

# Nivel de transporte

## - Clase práctica -

### Teoría de las Comunicaciones



DEPARTAMENTO  
DE COMPUTACION

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

# Agenda

## 1 Introducción

- Qué sabemos hasta ahora
- Nivel de transporte: conceptos esenciales

## 2 Nivel de transporte en el modelo TCP/IP

- El protocolo UDP
- TCP: repaso y ejercicios

## 3 Aplicaciones prácticas

- Port scanning
- Herramientas
- Firewalls
- NAT con puertos

# Agenda

## 1 Introducción

- Qué sabemos hasta ahora
- Nivel de transporte: conceptos esenciales

## 2 Nivel de transporte en el modelo TCP/IP

- El protocolo UDP
- TCP: repaso y ejercicios

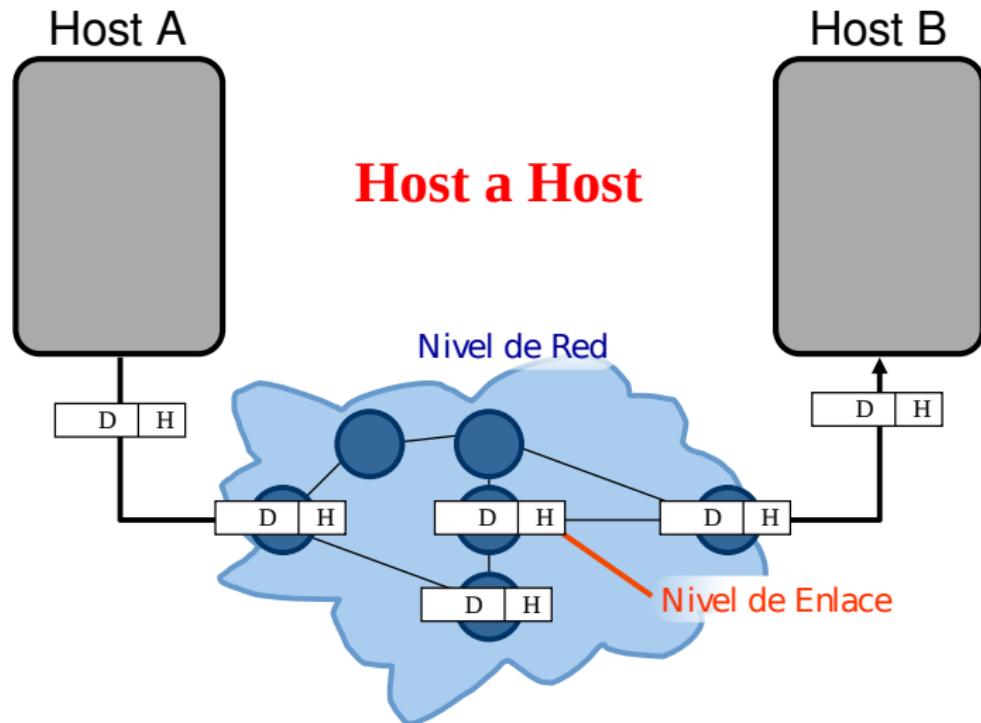
## 3 Aplicaciones prácticas

- Port scanning
- Herramientas
- Firewalls
- NAT con puertos

# Los niveles conocidos del modelo OSI

- **Nivel físico:** lida con el problema de cómo transmitir bits en un medio dado.
- **Nivel de enlace:** conecta dispositivos en una misma red local y transfiere tramas (frames) entre ellos.
- **Nivel de red:** define cómo interconectar redes y dirigir el tráfico de paquetes entre sus destinos.

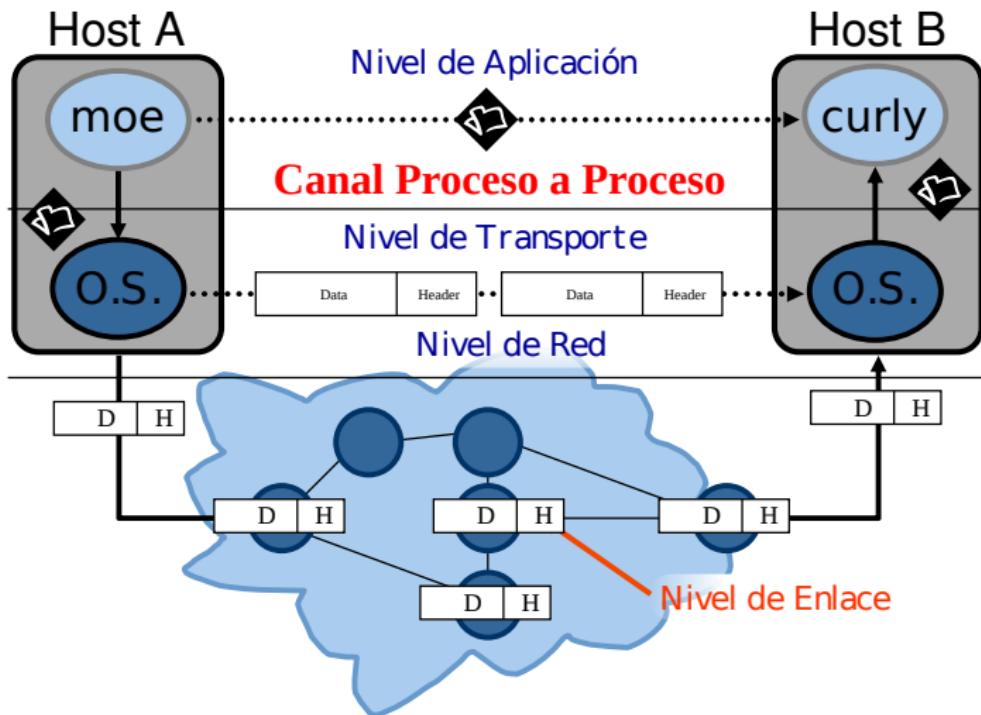
# El nivel de red y sus amigos ilustrados



## ¿Para qué está el nivel de transporte?

- En resumen, hasta ahora tenemos un mecanismo para comunicar hosts con hosts.
- Ahora, el objetivo está en implementar un canal de comunicación entre **procesos**.
- Este canal será en definitiva utilizado por las aplicaciones.
- Problema: capas inferiores suelen tener limitaciones y las aplicaciones pueden requerir servicios confiables.

# El nivel de transporte ilustrado



# Características esenciales

- El nivel de red ofrece un servicio de *mejor esfuerzo*:
  - ▶ Puede descartar mensajes,
  - ▶ Reordenar mensajes,
  - ▶ Entregar copias duplicadas de un mensaje,
  - ▶ Limitar los tamaños de mensaje a un valor fijo, o
  - ▶ Entregar mensajes con una demora arbitraria.
- Sin embargo, el nivel de transporte, para satisfacer a las aplicaciones, debe ofrecer:
  - ▶ Entrega de mensajes garantizada y respetando el orden,
  - ▶ No entregar mensajes duplicados,
  - ▶ Soportar tamaños de mensaje arbitrarios,
  - ▶ Control de flujo por parte del emisor hacia el receptor,
  - ▶ Soportar múltiples procesos en cada host, entre otras.
- ⇒ **Objetivo:** desarrollar algoritmos para alcanzar estos requerimientos.

