

# Introducción a las Redes de Computadoras

Redes y Comunicaciones

- Preguntar:
  - Genéricamente: Qué es una Red ?
  - Aunque seguramente no surja naturalmente la pregunta, sino la cuestión sería más específica:
    - Qué es Internet ?
    - Qué es la WEB ?
    - Qué pasa cuando navego por Internet, Accedo a Facebook, o miro un video en Youtube ?
- Optaremos por un enfoque de lo general a lo particular preguntándonos: **Qué es una red de computadoras / ordenadores?**

# Qué es una red de computadoras/ordenadores ?

- Análisis del punto de vista sistémico:
- **Red de Computadoras:** un grupo de computadoras/dispositivos interconectados.
- **Objetivo:** compartir recursos: dispositivos, información, servicios.
- El conjunto **computadoras, software de red, medios y dispositivos de interconexión** forma un sistema de comunicación.
- Ejemplos: red de la sala de PCs, red Universitaria, Internet.

# Componentes de un Sistema de Comunicación

- Fuente (Software).
- Emisor/Transmisor (Hardware).
- Medio de transmisión y dispositivos intermedios (Hardware).
- Procesos intermedios que tratan la información (Software y Hardware).
- Receptor (Hardware).
- Destino (Software).
- Otros: Protocolos (Software), Información, mensaje transmitido (Software).
- Señal de Información, materialización del mensaje sobre el medio (Hardware?).

# Componentes de un Sistema de Comunicación

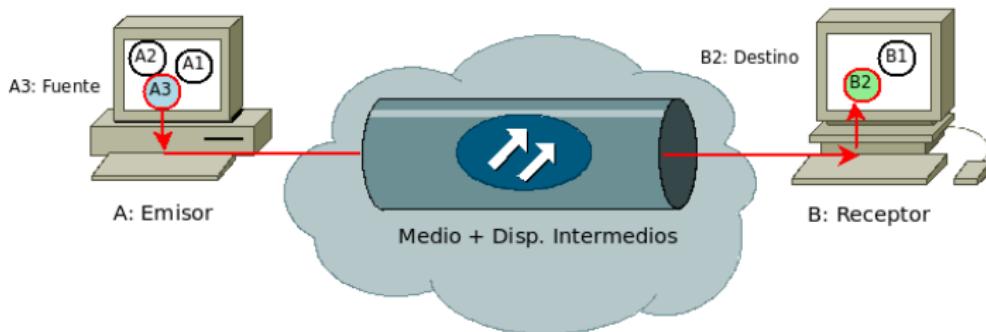


Figure: Sistema de Comunicaciones.

# Componentes de una Red

- Fuera del punto de vista sistémico podemos ver un gran numero de componentes:
  - Computadoras, en el modelo de Internet: Hosts (PCs, laptops, servidores).
  - Routers/switches, Gateways, AP (Access Points).
  - NIC (placas de red), Modems.
  - Vínculos/ enlaces: conformados por:
    - Medios: cables, fibras ópticas, señales electromagnéticas, antenas, interfaces, etc.
  - Programas: Browsers, Servidores Web, Clientes de Mail, Servidores de Streaming.
  - Etc...
- Las componentes de la red deben interactuar y combinarse a través de reglas.

# Protocolos

**Protocolo:** El conjunto de conductas y normas a conocer, respetar y cumplir no sólo en el medio oficial ya establecido, sino también en el medio social, laboral, etc.

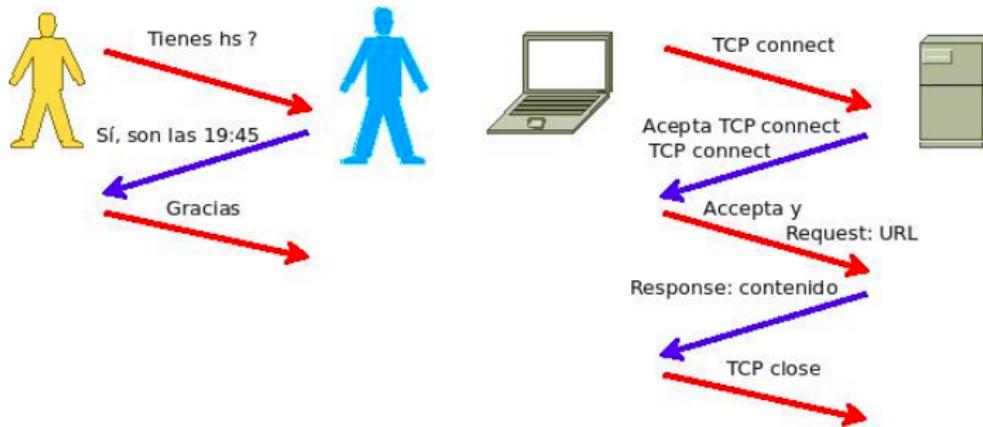


Figure: Protocolo

**Protocolo:** Un protocolo define el formato, el orden de los mensajes intercambiados y las acciones que se llevan a cabo en la transmisión y/o recepción de un mensaje u otro evento.

**Protocolo de Red:** conjunto de reglas que especifican el intercambio de datos u órdenes durante la comunicación entre las entidades que forman parte de una red. Permiten la comunicación y están implementados en las componentes.

- A principios de los 80', las compañías comenzaron a implementar redes propias (privadas y cerradas).
- Primeras **Redes Propietarias**.
- Consecuencia: Cada red tenía sus especificaciones propias (protocolos).
- Resultados: Incompatibilidad. La comunicación entre redes era muy difícil, evolución más lenta, carencia de estándares.
- Complejidad de modelos.

- La cantidad de componentes de red a interactuar genera complejidad, se requiere una organización de las mismas.
- Se requieren **Modelos de Organización**.
- Modelo en Capas: **Layering**, divide la complejidad en componentes reusables.
  - Reduce complejidad en componente más pequeñas.
  - Las capas de abajo **ocultan la complejidad** a las de arriba, **abstracción**.
  - Las capas de arriba **utilizan servicios** de las de abajo: **Interfaces**, similar a APIs.
  - Los **cambios en una capa no deberían afectar a las demás** si la interfaz se mantiene.
  - Facilita el desarrollo, evolución de las componentes de red asegurando interoperabilidad.
  - Facilita aprendizaje, diseño y administración de las redes.

# Modelo OSI (Open System Interconnection)

- Necesidad de desarrollar componentes estándares de red.
- Resultado: La ISO (International Standard Org.) crea el modelo OSI en 1984.
- Basado en los modelos de red (en capas):
  - DECNET (Digital).
  - SNA (IBM).
  - TCP/IP (DoD USA - Dept. of Defense USA).
- Modelo abierto y estándar.
- Modelo dividido en 7 (siete) capas.
- Modelo de Referencia.

# Modelo OSI (Cont'd)

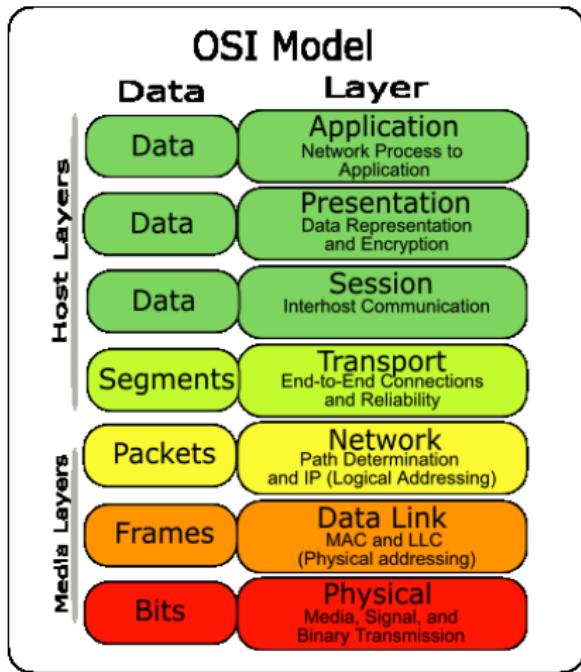


Figure: Modelo OSI

# Modelo OSI (Cont'd)

- Modelo en Capas: capa ofrece servicios a la capa superior, usa servicios de la capa inferior, mediante interfaz.

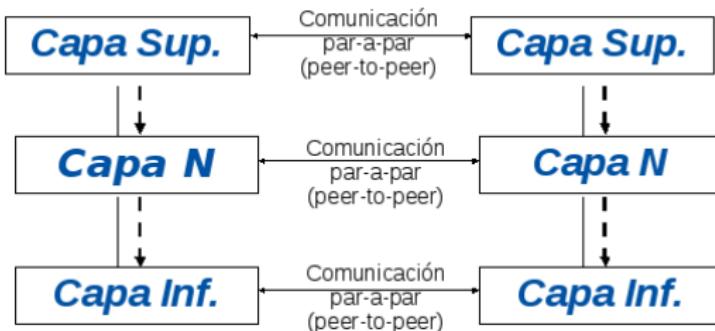


Figure: Comunicación en capas

**Capas de Host (Host layers):** 7,6,5,4, proveen envío de datos de forma confiable.

**Capas de Medio (Media layers):** 3,2,1, controlan el envío físico de los mensajes sobre la red.

# Funcionalidad por Capa (OSI)

Aplicación (7): servicios de red a los usuarios y a procesos, aplicaciones.

Presentación/Representación (6): formato de los datos.

Sesión (5): mantener track de sesiones de la aplicación.

Transporte (4): establecer y mantener canal “seguro” end-to-end (applic-to-applic).

Red (3): direccionar y rutear los mensajes host-to-host.  
Comunicar varias redes.

Enlace de Datos (2): comunicación entre entes directamente conectados. Comunicar una misma red. Acceso al Medio.

Física (1): transportar la información como señal por el medio físico. Características físicas. Información binaria, digital.

# Ejemplos de Implementaciones por Capa (OSI)

Aplicación (7): Telnet, HTTP, DNS, FTP, DNS, NCP, NDS, X.400.

Presentación/Representación (6): Postscript, JPEG, PNG, TIFF, MPEG, ZIP, XDR, ASN, HTML, CharSets(ASCII, ISO-8859-1, UTF-8, EBDIC).

Sesión (5): RPC de NFS, SQL, NetBIOS.

Transporte (4): TCP, UDP, SPX, ISO-TP.

Red (3): IP, ICMP, OSPF, IPX, CLNP, IS-IS.

Enlace de Datos (2): Ethernet, 802.3, 802.11, PPP, HDLC.

Física (1): RJ-45, EIA/TIA-568C, V.24, V.35, G.703(TEL), G.652(FO) RS-232, DOCSIS(Coax).

- Modelo que se convirtió en estándar.
- Qué protocolos se encuentran en Internet ?
  - Modelo Abierto.
  - Varios protocolos de nivel de enlace: Ethernet, 802.3, PPP, HDLC, Frame-Relay, 802.11a/b/g/n/ac, MPLS, DOCSIS, GPON, etc (No definidos por TCP/IP).
  - Protocolos propios de Internet y Transporte (Núcleo): ARP, IP, ICMP, TCP, UDP, OSPF, BGP, etc.
  - Protocolos de Aplicaciones: DNS, HTTP, FTP, SSH, SMTP, IMAP, etc.
  - API abierta para generar nuevos protocolos.

- Modelo de 5 (cinco) capas:
  - Capa de Aplicación (Process/Application).
  - Capa de Transporte o Host-to-Host.
  - Capa de Internet o Internetworking.
  - Capa de Enlace(Link Layer).
  - Capa de Física.
- Por simplicidad algunos autores hablan de 4 capas, agrupando a la Capa de Enlace y Capa física en una sola capa que llaman Capa de acceso a la Red.

# Modelo TCP/IP (Cont'd)

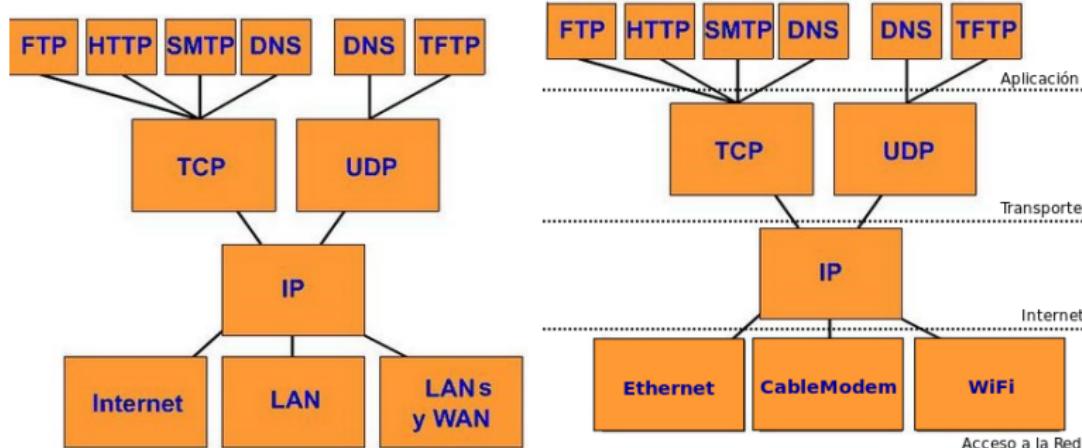


Figure: Modelo TCP/IP

# Comparación: OSI vs. TCP/IP

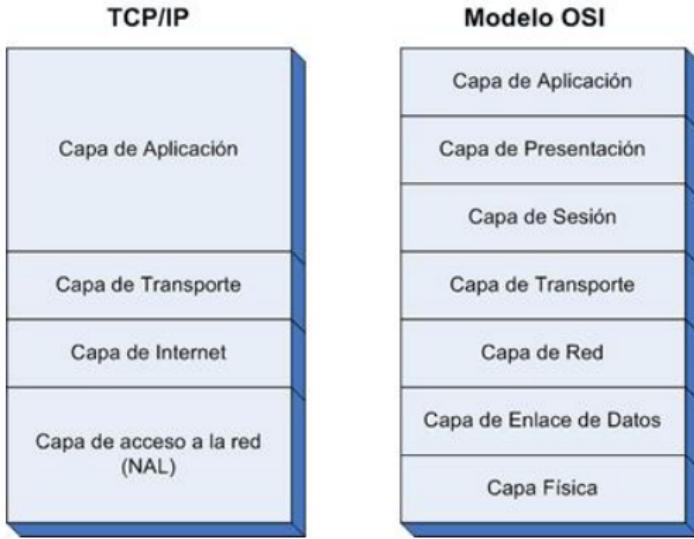


Figure: Modelo OSI vs TCP/IP

- Similitudes:

- Ambos se dividen en capas.
- Ambos tienen capas de aplicación, aunque incluyen servicios distintos.
- Ambos tienen capas de transporte similares.
- Ambos tienen capa de red similar pero con distinto nombre.
- Se supone que la tecnología es de conmutación de paquetes (no de conmutación de circuitos).
- Es importante conocer ambos modelos.

# Comparación: OSI vs. TCP/IP

- Diferencias:

- TCP/IP combina las funciones de la capa de presentación y de sesión en la capa de aplicación.
- TCP/IP combina la capas de enlace de datos y la capa física del modelo OSI en una sola capa.
- TCP/IP más simple porque tiene menos capas.
- Los protocolos TCP/IP son los estándares en torno a los cuales se desarrolló Internet, de modo que la credibilidad del modelo TCP/IP se debe en gran parte a sus protocolos.
- El modelo OSI es un modelo “más” de referencia, teórico, aunque hay implementaciones.

# Encapsulamiento

- Cada capa define su PDU: Protocol Data Unit.

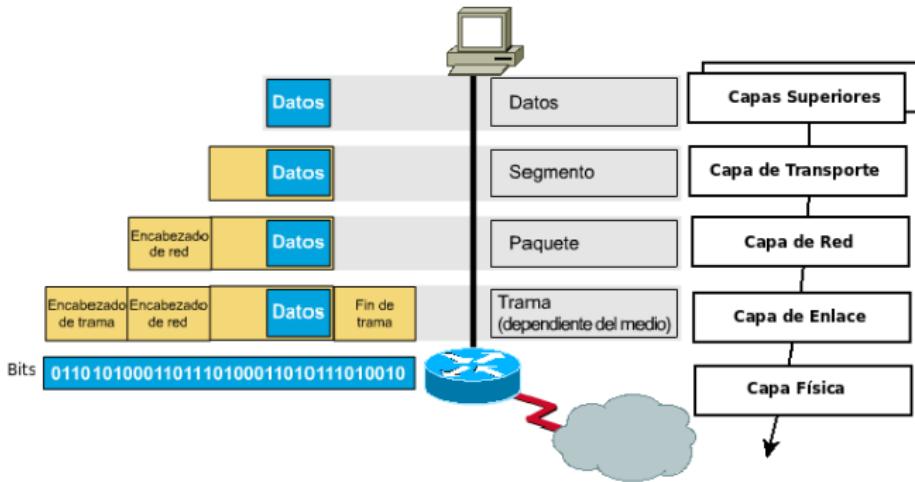


Figure: Encapsulamiento y PDUs

# Dispositivos y Capas

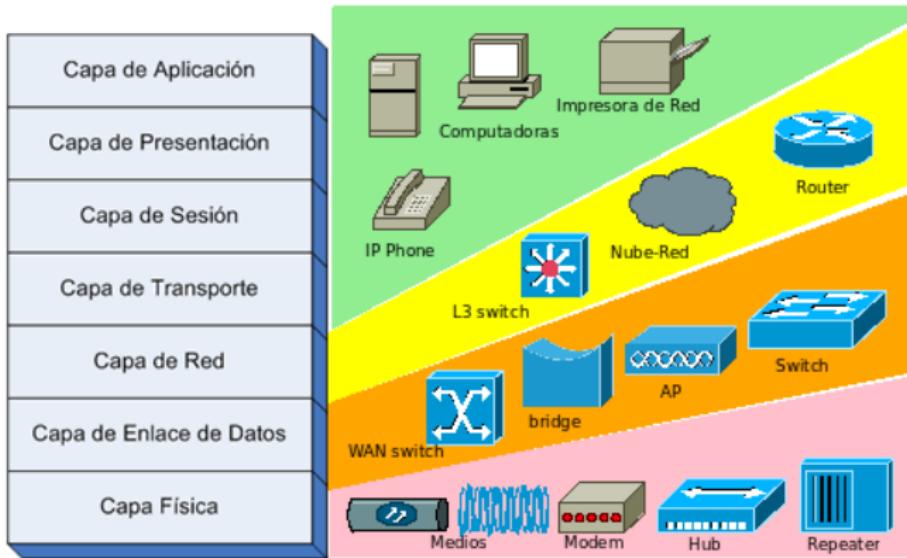


Figure: Dispositivos y Capas

# Comunicación entre Capas Peer-Peer

- Cada capa usa el servicio de la de abajo.
- Cada capa se comunica con la capa del otro extremo.

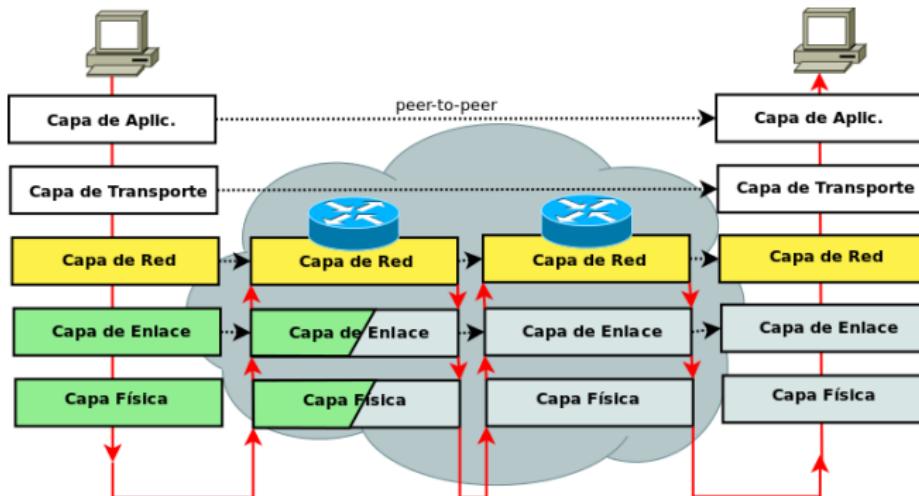


Figure: Comunicación Peer-Peer

# Comunicación entre Capas Peer-Peer(Cont'd)

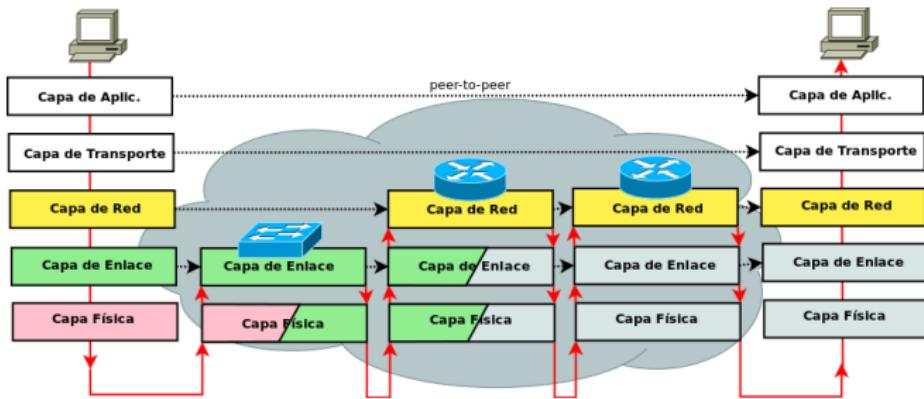


Figure: Comunicación Peer-Peer

- Diferentes clasificaciones de acuerdo a diferentes aspectos.
- Se pueden mencionar:
  - Clasificación por cobertura, distancia, alcance.
  - Clasificación por acceso abierto o privado.
  - Clasificación por topología física.
  - Clasificación por tipo de conexión/medio.
  - Etc.

