



Módulo 02

La Capa de Aplicaciones

(Pt. 1)



Redes de Computadoras
Depto. de Cs. e Ing. de la Comp.
Universidad Nacional del Sur



Copyright

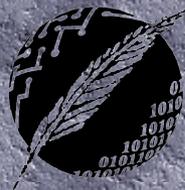
- Copyright © **2010-2024** A. G. Stankevicius
- Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la **GNU Free Documentation License**, versión 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera
- Una copia de esta licencia está siempre disponible en la página <http://www.gnu.org/copyleft/fdl.html>
- La versión transparente de este documento puede ser obtenida de la siguiente dirección:

<http://cs.uns.edu.ar/~ags/teaching>



Contenidos

- Servicios que requiere la capa de aplicaciones
- Protocolos de la capa de aplicaciones
 - HTTP
 - DNS
 - SMTP, POP e IMAP
- Arquitectura de las aplicaciones P2P
- Programación basada en sockets



ISO/OSI - TCP/IP



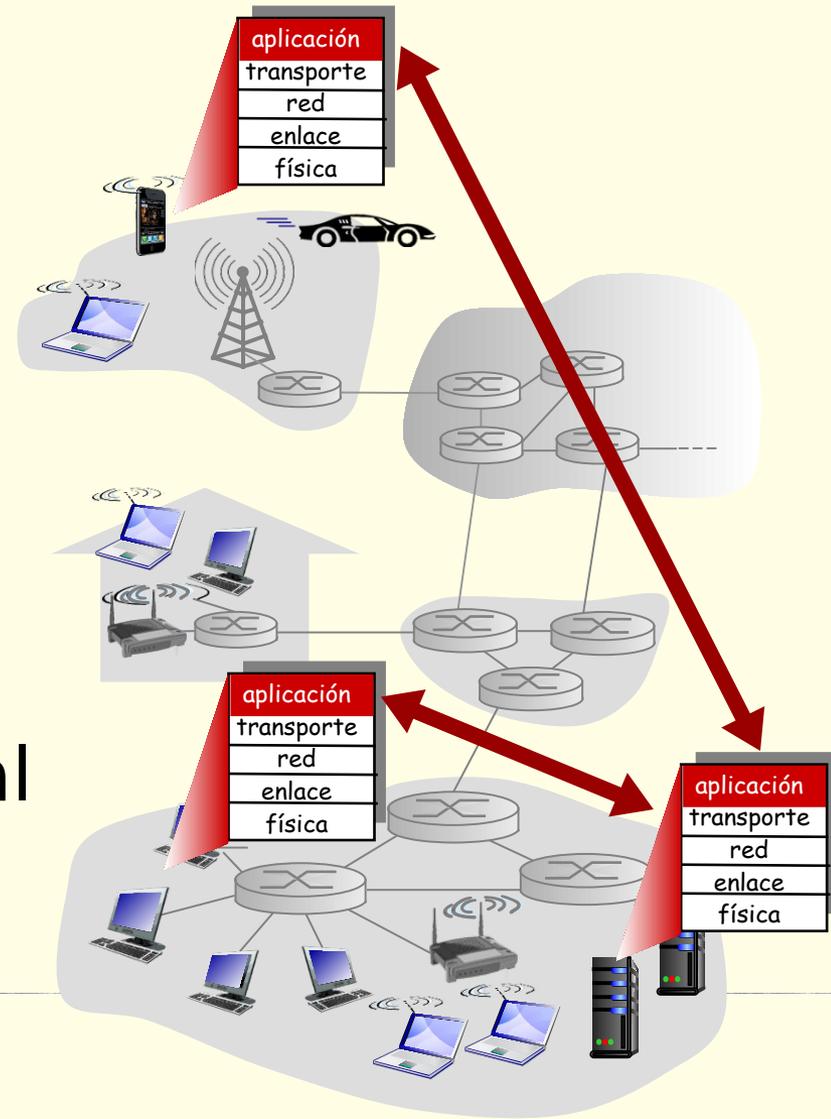
Un poco de nuestra jerga

- Denominaremos **proceso** a un programa en ejecución en una cierta computadora
- Los procesos se comunican entre sí principalmente de dos maneras:
 - ➔ Dentro de una misma computadora usando algún mecanismo de **IPC** (Inter Process Communication)
 - ➔ Entre procesos en distintas computadoras usando alguno de los **protocolos de la capa de aplicaciones**



¿Qué es una aplicación?