## Esto me suena de otro lado...

- De lo anterior puede pensarse que el nivel de transporte guarda similitudes con el nivel de enlace:
  - ▶ Ambos deben lidiar con control de errores, secuenciamiento, control de flujo, etc.
- **Pero** el nivel de enlace se apoya en un canal físico y el nivel de transporte tiene debajo una red:
  - ▶ Se requiere **direcccionamiento** explícito,
  - ▶ Establecer conexiones es indefectiblemente más complicado,
  - ▶ La red posee *capacidad de almacenamiento*: un paquete puede quedar almacenado en la red y llegar a destino más tarde,
  - ▶ Etc.

## Cómo identificamos los procesos

- Al iniciar la comunicación, los procesos deben especificar con qué proceso destino desean hablar.
- No sirve PIDs: scope local, y son variables.
- La arquitectura TCP/IP utiliza **puertos**: valor numérico (de 16 bits) que identifica procesos.
- Se utiliza junto con la dirección IP para dar una caracterización unívoca del destino.
- Usualmente implementados mediante **sockets**.

# Puertos: cómo funcionan

- Cuando un proceso se asocia a un puerto, está listo para recibir mensajes.
- Problema: los procesos externos deben tomar conocimiento de este puerto elegido.
- Soluciones:
  - ▶ Utilizar puertos **conocidos**: sucede en la práctica. Ejemplo: web server en puerto 80, mail server en puerto 25, etc.
  - ▶ Utilizar un **port mapper**: proceso que sabe cuál puerto utiliza cada servicio. Corre en un puerto fijo. Ejemplo: RPC.
- Los servidores saben cómo responder a los clientes: el puerto respectivo viene en los mensajes que ellos envían.

# Puertos y servicios en Linux

- En Linux, el archivo `/etc/services` lista los puertos conocidos de los distintos servicios:

ftp-data	20/tcp	
ftp	21/tcp	
fsp	21/udp	fspd
ssh	22/tcp	
ssh	22/udp	
telnet	23/tcp	
smtp	25/tcp	mail
time	37/tcp	timserver
time	37/udp	timserver

- Se usa para obtener, desde el código, el puerto de un servicio a través de su nombre.

# Clasificación de protocolos de transporte

- Podemos clasificar los protocolos de nivel de transporte a través de las siguientes características:
  - ▶ Orientación a conexión
  - ▶ Confiabilidad
- En TCP/IP se distinguen fundamentalmente estos protocolos:
  - ▶ **TCP**: con conexión y confiable.
  - ▶ **UDP**: sin conexión y no confiable.
- Hay otros. Ejemplo: RUDP (UDP extendido para hacerlo confiable y al mismo tiempo evitar el overhead de TCP).

# Agenda

## 1 Introducción

- Qué sabemos hasta ahora
- Nivel de transporte: conceptos esenciales

## 2 Nivel de transporte en el modelo TCP/IP

- El protocolo UDP
- TCP: repaso y ejercicios

## 3 Aplicaciones prácticas

- Port scanning
- Herramientas
- Firewalls
- NAT con puertos

- UDP -

# El protocolo UDP

- UDP (*User Datagram Protocol*), como hemos dicho, es no confiable ni orientado a conexión.
- Especificado en el RFC 768.
- Extiende el servicio de la capa de red subyacente transformándolo en un canal de comunicación entre procesos.
- ⇒ Agrega **demultiplexación** a IP.

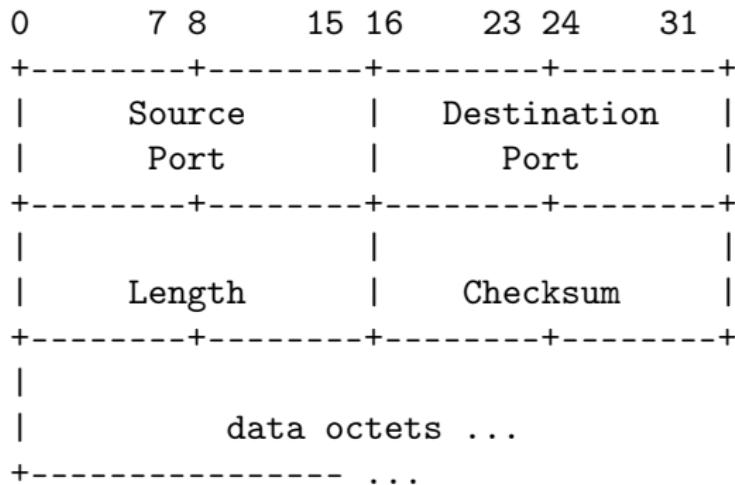
## Características de UDP

- Establece una comunicación poco costosa entre procesos.
- Evita la sobrecarga y las demoras propias de la entrega ordenada y confiable de mensajes.
- Soporta múltiples procesos de nivel de aplicación en cada máquina.
- Tráfico *egoísta*: puede congestionar las colas de entrada y salida de los routers.
- Sin control de flujo: el destinatario puede verse saturado y perder información en consecuencia.
- Esencialmente, **IP + puerto**.

# ¿Por qué UDP?

- Control preciso sobre qué información se envía y cuándo se envía.
- Sin demoras para establecimiento de conexiones.
  - ▶ UDP no contempla ningún procedimiento preliminar formal.
- Sin *estado de conexión*: no se reservan parámetros, números de secuencia, etc.
  - ▶ Esto facilita la manipulación de muchos clientes activos en simultáneo.
- Poco overhead por encabezado: sólo 8 bytes.

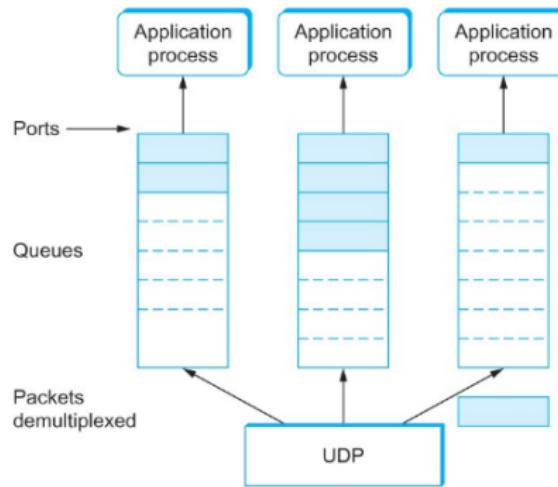
## Formato del datagrama (cortesía RFC)



- Length: longitud (en bytes) del datagrama completo. Tamaño mínimo: 8 bytes (¿por qué?).
- Checksum
  - ▶ Se computa sobre el header, los datos y el *pseudoheader*: campos Protocol, Source Address y Destination Address del header IP.
  - ▶ Opcional en IP; obligatorio en IPv6.