# Clasificación por Cobertura

**LAN:** (Local Area Network). Red de cobertura local.  
Ethernet, Wi-Fi.

**MAN:** (Metropolitan Area Network). red de cobertura metropolitana, dentro de una ciudad. MetroEthernet, MPLS, Wi-Max.

**WAN:** (Wide Area Network). red de cobertura de área amplia. Geográficamente distribuida. PPP, Frame-Relay, MPLS, HDLC, SONET/SDH.

**SAN:** (Storage Area Network). red de almacenamiento. iSCSI, Fibre Channel, ESCON.

**PAN:** red de cobertura personal. Red con alcance de escasos metros para conectar dispositivos cercanos a un individuo. Bluetooth, IrDA, USB.

**Otros términos:** CAN (Controller Area Network o Campus Area Network), NAN (Near-me AN, NFC), ...

Internet: red pùblica global, tecnología TCP/IP.

Intranet: red privada que utiliza la tecnología de Internet.

Extranet: red privada virtualizada sobre enlaces WAN: Internet.

Intranet con acceso de usuarios remotos. VPN

(Virtual Private Network) IPSec, PPTP, SSL,

OpenVPN, L2TP. Una intranet mapeada sobre una  
red pùblica como Internet.

# Clasificación Física de Redes

- Redes de Conmutación de Circuitos.
- Redes de Conmutación de Tramas/Paquetes.
  - Servicios Orientados a Conexión. Circuitos Virtuales.
  - Servicios NO Orientados a Conexión. Datagramas.

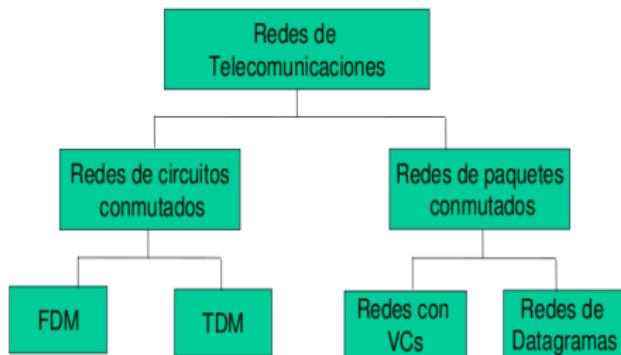


Figure: Clasificación de Redes

- Es una **red de redes de computadoras**, descentralizada, *pública*, que ejecutan el *conjunto abierto de protocolos* (suite) **TCP/IP**. Integra diferentes protocolos de un nivel más bajo:

## INTERNETWORKING

# Modelo de Internet

- Modelo de forma de reloj de arena (hourglass):

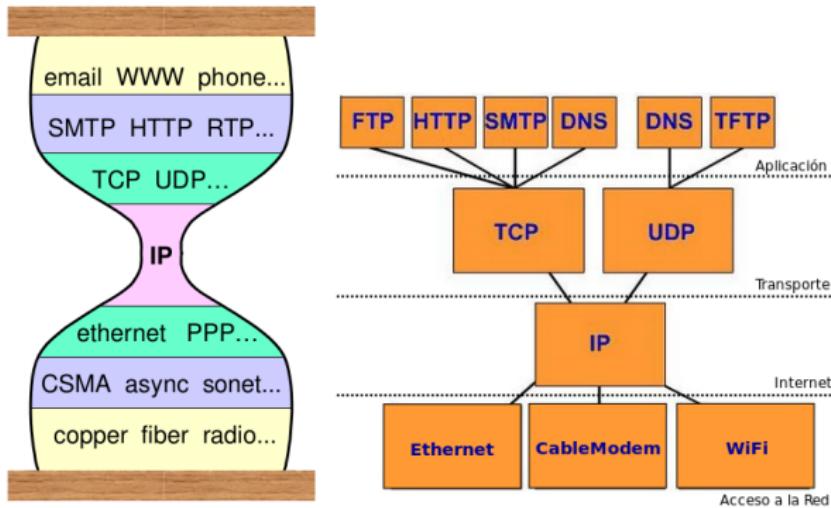
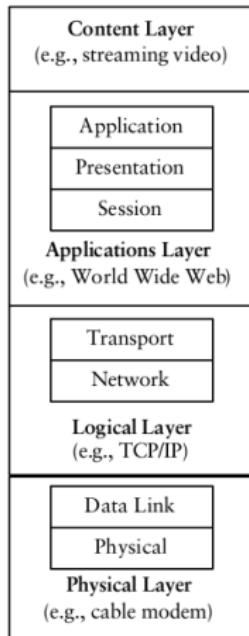


Figure: Modelo hourglass.

# Modelo Simplificado de Internet



**Figure:** Modelo Simplificado de Internet de 4 capas

# Qué es Internet ? (Cont'd)

Qué computadoras: PCs, Mainframes, Celulares, Laptops,  
Handhelds, Supercomputadoras, autos, heladeras, etc

...

Computadoras Especiales: routers y switches (sucesores de IMPs  
-Interface Message Processors-).

Qué medios: cobre, fibra óptica, wireless, satélites, etc.

Qué información: de todo !!!!! (de forma digital).

# Objetivos/Historia de Internet

**Inicios de 1960':** Red militar para la guerra fría ?? (aún no existía TCP/IP). Packet Switching Theory: paper de Kleinrock, usar paquetes en lugar de circuitos en 1960. ARPANET: RAND Corp, Leonard Kleinrock del MIT trabajan sobre la red, BBN implementa IMPs.

**Primera vez On-Line 1969:** Conectaba las Universidades: Stanford (SRI), Utah, UCLA, UCSB (UC Santa Barbara).

**Nuevo protocolo LAN 1973:** Ethernet, Bob Metcalfe en Xerox PARC.

**Cambio a TCP/IP 1983:** Desde NCP a TCP/IP. Vinton Cerf y Robert E. Kahn.

**Luego, NSFNET 1985:** Red Científica e Investigación, Usada por las Universidades.

**Continuando 1988:** Comienza como negocio, nuevas oportunidades.

**Hoy 2020:** Tele-trabajo, *Clases Virtuales*, Redes Sociales,



- Categoría **STANDARD TRACK**. RFC maturity levels.
  - Proposed Standard:** no se requiere implementaciones. Se asigna RFCnnnn.
  - Internet Standard (STD):** existen implementaciones y significante experiencia operacional. Se retiene el RFCnnnn y se agrega STDxxxx.

- Otras Categorías: “**Off-track**” .

**INFORMAL/EXPERIMENTAL**: otro proceso, se publica como Internet Draft, pero se coloca en otra Cat.

**BCP (Best Current Practices)**: otro proceso.

**HISTORIC (STD obsoletas)**: las RFCs se van actualizando o se pueden declarar obsoletas por otras.

**FYI: (For Your Information)**: como INFORMATIONAL.

# RFC (Request for Comments) (Cont'd)

- Algunos ejemplos: <http://www.rfc-editor.org/rfcxx00.html>.
  - RFC 791: IP, STD 5. 1981.
  - RFC 792: ICMP. STD 5. 1981.
  - RFC 793: TCP, STD 7. 1981.
  - RFC 768: UDP, STD 6. 1980.
  - RFC 854: TELNET, STD 8. 1983.
  - RFC 1035: DNS, STD 13. 1987.
  - RFC 2460 (desde 1998), luego 8200: IPv6, STD 86. 2017.
  - RFC 1945: HTTP 1.0, Informational. 1996.
  - RFC 2616: HTTP 1.1, aun Std Track. 1999.
  - RFC 5735, BCP 153: Special Use IPv4 Addresses. 2010.
  - RFC 5721, Experimental: POP3 Support for UTF-8. 2010.
  - RFC 1149, Experimental: Standard for the transmission of IP datagrams on avian carriers. 1 April 1990. April Fools' Day.
  - RFC 1267, Historic: BGP Border Gateway Protocol 3, 1991, obsoleta por RFC 4271: BGP-4.
  - RFC 1983, (Informational) FYI 18: Internet Users' Glossary. 1996.

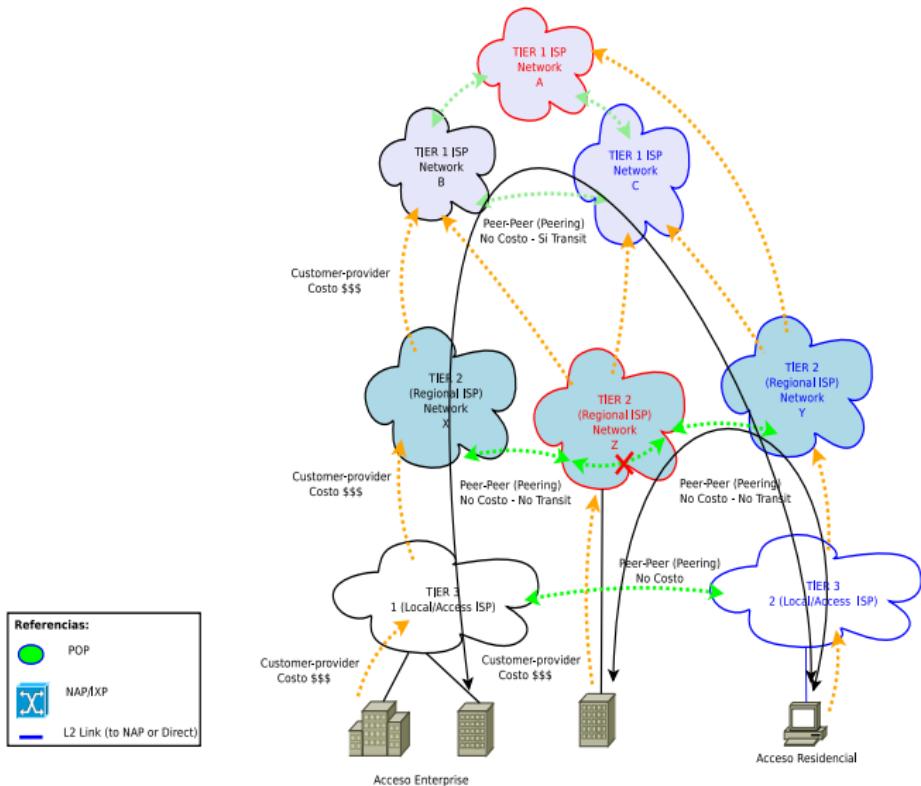
- Estructura en Jerárquica, en Tiers.
- **Capa de Acceso (Edge):** Acceso Residenciales, Acceso de Organizaciones.
- **Capa de núcleo (Core):** dividida en diferentes niveles.
  - Proveedores Regionales (Regional ISPs).
  - Proveedores Nacionales.
  - Proveedores Internacionales.
  - Proveedores Internacionales en el Tier 1.

- Diferentes tecnologías de última milla (acceso) y de redes locales.
- Despliegue local y de acceso de las personas y las organizaciones.
- Ver capa física y enlace modelo OSI.
- Últimos eslabones en la jerarquía.

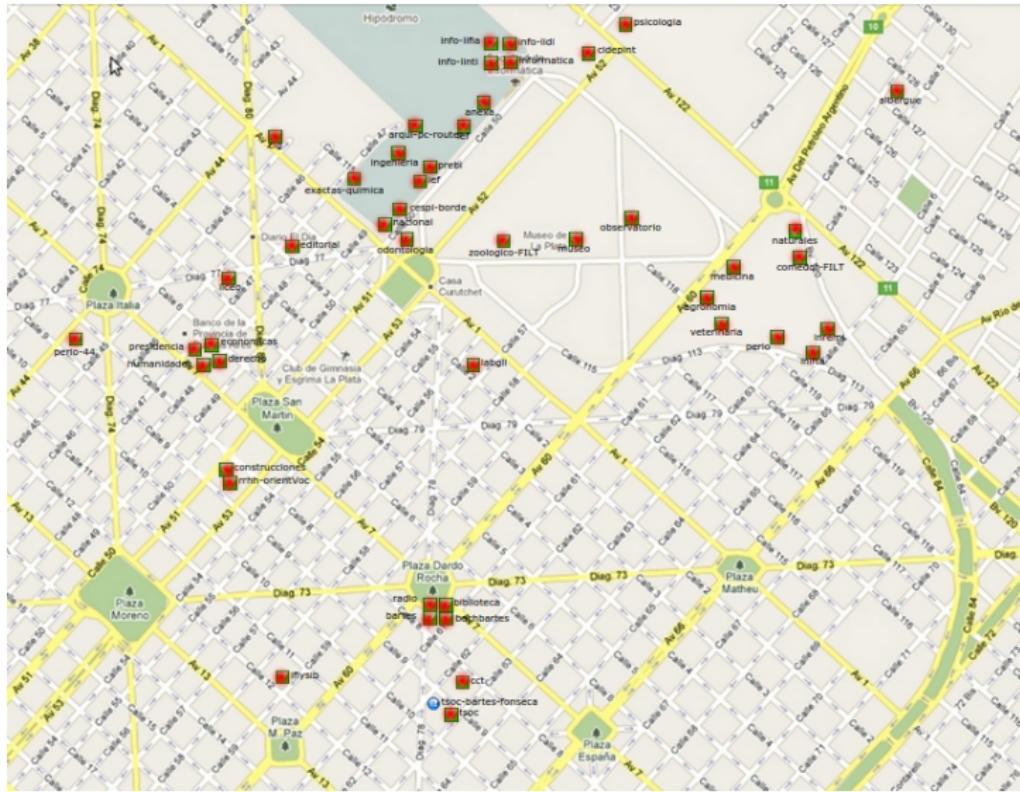
# Estructura de Internet (Core)

- Tecnologías de fibra óptica, Cobre, Satélites.
- Se interconectan mediante POPs (Point Of Presence) con Proveedores.
- Entre proveedores se interconectan mediante NAPs (Network Access Point) o conexiones Directas.
- Actualmente los NAPs se los llama IXPs (Internet Exchange Point)

# Estructura de Internet (Core Cont'd)



# Ejemplo de la Cobertura de la UNLP



- Concepto de protocolo y modelos en capas.
- Modelo OSI, Modelo TCP/IP, PDUs.
- Comunicación peer-to-peer.
- Clasificación de redes.
- Modelo TCP/IP en modelo de Internet.
- Estructura de Internet.
- Estándares de Internet RFCs.

- Kurose/Ross: Computer Networking (6th Ed).
- Andrew S. Tanenbaum. Computer Networks (5th Edition).
- Willam Stallings. Data & Computer Communications (8th Edition).
- Wikipedia <http://www.wikipedia.org>.
- <http://www.rfc-editor.org/overview.html>.
- <ftp://ftp.rfc-editor.org/in-notes/rfc2026.txt>.
- <http://www.isoc.org/internet/history/brief.shtml>.
- <http://isoc.org/wp/ietfjournal/?p=454>.
- Internet ...

# Introducción a Capa de Aplicación

## Redes y Comunicaciones

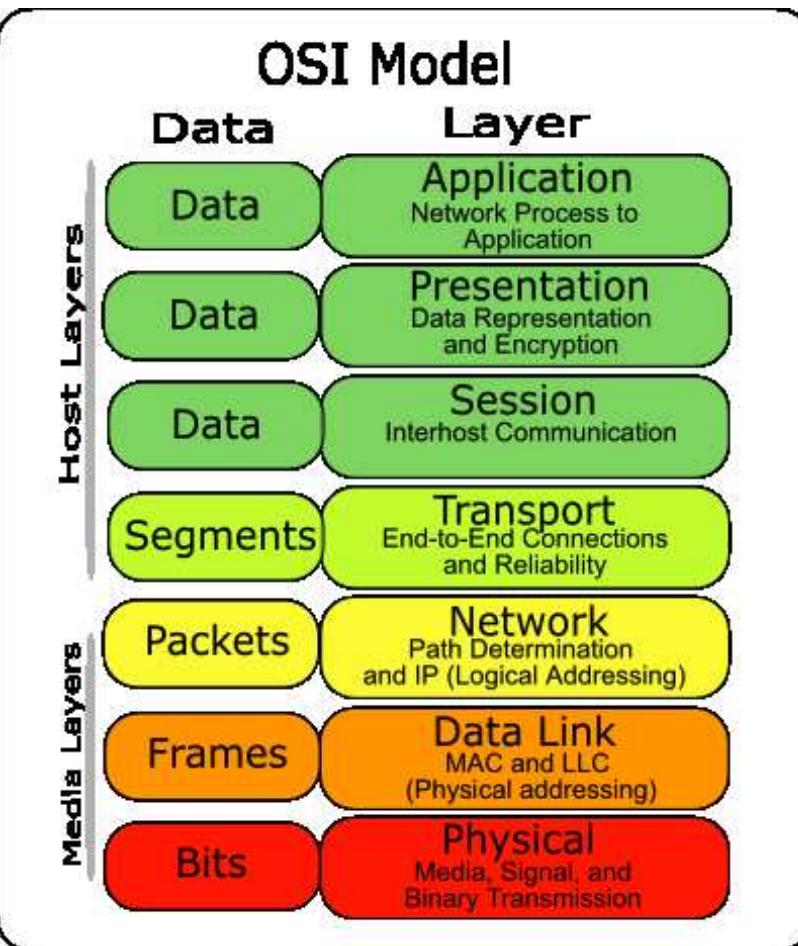
## Funciones de la Capa de Aplicación

- Provee servicios de comunicación a los usuarios ( Capa 8 ; ) y a las aplicaciones, incluye las aplicaciones mismas.
- Existe modelo de comunicación Machine to machine (M2M), no hay usuarios (personas).
- Interfaz con el usuario -User Interface (UI)- u otras aplicaciones/servicios.
- Las aplicaciones que usan la red pertenecen a esta capa.
- Los protocolos que implementan las aplicaciones también.
- Existen aplicaciones que NO son de red que deben trabajar con aplicaciones/servicios para lograr acceso a la red.

## Componentes de la Capa de Aplicación

- Elementos de capa de aplicación: Programas que corren en (diferentes) plataformas y se comunican entre sí y los protocolos que implementan.
- Las aplicaciones en la mayoría de los casos corren en los nodos finales(end-systems), no en el núcleo de la red. (más fácil el desarrollo y uso), siguen principio end-2-end.
- Qué cubre?: Se verá el enfoque orientado a Internet, en el cual la capa de Aplicación integra:
  - Capa de Aplicación propiamente dicha del modelo OSI.
  - Capa de Presentación del modelo OSI.
  - Capa de Sesión del modelo OSI.

# Modelo OSI



## Capa de Sesión

- Administra las conversaciones/diálogos entre las aplicaciones.
- Podría proveer mecanismos transaccionales o de sincronización: COMMIT, CHECKPOINT, ROLLBACK.
- Maneja actividad.
- Informar Excepciones.
- Ejemplo concreto RPC (Remote Procedure Call) en NFS.
- A menudo las facilidades del lenguaje de acceso a bases de datos: SQL podría verse como un ejemplo.
- Integrada en las aplicaciones de red mismas.
- Podría estar ausente.

## Capa de Sesión (Consideraciones)

- Básicamente la Capa de sesión se podría ver como un invento de la ISO.
- Ninguna de las redes existentes tenían esta capa (salvo algunos servicios de modelo OSI aparecen en SNA).
- La capa de sesión es muy “delgada”, con relativamente pocas características.
- En general no se utiliza.

## Capa de Presentación

- Conversión y codificación de datos a codificaciones comunes, ej: ASCII, EBDIC, charset ISO-8859-1, UTF-8, Unicode16.
- Compresión y descompresión de datos.
- Cifrado y de-cifrado de datos.
- Define formatos y algoritmos para esto: JPEG, MPEG, LZW, AES, DES, IDEA.
- Ejemplo concreto: XDR (External Data Representation) en NFS.
- Integrada en las aplicaciones de red mismas.

## Capa de Aplicación

- Define el formato de los mensajes. Existen protocolos que trabajan de forma binaria, por ejemplo usando ASN y otros en forma textual ASCII como HTTP.
- Define la semántica de cada uno de los mensajes.
- Define como debe ser el diálogo (intercambio de mensajes). Que mensajes se deben intercambiar.
- Ejemplo concreto: Protocolo HTTP y sus implementaciones mediante servidores WEB y browsers (navegadores).

