# Laboratorio: Diagrama de transición de estados de TCP

Cátedra de Redes y Sistemas Distribuidos
Sergio Canchi - 2015 (basado en lab-scapy)

### Descargo de responsabilidad

En este laboratorio se va a demostrar cómo capturar contenido enviado por red a la computadora en la que estamos trabajando. Si se distribuyen o reproducen públicamente copias de archivos cuyo contenido esta protegido por derecho de autor puede estarse cometiendo un delito si uno no cuenta con la autorización del tenedor de estos derechos.

Asimismo, si se captura y observa información dirigida a *otra* computadora (lo cual no se explica en este laboratorio) también se corre el riesgo de estar incurriendo en un delito puesto que, por ejemplo, leer correspondencia ajena esta penado con entre 15 días y 6 meses de prisión en Argentina.

Ni la cátedra de Redes y Sistemas Distribuidos, ni FaMAF, ni la UNC se hacen responsables del uso inapropiado del material dado para estas clases.

## Objetivos

- Aprender a usar una **herramienta de análisis de red.**
- Poder **capturar información** enviada en una conexión TCP.
- Implementar el **diagrama de transición de estados** de TCP, especificada en el RFC 793
- Comprender el **progreso de una conexión a través de transiciones entre estados**cubriendo tanto las transiciones típicas de un cliente o servidor y las transiciones raras que son posibles pero no son tan habituales

## Motivación

Una conexión TCP progresa a través de una serie de estados durante su tiempo de vida. La transición de un estado al estado siguiente es basado en la información contenida en los encabezados de los paquetes intercambiados, en consecuencia es posible reconstruir dicho progreso si se tiene acceso a estos paquetes.

Resulta interesante y educativo poder observar y analizar qué información se transmite en una conexión TCP y a partir de esta información visualizar el progreso de la conexión TCP (siempre y cuando no se violen derechos de autor ni licencias de uso)

## Introducción

TCP es *el* protocolo sobre el cual la mayor parte de las aplicaciones que dependen de internet están construidas. Entender su funcionamiento no es solo un buen entrenamiento en la prácticas comunes del diseño de protocolos sino que además puede resultar útil para capturar información que se transmite por la red.

Cuando se esta desarrollando un nuevo protocolo de red, o cuando se están buscando fallas de seguridad en un protocolo es de enorme utilidad poder observar (desde el punto de vista de un tercero) la información que intercambia el protocolo a través de la red. Las herramientas que permiten observar este tipo de actividad se llaman **herramientas de análisis de red**.

De entre estas la herramienta más básica es el **sniffer**, un programa que permite capturar los paquetes de datos tal cual son enviados por la placa de red. Los datos capturados por este programa pueden ser luego alimentados a otros que cumplen funciones de análisis específicas (estadísticas, reconstrucción de contenido, búsqueda de ataques, etc.).

En este laboratorio presentaremos a un sniffer clásico de varios *nix: **tcpdump**. Además utilizaremos **scapy**, una biblioteca python para el análisis de paquetes capturados con tcpdump.

## Tcpdump

Tcpdump lee paquetes de una interfaz de red y los puede presentarlos al usuario de varias formas. Por ejemplo:

```
rafael@tiber:~$ sudo tcpdump -i wlan2 -n -q
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan2, link-type EN10MB (Ethernet), capture size 96 bytes
```

```
12:09:34.478995 IP 192.168.1.108.22 > 192.168.1.110.34657: tcp 192
12:09:34.480783 IP 192.168.1.110.34657 > 192.168.1.108.22: tcp 0
12:09:34.488981 IP 192.168.1.108.22 > 192.168.1.110.34657: tcp 176
12:09:34.490766 IP 192.168.1.110.34657 > 192.168.1.108.22: tcp 0
12:09:34.495459 IP 122.107.108.110.16191 > 192.168.1.108.34909: tcp 0
12:09:34.495480 IP 192.168.1.108.34909 > 122.107.108.110.16191: tcp 1448
...
12:09:34.894421 IP 192.168.1.108.22 > 192.168.1.110.34657: tcp 240
^C
75 packets captured
75 packets received by filter
0 packets dropped by kernel
rafael@tiber:~$
```