## Puertos como colas de mensajes (cortesía Peterson)

- Cuando un paquete arriba a destino, éste se dirige al puerto correspondiente y aguardará allí encolado.
- De no haber espacio suficiente, se descartará sin más.
- La aplicación removerá el primer paquete encolado al momento de recibir mensajes.
- Si la cola está vacía, se bloqueará.



# - TCP -

# El protocolo TCP

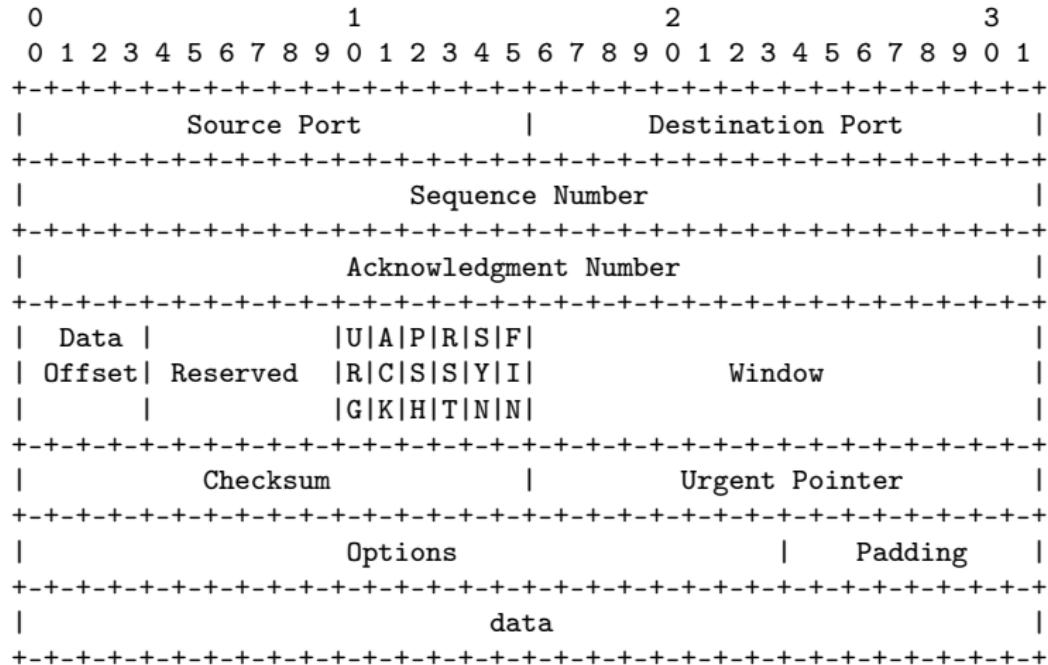
- Protocolo de nivel de transporte central en el modelo TCP/IP.
- Ofrece un servicio **confiable** y **orientado a conexión** a los protocolos de aplicación que corren sobre él.
  - ▶ Abstacta la red subyacente exponiendo un stream de bytes ordenado y libre de errores.
- Estandarizado originalmente en el RFC 793 (circa 1981).
  - ▶ No obstante, continuó evolucionando constantemente y fue rectificado por RFCs posteriores (e.g., RFC 1122).
  - ▶ Actualmente, existe un draft (RFC 793 bis) que volvería obsoleto a este RFC, delineando las características fundamentales de TCP en el contexto de la Internet moderna:

<http://www.ietf.org/id/draft-eddy-rfc793bis-05.txt>

# Diferencias básicas con UDP

- Transferencia de datos en orden
  - ▶ Esto es efectuado por el receptor a través de los números de secuencia de los segmentos.
- Garantía de transmisión de datos libre de errores
- Control de flujo
  - ▶ Determina un límite al volumen de datos que el emisor puede enviar.
  - ▶ El receptor informa cuántos bytes está dispuesto a aceptar a través de un campo en el header (similar a la RWS que ya estudiamos en protocolos de sliding window).
- Retransmisión de segmentos
  - ▶ Todo byte no reconocido es eventualmente retransmitido.
- Control de congestión
  - ▶ Sobre esto volveremos la semana que viene!

# Formato del segmento (cortesía RFC)



## Esquema de reconocimiento

- El número de secuencia sirve para identificar a **cada byte** enviado.
- Este valor se incrementa por cada byte que se envía.
- A su vez, cada byte debe reconocerse a través de paquetes ACK.
- El esquema de reconocimiento utilizado consta de ACKs **acumulativos**.
  - ▶ En particular, si  $\#ACK = n$ , esto significa que el próximo byte que el emisor del ACK espera recibir es el  $n$ .
  - ▶ O, en otras palabras, que recibió correctamente los bytes secuenciados  $0, \dots, n - 1$ .
- TCP también provee reconocimiento selectivo a través de opciones.
- Si un ACK no es recibido dentro de un tiempo estimado, se procederá a retransmitir los bytes en cuestión.
  - ▶ El funcionamiento de este mecanismo recae en la estimación del RTO (*retransmission timeout*), que a su vez depende de la estimación del RTT de la conexión.

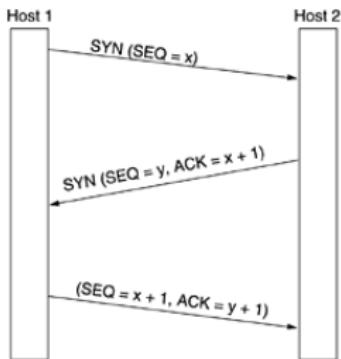
# TCP: Modo de operación

- El protocolo distingue tres fases de operación:
  - ▶ *Establecimiento de conexión*, en donde ambos extremos sincronizan números de secuencia y reservan estado,
  - ▶ *Transferencia de datos*, que corresponde a la etapa en la que ambas partes pueden enviar y recibir información, y
  - ▶ *Cierre de conexión*, donde se produce un cierre (potencialmente asimétrico) del canal y se liberan los recursos usados.
- A lo largo de estas fases, la conexión transitará diversos **estados**. Éstos se formalizan a través de una máquina de estados en la que distintos eventos causarán transiciones entre ellos (volvemos sobre esto en instantes).