## Capa de Aplicación

- Define el formato de los mensajes. Existen protocolos que trabajan de forma binaria, por ejemplo usando ASN y otros en forma textual ASCII como HTTP.
- Define la semántica de cada uno de los mensajes.
- Define como debe ser el diálogo (intercambio de mensajes). Que mensajes se deben intercambiar.
- Ejemplo concreto: Protocolo HTTP y sus implementaciones mediante servidores WEB y browsers (navegadores).

## Modelos de Comunicación de Aplicaciones

- Modelo Mainframe (dumb client).
- Modelo Cliente/Servidor.
- Modelo Peer to Peer (P2P).
- Modelo Híbrido.

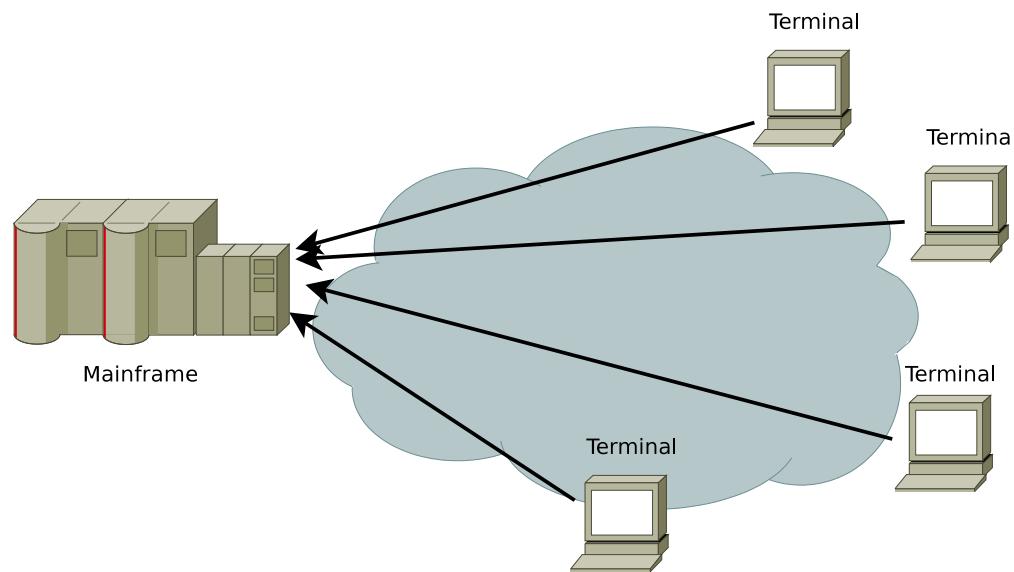
## Modelo de Mainframe Centralizado

- Modelo de Carga concentrada.
- El cliente es “tonto” (dumb) solo corre la comunicación y la interfaz física con el usuario (ej. terminal).
- El servidor pone todo el procesamiento.
- Modelo antiguo que resurge con thin-clients.
- Modelo puro: el mainframe es el que decide cuando le da el control al cliente
- Modelo puro: el mainframe maneja el diálogo de las comunicaciones.
- Cliente ejecuta en el mainframe.
- Ejemplo: Sistema SNA con Mainframe IBM S/370 y terminales

3270 (las terminales verdes del antiguo sistema de alumnos !!!).

- Servidor de Terminales: X11, VNC, Cytrix Metaframe, VMWARE PCoIP, LTS, Virtualización.

## Modelo de Mainframe Centralizado (Cont'd)



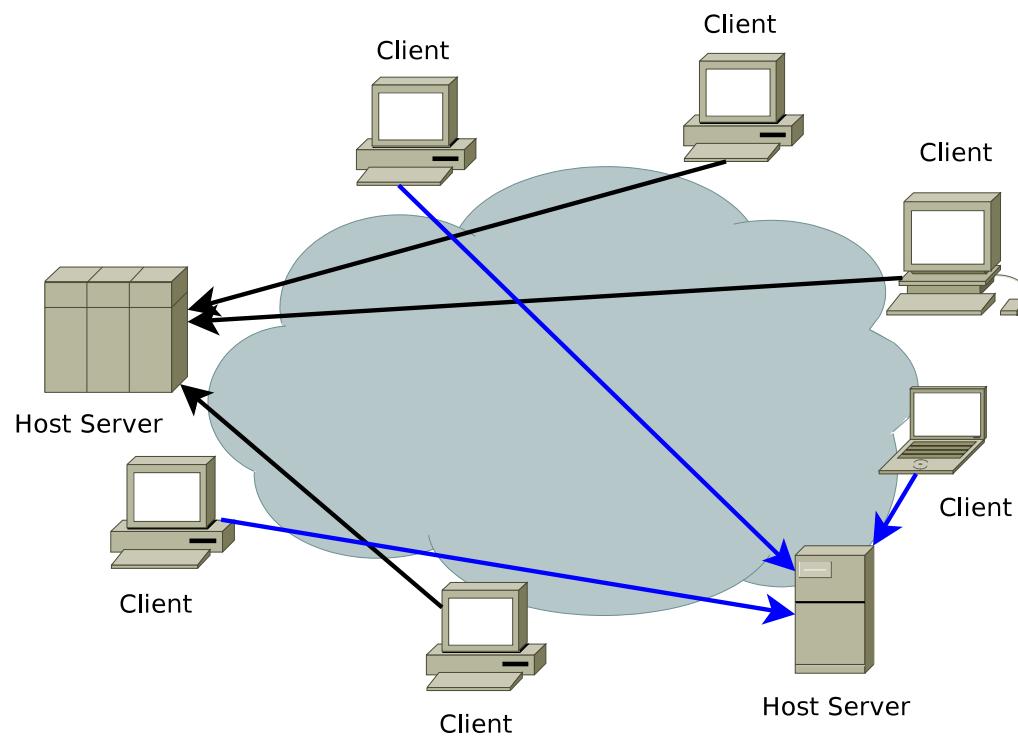
## Modelo de Mainframe Centralizado (Cont'd)



## Modelo de Cliente/Servidor

- Modelo de Carga compartida.
- Idea inicial: el cliente pone procesamiento de interfaz.
- El servidor pone el resto del procesamiento.
- Existen modelos en varios tiers (2 tiers, 3 tier o multi-tier).
- El servidor corre servicio esperando de forma pasiva la conexión.
- Cliente se conectan al servidor y se comunican a través de este.
- Ejemplo: File Server vía NFS o FTP.
- Moldeo asimétrico 1 a N, M a N (donde  $M < N$ ).

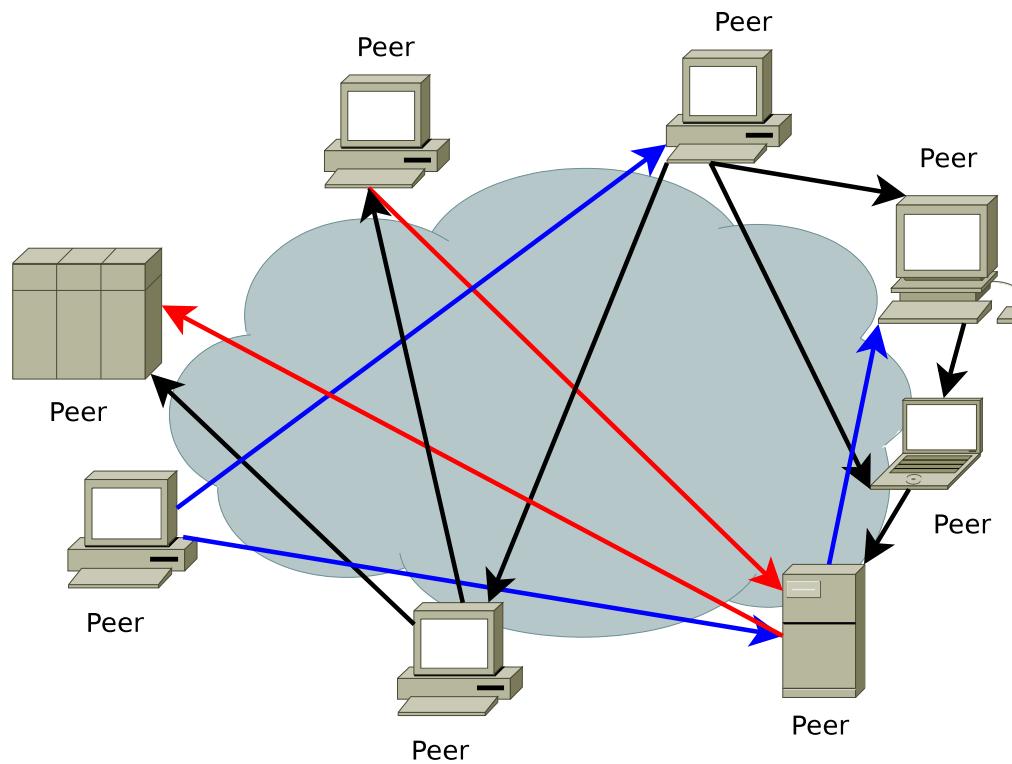
## Modelo de Cliente/Servidor (Cont'd)



## Modelo de Peer-to-Peer

- Modelo de Carga completamente compartida y distribuida.
- Los peers (participantes) pueden cumplir rol de cliente, servidor o ambos en un instante.
- Sistema escalable en cuanto a rendimiento.
- Sistema no escalable en cuanto a administración.
- Ejemplo: redes legadas para compartir archivos: Novell Lite, Microsoft Windows for Workgroup basado en LAN Manager (sobre NetBEUI), Algo más actual: Gnutella, BitTorrent. (servicio de file sharing totalmente P2P)
- Modelo asimétrico N a N.

## Modelo de Peer-to-Peer (Cont'd)



## Modelo de Peer-to-Peer Híbrido

- Modelo de Carga compartida y distribuida.
- Los peers (participantes) pueden cumplir rol de cliente, servidor o ambos en un instante.
- Existen diferentes tipos de nodos con diferente roles.
- Hay nodos centrales donde se registra la información y al resto de los nodos.
- Sistema escalable en cuanto a rendimiento.
- Sistema más escalable que P2P puro?.
- Ejemplos: eDonkey (y sus variantes aMule, eMule), Napster, IM, Skype.
- Moldeo asimétrico M a N.

## BitTorrent

- Protocolo para el intercambio de archivos grandes de forma masiva como peer-to-peer (P2P).
- Desarrollado originalmente por un programador, Bram Cohen (software libre).
- Cuando alguien quiere compartir un archivo genera un archivo .TORRENT.
- A diferencia de otros P2P no tiene índices de búsquedas, si existen WEBS que permite localizar archivos: Mininova, Thepiratebay, Ktorrents, TorrentReactor.

## BitTorrent (Cont'd)

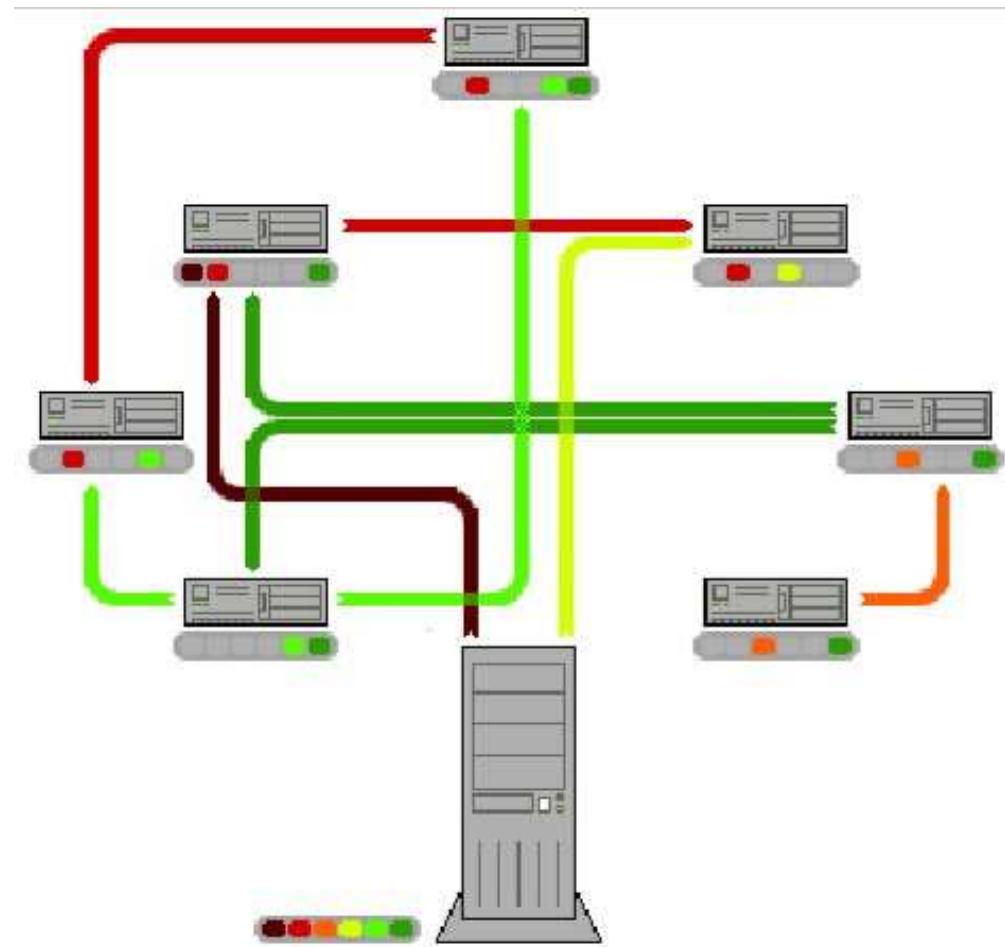
- .TORRENT. Dirección del TRACKER para unirse al grupo de PEERs (más información de archivo).
- Desde el TRACKER se obtiene PEERs: SEEDs (contienen archivo completo) y LEECHERs (contienen archivo parcial). El TRACKER se actualiza con el nuevo PEER.
- La comunicación con el TRACKER habitualmente se hace sobre HTTP, o podría ser sobre FTP.
- Los archivos se dividen en chunks, cada uno puede ser descargado de un PEER diferente.
- Se conecta con otros PEERs y comienza la descarga. Cada chunk se comparte con otros PEERs. Utiliza el port 6881 y escanea hacia arriba.

## BitTorrent (Cont'd)

- Cuando baja trabaja de forma aleatoria o “rarest first algorithm”, trata de bajar los chunks con menos números de copias.
- Por cada archivo puede haber como máximo 4 PEERs remotos descargando. Los peers se rankean, se deja bajar de aquellos que mantienen una buena relación de bajada y subida con el PEER.
- Se seleccionan PEERs aleatoriamente cada determinado tiempo.
- BitTorrent puede trabajar con NAT, si el cliente se conecta (outbound connection) a otro PEER que tiene IP pública y esta conexión se usa para download y upload. Es conveniente habilitar port forwarding.

- Se puede restringir download y upload rate.

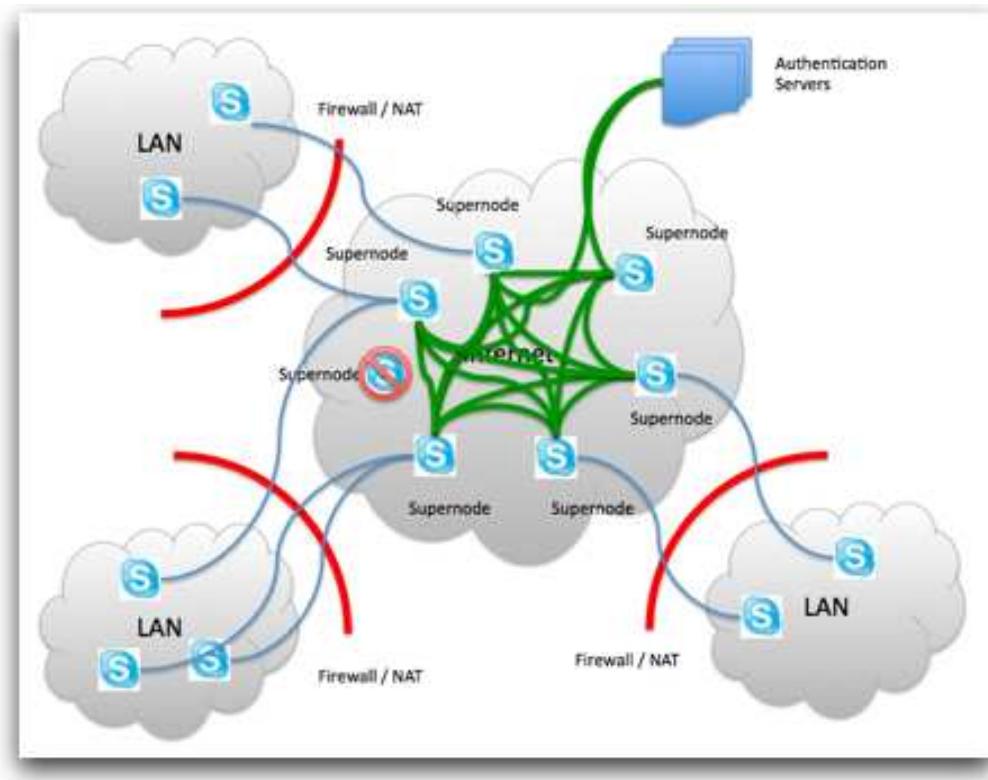
## BitTorrent (Cont'd)



## Skype

- Protocolo propietario que implementa VoIP.
- Nodos clientes pueden ser solo nodos o Super-nodos.
- Super-nodos actúan como directorios.
- Estos nodos hace de STUN, permitiendo conexión de computadoras detrás de NAT.
- Los super-nodos crean P2P Overlay networks (Una red sobre otra red).
- Corren el software tradicional de los clientes, pero sobre la Internet pública.
- Super-nodos se conectan entre si. Los Super-nodos utilizan servicios externos de autenticación.
- Existen Mega Super-nodos que pertenecen a Skype.

## Skype (Cont'd)



## Requerimientos de Aplicaciones

<u>Aplicación</u>	<u>Pérdidas</u>	<u>Bandwidth</u>	<u>Sensible a Time</u>
file transfer	no	Flexible	no
e-mail	no	Flexible	no
Web documents	no	Flexible	no
real-time audio/video	tolerante	audio: 5kbps-1Mbps video:10kbps-5Mbps	si, 100's msec
stored audio/video	tolerante	Igual al de arriba	si, pocos secs
interactive games	tolerante	Pocos Kbps	si, 100's msec
instant messaging	no	flexible	Si y no

- Cada aplicación puede tener diferentes requerimientos: seguridad, tiempo de respuesta, confiabilidad, optimizar ancho de banda, etc...
- Diferentes Alternativas de Transporte.

- Transport Control Protocol (TCP).
- User Datagram Protocol (UDP).
- Otras alternativas: Stream Control Transmission Protocol (SCTP),  
Reliable User Datagram Protocol (RUDP).

## Direccionamiento de Procesos

- Las aplicaciones son implementadas por los SO como procesos.
- Conceptos de IPC (Inter-Process Communication).
- Para que un proceso reciba un mensaje, éste debe tener un identificador único.
- Identificador de host no suficiente, se agrega identificador de proceso (independiente del SO) número de puerto.
- Servicio de multiplexación provisto por nivel de transporte.
- Accedido mediante la API de **Sockets BSD**, Winsocks de Microsoft o TLI/XTI de AT&T.

## Fuentes de Información

- Kurose/Ross: Computer Networking (6th Edition).
- Andrew S. Tanenbaum. Computer Networks (4th Edition).
- Wikipedia <http://www.wikipedia.org>.
- BitTorrent: <http://www.bittorrent.org>.
- Understanding Today's Skype Outage: Explaining Supernodes, Dan York, Dec 2010.
- Slides de Kurose/Ross Computer Networking 6ta edición.
- Internet ...

# Capítulo 2: Capa Aplicación

# Capítulo 2: Capa Aplicación

- 2.1 Principios de las aplicaciones de red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS

# Capítulo 2: Capa Aplicación

## Objetivos:

- Aspectos conceptuales y de implementación de los protocolos de aplicación.
  - Modelo de servicio de la capa transporte.
  - Paradigma cliente-servidor.
  - Paradigma peer-to-peer (par-a-par).
- Aprendizaje de protocolos examinando protocolos de aplicación populares
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Programación de aplicaciones de red
  - API de socket

# Principios de los protocolos de Capa Aplicación

- El software de una aplicación está distribuido entre dos o más sistemas.
- Ese “pedacito” de software es un proceso.
- Los procesos se comunican a través de mensajes.
  
- Aplicación de red <> Protocolo de capa de aplicación: el protocolo de la capa de aplicación es sólo una parte de la aplicación de red.

# Algunas aplicaciones de red

- E-mail
- Web
- Mensajería instantánea
- Login remoto
- Compartición de archivos P2P
- Juegos de red multi-usuarios
- Reproducción de clips de video almacenados
- Telefonía Internet
- Conferencias de video en tiempo real
- Computación paralela masiva.

