

# Trabajo Práctico - Conexiones

## Teoría de las Comunicaciones

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

23.10.2013

# Agenda

## 1 Introducción

- Objetivos del TP
- Background: sockets

## 2 Especificación del protocolo del TP

- Características básicas
- Formato del segmento
- Establecimiento y liberación de conexión
- Ventana deslizante y retransmisiones
- Estados

## 3 Tareas a desarrollar

- Primera parte: implementación
- Segunda parte: experimentación
- Tercera parte: análisis

## 4 Anexo: implementación

# Agenda

## 1 Introducción

- Objetivos del TP
- Background: sockets

## 2 Especificación del protocolo del TP

- Características básicas
- Formato del segmento
- Establecimiento y liberación de conexión
- Ventana deslizante y retransmisiones
- Estados

## 3 Tareas a desarrollar

- Primera parte: implementación
- Segunda parte: experimentación
- Tercera parte: análisis

## 4 Anexo: implementación

# Objetivos de este trabajo

- Poner en práctica las nociones estudiadas del nivel de transporte: conexiones, control de flujo, control de errores, etc.
- Tener contacto directo con el funcionamiento e implementación de protocolos de networking.
- Continuar profundizando el enfoque analítico de las instancias anteriores.

# ¿Qué es un socket?

- Como parte de la información contextual del trabajo, veremos con un poco más de detalle de qué tratan los **sockets**.
- Un socket es en esencia un canal de comunicación entre procesos.
- Los sockets entre procesos remotos se llaman **Internet sockets**.
- Hay otras variantes, como los sockets Unix: establecen un canal de comunicación entre procesos corriendo en el mismo SO.

# Tipos de Internet sockets

- Existen varios tipos de sockets, siendo tal vez los más relevantes los siguientes:
  - ▶ **Stream sockets,**
  - ▶ **Datagram sockets,** y
  - ▶ **Raw sockets.**
- Los stream sockets se apoyan en TCP, mientras que los datagram sockets en UDP.
- De los raw sockets hablaremos en unos minutos!

# Socket API

- Los sistemas operativos suelen ofrecer una interfaz para manipular sockets: conjunto de llamadas al sistema que abstraen al usuario de las problemáticas de networking que el SO resuelve.
- La API *standard* de hoy en día es básicamente la de los Berkeley sockets.
- Esta implementación de sockets apareció en los '80 en 4.2BSD.
- Luego evolucionó (aunque no demasiado) para dar origen a los sockets POSIX.
- POSIX: conjunto de standards de la IEEE para mantener compatibilidad entre distintos SOs, por lo general basados en Unix.

# Llamadas al sistema de la API de sockets

Algunas de las llamadas al sistema más importantes:

- `socket`: crea un nuevo socket de un tipo dado, identificado por un file descriptor.
- `bind`: típicamente usada por el servidor, liga el socket a una dirección local (i.e., IP más puerto).
- `listen`: usada por el servidor, lleva el socket (TCP) al estado de LISTEN. Recibe un argumento que indica la cantidad máxima de solicitudes a encolar.



## Llamadas al sistema de la API de sockets (cont.)

- `connect`: usada por el cliente, inicia el proceso de conexión a una dirección dada. Asigna un puerto local libre al socket.
- `accept`: acepta un pedido remoto de conexión enviado por un cliente y devuelve un nuevo socket con la conexión establecida.
- `send` y `recv`: envían y reciben datos de un socket.
- `close`: finaliza la conexión y libera los recursos reservados en el SO para el socket.

## Ejemplo (rápido) en Python: conexión entre dos procesos

```
>>> import socket
>>> sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> sock1.bind(('127.0.0.1', 12345))
>>> sock1.listen(1)
>>> sock, dst_address = sock1.accept()
```

```
>>> import socket
>>> sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> sock2.connect(('127.0.0.1', 12345))
```

## Ejemplo (rápido) en Python: intercambio de datos

```
...  
>>> sock2.send('hola' * 10)  
40
```

```
...  
>>> sock.recv(10)  
'holaholaho'  
>>> sock.recv(10)  
'laholahola'  
>>> sock.recv(100)  
'holaholaholaholahola'
```

# Sockets raw

- Los **sockets raw** conforman otra variante de sockets que posee capacidades poderosas no presentes en los otros tipos mencionados.
- Con ellos, un proceso puede leer y escribir datagramas IP con un número de protocolo no procesado por el kernel del SO.
  - ▶ Este campo de 8 bits indica qué protocolo viaja dentro de IP (e.g., 6 representa a TCP).
  - ▶ Como corolario, permite diseñar e implementar protocolos de transporte ad-hoc a nivel de usuario.
- También, con un socket raw, un proceso tiene la posibilidad de construir él mismo el datagrama IP (seteando la opción `IP_HDRINCL`).
- En este trabajo práctico utilizaremos sockets raw para enviar y recibir los segmentos de nuestro protocolo.