# Establecimiento de conexión

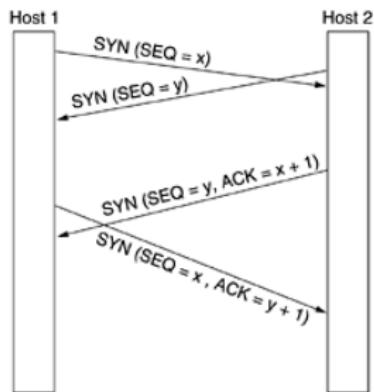
- El algoritmo para establecer una conexión TCP se conoce como *three-way handshake*.
- El objetivo es que ambas partes puedan sincronizar los números de secuencia iniciales (ISN).
- Si bien se trata de una conexión full-duplex, usualmente es posible identificar un actor *pasivo* ("servidor") y otro *activo* ("cliente").
  - ▶ El servidor debe hacer antes un *passive open*, que consiste en ligarse a un puerto local y ponerse a escuchar conexiones allí.
  - ▶ La acción del cliente, *active open*, consiste en enviar un paquete TCP con el flag de SYN prendido a dicho puerto.
- **Importante:** el flag de SYN es secuenciable! Esto significa que un paquete SYN consume un número de secuencia.

# Three-way handshake tradicional (cortesía Tanenbaum)



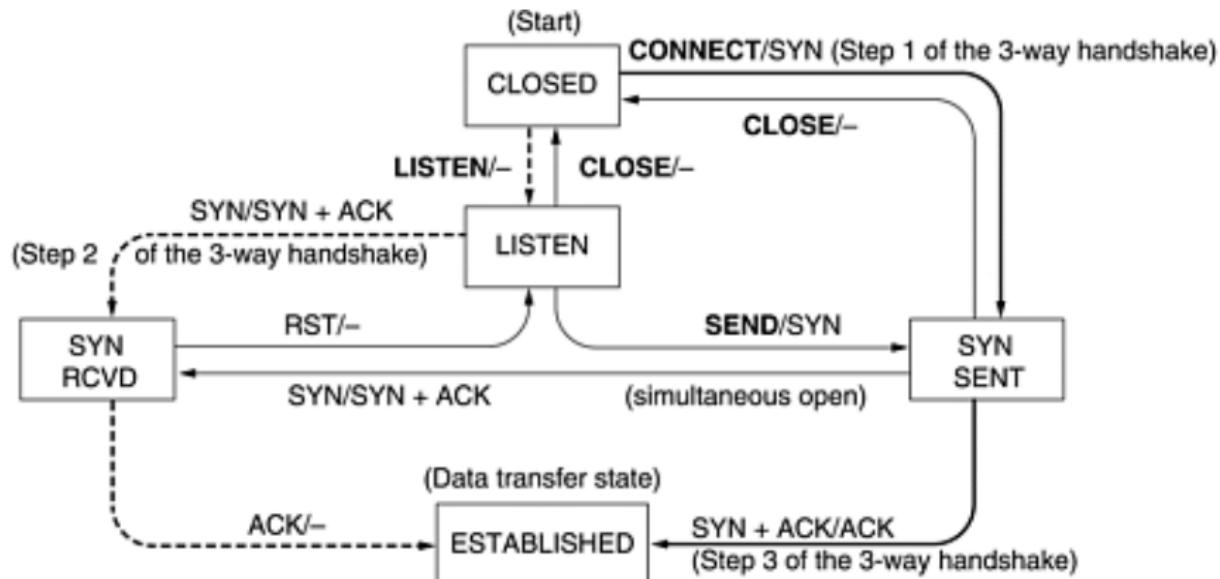
- El “servidor” (host 2) efectúa un LISTEN y el “cliente” un CONNECT, acción que dispara un paquete SYN con número de secuencia inicial  $x$ .
- El servidor responde con SYN+ACK, indicando su número de secuencia  $y$  y reconociendo el valor de  $x$  escogido por el cliente.
- Finalmente el cliente reconoce el valor de  $y$ , quedando así establecida la conexión.
- **Importante:** si en el host 2 no hubiera un proceso escuchando en el puerto solicitado por el cliente, el sistema operativo responderá con un paquete con el flag RST prendido.

## Otro escenario posible (cortesía Tanenbaum)



- Aunque es muy poco frecuente, el protocolo también especifica la situación de una **apertura simultánea**.
- En este caso, sendos hosts responderán al SYN de la contraparte con un SYN+ACK reconociendo el número de secuencia enviado.
- Al recibir los ACKs, cada host considerará que la conexión está establecida.

# Diagrama de estados: apertura (cortesía Tanenbaum)



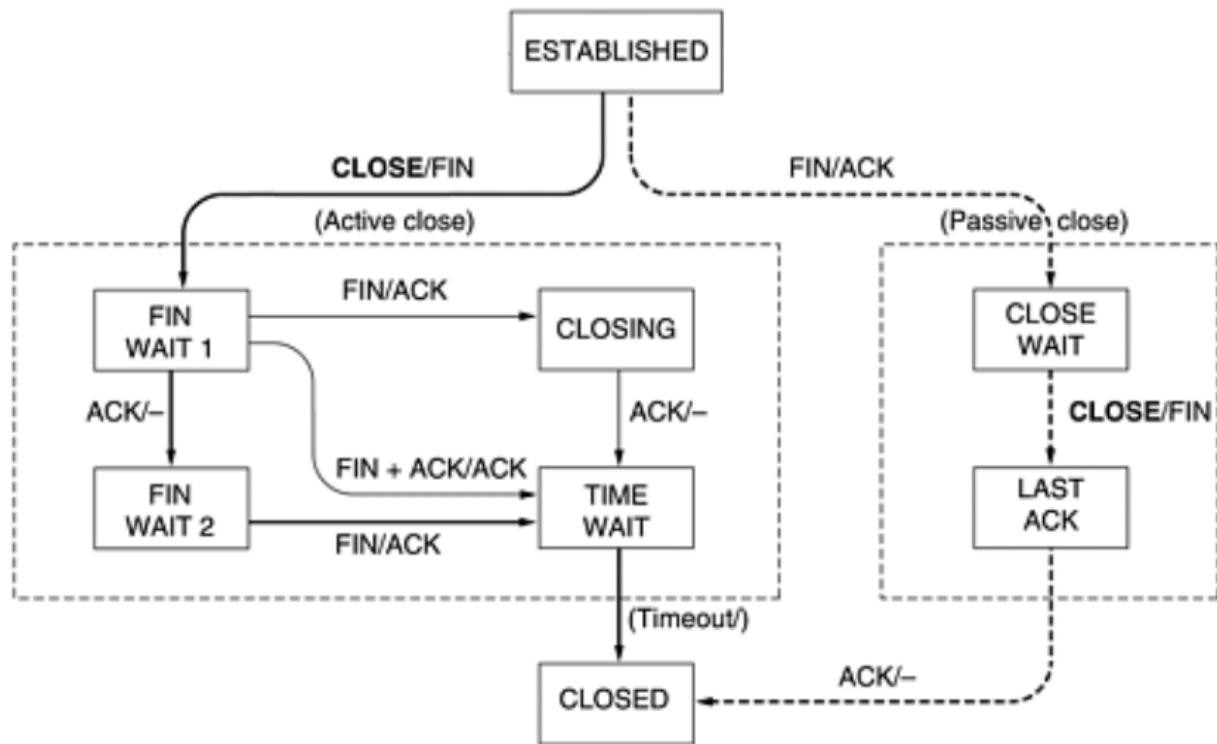
## Cierre de conexión

- Al cerrar la conexión, TCP admite asimetría: un extremo puede decidir cerrar su stream de escritura pero al mismo tiempo continuar recibiendo datos.
- Esto se logra a través del *four-way handshake*.
  - ▶ Un paquete FIN indica que el emisor no enviará más datos.
  - ▶ El receptor debe reconocer este FIN con un paquete ACK.
  - ▶ Eventualmente, el receptor enviará su FIN y éste será a su vez reconocido.
- No obstante, el escenario de cierre más común es simétrico:
  - ▶ Al FIN inicial se responde con FIN+ACK.
  - ▶ Luego, el emisor del primer FIN responde con un ACK.
- **Importante:** el flag FIN también es secuenciable!