# Protocolos de capa aplicación definen

- Tipos de mensajes intercambiados, e.g., mensajes de requerimiento y respuesta
  - Sintaxis de los tipos de mensajes: qué campos del mensajes y cómo éstos son delimitados.
  - Semántica de los campos, es decir el significado de la información en los campos
  - Reglas para cuándo y cómo los procesos envían y responden a mensajes
- Protocolos de dominio público:
- Definidos en RFCs
  - Permite inter-operatividad
  - eg, HTTP, SMTP
- Protocolos propietarios:
- eg, KaZaA

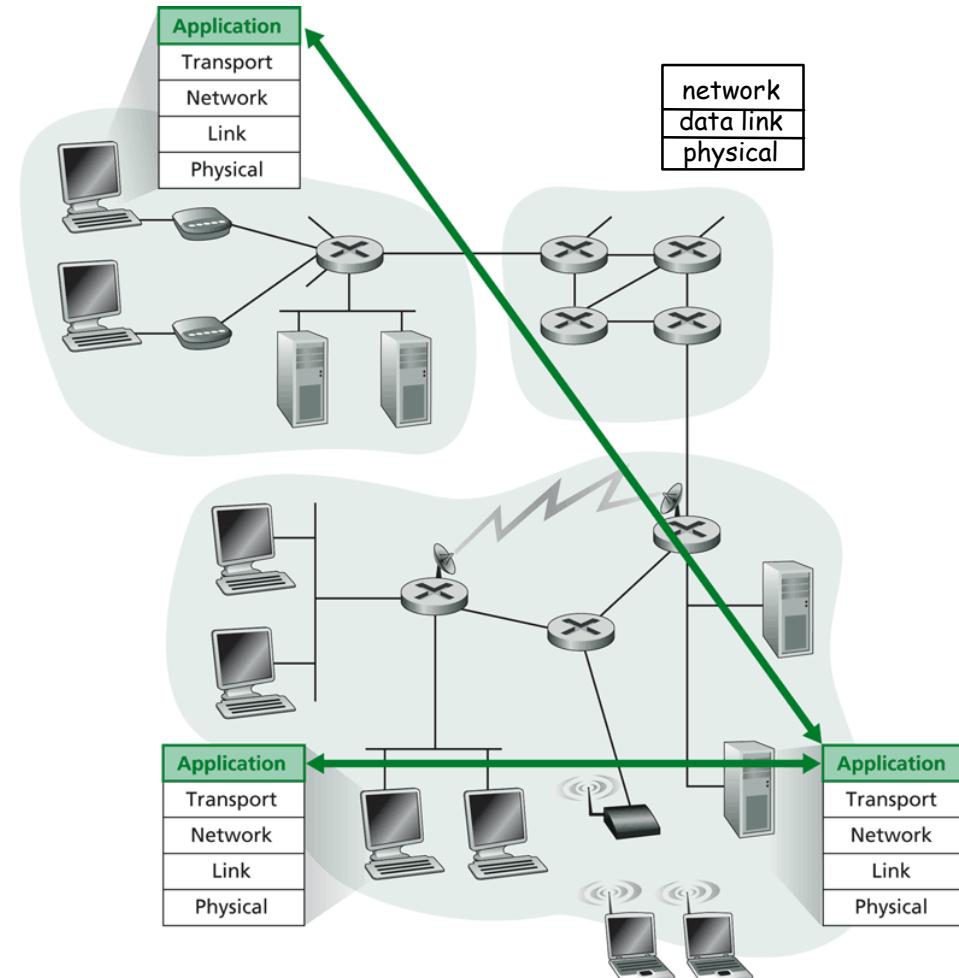
# Creación de una aplicación de red

Escribe un programa que

- Corra en diferentes sistemas y
- Se comunique por la red.
- e.g., Web: Programa del servidor Web se comunica con el programa del navegador

No se refiere al software escrito para los dispositivos en la red interna

- Dispositivos internos no funcionan en la capa aplicación
- Este diseño permite desarrollos rápidos

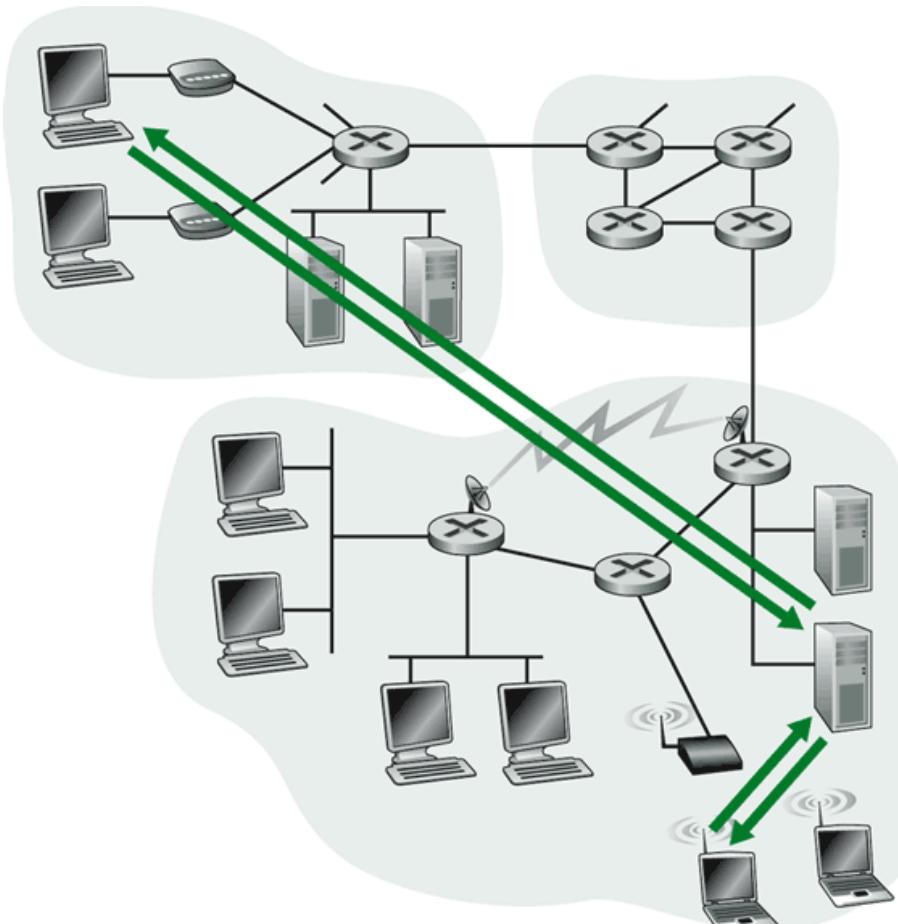


**Figure 2.1** ♦ Communication for a network application takes place between end systems at the application layer.

# Arquitecturas de Aplicación

- Cliente-servidor
- Peer-to-peer (P2P)
- Híbridos de cliente-servidor y P2P

# Arquitectura Cliente-servidor



a. Client-server application

servidor:

- Computador siempre on
- Dirección IP permanente

cliente:

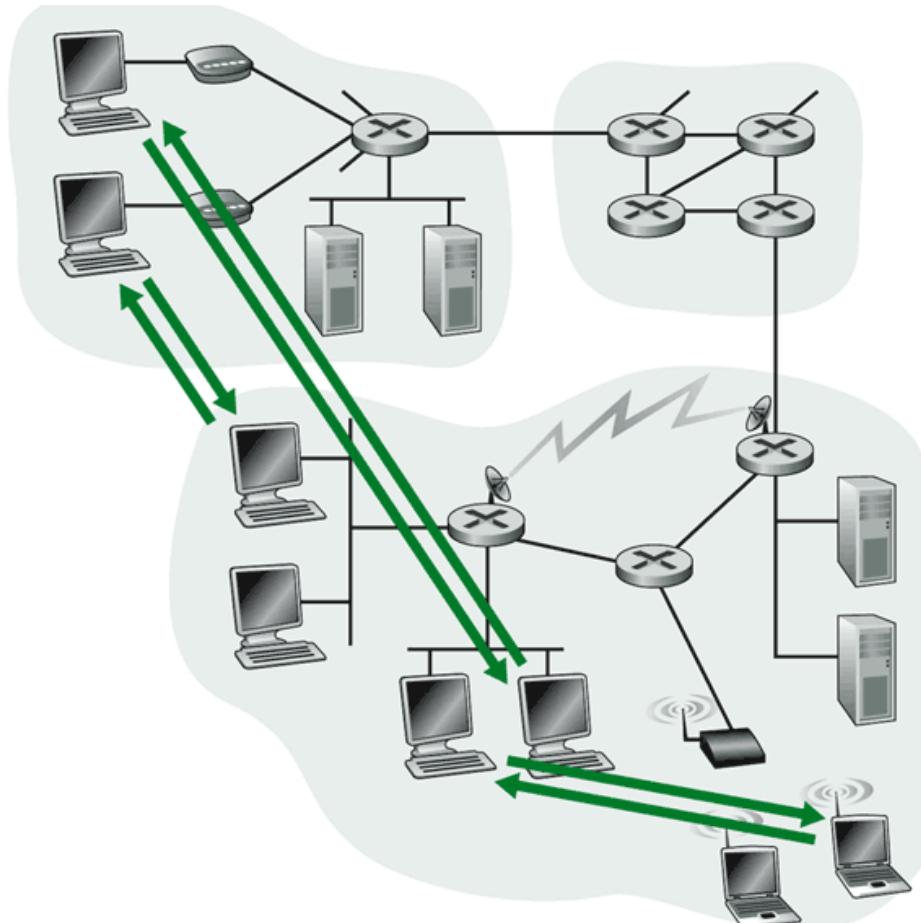
- Se comunica con un servidor
- Puede ser conectado intermitentemente
- Puede tener direcciones IP dinámicas
- No se comunican directamente entre sí (dos clientes puros)

# Arquitectura P2P Pura

- Servidor no siempre on
- Sistemas terminales arbitrarios se comunican directamente
- Pares se conectan intermitentemente y cambias sus direcciones IP
- ejemplo: Gnutella

Altamente escalable

Pero difícil de administrar



b. Peer-to-peer application

# Híbridos de cliente-servidor y P2P

## Napster

- Transferencia de archivos P2P
- Búsqueda de archivos centralizada:
  - Pares registran contenidos en servidor central
  - Pares consultan algún servidor central para localizar el contenido

## Mensajería Instantánea

- Diálogo es entre los usuarios es P2P
- Detección/localización de presencia es centralizada:
  - Usuario registra su dirección IP en un servidor central cuando ingresa al sistema
  - Usuarios contactan servidor central para encontrar las direcciones IP de sus amigos.

# Procesos que se comunican

Proceso: programa que corre en una máquina.

- Dentro de la máquina dos procesos se comunican usando comunicación entre proceso (definida por OS).
- Procesos en diferentes hosts se comunican vía intercambio de mensajes

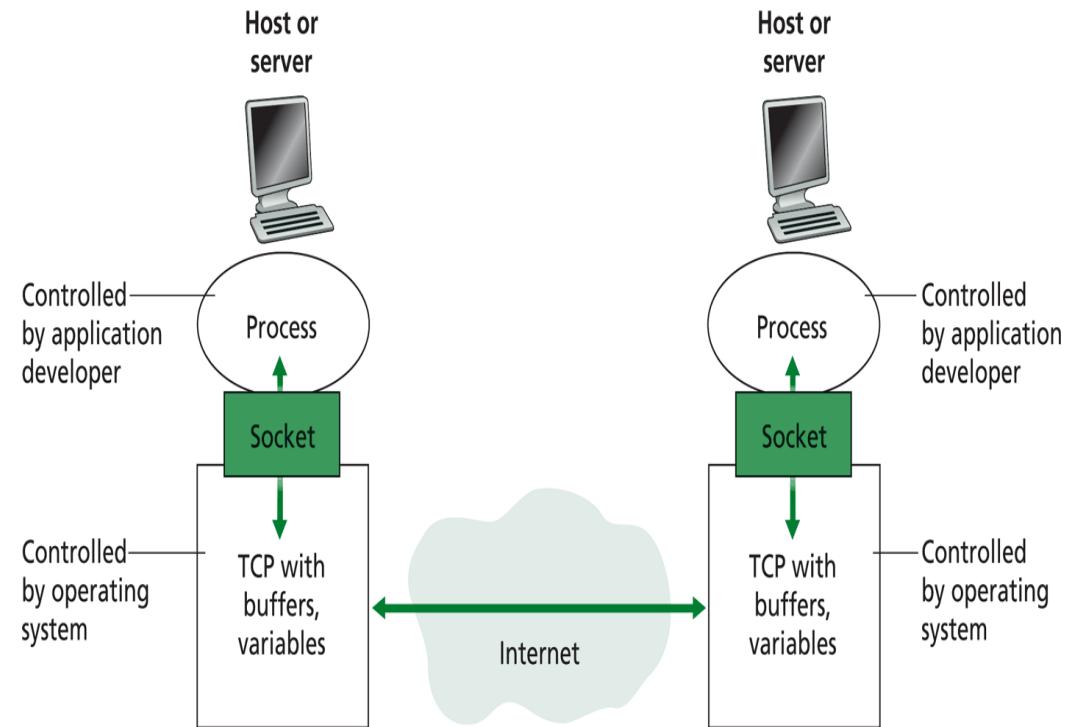
Proceso Cliente: proceso que inicia la comunicación (crea y envía mensajes sobre la red)

Proceso servidor: proceso que espera por ser contactado (recibe los mensajes y, posiblemente responde enviando mensajes)

- Nota: Aplicaciones con arquitectura P2P tienen procesos clientes y procesos servidores

# Sockets

- Los procesos envían/reciben mensajes a/desde sus socket
- socket son análogos a puertas
  - Proceso transmisor saca mensajes por la puerta
  - Proceso transmisor confía en la infraestructura de transporte al otro lado de la puerta la cual lleva los mensajes al socket en el proceso receptor
- API: (1) debemos elegir el protocolo de transporte; (2) podemos definir algunos parámetros (volveremos más adelante)



**Figure 2.3** ♦ Application processes, sockets, and underlying transport protocol

## Direccionamiento de procesos

- Para que un proceso reciba un mensaje, éste debe tener un identificador
- Un host tiene una dirección IP única de 32 bits.
- Q: ¿Es suficiente la dirección IP para identificar un proceso en un host?
- Respuesta: No, muchos procesos pueden estar corriendo en el mismo host.
- El identificador incluye la dirección IP y un número de puerta asociado con el proceso en el host.
- Ejemplo de números de puertas:
  - Servidor HTTP: 80
  - Servidor de Mail: 25

## Capa de aplicación: Agentes de usuario

### Agente de usuario

- Es una interfaz entre el usuario y la aplicación de red.
- Por ejemplo: en la WEB, el agente de usuario es el navegador, el cual permite al usuario visualizar las páginas WEB e interactuar con los elementos de la misma. El navegador es un proceso que envía/recibe mensajes por medio de un socket y además brinda la interfaz al usuario.*

# ¿Qué servicios de transporte necesita una aplicación?

## Pérdida de Datos

- Algunas aplicaciones (e.g., audio) pueden tolerar pérdida
- otras transferencia (e.g., de archivos, telnet) requieren transferencia 100% confiable

## Retardo

- Algunas Aplicaciones (e.g., Telefonía Internet, juegos interactivos) requieren bajo retardo para ser “efectivas”

## Bandwidth

- Algunas aplicaciones multimedia) requieren cantidad mínima de ancho de banda para ser “efectivas”
- otras (“aplicaciones elásticas”) hacen uso del bandwidth que obtengan

## Requerimientos de servicio de transporte de aplicaciones comunes

<b>Aplicación</b>	<b>Pérdidas</b>	<b>Bandwidth</b>	<b>Sensible a Time</b>
file transfer	no	Flexible	no
e-mail	no	Flexible	no
Web documents	no	Flexible	no
real-time audio/video	tolerante	audio: 5kbps-1Mbps video:10kbps-5Mbps	si, 100's msec
stored audio/video	tolerante	Igual al de arriba	si, pocos secs
interactive games	tolerante	Pocos Kbps	si, 100's msec
instant messaging	no	flexible	Si y no

# Servicios de los protocolos de transporte en Internet

## Servicio TCP:

- Orientado a la conexión* acuerdo requerido entre procesos cliente y servidor antes de transferencia
- Transporte confiable* entre proceso Tx y Rx
- Control de flujo*: Tx no sobrecargará al Rx
- Control de congestión*: frena al Tx cuando la red está sobrecargada
- No provee*: garantías de retardo ni ancho de banda mínimos

## Servicio UDP:

- Transferencia de datos no confiable entre proceso Tx y Rx.
- No provee: acuerdo entre los procesos, confiabilidad, control de flujo, control de congestión, garantías de retardo o ancho de banda

Q: ¿Por qué molestarse?  
¿Por qué existe UDP?

## Aplicaciones Internet: aplicación, protocolo de transporte

<b>Aplicación</b>	<b>Protocolo capa aplicación</b>	<b>Protocolo de transporte que lo sustenta</b>
	SMTP [RFC 2821]	
e-mail	Telnet [RFC 854]	TCP
remote terminal access	HTTP [RFC 2616]	TCP
Web	FTP [RFC 959]	TCP
file transfer	proprietary	TCP
streaming multimedia	(e.g. RealNetworks)	TCP or UDP
	proprietary	
Internet telephony	(e.g., Dialpad)	typically UDP

# Capítulo 2: Capa Aplicación

- 2.1 Principios de las aplicaciones de red
- 2.2 Web y HTTP
- 2.3 FTP
- 2.4 Correo Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS

# Protocolo HTTP(hasta 1.1) y WEB-Cache

Redes y Comunicaciones

# Historia

- WWW (red de sistemas de hipertexto inter-linkeados accesibles vía Internet).
- Desarrollada en 1990 por Tim Berners-Lee en el CERN.
  - Desarrolla el protocolo HTTP y el lenguaje HTML.
- 1993: el primer cliente/browser GUI: Mosaic.
- 1994: Netscape Navigator 1.0.
- Desarrollo de servidores, por ejemplo 1995: Apache Server.
- Buscadores, Indexadores son parte importante del servicio.
- Modelo anterior de interacción: servidor produce, cliente consume.
- Nuevos Modelos: Web 2.0, Tim O'Reilly en 2004, modelo de interacción entre usuarios, consumen y producen mediante servicios especiales: blogs, redes sociales, wikis, multimedia, etc.

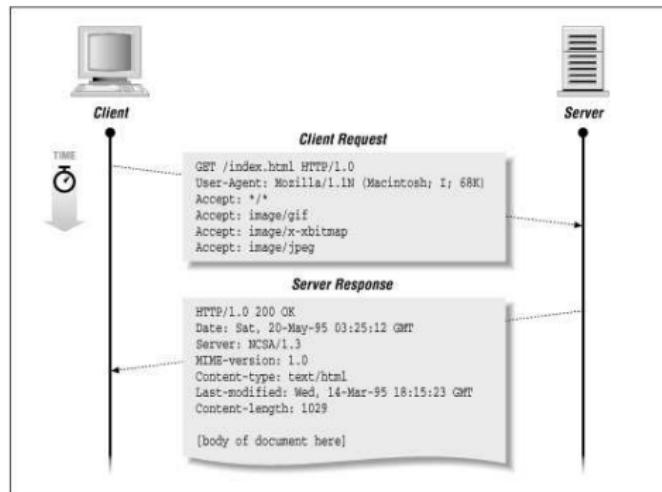
# Elementos WEB

- Recurso u Objeto HTTP: ej. web page.
- Referenciado por una URI (Uniform Resource Id): URL (Uniform Resource Location) o URN (Name).
- Formato de URL:  
`protocol://[user:pass@]host:[port]/[path].`
- Ejemplo: `http://www.NN.unlp.edu.ar:8080/dir/index.html`.
- URN: no indica ubicación, solo identifican, categorías, poco impl. ej. `urn:isbn:0132856204`, `urn:ietf:rfc:2616`.
- Objetos pueden ser página web (web page), imágenes JPEG, PNG, GIF, Java Applet, archivos de multimedia: MP3, AVI, etc.
- Páginas web: archivo HTML que incluye vínculos o directamente otros objetos.

# Funcionamiento de HTTP

- Modelo cliente/servidor, Request/Response (sin estados - stateless).
- Protocolo corre sobre TCP (requiere protocolo de transporte confiable), puerto 80 el default.
- El cliente escoge cualquier puerto no privilegiado.
- Trabaja sobre texto ASCII, permite enviar información binaria con encabezados MIME.
- Clientes (llamados browsers o navegadores): Firefox, IE, Opera, Safari, Chrome.
- Servidores: Apache Server, MS IIS, NGINX , Google GWS, Tomcat.