- Bajo esta perspectiva, ¿en qué consiste una aplicación de red?
 - ➔ Una aplicación de red es, en esencia, un conjunto de procesos distribuidos que se comunican entre sí
- La definición resulta deliberadamente universal
 - ➔ Se puede aplicar a la web, al ChatGPT, etc.



¿Qué es una aplicación?

- Tareas a cargo del programador:
 - ➔ Escribir el código que correrá en la frontera de la red posiblemente en diferentes computadoras
 - ➔ Definir los protocolos que se usarán para comunicarse a través de la red
 - ➔ No necesita preocuparse por escribir código para el núcleo de la red, puede asumir que funciona de acuerdo a su especificación
- Extraordinaria decisión de diseño: **ique la complejidad radique en la frontera de la red!**



Aplicaciones de red

● En la actualidad contamos con aplicaciones de red del más variado tipo:

- Navegador
- Correo electrónico
- Mensajería instantánea
- Transferencia de archivos
- Distribución **P2P** de archivos
- Juegos multiusuario en línea
- Desarrollo colaborativo



Aplicaciones de red

● Continúa:

- Operación remota de computadoras
- Reproducción remota de contenidos multimediales.
- Telefonía sobre **IP (VoIP)**
- Video conferencia en tiempo real
- Computación basada en la nube (cloud computing)
- Realidad aumentada y virtual
- ...
- Ah, y educación a distancia... ¡como todos padecemos!



Modelo cliente-servidor

- El modelo **cliente-servidor** permite separar las tareas en dos grupos de procesos:
 - ➔ Por un lado los procesos clientes, que son los encargados de iniciar los requerimientos a los servidores
 - ➔ Por otro lado los procesos servidores, que son los encargados de atender y responder a esos requerimientos
- El servidor **debe estar siempre a disposición** para atender nuevos requerimientos



Cliente prototípico

- El cliente tiene como responsabilidad **gestionar la interfaz** con la que el usuario final interactúa
- Tiene que **iniciar las solicitudes** a los servidores que correspondan
 - ➔ Usualmente la solicitud es producto de una acción del usuario, pero también es factible que el cliente genere solicitudes de forma autónoma
- Una vez obtenida la respuesta a una solicitud, debe **poner a disposición del usuario la información recibida** de una manera acorde



Servidor prototípico

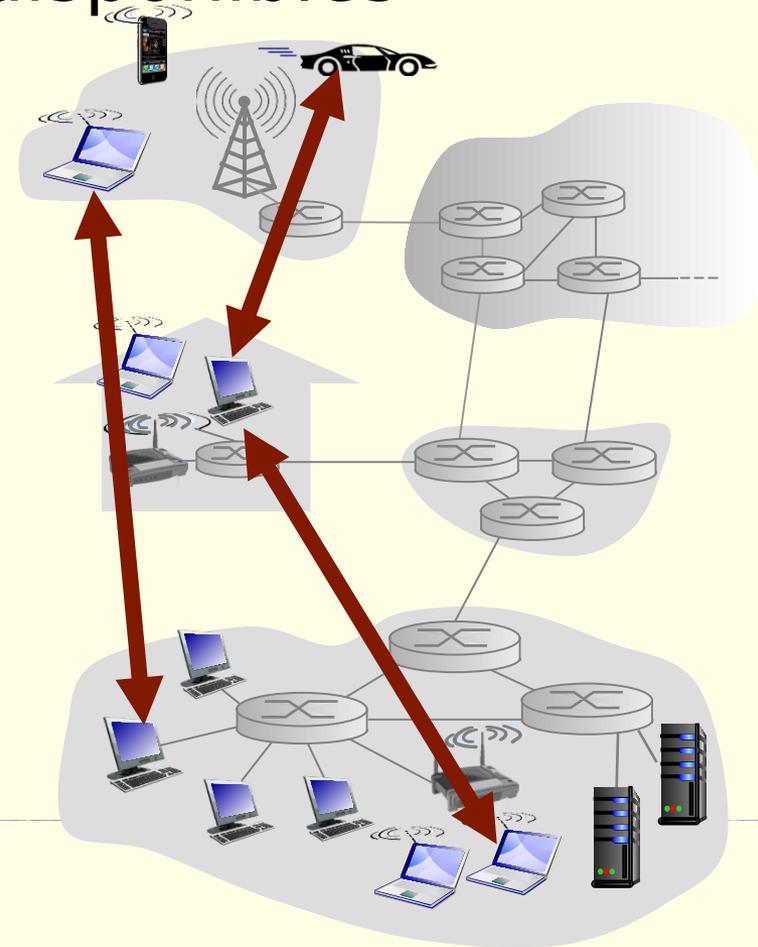
- El servidor tiene como responsabilidad **estar en todo momento a la espera de nuevas solicitudes de servicio**
 - ➔ El servidor suele ser un proceso que está corriendo sin supervisión todo el tiempo (esto es, un **daemon**)
- Al recibir cada nueva solicitud debe **generar una respuesta acorde y suministrar esta respuesta al cliente correspondiente**
 - ➔ Hay muchas maneras de optimizar este aspecto, es acá donde vale la pena focalizar nuestro esfuerzo