Aquí se muestra a tcpdump leyendo paquetes desde wlan2 (una interfaz wireless) y mostrando un resumen por salida estándar. Notar que tcpdump necesita ejecutarse con privilegios de root.

```
rufus@gavilan:~$ sudo tcpdump -i eth1 -n -s 0 -w textfile.pcap
tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
^C179 packets captured
179 packets received by filter
0 packets dropped by kernel
rufus@gavilan:~$
```

Este otro ejemplo hace algo más interesante: Guarda una copia de cada paquete que paso por la interfaz wlan2 en el archivo textfile.pcap. Pcap es un formato de archivo más o menos estándar para almacenar **p**aquetes **cap**turados.

```
[sergio@localhost tmp]# sudo tcpdump -s 0 port http -i enp3s0 -w mypdfcap1.pcap
tcpdump: listening on enp3s0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C60 packets captured
60 packets received by filter
0 packets dropped by kernel
```

También se puede filtrar la captura de paquetes de un determinado puerto con la opción*port*.

# Scapy

Scapy es una biblioteca para python que permite (entre otras cosas) manipular paquetes capturados en archivos pcap. En el kickstart se entrega un pequeño código (`tryme.py`) que carga en el interprete interactivo tres listas:

- **`xs`**: Una lista con todos los paquetes que hay en textfile.pcap
- **`ys`**: Una lista con todos los paquetes de xs que son TCP y que tienen un payload
- **`ws`**: Una lista con todos los paquetes de xs que son TCP y que no tienen un payload

```
[sergio@localhost tmp]$ ipython tryme.py
Python 2.7.5 (default, Jun 17 2014, 18:11:42)
Type "copyright", "credits" or "license" for more information.

IPython 3.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [17]: len(xs)
Out[17]: 179

In [18]: len(ys)
Out[18]: 3

In [19]: len(ws)
Out[19]: 11

In [20]: ws[0].show()
###[ Ethernet ]###
  dst       = 94:0c:6d:bd:3e:f6
  src       = 00:18:f3:54:c5:1c
  type      = IPv4
###[ IP ]###
     version   = 4L
     ihl       = 5L
     tos       = 0x0
     len       = 60
     id        = 7723
     flags     = DF
     frag      = 0L
     ttl       = 64
     proto     = tcp
```

```
     chksum    = 0x8133
     src       = 192.168.1.110
     dst       = 200.16.17.55
     \options   \
###[ TCP ]###
        sport     = 49543
        dport     = http
        seq       = 802911101
        ack       = 0
        dataofs   = 10L
        reserved  = 0L
        flags     = S
        window    = 5840
        chksum    = 0xc33c
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', ''), ('Timestamp', (1076566, 0)), ('NOP', None), ('WScale', 6)]

In [21]: ws[0].sprintf("%IP.src%:%TCP.sport%")
Out[21]: '192.168.1.110:49543'

In [22]: ws[0].sprintf("%TCP.flags%")
Out[22]: 'S'
```

Como se puede ver, solo hay 3 paquetes con payload mientras que hay 11 paquetes sin payload y que el host con address `192.168.1.110:49543` envía un `SYN` en el encabezado TCP