# Sockets raw: cómo funcionan

- La creación de un socket raw requiere permisos elevados: esto prohíbe a usuarios normales la inyección de datagramas IP arbitrarios en la red.
- Al crearse, un socket raw define con qué número de protocolo va a trabajar:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, protocol)
```

- Cuando se define la opción `IP_HDRINCL`, el output de datos en el socket se espera que contenga el header IP seguido del payload arbitrario.
  - ▶ No obstante, el kernel calcula el checksum de IP.
- Observar que no es necesario invocar a `connect` con un socket raw.

# Sockets raw: qué hace el kernel al recibir datagramas

- Al recibir datagramas IP, el kernel **no** reenvía segmentos UDP ni TCP a los sockets raw.
- En cambio, reenvía los paquetes IP con número de protocolo desconocido.
  - ▶ Sólo realiza chequeos básicos sobre estos paquetes (checksum, dirección destino, versión de IP, etc.)
- En este proceso, el kernel examinará **todos** los sockets raw de **todos** los procesos.
- Una copia del datagrama será entregada a cada socket que satisfaga una serie de tests que involucran el chequeo del protocolo y de las direcciones especificadas.

# Agenda

## 1 Introducción

- Objetivos del TP
- Background: sockets

## 2 Especificación del protocolo del TP

- Características básicas
- Formato del segmento
- Establecimiento y liberación de conexión
- Ventana deslizante y retransmisiones
- Estados

## 3 Tareas a desarrollar

- Primera parte: implementación
- Segunda parte: experimentación
- Tercera parte: análisis

## 4 Anexo: implementación

# PTC: características básicas

- **Unidireccionalidad:** es un protocolo simplex en el que un cliente activo envía datos y un servidor pasivo los recibe, sin la posibilidad de enviar sus propios datos.
- **Orientación a conexión:** contempla procesos formales de establecimiento y liberación de conexión.
- **Confiabilidad:** a través de un algoritmo de ventana deslizante, garantiza que los datos enviados por el cliente lleguen correctamente a destino.



## Formato del segmento

0					15					16					31				
PUERTO ORIGEN										PUERTO DESTINO									
A	N	R	S	F	(CERO)					(CERO)									
#SEQ										#ACK									

- El encabezado es de tamaño fijo: 12 bytes.
- A es el flag de ACK; prendido sólo en los mensajes del servidor para reconocer la correcta recepción de un paquete.
- S es el flag de SYN; es enviado sólo por el cliente al requerir establecer una nueva conexión.
- F es el flag de FIN; es enviado únicamente por el cliente cuando desea finalizar una conexión existente.
- Los flags N y R actualmente no tienen uso.

## Segmentos del protocolo

- Inmediatamente después de este encabezado siguen los datos (conteniendo entre 1 y 1000 bytes).
- El número de secuencia no trabaja a nivel de byte como ocurre en TCP sino que identifica a toda la porción de los datos contenida en el paquete.
- Los paquetes de **PTC** viajan dentro de IP: para que los hosts puedan reconocer este hecho, el campo proto del header IP debe definirse con valor 202, tradicionalmente sin uso.

# Algoritmo de establecimiento de conexión

- Para iniciar el proceso de conexión, el cliente debe enviar un segmento con el flag SYN prendido y con un valor arbitrario en el campo #SEQ.
- El resto de los campos pueden tener un valor arbitrario; no serán tenidos en cuenta por el servidor.
- El servidor responderá con un segmento con el flag ACK prendido y el campo #ACK reconociendo el valor de #SEQ previamente recibido.
- Una vez hecho esto, el servidor considerará que la conexión está establecida. Lo mismo hará el cliente al recibir este reconocimiento.
- En caso de no llegar, el cliente eventualmente retransmitirá el paquete inicial.

# Finalización de conexión

- Para cerrar la conexión, la secuencia a seguir es similar, aunque cambiando el flag de SYN por el flag de FIN.
- El segmento FIN enviado por el cliente también debe ir secuenciado como cualquier otro segmento de datos previamente enviado.
- En caso de no recibir el ACK respectivo, el cliente también retransmitirá el FIN.

# Algoritmo de ventana deslizante

- **PTC** implementa ventana deslizante, particularmente GO-BACK-N.
- De esta manera, el servidor tiene una ventana de recepción de tamaño 1.
- El cliente tiene una ventana de emisión de tamaño que fijamos en un valor constante arbitrario, potencialmente mayor que 1.
- Llamaremos SEND\_WINDOW a este valor.

# Algoritmo de ventana deslizante: funcionamiento

- El cliente, entonces, podrá enviar hasta `SEND_WINDOW` paquetes en simultáneo antes de bloquearse.