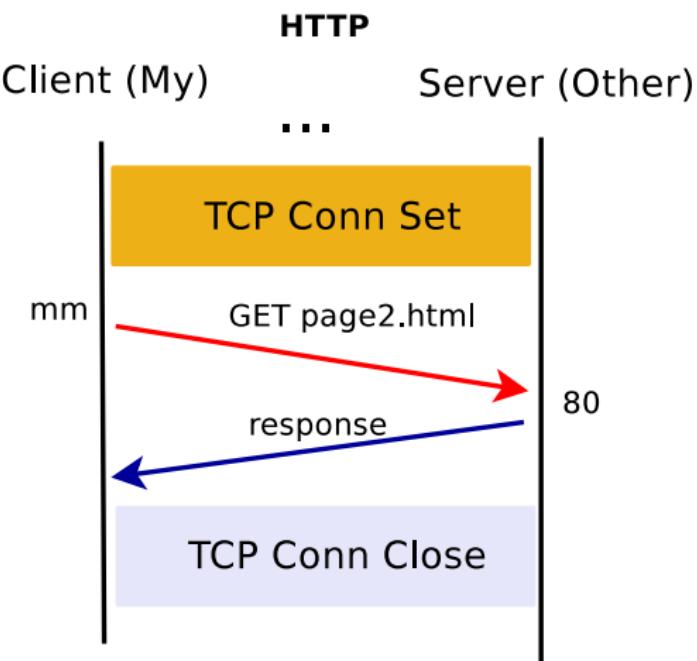
# Esquema de HTTP



# Versión HTTP 0.9

- Primera versión de HTTP fue la 0.9 nunca se estandarizó.
- Pasos para obtener un documento:
  - ① Establecer la conexión TCP
  - ② HTTP Request vía comando GET.
  - ③ HTTP Response enviando la página requerida.
  - ④ Cerrar la conexión TCP por parte del servidor.
  - ⑤ Si no existe el documento o hay un error directamente se cierra la conexión.

# Diagrama de Pasos HTTP



# Versión HTTP 0.9 (cont'd)

- Solo una forma de Requerimiento.
- Solo una forma de Respuesta.
- Request/Response sin estado.

Request ::= GET <document-path> <CR><LF>

Response ::= ASCII chars HTML Document.

GET /hello.html <CR><LF>

GET / <CR><LF>

# Versión HTTP 1.0

- Versión de HTTP 1.0 estándar [RFC-1945].
- Define formato, proceso basado en HTTP 0.9:
  - Se debe especificar la versión en el requerimiento del cliente.
  - Para los Request, define diferentes métodos HTTP.
  - Define códigos de respuesta.
  - Admite repertorio de caracteres, además del ASCII, como: ISO-8859-1, UTF-8, etc.
  - Admite MIME (No solo sirve para descargar HTML e imágenes).
  - Por default NO utiliza conexiones persistentes.

# Request HTTP 1.0

<Method> <URI> <Version>

[<Headers Opcionales>]

<Blank>

[<Entity Body Opcional>]

<Blank>

<Method HTTP 1.0> ::= GET, POST, HEAD, PUT,  
DELETE, LINK, UNLINK

# Response HTTP 1.0

```
<HTTP Version> <Status Code> <Reason Phrase>
[<Headers Opcionales>]
<Blank>
[<Entity Body Opcional>]
```

# Ejemplos de respuestas HTTP/1.0

HTTP/<version> 200 OK

HTTP/<version> 301 Moved Permanently

HTTP/<version> 400 Bad Request

HTTP/<version> 403 Access Forbidden

HTTP/<version> 404 Not Found

HTTP/<version> 405 Method Not Allowed

HTTP/<version> 500 Internal Server Error (CGI Error)

HTTP/<version> 501 Method Not Implemented

# Métodos HTTP/1.0

**GET:** obtener el documento requerido. Puede enviar información, pero no demasiada. Es enviada en la URL. Formato ?var1=val1&var2=val2.... Limitación de tamaño de URL por parte de las implementaciones. NO espera recibir datos en body.

**HEAD:** idéntico a GET, pero sólo requiere la meta información del documento, por ejemplo, su tamaño. Usado por clientes con caché.

**POST:** hace un requerimiento de un documento, pero también envía información en el Body. Generalmente, usado en el fill-in de un formulario HTML(FORM). Puede enviar mucha más información que un GET.

# Métodos HTTP/1.0 (Cont'd)

**PUT:** usado para reemplazar un documento en el servidor. En general, deshabilitada. Utilizado, por ejemplo, por protocolos montados sobre HTTP, como WebDAV [WDV].

**DELETE:** usado para borrar un documento en el servidor. En general, deshabilitada. También, puede ser utilizada por WebDAV.

**LINK, UNLINK:** establecen/des-establecen relaciones entre documentos.

# GET/Response HTTP 1.0

```
GET /index2.html HTTP/1.0
User-Agent: telnet/andres (GNU/Linux)
Host: estehost.com
Accept: */*
```

```
HTTP/1.1 200 OK
Date: Mon, 21 Apr 2008 00:28:51 GMT
Server: Apache/2.2.4 (Ubuntu)
Last-Modified: Mon, 21 Apr 2008 00:18:14 GMT
ETag: "a3b36-1f-91d5d80"
Accept-Ranges: bytes
Content-Length: 31
Connection: close
Content-Type: text/plain
```

# GET/Response HTTP 1.0 (Cont'd)

```
<HTML>
<H1> HOLA </H1>
...
</HTML>
```

# HEAD/Response HTTP 1.0

HEAD /index.html HTTP/1.0  
User-Agent: telnet/andres (GNU/Linux)  
Host: estehost.com  
Accept: \*/\*

HTTP/1.1 200 OK  
Date: Mon, 21 Apr 2008 00:40:25 GMT  
Server: Apache/2.2.4 (Ubuntu)  
Last-Modified: Mon, 21 Apr 2008 00:18:14 GMT  
ETag: "a3b36-1f-91d5d80"  
Accept-Ranges: bytes  
Content-Length: 31  
Connection: close  
Content-Type: text/plain

# POST/Response HTTP 1.0

```
POST /index.html HTTP/1.0
User-Agent: telnet/andres (GNU/Linux)
Host: estehost.com
Accept: */
Content-Type: text/plain
Content-Length: 10
```

1234567890

```
HTTP/1.1 200 OK
Date: Mon, 21 Apr 2008 00:37:22 GMT
Server: Apache/2.2.4 (Ubuntu)
Last-Modified: Mon, 21 Apr 2008 00:18:14 GMT
ETag: "a3b36-1f-91d5d80"
Accept-Ranges: bytes
```

# POST/Response HTTP 1.0 (Cont'd)

Content-Length: 31

Connection: close

Content-Type: text/plain

<HTML>

<H1> HOLA </H1>

...

</HTML>

# GET HTTP/1.0 con Host Virtuales

- Mediante el parámetro Host se pueden multiplexar varios servicios sobre un mismo host.

```
? ./mozilla http://www.uno.test
```

```
? ./mozilla http://www.dos.test
```

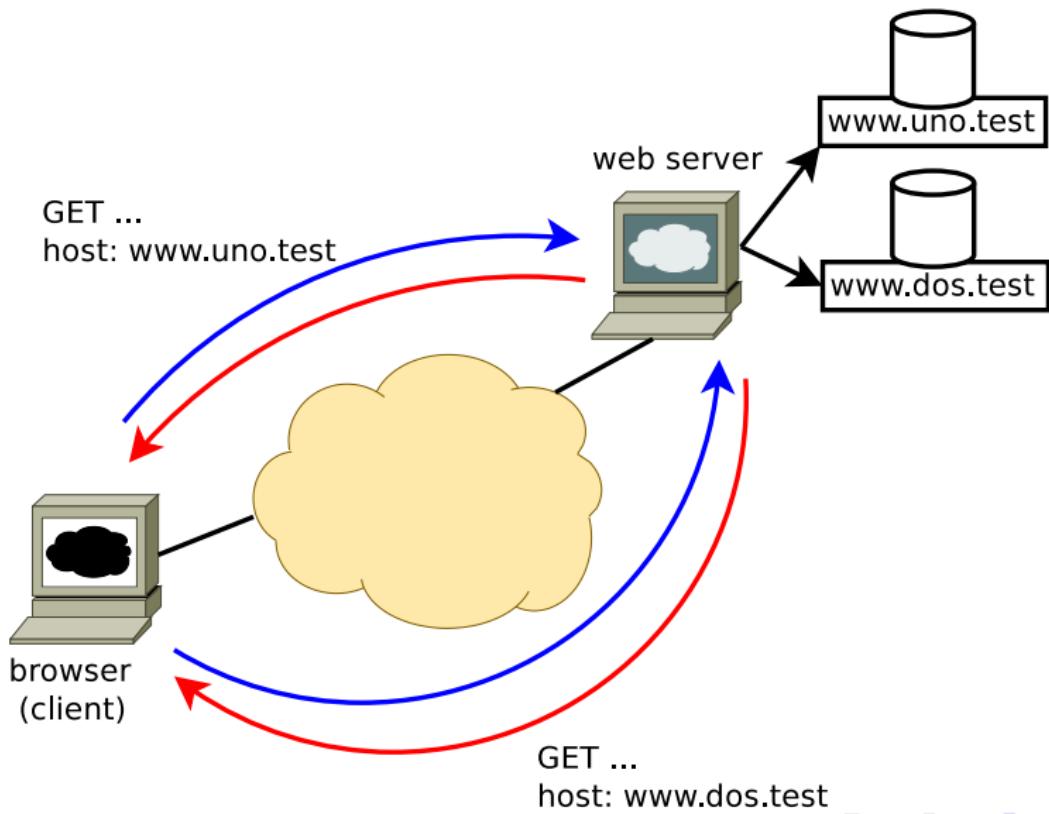
```
? host www.uno.test
```

```
www.uno.test has address 192.168.0.2
```

```
? host www.dos.test
```

```
www.dos.test has address 192.168.0.2
```

# Virtual Hosts



# PUT, DELETE HTTP 1.0

- Los método PUT/DELETE ya existían en HTTP/1.0.
- No se implementan directamente en el servidor.
- Se deben agregar como una extensión CGI para manejarlos.
- Debe leer el campo de longitud y leer esa cantidad de información de la entrada, luego generando el archivo.
- Los utilizan protocolos como WebDAV, MS File sharing/OpenCloud.
- Otra forma de subir o borrar archivos es usando el método POST y programando una CGI adecuada.
- La diferencia con el POST que el archivo que se indica puede no existir y el script de procesamiento es el mismo.

# Ejemplo de PUT

```
LoadModule actions_module .../mod_actions.so  
...  
Script PUT /cgi-bin/put.pl
```

```
PUT /otro.txt HTTP/1.0  
Host: otrohost.com  
Accept-Encoding: gzip, deflate  
Content-Type: text/xml  
User-Agent: telnet/andres (GNU/Linux)  
Content-Length: 10
```

0123456789

# Autenticación HTTP

- HTTP/1.0 contempla autenticación con `WWW-Authenticate Headers`.
- Encabezados, el cliente y el servidor intercambiar información auth.
- El servidor, ante un requerimiento de un documento que requiere autenticación, enviará un mensaje 401 indicando la necesidad de autenticación y un Dominio/Realm.
- El navegador solicitará al usuario los datos de user/password (si es que no los tiene cachedos) y los enviará en texto claro al servidor.
- El servidor dará o no acceso en base a esos valores.
- Para los siguientes requerimientos, el navegador usará los valores que tiene almacenados para el Realm solicitado.

# Autenticación HTTP (Cont'd)

## Authorization Required

This server could not verify that you are authorized to access the document requested. Either you supplied the wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials required.

Apache/2.2.9 (Debian) DAV/2 SVN/1.5.1 PHP/5.2.6-1+lenny3 with Suhosin-Patch mod\_ssl/2.2.9 OpenSSL/0.9.8g Server at

Port 443



# HTTP/1.0 con conexiones persistentes

- En HTTP/1.0 no se contemplaron las conexiones persistentes por default.
- A partir de HTTP/1.1 [RFC-2068], si.
- En HTTP/1.0 se pueden solicitar de forma explícita.

GET /index.html HTTP/1.0

Connection: Keep-Alive

User-Agent: telnet/andres (GNU/Linux)

Host: estehost.com

Accept: \*/\*

HTTP/1.1 200 OK

Date: Tue, 22 Apr 2008 01:31:56 GMT

Server: Apache/2.2.4 (Ubuntu)

Last-Modified: Mon, 21 Apr 2008 00:18:14 GMT

# HTTP/1.0 con conexiones persistentes (Cont'd)

```
ETag: "a3b36-1f-91d5d80"  
Accept-Ranges: bytes  
Content-Length: 31  
Keep-Alive: timeout=15, max=10  
Connection: Keep-Alive  
Content-Type: text/html
```

```
<HTML>  
<H1> HOLA </H1>  
</HTML>
```

```
GET /index.html HTTP/1.0  
Connection: Keep-Alive
```

...

Connection closed by foreign host.

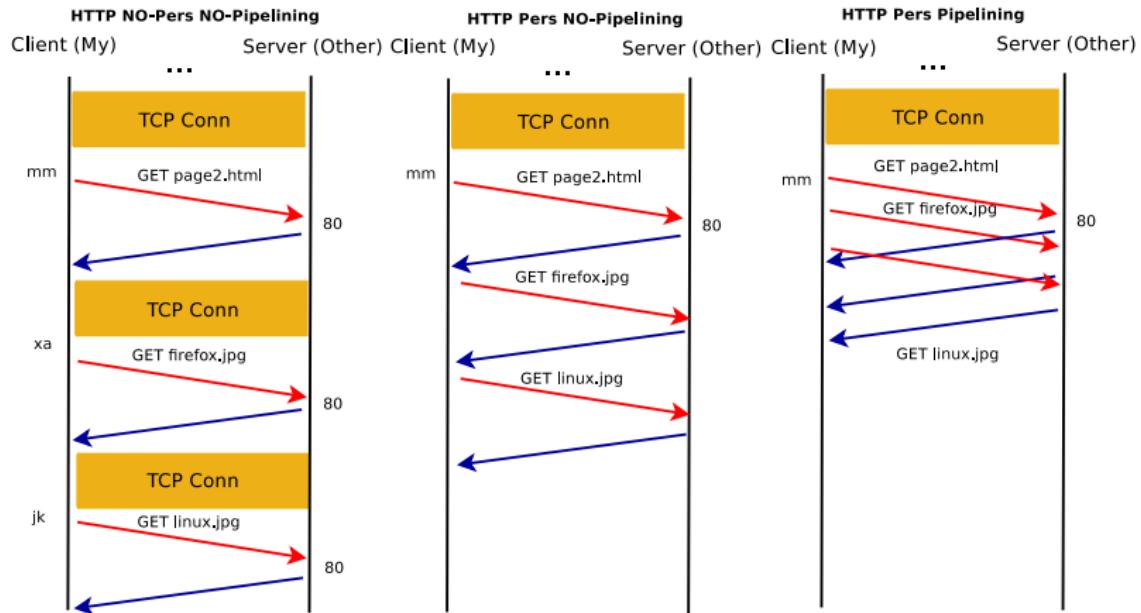
# Versión HTTP 1.1

- HTTP/1.1, 1997 con la [RFC-2068] se actualiza con [RFC-2616].
- Nuevos mensajes HTTP 1.1: OPTIONS, TRACE, CONNECT.
- Conexiones persistentes por omisión.
- Pipelining, mejora tiempo de respuestas.

# Pipelining HTTP 1.1

- No necesita esperar la respuesta para pedir otro objeto HTTP.
- Solo se utiliza con conexiones persistentes.
- Mejora los tiempos de respuestas.
- Sobre la misma conexión se debe mantener el orden de los objetos que se devuelven.
- Se pueden utilizar varios threads para cada conexión.
  - Sin pipelining:  $1RTT + FT$  por cada objeto,  $n$  objetos  $nRTT + nFT$ .
  - Con pipelining óptimo:  $n$  objetos:  $1RTT + nFT$ .

# Pipelining HTTP 1.1 (Cont'd)



# TRACE y CONNECT HTTP 1.1

- TRACE utilizada para debugging.
- El servidor debe copiar el requerimiento tal cual.
- El cliente puede comparar lo que envía con lo que recibe.
- CONNECT utilizada para generar conexiones a otros servicios montadas sobre HTTP.
- Proxy-Agent genérico.

# Ejemplo de CONNECT

CONNECT 127.0.0.1:25

HTTP/1.0 200 Connection Established

Proxy-agent: Apache/2.2.4 (Ubuntu)

220 khartum ESMTP Postfix (Ubuntu)

HELO otrohost.com

250 khartum

QUIT

221 2.0.0 Bye

...

# Redirects HTTP

- Redirect temporal 302, indicando la nueva URL/URI.
- El user-agent no debería re-direccionarlo salvo que el usuario confirme.
- Moved Permanently 301, se indica que cualquier acceso futuro debe realizarse sobre la nueva ubicación (mejora Indexadores).
- Se pueden generar problemas con Cookies.

# CGIs Scripts y JavaScript

- Necesidad de dinamismo para generar aplicaciones.
- Server-Side Script:
  - Ejecuta del lado del servidor.
  - CGI (Common Gateway Interface): aplicación que interactúa con un servidor web.
  - CGIs leen de STDIN (POST) o de variables de entorno (GET): QUERY\_STRING datos de usuario.
  - Escriben en la STDOUT response.
  - Deben anteponer el content-type en el header.
  - POST permite enviar más datos.
  - Lenguajes de scripting más flexibles y seguros: PHP, ASP, JSP.
  - Implementados como CGIs, dentro o módulos propios del web-server.

# CGIs Scripts y JavaScript (Cont'd)

- Client-Side Script:

- Ejecuta del lado del cliente, en el browser.
- JavaScript estándar W3C.
- Usan modelo de objetos DOM (Document Object Model).
- Otros lenguajes JScript, VBScript.
- Permiten extensiones como AJAX (Asynchronous JavaScript And XML).
- AJAX hace requerimientos particulares y no necesita recargar toda la página.
- Parseo XML para comunicarse.
- Existen numerosos frameworks que encapsulan esta funcionalidad brindando una interfaz de programación, API fácil de utilizar.

# CGIs Scripts y JavaScript (Cont'd)

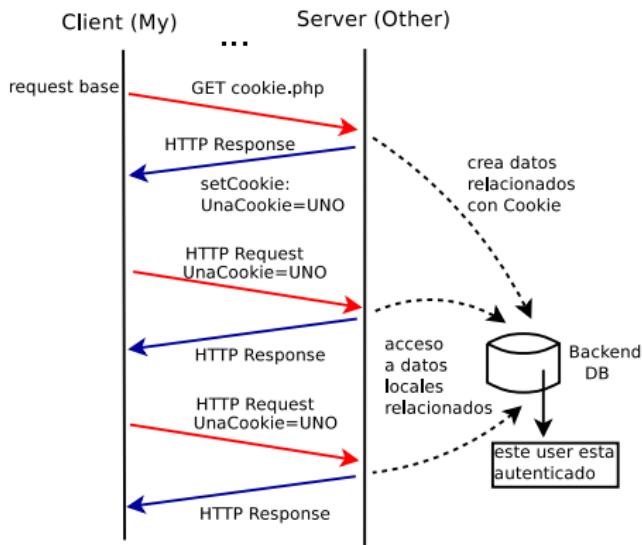
- Server-Side to Server-Side Script:

- Permiten comunicación entre servidores.
- Modelo de “objetos” y servicios distribuidos.
- Conjunto de convenciones para implementar RMI (Remote Method Invocation) sobre HTTP (u otro protocolo de texto).
- Previo XML-RPC.
- SOAP (Simple Object Access Protocol).
- Web-Services.
- REST.

# Cookies

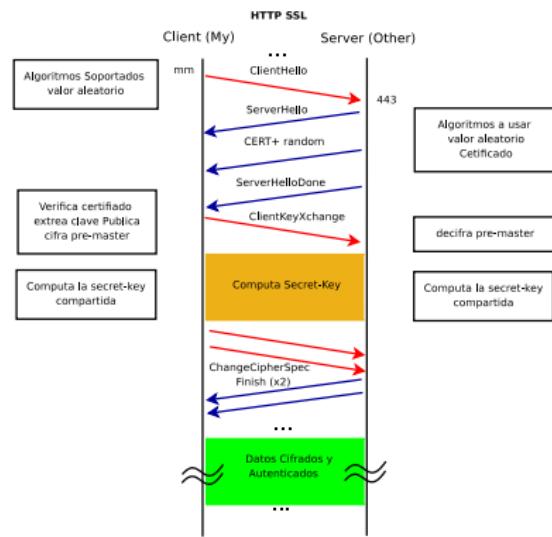
- Mecanismo que permite a las aplicaciones web del servidor “manejar estados”.
- El cliente hace un request.
- El servidor retorna un recurso (un objeto HTTP, como una página HTML) indicando al cliente que almacene determinados valores por un tiempo.
- La Cookie es introducida al cliente mediante el mensaje en el header `Set-Cookie:` mensaje que indica un par (nombre,valor).
- El cliente en cada requerimiento luego de haber almacenado la Cookie se la enviará al servidor con el header `Cookie:`.
- El servidor puede utilizarlo o no.
- El servidor puede borrarlo.
- Esta información puede ser usada por client-side scripts.

# Ejemplo de Cookie



# HTTPS (HTTP sobre TLS/SSL)

- Utiliza el port 443 por default.
- Etapa de negociación previa.
- Luego se cifra y autentica todo el mensaje HTTP (incluso el header).