## Cierre simultáneo

- Al igual que durante el inicio de conexión, el protocolo también especifica una situación de **cierre simultáneo**.
- En ésta, cada interlocutor envía en forma independiente su FIN antes de que el FIN del otro sea correctamente recibido.
- La secuencia de estados atravesada en este caso será distinta a la del flujo asimétrico anterior.

# Diagrama de estados: cierre (cortesía Tanenbaum)



## Ejercicio 5 - práctica 6

- a. Asociar cada segmento con cada una de las conexiones indicando el criterio usado para determinar a qué conexión pertenece un segmento. ¿Cuántas son?

#	ORIG	DEST	FLAGS	#SEQ	#ACK	LENGTH
1	1.2.3.4:5678	20.232.1.1:80	S	0	—	0
2	1.1.1.1:2222	3.3.3.3:4444	S	42	—	0
3	20.232.1.1:80	1.2.3.4:5678	SA	10000	1	0
4	1.2.3.4:5678	20.232.1.1:80	A	1	10001	0
5	3.3.3.3:4444	1.1.1.1:2222	SA	54321	43	0
6	1.2.3.4:5678	20.232.1.1:80	A	1	10001	50
7	1.1.1.1:2222	3.3.3.3:4444	A	43	54322	0
8	20.232.1.1:80	1.2.3.4:5678	A	10001	51	0
9	20.232.1.1:80	1.2.3.4:5678	A	10001	51	200
10	1.1.1.1:2222	3.3.3.3:4444	SAU	43	64334	0
11	20.232.1.1:80	1.2.3.4:5678	F	10201	—	0
12	1.2.3.4:5678	20.232.1.1:80	FA	51	10202	0
13	3.3.3.3:4444	1.1.1.1:2222	RA	54321	43	0
14	20.232.1.1:80	1.2.3.4:5678	A	10202	52	0

## Ejercicio 5 - práctica 6

- a. Asociar cada segmento con cada una de las conexiones indicando el criterio usado para determinar a qué conexión pertenece un segmento. ¿Cuántas son?

#	ORIG	DEST	FLAGS	#SEQ	#ACK	LENGTH
1	1.2.3.4:5678	20.232.1.1:80	S	0	—	0
2	1.1.1.1:2222	3.3.3.3:4444	S	42	—	0
3	20.232.1.1:80	1.2.3.4:5678	SA	10000	1	0
4	1.2.3.4:5678	20.232.1.1:80	A	1	10001	0
5	3.3.3.3:4444	1.1.1.1:2222	SA	54321	43	0
6	1.2.3.4:5678	20.232.1.1:80	A	1	10001	50
7	1.1.1.1:2222	3.3.3.3:4444	A	43	54322	0
8	20.232.1.1:80	1.2.3.4:5678	A	10001	51	0
9	20.232.1.1:80	1.2.3.4:5678	A	10001	51	200
10	1.1.1.1:2222	3.3.3.3:4444	SAU	43	64334	0
11	20.232.1.1:80	1.2.3.4:5678	F	10201	—	0
12	1.2.3.4:5678	20.232.1.1:80	FA	51	10202	0
13	3.3.3.3:4444	1.1.1.1:2222	RA	54321	43	0
14	20.232.1.1:80	1.2.3.4:5678	A	10202	52	0

# Establecimiento de la conexión

Siguiendo con el ejercicio 5 de la práctica 6.

- b. Indicar los segmentos correspondientes al 3-ways handshake de ambas conexiones.

#	ORIG	DEST	FLAGS	#SEQ	#ACK	LENGTH
1	1.2.3.4:5678	20.232.1.1:80	S	0	—	0
2	1.1.1.1:2222	3.3.3.3:4444	S	42	—	0
3	20.232.1.1:80	1.2.3.4:5678	SA	10000	1	0
4	1.2.3.4:5678	20.232.1.1:80	A	1	10001	0
5	3.3.3.3:4444	1.1.1.1:2222	SA	54321	43	0
6	1.2.3.4:5678	20.232.1.1:80	A	1	10001	50
7	1.1.1.1:2222	3.3.3.3:4444	A	43	54322	0
8	20.232.1.1:80	1.2.3.4:5678	A	10001	51	0
9	20.232.1.1:80	1.2.3.4:5678	A	10001	51	200
10	1.1.1.1:2222	3.3.3.3:4444	SAU	43	64334	0
11	20.232.1.1:80	1.2.3.4:5678	F	10201	—	0
12	1.2.3.4:5678	20.232.1.1:80	FA	51	10202	0
13	3.3.3.3:4444	1.1.1.1:2222	RA	54321	43	0
14	20.232.1.1:80	1.2.3.4:5678	A	10202	52	0

# Cierre de la conexión

## Ejercicio 5 de la práctica 6.

- e) Detectar el cierre anómalo de una de las conexiones e indique una posible causa.

#	ORIG	DEST	FLAGS	#SEQ	#ACK	LENGTH
1	1.2.3.4:5678	20.232.1.1:80	S	0	—	0
2	1.1.1.1:2222	3.3.3.3:4444	S	42	—	0
3	20.232.1.1:80	1.2.3.4:5678	SA	10000	1	0
4	1.2.3.4:5678	20.232.1.1:80	A	1	10001	0
5	3.3.3.3:4444	1.1.1.1:2222	SA	54321	43	0
6	1.2.3.4:5678	20.232.1.1:80	A	1	10001	50
7	1.1.1.1:2222	3.3.3.3:4444	A	43	54322	0
8	20.232.1.1:80	1.2.3.4:5678	A	10001	51	0
9	20.232.1.1:80	1.2.3.4:5678	A	10001	51	200
10	1.1.1.1:2222	3.3.3.3:4444	SAU	43	64334	0
11	20.232.1.1:80	1.2.3.4:5678	F	10201	—	0
12	1.2.3.4:5678	20.232.1.1:80	FA	51	10202	0
13	3.3.3.3:4444	1.1.1.1:2222	RA	54321	43	0
14	20.232.1.1:80	1.2.3.4:5678	A	10202	52	0

## Ejercicio 5 - práctica 6 (item c.)

Detectar el cierre anómalo de una de las conexiones e indique una posible causa.