- Cada `ACK` recibido permitirá desplazar la ventana y así hacer lugar para el envío de nuevos segmentos.
- Los `ACKs` del servidor pueden ser acumulativos.
- En otras palabras, un paquete `ACK` puede reconocer más de un paquete con números de secuencia contiguos enviados por el cliente.

# Retransmisiones

- Al enviar un segmento con datos, el cliente también encolará este segmento en la cola de retransmisión.
- Éste permanecerá allí hasta ser eventualmente reconocido.
- El cliente también define un tiempo máximo de espera `RETRANSMISSION_TIMEOUT` para esperar por estos reconocimientos.
- De superarse, se asumirá que el paquete se extravió y por ende será retransmitido junto con todo segmento posterior aguardando en la cola de retransmisión.

## Cantidad máxima de retransmisiones

- Se define también un número máximo admisible de retransmisiones, `MAX_RETRANSMISSION_ATTEMPTS`.
- Si algún segmento del cliente debiera ser retransmitido más veces que esta cantidad, se debe asumir que la conexión se perdió, y se pasará a cerrarla sin enviar FIN.



# Estados del cliente

Estado	Descripción	Estado anterior
CLOSED	Representa la ausencia de conexión	FIN_SENT o ninguno
SYN_SENT	El cliente desea iniciar una conexión	CLOSED
ESTABLISHED	El cliente ya puede enviar datos	SYN_SENT
FIN_SENT	El cliente desea terminar la conexión	ESTABLISHED

# Estados del servidor

Estado	Descripción	Estado anterior
CLOSED	Representa la ausencia de conexión	FIN_RECEIVED o ninguno
SYN_RECEIVED	El cliente desea iniciar una conexión	CLOSED
ESTABLISHED	El servidor está esperando recibir datos	SYN_RECEIVED
FIN_RECEIVED	El cliente desea terminar la conexión	ESTABLISHED

# Agenda

## 1 Introducción

- Objetivos del TP
- Background: sockets

## 2 Especificación del protocolo del TP

- Características básicas
- Formato del segmento
- Establecimiento y liberación de conexión
- Ventana deslizante y retransmisiones
- Estados

## 3 Tareas a desarrollar

- Primera parte: implementación
- Segunda parte: experimentación
- Tercera parte: análisis

## 4 Anexo: implementación

## Primera parte: implementación

Tomando como punto de partida el código parcial suministrado por la cátedra, se debe completar la implementación en Python del cliente del protocolo **PTC**. Puntualmente, se debe completar lo siguiente en el archivo `client.py`:

- El método `handle_incoming` de `PTCClientProtocol`, que es invocado cada vez que llega un paquete desde el servidor. El argumento `packet` indica el paquete recibido.
- El método `handle_timeout` de `PTCClientProtocol`, que es invocado siempre que el tiempo de espera del primer paquete encolado en la cola de retransmisión se agota.
- La lógica de la clase `ClientControlBlock`, que maneja las variables de la ventana deslizante del protocolo. Fundamentalmente, se debe poder verificar si es posible enviar (método `send_allowed`) y además definir métodos para determinar si un **ACK** es aceptado y para reajustar la ventana dado un **ACK** aceptado.

## Segunda parte: experimentación

Utilizando el servidor provisto por la cátedra y la implementación completa del cliente del punto anterior, realizar las siguientes pruebas en el contexto de una red local (LAN):

- Enviar archivos de distinto tamaño (e.g., 1 KB, 5 KB, 10 KB, 50 KB, 100 KB, etc.) y medir el tiempo total de transmisión de los mismos. Tomar la medida representativa como el promedio de un número  $N$  de experimentos.
- Variar el tamaño de la ventana de emisión y repetir los envíos.
- Observar en Wireshark cómo impacta lo anterior en la cantidad de retransmisiones.

## Tercera parte: análisis

A partir de los experimentos hechos, graficar lo siguiente y sacar conclusiones:

- Throughput percibido (i.e., cantidad de bits por segundo) en función del tamaño de archivo.
- Throughput en función de la ventana de emisión, para un tamaño de archivo fijo.
- Cantidad de retransmisiones en función de la ventana de emisión, para un tamaño de archivo fijo.

# Agenda

## 1 Introducción

- Objetivos del TP
- Background: sockets

## 2 Especificación del protocolo del TP

- Características básicas
- Formato del segmento
- Establecimiento y liberación de conexión
- Ventana deslizante y retransmisiones
- Estados

## 3 Tareas a desarrollar

- Primera parte: implementación
- Segunda parte: experimentación
- Tercera parte: análisis

## 4 Anexo: implementación

# Referencias



W. Richard Stevens (2003)

Unix Network Programming, Volume 1: The Sockets Networking API  
Addison-Wesley



*Beej's Guide to Network Programming*

<http://beej.us/guide/bgnet/>



A. Tanenbaum (1996)

Computer Networks, 3ra Ed. Capítulo 3: págs. 207-213.  
Prentice Hall