# WEB-Cache

- “Proxiar” y Chachear recursos HTTP.
- Objetivos:
  - Mejorar tiempo de respuesta (reducir retardo en descarga).
  - Ahorro de BW (recursos de la red).
  - Balance de carga, atender a todos los clientes.
- Se solicita el objeto, si esta en cache y esta “fresco” se retorna desde allí (HIT).
- Si el objeto no esta o es viejo se solicita al destino y se cachea (MISS).
- Se puede realizar control de acceso.

# WEB-Cache (Cont'd)

- Cache del lado del cliente.
- Los web browser tienen sus propias cache locales.
- Los servidores agregan headers:
  - Last-Modified: date
  - ETag: (entity tag) hash
- Requerimientos condicionales desde los clientes:
  - If-Modified-Since: date
  - If-None-Match: hash
- Respuestas de los servidores:
  - 304 Not Modified.
  - 200 OK.

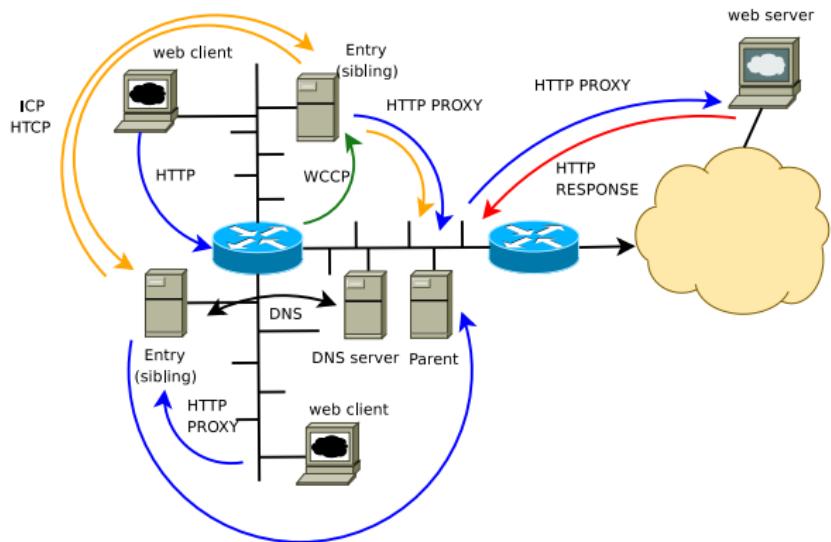
# WEB-Cache (Cont'd)

- Los cache como servers funcionan como Proxy.
- Son servidores a los clientes y clientes a los servidores web.
- Los instalan ISP o redes grandes que desean optimizar el uso de los recursos.
- Existen:
  - Proxy no-transparente.
  - Proxy transparentes.
  - Proxy en jerarquía o mesh (ICP y HTCP).
  - CDN (Content Delivery Network), funcionan por DNS.

# WEB-Cache (Cont'd)

- Protocolos de comunicación entre web-cache servers.
  - ICP (Internet Cache Protocol).
  - (HTCP) Hyper Text Caching Protocol.
- Diferentes relaciones: parent, siblings
- Protocolo de comunicación entre router y web-cache servers.
  - WCCP (Web Cache Control Protocol).
- En general corren sobre UDP.

# WEB-Cache (Cont'd)



- [StevI2] TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, (2nd. Ed). 2011. Kevin R. Fall, W. Richard Stevens.
- [KR] Computer Networking: A Top-Down Approach, Addison-Wesley, (6th Edition). 2012. Kurose/Ross.
- [LX] The Linux Home Page: <http://www.linux.org/>.
- [Siever] Linux in a Nutshell, Fourth Edition June, 2003. O'Reilly. Ellen Siever, Stephen Figgins, Aaron Weber.
- [RFC-793] <http://www.rfc-editor.org/rfc/rfc793.txt>. TCP Transmission Control Protocol (Jon Postel 1981 USC-ISI IANA).
- [HTTP0.9] The Original HTTP as defined in 1991.  
<http://www.w3.org/Protocols/HTTP/AsImplemented.html>.
- [RFC-1738] <http://www.faqs.org/rfcs/rfc1738.html>. Uniform Resource Locators (URL). (Berners-Lee, T., Masinter, L., and M. McCahill, CERN, Xerox PARC, University of Minnesota, 1994).
- [RFC-1866] <http://www.faqs.org/rfcs/rfc1866.html>. Hypertext Markup Language - 2.0. (Berners-Lee (MIT/W3C) , D. Connolly, 1995).
- [HTML30] <http://www.w3.org/MarkUp/html3/CoverPage>. Raggett, D., "HyperText Markup Language Specification Version 3.0", September 1995. (Available at
- [HTML401] <http://www.w3.org/TR/html401> Raggett, D., et al., "HTML 4.01 Specification", W3C Recommendation, December 1999.
- [XHTML1] <http://www.w3.org/TR/xhtml1>. "XHTML 1.0: The Extensible HyperText Markup Language: A Reformulation of HTML 4 in XML 1.0", W3C Recommendation, January 2000.
- [RFC-1945] <http://www.faqs.org/rfcs/rfc1945.html>. Hypertext Transfer Protocol – HTTP/1.0. (T. Berners-Lee (MIT/LCS) , R. Fielding (UC Irvine) , H. Frystyk (MIT/LCS) , 1996).
- [RFC-2068] HTTP/1.1. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee, "Hypertext Transfer Protocol HTTP/1.1", RFC 2068, 1997.
- [RFC-2616] <http://www.faqs.org/rfcs/rfc2616.html>. HTTP/1.1. (R. Fielding (UC Irvine) , J. Gettys (Compaq/W3C) , J. Mogul (Compaq) , H. Frystyk (W3C/MIT) , L. Masinter (Xerox) , P. Leach (Microsoft) , T. Berners-Lee (W3C/MIT), 1999)
- [RFC-2246] <http://www.ietf.org/rfc/rfc2246.txt>. The TLS Protocol Version 1.0. (Dierks, C. Allen (Certicom), 1999).

- [CGI1.1] <http://www.w3.org/CGI/>. The WWW Common Gateway Interface Version 1.1.
- [PUTs] Referencias al método HTTP/1.0 PUT. <http://www.apacheweek.com/features/put.html>. <http://www.w3.org/Amaya/User/Put.html>. <http://www.w3.org/Library/Examples/>.
- [APACHE] <http://httpd.apache.org/docs/2.0/server-wide.html>
- [MOZ] Mozilla Firefox. <http://www.mozilla.com>.
- [WCPP] <http://www.oreilly.com/openbook/webclient/> Web Client Programming with Perl. Clinton Wong, 1997.
- [OpenSSL] OpenSSL project: <http://www.openssl.org/>.
- [WDV] WebDAV: <http://www.webdav.org/>. RFC-4918.
- [BR1] <http://marketshare.hitslink.com>.  
<http://www.nationmaster.com>.  
<http://www.w3counter.com/globalstats.php>.  
[http://en.wikipedia.org/wiki/Usage\\_share\\_of\\_web\\_browsers](http://en.wikipedia.org/wiki/Usage_share_of_web_browsers).
- [SR1] <http://news.netcraft.com>.
- [CAIDAtnotb98] The nature of the beast: recent traffic measurements from an Internet backbone. K Claffy, caida, kc@caida.org. Greg Miller and Kevin Thompson, MCI/vBNS, gmiller,kthomp@mci.net. 1998.
- [YT] <http://www.youtube.com/>.
- [COM05] Ethereal, Wireshark. Autor original Gerald Combs, 2005.  
<http://www.ethereal.com/>.  
<http://www.wireshark.org/>.
- [SOAP] W3C Recommendation (27 April 2007). SOAP Version 1.2 Part 0: Primer (Second Edition). W3C.  
<http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>.
- [COOKIE] Cookies Spec. [http://curl.haxx.se/rfc/cookie\\_spec.html](http://curl.haxx.se/rfc/cookie_spec.html).

# ¿Qué pasa con la seguridad cuando se transmite información?

Hay algunas preguntas que podemos hacernos:

- ¿Estamos seguros que quien nos envía un mensaje es quien dice ser?
- Si alguien logra interceptar una comunicación ¿puede ver el contenido del mensaje?
- Si alguien accede al contenido del mensaje ¿puede modificarlo sin que el que lo reciba se dé cuenta?

# ¿Qué pasa con la seguridad cuando se transmite información?

Hay distintos mecanismos para proteger la información y garantizar:

- Autenticidad
- Confidencialidad
- Integridad

Depende de lo que queramos garantizar qué mecanismos vamos a usar!

# Un ejemplo de intercambio de Información Seguro

Pensemos en un servicio de mensajería de paquetes que usa un protocolo para proteger la comunicación.

El protocolo sería:

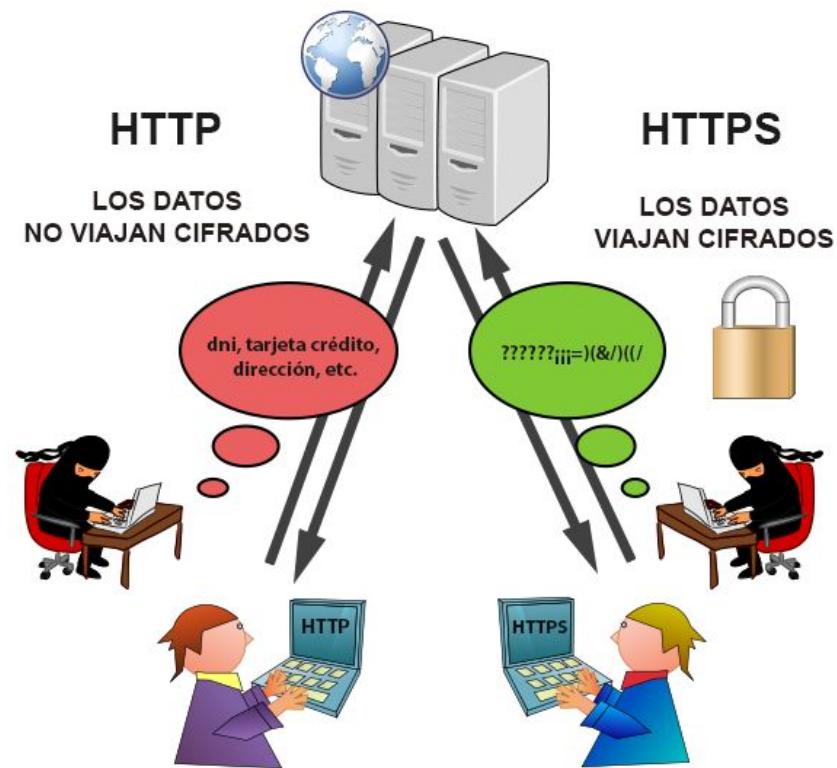
- Se usa una caja con candado para guardar los mensajes a enviar. Existe una única llave.
- Inicialmente el receptor debe hacer llegar al emisor el candado que éste debe utilizar.
- Para transmitir un mensaje secreto, el emisor guarda el mensaje en la caja y la cierra con el candado que el receptor le dio.
- El receptor recibe la caja, y luego de abrir el candado con la llave que **solo él posee** puede acceder al mensaje.



# Navegando en sitios seguros

Cuando navegamos en Internet, los protocolos de comunicación más usados son HTTP y HTTPS.

Estos protocolos son los que utilizan los navegadores para conectarse a páginas web.



# Navegando en sitios seguros - HTTPS

A diferencia de **HTTP**, el protocolo de navegación **HTTPS** provee un canal de comunicación seguro entre el cliente y el servidor.

La seguridad que ofrece **HTTPs** implica que:

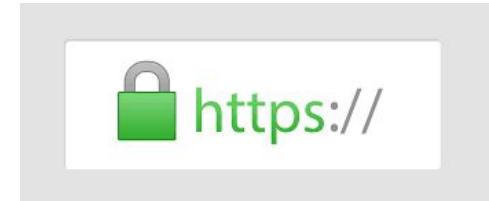
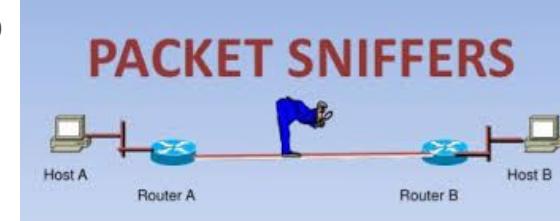
- La información viaja de manera cifrada de extremo a extremo.
- El cliente puede verificar la autenticidad del sitio visitado.
- Opcionalmente el servidor puede requerir autenticación del cliente.

# Navegando en sitios seguros - HTTPS

A diferencia de **HTTP**, el protocolo de navegación **HTTPS** provee un canal de comunicación seguro entre el cliente y el servidor.

La seguridad que ofrece **HTTPs** implica que:

- La información viaja de manera cifrada de extremo a extremo  
**Un sniffer (aplicación que permite analizar el tráfico de red con el propósito de espionar) no podrá entender los mensajes intercambiados entre el cliente y el servidor**
- El cliente puede verificar la autenticidad del sitio visitado  
**El navegador tiene la posibilidad de utilizar la criptografía asimétrica para verificar la autenticidad del sitio visitado**



# Cómo se distinguen los navegadores HTTP y HTTPS

Comparación en cómo se muestra un sitio HTTP y uno HTTPS válido

A screenshot of a web browser window titled "HTTP vs HTTPS — Test this site". The address bar shows "HTTP vs HTTPS — Test this site" and "https://httpvshttps.com". The content area displays the text "HTTP vs HTTPS Test" and "Encrypted Websites Protect Our Privacy and are Significantly Faster". Below this, it says "Compare load times of the unsecure HTTP and encrypted HTTPS versions of this page. Each test loads 360 unique, non-cached images (0.62 MB total). For fastest results, run each test 2-3 times in a private/incognito browsing session." At the bottom, there is a row of green checkmarks. To the right of the content area, there is a legend with "HTTP" and "HTTPS" and a performance summary: "1.557 s" and "88% faster than HTTP".

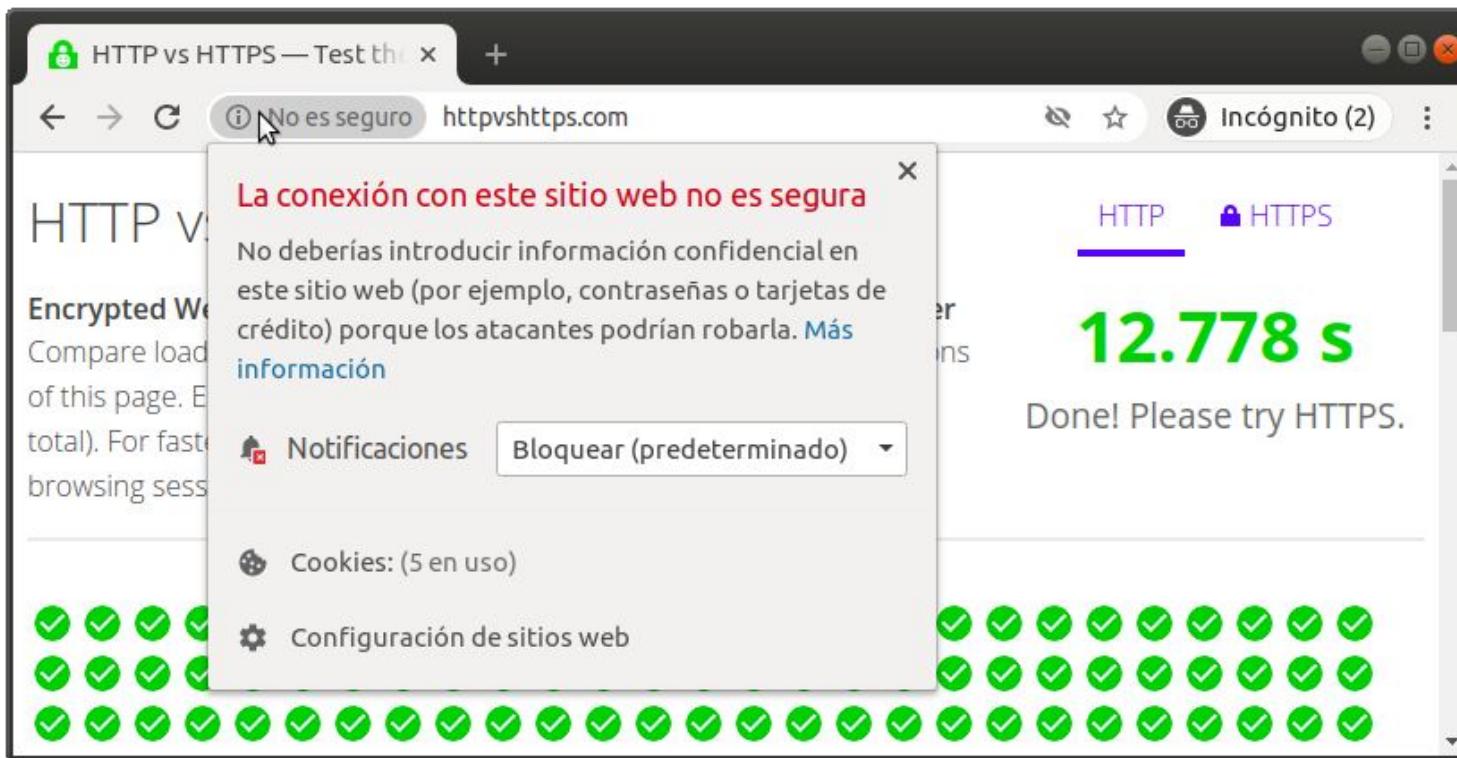
HTTP      **HTTPS**

**1.557 s**

88% faster than HTTP



# ¿Qué significa ese: No es seguro?

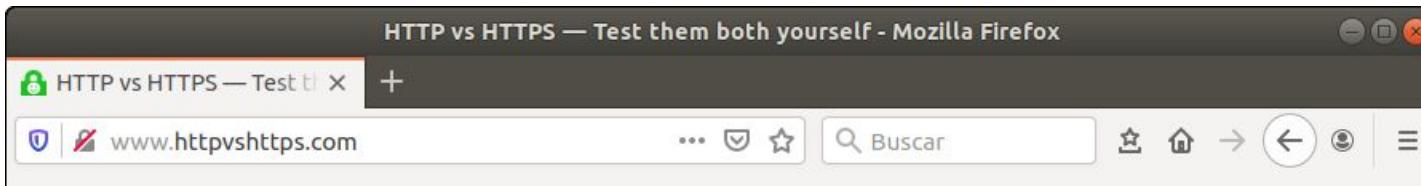


A screenshot of a web browser window titled "HTTP vs HTTPS — Test this site". The address bar shows "http://https://testthissite.com". A tooltip message "No es seguro" (Not secure) is visible over the address bar. The main content area displays a red warning message: "La conexión con este sitio web no es segura. No deberías introducir información confidencial en este sitio web (por ejemplo, contraseñas o tarjetas de crédito) porque los atacantes podrían robarla." Below this, there are dropdown menus for "Notificaciones" (Notifications) set to "Bloquear (predeterminado)" (Block) and "Cookies: (5 en uso)" (Cookies: (5 in use)). At the bottom, there is a "Configuración de sitios web" (Site settings) button. The background of the browser window features a grid of green checkmarks.



# Como se distinguen los navegadores HTTP y HTTPS

Comparación en como se muestra un sitio HTTP y uno HTTPS válido



A screenshot of a Mozilla Firefox browser window. The title bar says "HTTP vs HTTPS — Test them both yourself - Mozilla Firefox". The address bar shows "HTTP vs HTTPS — Test them both yourself" and "https://www.httpvshttps.com". The page content is identical to the HTTP version but includes a "HTTPS" link under the heading and a green "1.288 s" badge indicating a faster load time. The Firefox logo is partially visible on the right side of the window.



# Navegando en un sitio seguro

A screenshot of a Google search results page. The browser window title is "Google" and the address bar shows "google.com". The page features a colorful, whimsical illustration of various cartoon characters, including a lion, a monkey, and a bear, playing with kites and a sailboat on a beach-like setting. At the top right of the page are links for "Gmail", "Imágenes", and a "Iniciar sesión" button. Below the illustration is a search bar with a magnifying glass icon and a microphone icon for voice search. At the bottom of the page, there are navigation links for "Publicidad", "Negocios", "Sobre Google", "Cómo funciona la Búsqueda", "Buscar con Google", "Mostrar con cuadro", "Privacidad", "Condiciones", and "Preferencias".



# Navegando en un sitio seguro

Google

google.com

Incógnito

La conexión es segura

Tu información (por ejemplo, las contraseñas o los números de las tarjetas de crédito) es privada cuando se envía a este sitio web. [Más información](#)

Ubicación: Bloquear

Notificaciones: Bloquear (predeterminado)

Certificado (válido)

Cookies: (6 en uso)

Configuración de sitios web

Gmail Imágenes Iniciar sesión

Publicidad

Privacidad Condiciones Preferencias

This screenshot shows a Google search results page for 'google.com'. A security warning overlay is displayed, stating 'La conexión es segura' (The connection is secure) and explaining that user information like passwords or credit card numbers are private when sent to this site. It includes links to 'Más información' and 'Bloquear' (Block) for location and notifications. Below the overlay, the main search results page is visible, featuring a colorful illustration of children playing with paper boats and balloons. Navigation bars for Gmail, Images, and Sign In are at the top right, along with a 'Publicidad' (Advertisement) section. At the bottom, there are links for Privacy, Conditions, and Preferences.



