la cantidad de hosts especificados y el porcentaje de pérdida de paquetes de cada enlace en la topología. Para ejecutar Mininet se utilizará el siguiente comando:

```
$ sudo mn --custom ~/mininet/custom/custom.py --topo custom --mac -x
```

Se pedirá entregar el código y se exige no distribuirlo. Facilitar la distribución de código, como así también el uso de código desarrollado por otros, van en contra de la conducta que la Facultad de Ingeniería pretende de los alumnos. Más aún, nuestro objetivo en la cátedra es fomentar el aprendizaje y el interés por temas actuales e innovadores, por lo cual pretendemos que el desarrollo del trabajo sea motivador y enriquecedor para los alumnos.

3.1. Topología

Se propone desarrollar una topología parametrizable. Se tendrá una cantidad de hosts variable, formando una arquitectura en donde haya un único servidor que se encuentra conectado a la cantidad de clientes que se recibió como parámetro.

Asimismo, cada enlace que conecta a un cliente con el servidor, deberá tener la pérdida de paquetes que se introdujo como parametro en el comando de ejecución.

A continuación se muestra una clase con una topología que cuenta con un servidor conectado a tres hosts con una perdida de paquetes del 10 %.

```
from mininet.topo import Topo
from mininet.link import TCLink

class Topo( Topo ):
    def __init__( self ):
        # Initialize topology
        Topo.__init__( self )

        # Create hosts
        h1 = self.addHost('host_1'))
        h2 = self.addHost('host_2'))
        h3 = self.addHost('host_3'))
        h4 = self.addHost('host_4'))

        # Add links between server and hosts self.addLink(s1, s2)
        self.addLink(h1, h2, cls=TCLink, loss=10)
        self.addLink(h1, h3, cls=TCLink, loss=10)
        self.addLink(h1, h4, cls=TCLink, loss=10)

topos = { 'customTopo': Topo }
```

3.2. Protocolo de Entrega Confiable

Una vez verificado el correcto armado de la red, se utilizará dicha arquitectura para ejecutar distintas pruebas entre el o los clientes y el servidor. Para ello, se correará el código con el protocolo confiabale en dichos hosts y se transferirán distintos archivos de tamaño variable.

Una vez concluida la transferencia (subida y/o descarga) de un archivo, se verificará que el archivo no haya sido modificado.

4. Análisis

Comparar la performance de la versión Selective Repeat del protocolo y la versión Stop&Wait utilizando archivos de distintos tamaños y bajo distintas configuraciones de pérdida de paquetes.

5. Preguntas a responder

1. Describa la arquitectura Cliente-Servidor.
2. ¿Cuál es la función de un protocolo de capa de aplicación?
3. Detalle el protocolo de aplicación desarrollado en este trabajo.
4. La capa de transporte del stack TCP/IP ofrece dos protocolos: TCP y UDP. ¿Qué servicios proveen dichos protocolos? ¿Cuáles son sus características? ¿Cuando es apropiado utilizar cada uno?

6. Entrega

La entrega consta de un informe, el código fuente de la aplicación desarrollada y un archivo README en formato MarkDown[3] con los detalles necesarios para ejecutar la aplicación. La codificación debe cumplir el standard PEP8 [4], para ello se sugiere utilizar el linter flake8 [5]. La figure 2 muestra la estructura y el contenido del archivo ZIP entregable.

Figura 2: Estructura del archivo zip entregable

6.1. Fecha de entrega

La entrega se hará a través del campus. La fecha de entrega está pautada para el día martes 25 de Abril de 2023 a las 19.00hs. **Cualquier entrega fuera de término no será considerada.**

6.2. Informe

La entrega debe contar con un informe donde se demuestre conocimiento de la interfaz de sockets, así como también los resultados de las ejecuciones de prueba (capturas de ejecución de cliente y logs del servidor). El informe debe describir la arquitectura de la aplicación. En particular, se pide detallar el protocolo de red implementado para cada una de las operaciones requeridas.

El informe debe tener la siguientes secciones:

- Introducción
- Hipótesis y suposiciones realizadas
- Implementación
- Pruebas
- Preguntas a responder
- Dificultades encontradas
- Conclusión

Referencias

[1] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[2] Python Software Foundation. *socket — Low-level networking interface*, 2020. `https://docs.python.org/3/library/socket.html` [Accessed: 15/10/2020].

[3] John Gruber. *Markdown*, 2020. `https://daringfireball.net/projects/markdown/` [Accessed: 15/10/2020].

[4] Guido van Rossum, Barry Warsaw, and Nick Coghlan. Style guide for Python code. PEP 8, Python Software Foundation, 2001.

[5] Ian Stapleton Cordasco. *Flake8: Your Tool For Style Guide Enforcement*, 2020. `https://flake8.pycqa.org/en/latest/` [Accessed: 15/10/2020].