Modelo par-a-par puro

● Bajo el modelo **par-a-par puro**, los servidores no requieren estar siempre disponibles

- Los pares se conectan entre sí intermitentemente y de manera directa
- También pueden cambiar su dirección sin previo aviso
- **Resulta altamente escalable**, si bien al mismo tiempo es **más complicado de gestionar**



A lo suyo ChatGPT...

AG

You

contrastame los estilos arquitectónicos cliente-servidor y par-a-par, señalando sus fortalezas y debilidades

AG

ChatGPT

Por supuesto, aquí tienes una comparación entre los estilos arquitectónicos cliente-servidor y par-a-par (P2P), destacando sus fortalezas y debilidades:

Cliente-Servidor:

• Fortalezas:

1. **Centralización del control:** En un modelo cliente-servidor, el servidor actúa como un punto centralizado de control y gestión, lo que facilita la administración y la implementación de políticas de seguridad y acceso.
2. **Escalabilidad vertical:** Los servidores pueden actualizarse verticalmente (aumentar la potencia de procesamiento, memoria, etc.) para manejar un mayor número de clientes, lo que facilita la escalabilidad en entornos controlados.
3. **Mejor control de recursos:** Los servidores pueden gestionar y distribuir los recursos de manera más eficiente, garantizando un uso equitativo y optimizado de los recursos disponibles.
4. **Seguridad centralizada:** Al centralizar la gestión de usuarios y recursos en el servidor, es más fácil aplicar medidas de seguridad centralizadas y controlar el acceso a los datos y servicios.

• Debilidades:

1. **Punto único de fallo:** Si el servidor experimenta una interrupción, toda la red o servicio puede volverse inaccesible para los clientes.
2. **Cuellos de botella de rendimiento:** El servidor puede convertirse en un cuello de botella si tiene dificultades para manejar el tráfico de múltiples clientes simultáneamente.
3. **Costo y complejidad:** La implementación y mantenimiento de servidores robustos puede ser costosa y requerir una planificación y administración detalladas.
4. **Escalabilidad limitada:** La escalabilidad en un entorno cliente-servidor puede ser limitada por la capacidad del servidor para manejar un número creciente de clientes.

Par-a-Par (P2P):

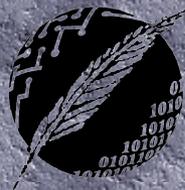
• Fortalezas:

1. **Descentralización:** En un entorno P2P, no hay un punto centralizado de control, lo que proporciona una mayor resiliencia y redundancia frente a fallos individuales.
2. **Escalabilidad horizontal:** La escalabilidad en un entorno P2P es más fácil de lograr ya que cada nodo puede contribuir con recursos y capacidad de procesamiento a la red, permitiendo un crecimiento más orgánico y distribuido.
3. **Menos dependencia de la infraestructura:** Al no depender de servidores centralizados, las redes P2P pueden ser más flexibles y adaptativas, lo que las hace ideales para entornos donde la infraestructura puede ser limitada o inestable.
4. **Distribución de costos:** La carga de la red se distribuye entre los nodos, lo que puede reducir los costos de infraestructura en comparación con un modelo cliente-servidor.

• Debilidades:

1. **Seguridad y confiabilidad:** La descentralización puede dificultar la aplicación de medidas de seguridad coherentes y el control de acceso a recursos y datos.
2. **Rendimiento variable:** La velocidad y el rendimiento pueden variar dependiendo de la disponibilidad y capacidad de los nodos individuales en la red P2P.
3. **Gestión de recursos:** La gestión de recursos en un entorno P2P puede ser más compleja debido a la variabilidad de los nodos y la falta de una autoridad central.
4. **Potencial para abuso:** La descentralización también puede facilitar actividades no deseadas, como el intercambio ilegal de archivos o el uso indebido de recursos de red.