# Datos del certificado del sitio visitado

The screenshot shows the 'Visor de certificados' (Certificate Viewer) for the domain \*.google.com. The 'General' tab is selected. Key information displayed includes:

- Este certificado se ha verificado para los siguientes usos:** Certificado de servidor SSL.
- Enviado a**:
  - Nombre común (CN): \*.google.com
  - Organización (O): Google LLC
  - Unidad organizativa (OU): <No incluido en el certificado>
- Emitido por**:
  - Nombre común (CN): GTS CA 1 O1
  - Organización (O): Google Trust Services
  - Unidad organizativa (OU): <No incluido en el certificado>
- Período de validez**:
  - Emitido el: miércoles, 15 de julio de 2020, 5:29:16
  - Vencimiento el: miércoles, 7 de octubre de 2020, 5:29:16
- Huellas digitales**:
  - Huella digital SHA-256: F4 57 71 F0 87 49 49 FE 10 C7 2F 56 9D 7B CA 4A  
24 C6 9B 9B FC 2F 30 11 D8 F2 5A 5B 43 29 CE 01
  - Huella digital SHA-1: F8 94 0F 5D C6 9E 48 AB 4A D8 FA 83 93 B0 53 A1  
2E 7D 4C 54

Datos de la identidad del sitio para el cual fue emitido este certificado.  
**\*.google.com** vale para diferentes sitios en el dominio **.google.com**

Datos sobre el período de validez del presente certificado.

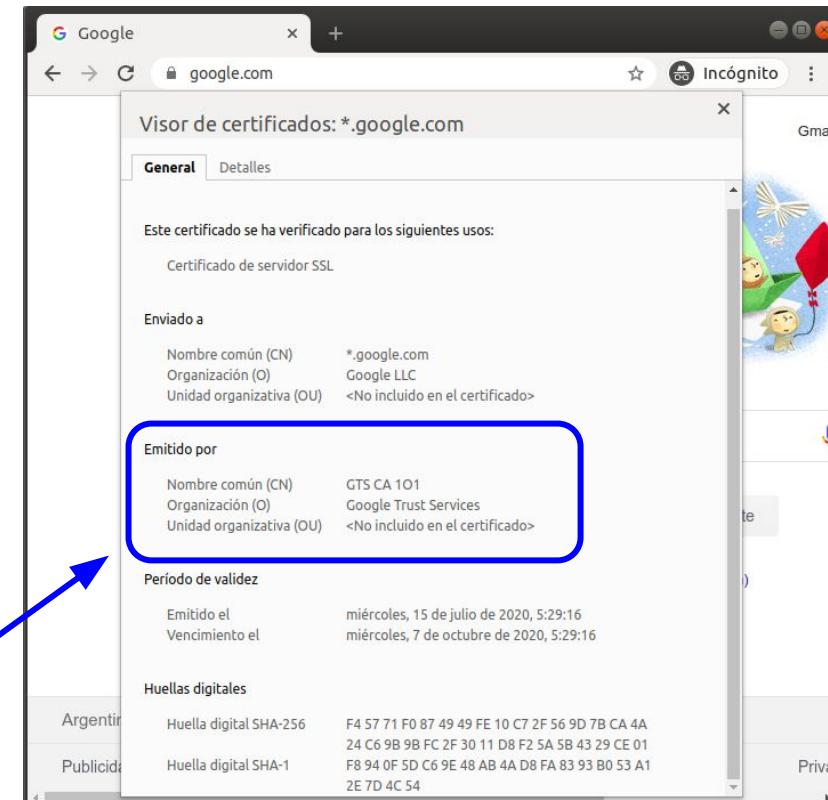
# El certificado está firmado digitalmente por una CA

Los **certificados digitales**, son reconocidos como válidos, porque fueron emitido por una **Autoridad de certificación en la que confiamos**.

La autoridad de certificación es quien asegura que el par de claves pertenece a determinado sitio web.

Un certificado digital, está firmado por la autoridad de certificación.

**Entidad en la que confiamos!!!**



# Algunas Autoridades de Certificación en las que Chrome confía

Por defecto nuestros los navegadores vienen con conjunto de **Autoridades de certificación en la que se confía**.

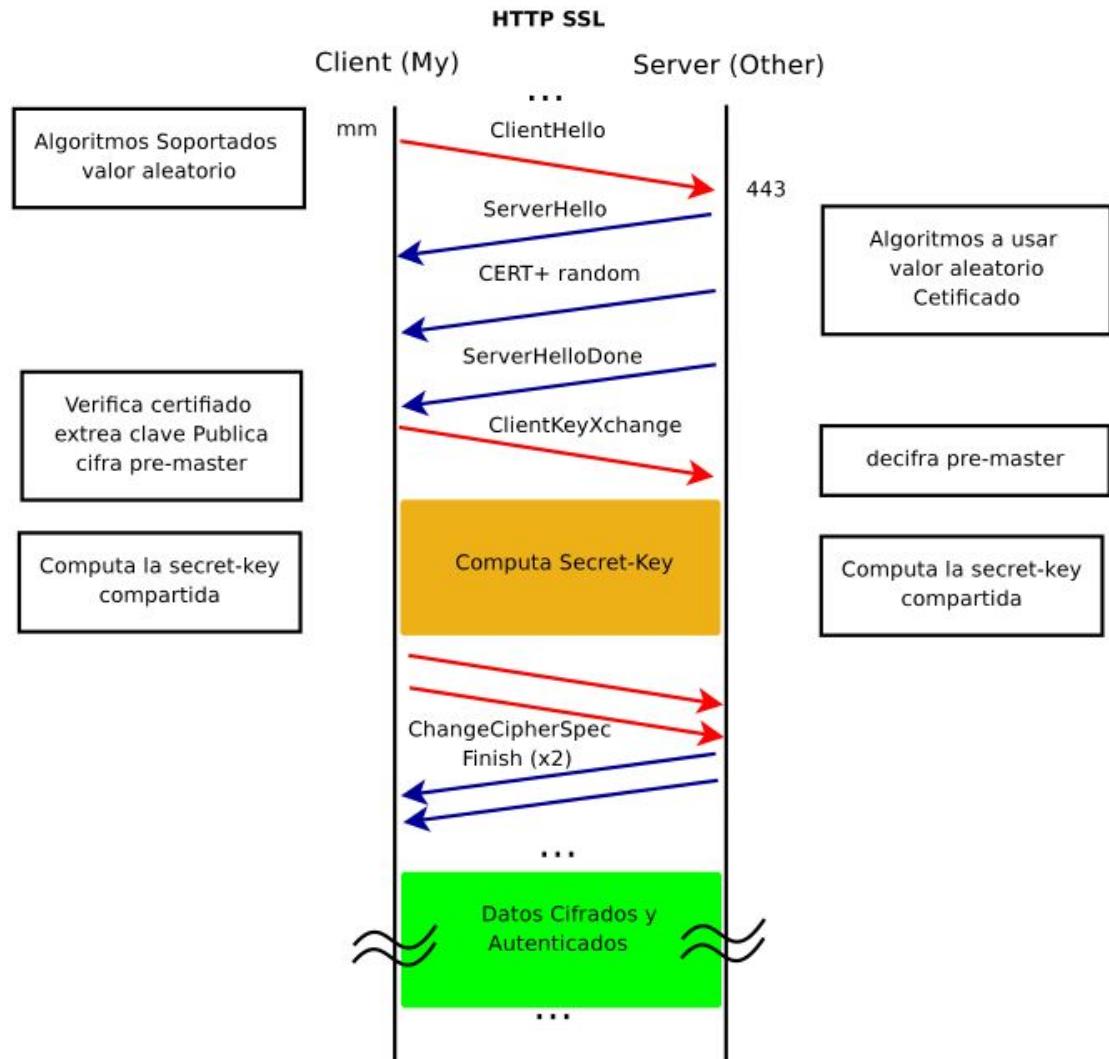
Nosotros podríamos agregar o borrar alguna **autoridad de certificación**, pero no es algo que se suela hacer.

org-E-Tuğra EBG Bilişim Teknolojileri ve Hizmetleri A.Ş.	▼
org-Entrust, Inc.	▼
org-Entrust.net	▼
org-FNMT-RCM	▼
org-GeoTrust Inc.	▼
org-GlobalSign	▲
GlobalSign	⋮
org-GlobalSign nv-sa	▼
org-GoDaddy.com, Inc.	▼
org-Government Root Certification Authority	▼
org-GUANG DONG CERTIFICATE AUTHORITY CO.,LTD.	▼
org-Hellenic Academic and Research Institutions Cert. Authority	▼
org-Hongkong Post	▼

# HTTPS

## HTTP sobre TLS/SSL

- Utiliza por defecto el puerto 443
- Lleva a cabo un handshake (negociación) antes del intercambio de datos.



# Otros protocolos seguros

El protocolo SSL/TLS tiene multitud de aplicaciones en uso actualmente. La mayoría de son versiones seguras de programas que emplean protocolos que no lo son. Ejemplos:

- SSH utiliza SSL/TLS por debajo.
- SMTP y NNTP pueden operar también de manera segura sobre SSL/TLS.
- POP3 e IMAP4 sobre SSL/TLS son POP3S i IMAPS.

# Problemas cuando usamos HTTPs

Cuando accedemos a un sitio HTTPs, se pueden presentar distintos problemas que no permiten al navegador asegurar la identidad del sitio visitado.

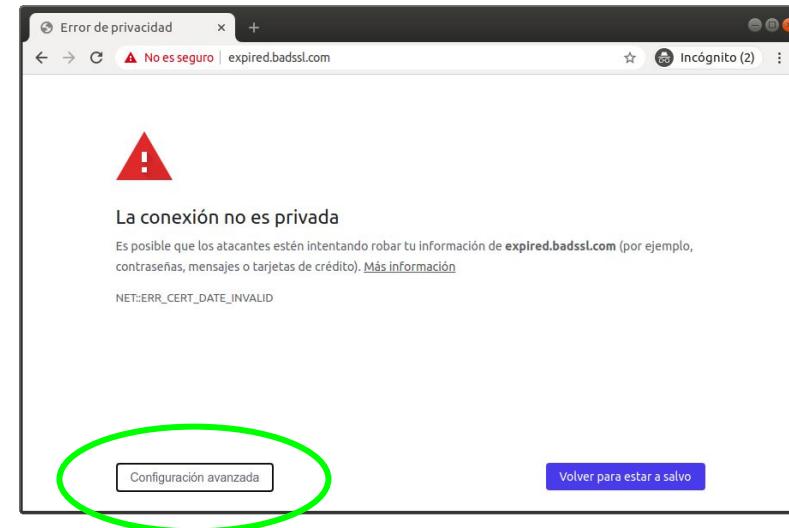
Cuando ocurre alguna de estas situaciones, el navegador alerta al usuario. Es importante que el usuario entienda que el problema es que **no se puede asegurar la identidad del sitio visitado**.

Estos problemas pueden deberse tanto a errores en la configuración del sitio como así también a ataques de phishing contra los usuarios.

# Más información

En caso que no se trate de un phishing, algunas situaciones en las que podemos ver este tipo de alertas es cuando:

- El certificado está vencido.
- No coincide la URL del sitio visitado con la identidad del certificado presentado por el sitio web.
- El certificado está autofirmado.
- El certificado está firmado por una CA en la que no se confía.
- Cuidado con la fecha/hora del sistema!!



# Problemas cuando usamos HTTPs

En estos dos videos pueden encontrar más información para complementar:

- <https://youtu.be/tHhFQaurGAg> (muy básico y sencillo)
- <https://youtu.be/pOeWmStBOYY> (un poquito más detallado)
- <https://www.websecurity.digicert.com/es/es/security-topics/what-is-ssl-tls-https> (un poco más de info en Digicert!)

# HTTP/2

Redes y Comunicaciones (v0.2)

# Qué es HTTP/2?

- Reemplazo de cómo HTTP se transporta.
- No es un reemplazo del protocolo completo.
- Se conservan métodos y semántica.
- Base del trabajo protocolo desarrollado por Google SPDY/2.
- Definido en:
  - RFC7540: Hypertext Transfer Protocol version 2.
  - RFC7540: HPACK - Header Compression for HTTP/2 RFC7541.
- Otro protocolo HTTP/3, basado en HTTP over QUIC(UDP).

# Problemas con HTTP/1.0, HTTP/1.1

- Un request por conexión, por vez, muy lento.
- Alternativas (evitar HOL):
  - Conexiones persistentes y pipelining.
  - Generar conexiones paralelas.
- Problemas:
  - Pipelining requiere que los responses sean enviado en el orden solicitado, HOL posible.
  - POST no siempre pueden ser enviados en pipelining.
  - Demasiadas conexiones genera problemas, control de congestión, mal uso de la red.
  - Muchos requests, muchos datos duplicados (headers).

# Diferencias principales con HTTP/1.1

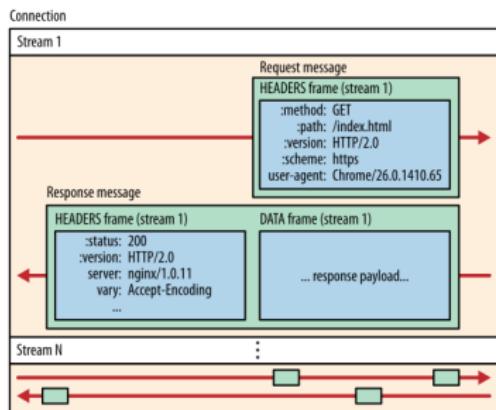
- Protocolo binario en lugar de textual(ASCII), binary framing: (más eficiente).
- Multiplexa varios request en una petición en lugar de ser una secuencia ordenada y bloqueante.
- Utilizar una conexión para pedir/traer datos en paralelos, agrega: datos fuera de orden, priorización, flow control por frame.
- Usa compresión de encabezado.
- Permite a los servidores “pushear” datos a los clientes.
- La mayoría de las implementaciones requieren TLS/SSL, no el estándar.

# HTTP/2 mux stream, framing

- Puede generar una o más conexiones TCP. Trata de aprovechar las que tiene establecidas.
- Un stream es como una sub-conexión (una “conexión” http2 dentro de una conexión TCP).
- Un stream tiene un ID y una prioridad(alternativa) y son bidireccionales.
- Sobre una conexión TCP multiplexa uno o más streams (“conexiones http2”).
- Los streams transportan mensajes.
- Los mensajes http2 (Request, Response) se envían usando un stream.
- Los mensajes http2 son divididos en frames dentro del mismo stream.
- Un frame es una porción de mensaje: header fijo+payload variable (unidad mínima).
- El mismo stream puede ser usado para llevar diferentes msj.

# HTTP/2 mux stream, framing (cont.)

- Los mensajes están compuestos por frames, que podrían ser de diferentes tipos.
- Los streams van en una misma conexión.
- Los streams son identificados y divididos en frames.
- Frame types: HEADERS, DATA, PUSH\_PROMISE, WINDOW\_UPDATE, SETTINGS, etc.



fuente: <https://web.dev/performance-http2/#streams,-messages,-and-frames>

# HTTP/2 mux stream, framing (cont.)

http2\_native.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http2

No.	Time	Source	Destination	Protocol	Length Info
4	0.414129282	192.168.0.228	139.162.123.134	HTTP2	96 Magic
5	0.420556333	192.168.0.228	139.162.123.134	HTTP2	152 SETTINGS[0], WINDOW_UPDATE[0], HEADERS[1]: G
7	0.839101976	139.162.123.134	192.168.0.228	HTTP2	100 SETTINGS[0]
10	0.839139220	139.162.123.134	192.168.0.228	HTTP2	76 SETTINGS[0]
12	0.839167233	139.162.123.134	192.168.0.228	HTTP2	369 HEADERS[1]: 200 OK, DATA[1] (text/plain)
14	0.839280935	192.168.0.228	139.162.123.134	HTTP2	75 SETTINGS[0]
15	0.845040952	192.168.0.228	139.162.123.134	HTTP2	93 HEADERS[3]: GET /humans.txt
16	1.161001606	139.162.123.134	192.168.0.228	HTTP2	124 HEADERS[3]: 404 Not Found, DATA[1] (text/plain)

```

> Transmission Control Protocol, Src Port: 38564, Dst Port: 80, Seq: 25, Ack: 1, Len: 86
> HyperText Transfer Protocol 2
  > Stream: SETTINGS, Stream ID: 0, Length 18
  > Stream: WINDOW_UPDATE, Stream ID: 0, Length 4
  > Stream: HEADERS, Stream ID: 1, Length 37, GET /robots.txt
    Length: 37
    Type: HEADERS (1)
    Flags: 0x05
    0... .... .... .... .... = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
    [Pad Length: 0]
    Header Block Fragment: 82048862c3c674a174f94f864188aa69d29ac4b9ec9b7a88...
    [Header Length: 136]
    [Header Count: 6]
    > Header: :method: GET
    > Header: :path: /robots.txt
    > Header: :scheme: http
    > Header: :authority: ngnhttp2.org
    > Header: user-agent: curl/7.61.0
    > Header: accept: */*

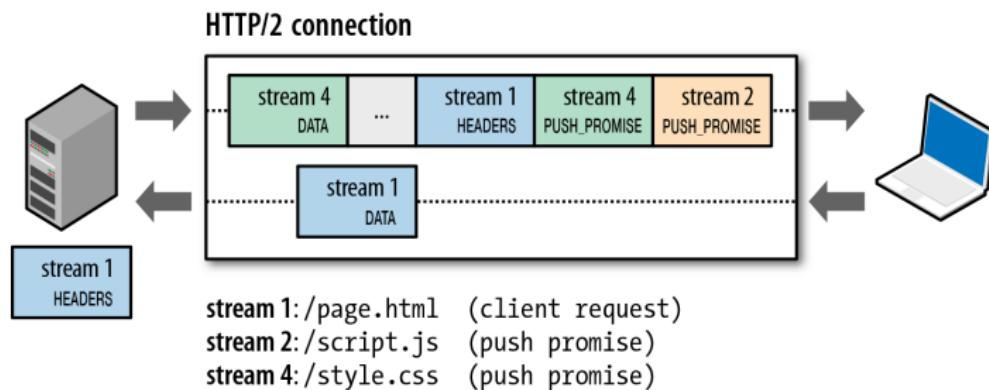
```

HyperText Transfer Protocol 2: Protocol

Packets: 21 - Displayed: 8 (38.1%)

Profile: Classic

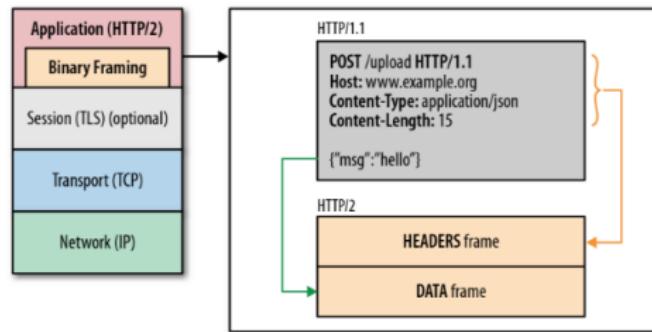
# HTTP/2 mux stream, framing (cont.)



fuente: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>

# HTTP/2 mux stream, framing (Cont.)

- Streams codificados en binario y cada frame con header común fijo(9B).



fuente: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>

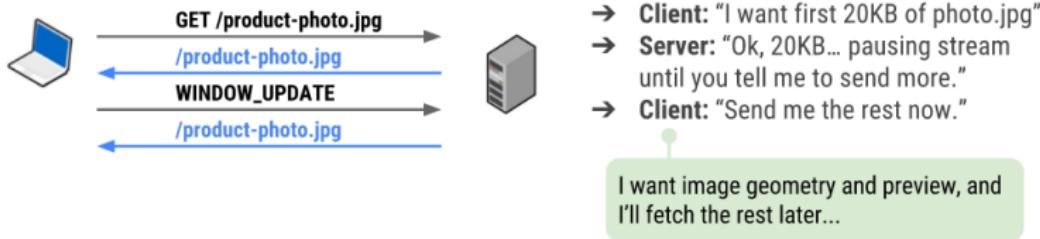
# HTTP/2 HEADERS

- Se mantienen casi todos los HEADERs de HTTP/1.1.
- No se codifican más en ASCII.
- Surgen nuevos pseudo-headers que contiene información que estaba en el método y otros headers.
- Por ejemplo: HEAD /algo HTTP/1.1 se reemplaza con http2:
  - :method: head
  - :path: /algo
  - :scheme: https o http
  - :authority: www.site.com reemplaza al headerHost:.