## Ejercicio 5 - práctica 6 (item c.)

Detectar el cierre anómalo de una de las conexiones e indique una posible causa.

Conexión #2, paquete #10

#	ORIG	DEST	FLAGS	#SEQ	#ACK	LENGTH
2	1.1.1.1:2222	3.3.3.3:4444	S	42	—	0
5	3.3.3.3:4444	1.1.1.1:2222	SA	54321	43	0
7	1.1.1.1:2222	3.3.3.3:4444	A	43	54322	0
10	1.1.1.1:2222	3.3.3.3:4444	SAU	43	64334	0
13	3.3.3.3:4444	1.1.1.1:2222	RA	54321	43	0

## Ejercicio 5 - práctica 6 (item d.)

Para la otra conexión, detallar los cambios de estado en cada extremo a lo largo de toda la comunicación.

#	ORIG	DEST	FLAGS	#SEQ	#ACK	LENGTH
1	1.2.3.4:5678	20.232.1.1:80	S	0	—	0
3	20.232.1.1:80	1.2.3.4:5678	SA	10000	1	0
4	1.2.3.4:5678	20.232.1.1:80	A	1	10001	0
6	1.2.3.4:5678	20.232.1.1:80	A	1	10001	50
8	20.232.1.1:80	1.2.3.4:5678	A	10001	51	0
9	20.232.1.1:80	1.2.3.4:5678	A	10001	51	200
11	20.232.1.1:80	1.2.3.4:5678	F	10201	—	0
12	1.2.3.4:5678	20.232.1.1:80	FA	51	10202	0
14	20.232.1.1:80	1.2.3.4:5678	A	10202	52	0

## Ejercicio de recuperatorio - 1er cuatrimestre de 2014

La siguiente captura de paquetes TCP corresponde a un programa corriendo en un host A comunicándose con el servicio corriendo en el puerto 5900 del host B:

No.	Source	Dest	Info
1	A	B	26574 > 5900 [SYN] Seq=0 Ack=0 Len=0
2	B	A	5900 > 26574 [SYN,ACK] Seq=0 Ack=1 Len=0
3	A	B	26574 > 5900 [ACK] Seq=1 Ack=1 Len=0
4	A	B	26574 > 5900 [PSH,ACK] Seq=1 Ack=1 Len=1024
5	B	A	5900 > 26574 [ACK] Seq=1 Ack=1025 Len=200
6	A	B	26574 > 5900 [ACK] Seq=1025 Ack=201 Len=0

- a) Extender la captura proponiendo una serie de paquetes que hagan que el socket del host A atraviese los siguientes estados:

ESTABLISHED - FIN-WAIT-1 - FIN-WAIT-2 - TIME-WAIT -  
CLOSED

- b) Proponer otros dos escenarios de cierre que podrían darse en esta conexión mencionando los estados transitados por cada socket.

# Agenda

## 1 Introducción

- Qué sabemos hasta ahora
- Nivel de transporte: conceptos esenciales

## 2 Nivel de transporte en el modelo TCP/IP

- El protocolo UDP
- TCP: repaso y ejercicios

## 3 Aplicaciones prácticas

- Port scanning
- Herramientas
- Firewalls
- NAT con puertos

# - PORT SCANNING -

## A por los puertos!

- La identificación de los puertos disponibles se conoce como *port scanning*.
- Procedimiento que barre una secuencia de puertos en un host dado enviando paquetes y analizando las posibles respuestas.
- Se puede materializar de formas variadas, según la heurística que se desee utilizar.
- Hoy mencionaremos sólo los dos más importantes: SYN scan y connect scan.
- Se apoyan fuertemente en la compatibilidad con la especificación de TCP (RFC 793).

# Clasificación de puertos

- Usualmente, un scan clasificará a un puerto de la siguiente manera:
- **open** (abierto) si pudo inferir de su heurística que hay un servicio escuchando en dicho puerto.
- **closed** (cerrado) si determinó que no hay un proceso escuchando allí.
- **filtered** (filtrado) si hay chances de que alguna entidad intermedia (e.g., firewall) descarte el tráfico correspondiente.
- Puede ocurrir que la heurística del scanner no pueda discernir completamente entre estas categorías.

## SYN scan (o half-open scan)

- El scan más popular!
- Envía un paquete TCP con flag SYN y espera la respuesta:
  - ▶ SYN/ACK: indica que el puerto está abierto.
  - ▶ RST: indica que el puerto está cerrado.
  - ▶ Si no hay respuesta, se asume filtrado. Ídem si la respuesta es ICMP de tipo *destination unreachable*.
- No se completa la conexión.

## SYN scan: ventajas y desventajas

- Suele ser rápido.
- Poco invasivo y discreto, al no completar las conexiones.
- Por otro lado, requiere enviar y recibir paquetes ad-hoc, lo cual requiere permisos elevados en el sistema.

## Connect scan (o TCP scan)

- Usa la API del sistema operativo para establecer una conexión: primitiva connect de sockets.
- Así se inicia una conexión usualmente en clientes web, clientes de mail, clientes FTP, etc.
- Observar que se instancia por completo el algoritmo de three-way handshake!