En resumen, ambos estilos arquitectónicos tienen sus propias fortalezas y debilidades, y la elección entre ellos depende de los requisitos específicos y las características de la aplicación o sistema que se esté desarrollando. Mientras que el modelo cliente-servidor ofrece centralización y control, el modelo P2P proporciona descentralización y escalabilidad horizontal.



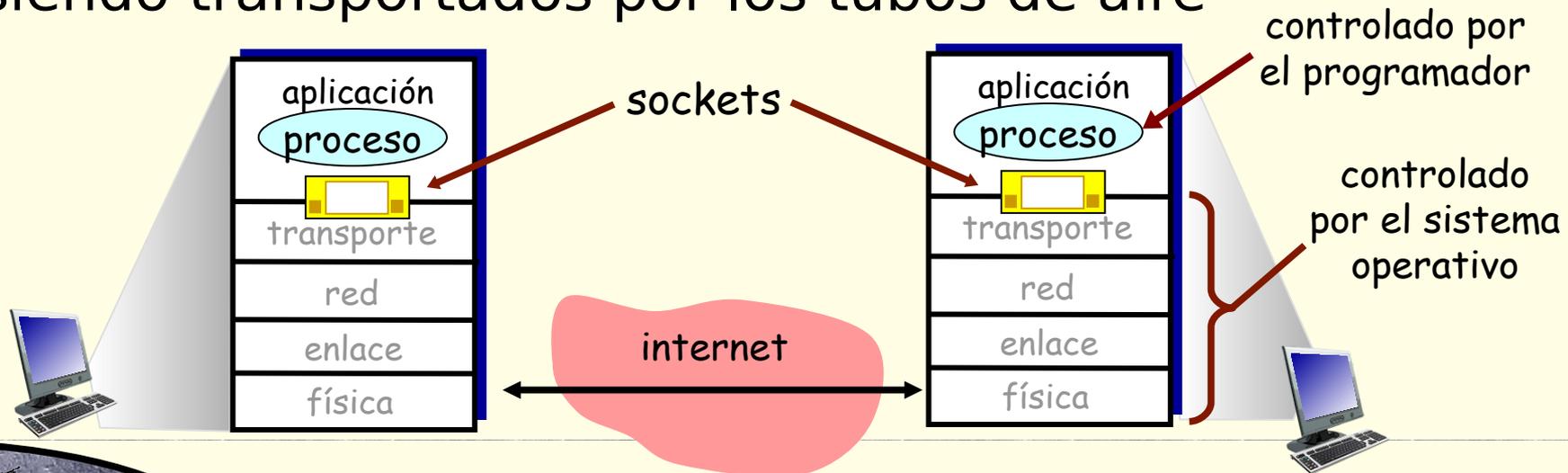
Modelo híbrido

- En ocasiones se logra el mejor desempeño siguiendo un **modelo híbrido**
- Por caso, las aplicaciones de telefonía **VoIP**:
 - ➔ Las llamadas entre dos usuarios se realizan de manera directa entre sí (es decir, de manera **P2P**)
 - ➔ Existe un servidor centralizado que registra cuáles son y dónde están los usuarios en línea en ese momento
 - ➔ La aplicación consulta al servidor como paso previo a iniciar una llamada con otro usuario



La interfaz sockets

- Los procesos se comunican enviando y recibiendo información por un **socket**
- El socket funciona como un sistema de mensajería neumático, podemos mandar y recibir sin realmente saber qué pasa con los mensajes mientras están siendo transportados por los tubos de aire



Espacio de nombres de la red

- Para que un proceso sea capaz de recibir un mensaje debe contar con un identificador unívoco a lo ancho de toda la red
 - En el espacio de nombres adoptado cada computadora cuenta con **una dirección IP propia**
- ¿Será suficiente con poder identificar cada una de las computadoras de la red?
 - Para poder distinguir a los distintos procesos dentro de una determinada computadora también se debe suministrar un número de **puerto** (port)



El rol de los protocolos

- Los **protocolos de la capa de aplicaciones** cumplen un rol central en las aplicaciones:
 - ➔ Su implementación constituyen una parte integral de la aplicación de red (la otra parte es la **GUI**)
 - ➔ Se caracteriza definiendo los mensajes que han de ser intercambiado así como las acciones que se deben tomar al recibir dichos mensajes
 - ➔ La comunicación entre procesos es provista como servicio por la capa inmediata inferior y es implementada a través de sus protocolos