# pcap2py.py

El programa `pcap2py.py` provisto en el kickstart es una herramienta que toma como entrada un archivo .pcap y produce como salida una lista de segmentos TCP, donde cada segmento contiene la información de los encabezados necesarios para realizar una transición de estado. A esta lista de segmentos llamaremos **traza**. Por ejemplo

```
[sergio@localhost tmp]$ python pcap2py.py mypdfcap1.pcap
[
    {'src': '192.168.1.3:38774', 'ack': False, 'dst': '200.16.17.104:http', 'syn': True, 'rst': False, 'fin': False},
    {'src': '200.16.17.104:http', 'ack': True, 'dst': '192.168.1.3:38774', 'syn': True, 'rst': False, 'fin': False},
    {'src': '192.168.1.3:38774', 'ack': True, 'dst': '200.16.17.104:http', 'syn': False, 'rst': False, 'fin': False},
    {'src': '192.168.1.3:38774', 'ack': False, 'dst': '200.16.17.104:http', 'syn': False, 'rst': False, 'fin': False},
    {'src': '200.16.17.104:http', 'ack': True, 'dst': '192.168.1.3:38774', 'syn': False, 'rst': False, 'fin': False},
...
]
```

Este programa hace uso de scapy para realizar analisis de paquetes

# Walnut

Walnut es una herramienta educativa que permite diseñar y simular algoritmos. El simulador de Walnut permite además representar graficamente la ejecución de estos algoritmos a partir del log de la simulación.
Un proyecto en Walnut incluye cuatro secciones: Definición del mundo, Problemas, Visualizaciones y Agentes. En este laboratorio se va a modificar las secciones Problemas y Agentes.

# Tarea

El trabajo en este laboratorio consiste en hacer una implementación del diagrama de transición de estados de TCP de acuerdo a la especificación dada en el RFC793.

Para esto estaremos usando la plataforma Walnut para simular y visualizar la implementación del diagrama de transición de estados para una traza dada.

Deberán

1. Crear una cuenta en Walnut (http://aimara.machinalis.com)
2. Iniciar sesión en Walnut
    1. Realizar un *fork* del proyecto a partir de la siguiente URL (http://aimara.machinalis.com/simulations/edit_world/96/)
    2. Confirmar el fork, presionando *save*
    3. Marcar el Agente *State transition* como privado (por ningún motivo esta permitido hacerlo público)
3. Se deberá completar el programa de la sección Agente *State transition*, responsable de realizar la transición de estado
4. Ejecutar la simulación para las 3 trazas provistas en el kickstart (trace_test[1-3].py). Para ejecutar una simulación se debe realizar los siguientes pasos:

    1. Copiar y pegar una traza en el agente *State transition*
    2. Configurar el problema *Two hosts*: solo se debe modificar los campos *address*tanto del host cliente y del host servidor, usando la información de la traza (valores de 'src' y 'dst' respectivamente)
    3. Guardar los cambios y ejecutar la simulación
5. Se deberá crear manualmente al menos dos trazas que contengan una o mas transiciones raras (transiciones posibles pero no son tan habituales).
6. Opcional: Generar dos nuevas trazas y ejecutar las simulaciones con estas trazas
7. En la entrega se deberá incluir (i) copia del programa modificado del agente *State transition* en un archivo con nombre agent.py, (2) las trazas creadas manualmente y (3) opcionalmente las nuevas trazas generadas.

## Ayudas

- Jugar con `tryme.py` un rato les va a resultar muy provechoso
- RFC 793 (v4, 1981) que especifica el diagrama de transición de estados de TCP.

## Requisitos del código a entregar

- La fecha máxima de entrega es el día lunes 15 de junio, a las 23:59.
- Las entregas serán a través del repositorio provisto por la Cátedra. Recordar que no esta permitido hacer público el programa del agente *State transition* en Walnut
- Junto con el código, se deberá entregar un **brevísimo informe** en el cual se explique los casos especiales en los que no funciona el código, las dificultades con las que se encontraron, y cómo las resolvieron. El informe deberá estar en texto plano, Markdown o reStructuredText.
- Se deberá mantener el estilo de codificación PEP8.
- El trabajo es grupal. Todos los integrantes del grupo deberán poder explicar el código presentado.
- No está permitido compartir código entre grupos.

*$Date$, $Revision$*