## Connect scan: ventajas y desventajas

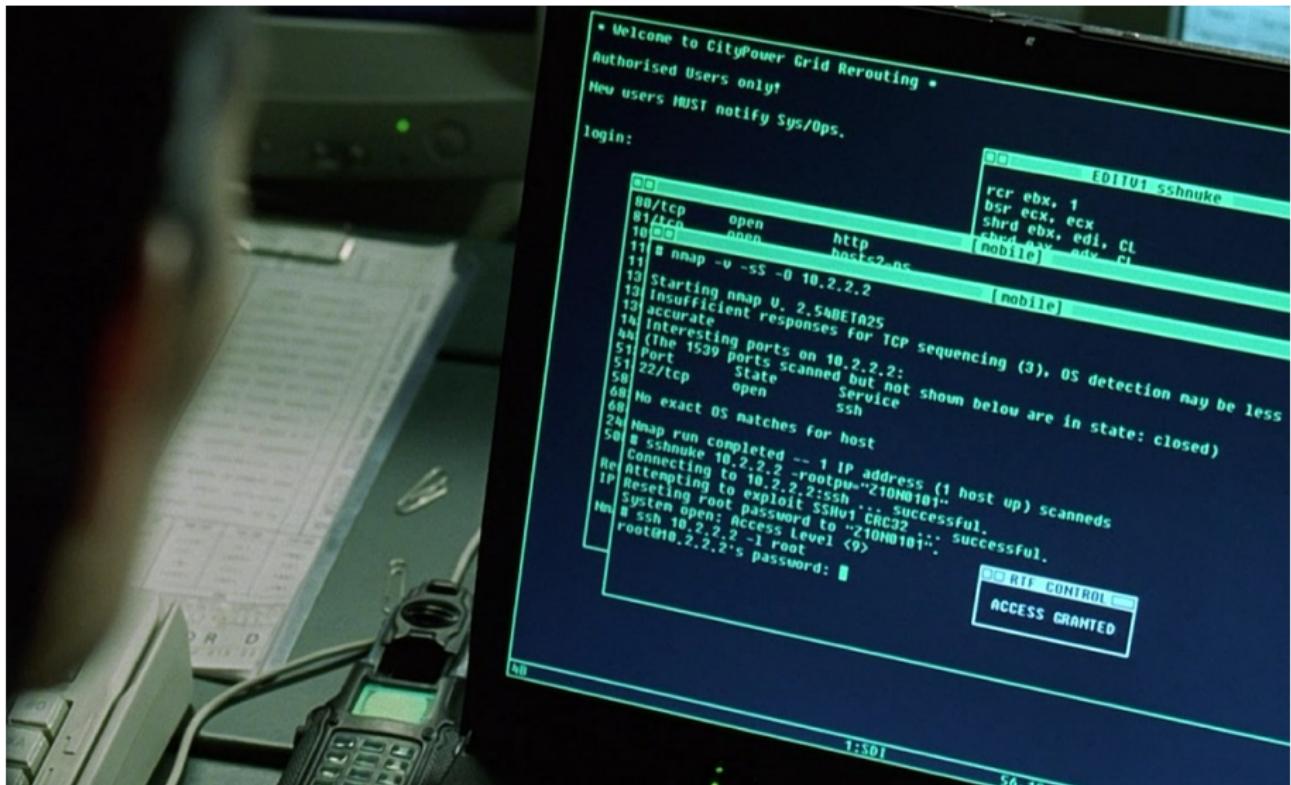
- No requiere permisos elevados para correr.
- Pero tiene varias contras...
  - ▶ Más lento y con más volumen de tráfico, al completar conexiones.
  - ▶ Menos discreto: los hosts posiblemente registren en logs estas conexiones.

# - HERRAMIENTAS -

## nmap

- Es una herramienta de discovery muy famosa y muy completa:
  - ▶ Discovery de hosts,
  - ▶ Escaneo de puertos (implementa todas las técnicas mencionadas hace un rato y algunas otras más esotéricas),
  - ▶ Detección de servicios,
  - ▶ Detección de sistemas operativos.
- Además tiene soporte para scripts custom.
- Apareció por 1997 y tiene soporte para múltiples plataformas.
- El nombre viene de *network mapper*.

# ...y lo usa Trinity en Matrix!



## nmap : fragmento de una corrida

```
lucio@knuth:~$ sudo nmap -sS -sV 192.168.1.100
```

```
Starting Nmap 5.21 ( http://nmap.org ) at 2012-10-30 01:56 ART
Nmap scan report for 192.168.1.100
Host is up (0.013s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 5.3p1 Debian 3ubuntu6 (protocol 2.0)
80/tcp    open  http?
1 service unrecognized despite returning data.
MAC Address: 00:24:8C:96:57:8B (Asustek Computer)
Service Info: OS: Linux

Service detection performed. Please report any incorrect results
at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 134.77 seconds
```

## telnet

- En realidad es un protocolo de nivel de aplicación (sobre TCP)...
- Está especificado en el RFC 854.
- Provee acceso a una interfaz de línea de comandos en la máquina destino.
- Muy usado, no obstante, para testear conectividad.
- Hoy por hoy su uso para acceso remoto quedó obsoleto por ser poco seguro (a diferencia de SSH).

## telnet : un ejemplo

- Veamos cómo usar telnet para conectarnos a un servidor de mails:

```
lucio@knuth:~$ telnet proxymail1.sion.com 25
Trying 200.81.186.15...
Connected to proxymail1.sion.com.
Escape character is '^]'.
220 mx2.sion.com ESMTP (SION)
QUIT
221 2.0.0 Bye
Connection closed by foreign host.
```

## netstat

- Sirve para ver y analizar las conexiones TCP (...y UDP).
- Puede indicar los procesos ligados a cada socket.
- Muestra el estado de las conexiones TCP (según el famoso diagrama de estados que ya conocemos).

## netstat : fragmento de una corrida

```
lucio@knuth:~$ netstat -an | more
(...)
Proto Local Address          Foreign Address        State
tcp   127.0.0.1:631          0.0.0.0:*
tcp   127.0.0.1:5432         0.0.0.0:*
tcp   0.0.0.0:902           0.0.0.0:*
tcp   0.0.0.0:80            0.0.0.0:*
tcp   192.168.1.101:46670    65.55.71.176:1863  ESTABLISHED
tcp   192.168.1.101:37853    74.125.130.125:5222 ESTABLISHED
tcp   192.168.1.101:57385    192.168.1.100:22   ESTABLISHED
(...)
```

## netcat

- La navaja suiza para TCP/IP.
- Múltiples y variadas funcionalidades:
  - ▶ Hablar con servidores (como hicimos con telnet )
  - ▶ Escaneo de puertos
  - ▶ Transferencia de archivos
  - ▶ Escuchar en un puerto dado

## netcat : ejemplo de port scanning

```
lucio@knuth:~$ nc -vz 192.168.1.100 77-83
nc: connect to 192.168.1.100 port 77 (tcp) failed
nc: connect to 192.168.1.100 port 78 (tcp) failed
nc: connect to 192.168.1.100 port 79 (tcp) failed
Connection to 192.168.1.100 80 port [tcp/www] succeeded!
nc: connect to 192.168.1.100 port 81 (tcp) failed
nc: connect to 192.168.1.100 port 82 (tcp) failed
nc: connect to 192.168.1.100 port 83 (tcp) failed
```

## netcat : ejemplo de transferencia de archivos

```
lucio@knuth:~$ nc -l localhost 12345 > file &
[1] 5557
lucio@knuth:~$ echo "eh amigo" > eh_amigo
lucio@knuth:~$ nc localhost 12345 < eh_amigo
[1]+  Done                      nc -l localhost 12345 > file
lucio@knuth:~$ cat file
eh amigo
```

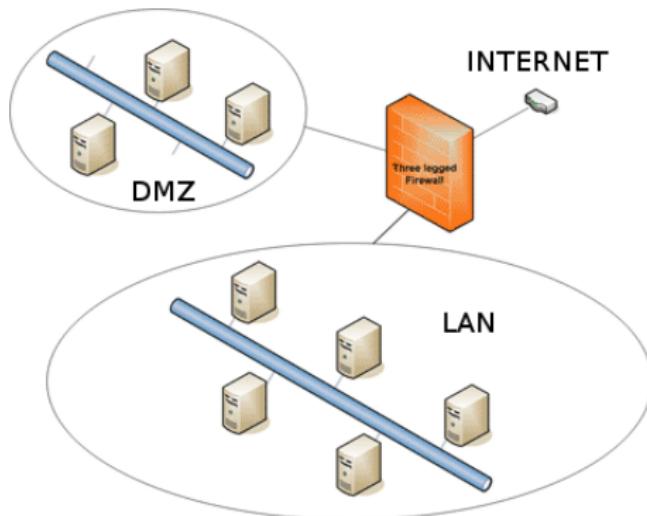
- *socket statistics*: herramienta para investigar el estado interno de los sockets.
- Similar a `netstat`, aunque permite obtener información de las variables internas de TCP:
  - ▶ Estimación actual del RTT y su varianza (RTTVAR)
  - ▶ Estimación actual del RTO
  - ▶ Información sobre las variables de control de congestión
  - ▶ Cantidad de retransmisiones y de segmentos no ACKeados
  - ▶ Etc.
- Además de la `man page`, más info (aunque no mucha) en:  
<http://www.cyberciti.biz/files/ss.html>