Definición de un protocolo

- Para definir un protocolo hace falta especificar diversos aspectos:
 - Los **tipos de mensajes** a ser intercambiados
 - La sintaxis **de estos mensajes** (esto es, definir sus campos y explicitar cómo se delimitan)
 - La **semántica** de la información contenida en los **campos de los mensajes** contemplados
 - La información acerca del **secuenciamiento de los mensajes** (esto es, cuándo y cómo deben interactuar los procesos que implementen el protocolo)



Definición de un protocolo

- La definición formal de un protocolo puede ser de acceso **público** o **privado**
- Si es de acceso público, nos aseguramos una **amplia difusión** y una **gran compatibilidad** entre las distintas implementaciones
 - ➔ Se suelen definir dentro de un documento técnico denominado **RFC** (Request For Comments)
- Si en cambio son privados, nos aseguramos el monopolio **excluyendo a la competencia**
 - ➔ Al menos por un tiempo (hasta la ingeniería reversa)



Requerimientos de transporte

- Las aplicaciones tienen diversos requerimientos de transporte de datos:
 - Algunas necesitan asegurar la **integridad de los datos transferidos** (web, chat, etc.). Otras son en cambio tolerantes a las pérdidas (streaming de audio, etc.)
 - Algunas requieren **baja latencia** (juegos online o telefonía sobre internet), mientras que otras no se ven tan afectadas por los retardos
 - Algunas sólo funcionan si cuentan con un dado **ancho de banda** a su disposición, mientras que otras aceptan lo mucho o lo poco que se tenga a disposición



Requerimientos de transporte

Tipo de Aplicación	Integridad de los datos	Ancho de banda	Tolerante a retardos
transferencia de archivos	requerido	elástico	si
envío y recepción de email	requerido	elástico	si
navegar la web	requerido	elástico	si
audio/video en tiempo real	no requerido	no elástico	no (muy exigente)
audio/video almacenado	no requerido	no elástico	no (menos exigente)
juegos en línea	no requerido	no elástico	no (muy exigente)
mensajería instantánea	requerido	elástico	más o menos
consultas al servidor DNS	no requerido	elástico	si
terminal remota segura	requerido	elástico	si



Protocolos de transporte

- La capa de transporte brinda dos servicios:
 - ➔ **TCP**: Un servicio de transporte orientado a la conexión, seguro, confiable, que implementa control de flujo y gestión de congestiones, pero que no da garantías acerca del ancho de banda ni de la latencia
 - ➔ **UDP**: Un servicio de transporte no orientado a la conexión, que no asegura la integridad de los datos ni implementa control de flujo, y tampoco da garantías acerca del ancho de banda ni de la latencia
- Internet es una red estilo “**best effort**”
 - ➔ No da garantías de latencia ni de ancho de banda



Protocolos de transporte

Tipo de Aplicación	Protocolo	Definición Formal	Protocolo de Transporte
transferencia de archivos	FTP	RFC 959	TCP
envío y recepción de email	SMTP	RFC 5321	TCP
	POP3	RFC 1939	TCP
	IMAP	RFC 9051	TCP
navegar la web	HTTP	RFC 7540	TCP
audio/video en tiempo real	Skype	privado	TCP y UDP
audio/video almacenado	RTMP	privado	TCP (usa HTTP)
juegos en línea	battle.net	privado	TCP y UDP
mensajería instantánea	MSN	privado	TCP
consultas al servidor DNS	DNS	RFC 1034/5	UDP
terminal remota segura	SSH	RFC 4250/6	TCP



TCP (más) seguro

- Cabe destacar que ni **TCP** ni **UDP** encriptan la información transportada
 - Es decir, las contraseñas viajan por la red a la vista de todos los intermediarios
- Recordemos que en el comienzo de internet la seguridad no era una preocupación
 - Sin duda, llegado el caso una aplicación podría encargarse de encriptar y de desencriptar la información antes de encauzarla a través del socket



TCP (más) seguro

- En la actualidad, tenemos a disposición la librería **SSL** (Secure Socket Layer)
 - Esta librería brinda conexiones **TCP** encriptadas, asegurando la integridad de los datos y también posibilita autenticar a los interlocutores
 - **SSL perfecciona** y **extiende** el servicio básico provisto por el protocolo **TCP** permitiendo que la información sensible viaje protegida a través del núcleo de la red
- Retomaremos este aspecto crucial más adelante, en último módulo de la materia



¿Preguntas?