## ss : ejemplo de la demo

```
Recv-Q Send-Q      Local Address:Port      Peer Address:Port
0        0          127.0.0.1:33088        127.0.0.1:5555
          cubic wscale:7,7 rto:220 rtt:21.5/10 mss:65483
          cwnd:27 ssthresh:26 send 657.9Mbps rcv_space:43690
```

# Firewall - Definición

Conjunto de sistemas encargado de mediar todas las comunicaciones entre dos redes implementando una política de seguridad de **control de acceso** mediante el **filtrado de paquetes**.



Esencialmente verifica cada paquete independientemente de los demás y permite o denegado su paso dependiendo de un **conjunto de reglas**.

# Distintos tipos de Firewalls

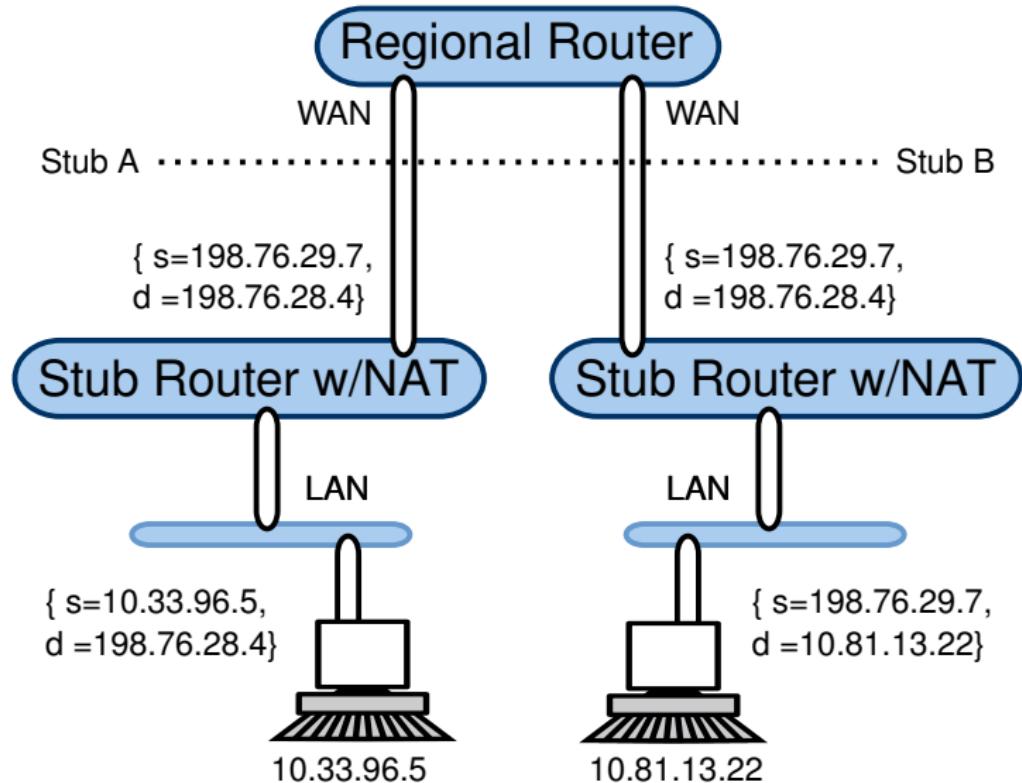
- **Stateless:** Utiliza reglas sencillas sobre los mensajes entrantes / salientes del tipo:  
 $< dir_{origen}, puerto_{origen}, dir_{dst}, puerto_{dst}, protocolo >$
- **Stateful:** Mantiene estado de conexiones entrantes o salientes y puede permitir o denegar teniendo en cuenta la sesión a la que pertenece.
- **Gateways de aplicación:** Proxy inteligente. Entiende ciertos protocolos. Sirve para aplicar filtros de capa de aplicación.

# - NAT CON PUERTOS -

# ¿Qué es NAT?

- NAT (*Network Address Translation*) es un método para mapear direcciones IP de un dominio (usualmente privado) a otro (usualmente público).
- Su forma más básica está especificada en el RFC 1631.
- Función implementada en los gateways: transparente para los hosts.
- Se ideó para atacar el problema de la escasez de direcciones IP.
- Pools de direcciones privadas pueden reutilizarse arbitrariamente.

## NAT ilustrado (cortesía RFC)



## Detalles...

- NAT no sólo modifica la dirección IP de los paquetes.
- **Debe** modificar el checksum.
- ...y también el checksum de nivel de transporte por el pseudoheader!
- En general, debe tocar cualquier parte donde aparezca la IP.
- Ejemplo: ICMP (payload).
- Potenciales problemas:
  - ▶ Aplicaciones podrían romperse.
  - ▶ Seguridad.

## NAT con puertos (o NAPT)

- Una extensión: NAPT (*Network Address and Port Translation*).
- Mapeo de muchos-a-uno a diferencia del mapeo biyectivo anterior.
- La dirección pública es la misma para todos los hosts de la red privada.
- Los puertos (públicos) son los que identifican el tráfico y definen el mapeo.
- Terminología en el RFC 2663.

## Cómo trabaja NAPT

- Cuando un host quiere contactar un servicio externo, el paquete saliente de su red privada contendrá también ahora un puerto origen **distinto**.
- El gateway realiza esto y guarda la entrada en su tabla de mapeos.
- Al revés no: un host externo no puede iniciar la comunicación.
- Solución: **port forwarding**.
  - ▶ Reglas definidas en el gateway para asociar un puerto externo a un servicio interno.
  - ▶ Ejemplo:  $(190.172.12.16, 8080) \rightarrow (192.168.0.100, 80)$

# Referencias

-  RFCs 768, 793, 1631 y 2663.
-  *Nmap Port Scanning Techniques*  
<http://nmap.org/book/man-port-scanning-techniques.html>
-  S. McClure, J. Scambray, G. Kurtz (2012)  
Hacking Exposed 7: Network Security Secrets & Solutions - Cap. 2: Scanning  
New York: McGraw-Hill.
-  man pages de telnet , netstat , netcat y nmap .