Para las respuestas: :status: códigos de retornos 200, 301, 404, etc.

# HTTP/2 priorización y flow-control

- Los streams dentro de una misma conexión tienen flow-control individual.
- Los streams pueden tener un weight (prioridad).
- Los streams pueden estar asociados de forma jerárquica, dependencias.



fuente: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>

# HTTP/2 inline vs. push

- Cuando el cliente solicita una página, “parsea” el primer response HTML luego solicita el resto.
- El server puede enviar el HTML más otros datos, por ejemplo CSS o Javascript.
- No siempre es lo que necesita el cliente, depende de que funcionalidad ofrece.

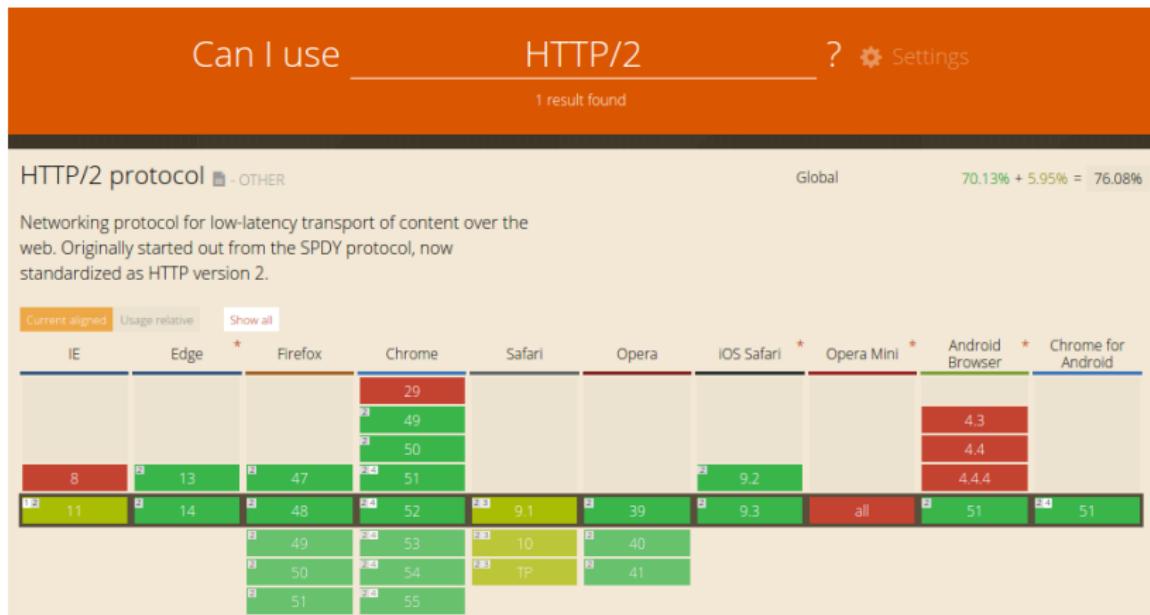


fuente: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>

# Compresión y Soporte

- Compresión de encabezados.
- SPDY/2 propone usar GZIP.
- GZIP + cifrado, tiene “bugs” utilizados por atacantes.
- Se crea un nuevo compresor de Headers: HPACK.
- H2 y SPDY, soportados en la mayoría de los navegadores.

# Soporte en clientes para 2016



fuente: <http://caniuse.com/#search=HTTP%2F2>

# Otras Características

- HTTP/1.1, posibilidad de hacer un upgrade durante la conexión:  
**Upgrade Header.** Connection: Upgrade, HTTP2-Settings  
Upgrade: h2c|h2.
- http2: Negociar el protocolo de aplicación:  
**ALPN:** Application-Layer Protocol Negotiation.  
Se negocia como extensión de SSL en Hello (Anteriormente NPN). Se ofrece: h2, h3, http/1.1, ...
- Posibilidad de negociar protocolo alternativo:  
**Alterantive Service:** alt-svc.

# Application-Layer Protocol Neg.

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Length	Info
4	0.000114	127.0.0.1	127.0.0.1	56	[TCP Window Update] 8443->61946 [ACK] Seq=1
5	0.000285	127.0.0.1	127.0.0.1	461	Client Hello
6	0.000328	127.0.0.1	127.0.0.1	56	8443->61946 [ACK] Seq=1 Ack=406 Win=146576
7	0.001662	127.0.0.1	127.0.0.1	1122	Server Hello, Certificate, Server Hello Done

▼ Extension: Application Layer Protocol Negotiation  
 Type: Application Layer Protocol Negotiation (0x0010)  
 Length: 52  
 ALPN Extension Length: 50

▼ ALPN Protocol  
 ALPN string length: 17  
 ALPN Next Protocol: HTTP-draft-04/2.0  
 ALPN string length: 8  
 ALPN Next Protocol: spdy/4a2  
 ALPN string length: 8  
 ALPN Next Protocol: spdy/3.1  
 ALPN string length: 6  
 ALPN Next Protocol: spdy/3  
 ALPN string length: 6  
 ALPN Next Protocol: spdy/2

▼ Extension: status\_request  
 Type: status\_request (0x0005)  
 Length: 5

File: "/home/andres/docs/mvdoc... | Packets: 202 - Disp... | Profile: Default

# Debugging (2016)

Google - Google Chrome

Google https://www.google.com.ar/#gfe\_rd=cr&ei=6XPEV9nsCc0gxg55... Andres

Google Argentina

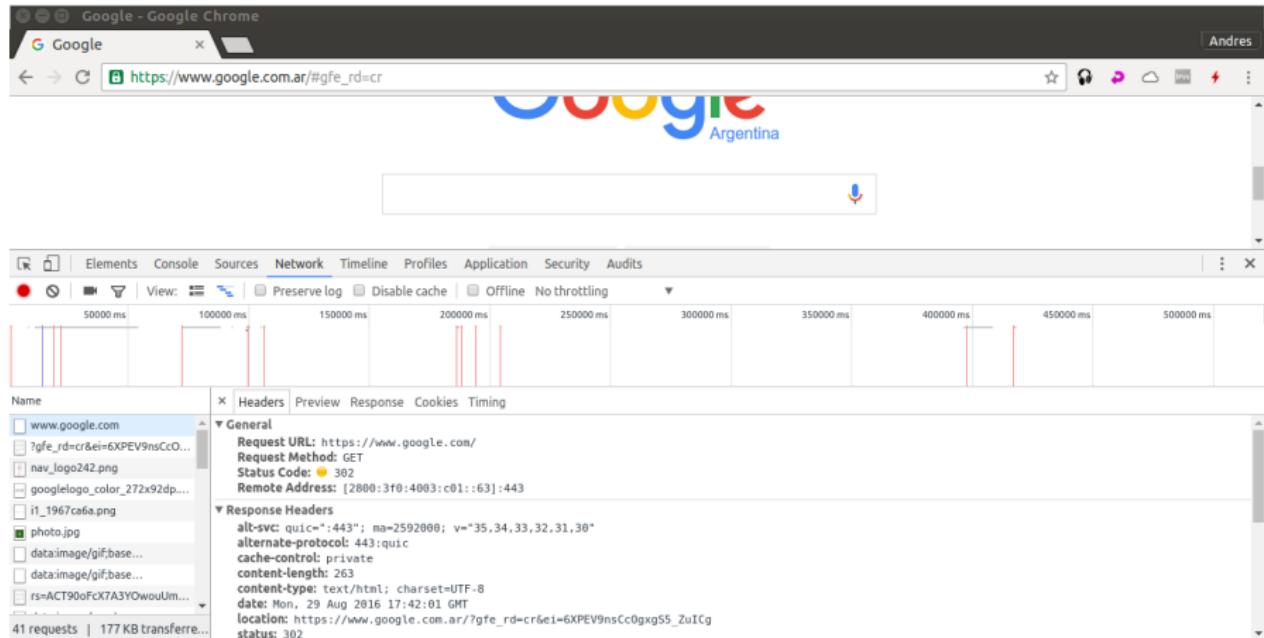
Network Timeline Profiles Application Security Audits

View: Preserve log Disable cache Offline No throttling

Name	Method	Status	Protocol	Type	Initiator	Size	Time	Timeline – Start Time	4.00 s	6.00 s
www.google.com	GET	302	h2	text/html	Other	399 B	33 ms			
?gfe_rd=cr&ei=6XPEV9nsCc0gxg55...	GET	200	quic/1+spdy/3	document	https://www.goog...	62.7 KB	335 ms			
nav_logo242.png	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...	2 ms			
googlelogo_color_272x92dp.png	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...	3 ms			
l_1967ca6a.png	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...	5 ms			
photo.jpg	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...	4 ms			
data:image/gif;base64...	GET	200	data	gif	?gfe_rd=cr&ei=6X...	(from cac...	0 ms			
data:image/gif;base64...	GET	200	data	gif	?gfe_rd=cr&ei=6X...	(from cac...	0 ms			

25 requests | 64.3 KB transferred | Finished: 7.34 s | DOMContentLoaded: 536 ms | Load: 535 ms

# Debugging (Cont.)



The screenshot shows the Google Chrome browser interface with the Network tab selected in the developer tools. The address bar displays the URL [https://www.google.com.ar/#gfe\\_rd=cr&ei=6XPEV9nsCc0gxg55\\_ZuICg](https://www.google.com.ar/#gfe_rd=cr&ei=6XPEV9nsCc0gxg55_ZuICg). The main content area shows the Google Argentina homepage. The Network tab timeline shows several requests, with the first one being a GET request to `www.google.com` with a status code of 302.

Name	Headers	Preview	Response	Cookies	Timing
<code>www.google.com</code>	General				
	Request URL: <a href="https://www.google.com/">https://www.google.com/</a>				
	Request Method: GET				
	Status Code: 302				
	Remote Address: [2800:3f0:4003:c01::63]:443				
<code>?gfe_rd=cr&amp;ei=6XPEV9nsCc0gxg55_ZuICg</code>	Response Headers				
	alt-svc: quic=":443"; ma=2592000; v="35,34,33,32,31,30"				
	alternate-protocol: 443:quic				
	cache-control: private				
	content-length: 263				
	content-type: text/html; charset=UTF-8				
	date: Mon, 29 Aug 2016 17:42:01 GMT				
	location: <a href="https://www.google.com.ar/?gfe_rd=cr&amp;ei=6XPEV9nsCc0gxg55_ZuICg">https://www.google.com.ar/?gfe_rd=cr&amp;ei=6XPEV9nsCc0gxg55_ZuICg</a>				
41 requests   177 KB transferred...					

# Debugging (Cont.)

chrome://net-internals/#http2

Capturing halted

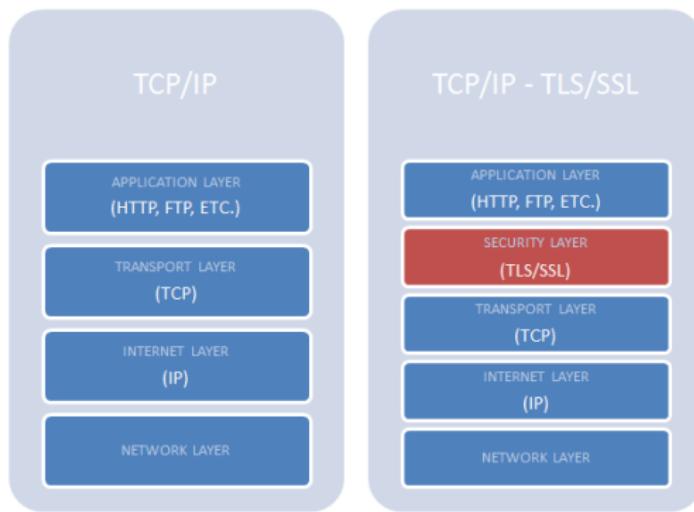
- Export
- Import
- Proxy
- Events
- Timeline
- DNS
- Sockets
- Alt-Svc
- HTTP/2**
- QUIC
- SDCH
- Cache
- Modules
- HSTS
- Bandwidth
- Prerender

HTTP/2 sessions

[View live HTTP/2 sessions](#)

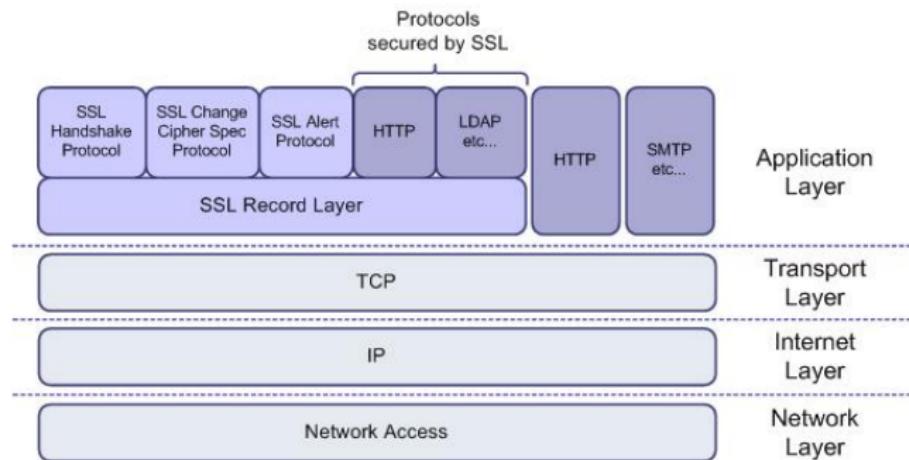
Host	Proxy	ID	Protocol Negotiated	Active streams	Unclaimed pushed	Max	Initiated	Pushed	Pushed and claimed	Abandoned	Received frames	Secure	Sent settings	Received settings
accounts.google.com:443	direct://	231	h2	0	0	100	0	0	0	0	0	true	true	true
s2.googleusercontent.com:443	direct://	370	h2	0	0	100	0	0	0	0	0	true	true	true
ssl.google-analytics.com:443	direct://	265	h2	0	0	100	0	0	0	0	0	true	true	true
ssl.gstatic.com:443	direct://	450	h2	0	0	100	0	0	0	0	0	true	true	true
www.google-analytics.com:443	direct://	339	h2	0	0	100	0	0	0	0	0	true	true	true
www.google.com:443	direct://	516	h2	0	0	100	0	0	0	0	0	true	true	true
www.google.com:443	direct://	335	h2	0	0	100	0	0	0	0	0	true	true	true
accounts.google.com:443	direct://	328	h2	0	0	100	0	0	0	0	0	true	true	true
clients2.google.com:443	direct://	654	h2	0	0	100	0	0	0	0	0	true	true	true
www.googleapis.com:443	direct://	120	h2	0	0	100	0	0	0	0	0	true	true	true

# SSL/TLS



fuente: <https://www.simple-talk.com/dotnet/net-framework/tlssl-and-net-framework-4-0/>

# SSL/TLS



fuente: [http://nicolascormier.com/documentation/bin/apache/apache2\\_with\\_ssl\\_tls/part1.htm](http://nicolascormier.com/documentation/bin/apache/apache2_with_ssl_tls/part1.htm)

[HTTP/2] [https://http2.github.io/.](https://http2.github.io/)

[Ilya Grigorik] HTTP/2 is here, let's optimze!

[https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19.](https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19)

[HTTP/2-dev] <https://web.dev/performance-http2/#streams,-messages,-and-frames>

[HTTP/2-undertow] <https://undertow.io/blog/2015/04/27/An-in-overview-of-HTTP2.html>

# Protocolo DNS

Redes y Comunicaciones

# Historia

- Internet: necesidad de utilizar nombres en lugar de direcciones IP.
- URL:  
<https://www.info.unlp.edu.ar/resena-historica/index.php>.
- Mecanismos para mapear nombres de internet (nombres de dominio) a dir. IP.
- 1973, archivo global, HOSTS.TXT, SRI (Stanford Research Institute, hoy SRI International) RFC-606, [RFC-608].
- 1980 el servicio muy difícil de mantener y no escalable.
- 1983 En USC, se desarrolla DNS (Domain Name System): [RFC-882],[RFC-883].
- 1984 Primeras implementaciones Unix BSD.
- Modificaciones: [RFC-1034], [RFC-1035], etc.

# Aspectos y Elementos de DNS

- Espacio de nombres (sintaxis y zonas, dominios).
- Procedimiento de Delegación y Arquitectura.
- Base de datos distribuida y Servidores.
- Define las componentes y el protocolo para su comunicación.
- Procedimiento de búsqueda/resolución (Protocolos).

# Elementos de DNS, FQDN

- Nombre de dominio FQDN: lista de etiquetas (labels) separadas por puntos.
- Se leen desde el nodo/etiqueta de la izquierda hasta la raíz del árbol (el punto), estructura jerárquica con sub-nombres (niveles).
  - La sintaxis jerárquica refleja la delegación de autoridad.
  - No son case-sensitive, cada etiqueta Máximo 63 chars.
  - Max etiquetas 127, nombre max. 255 chars, acepta valores internacionales, UTF-8, Unicode.

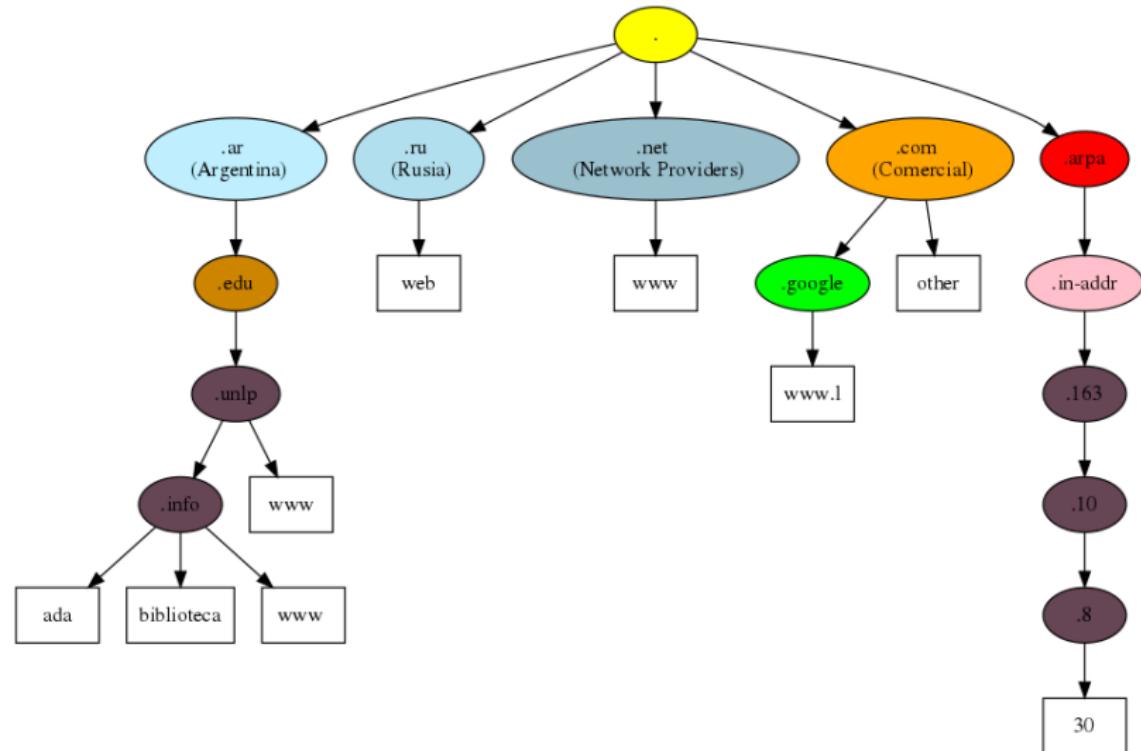
www, biblioteca, ada (No FQDN)

www.info.unlp.edu.ar. , biblioteca.info.unlp.edu.ar. (FQDN)

ada.info.unlp.edu.ar. (FQDN)

www.info.unlp.edu.ar (Considerado FQDN)

# Esquema de Nombres de DNS



# TLDs (Top Level Domains)

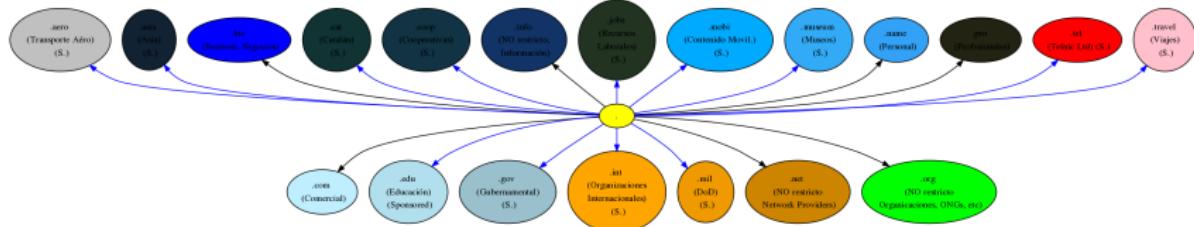
- Los TLDs se podrían clasificar en 3 grupos:

**gTLDs, Generic TLDs:** contienen dominios con propósitos particulares, de acuerdo a diferentes actividades. políticas definidas por el ICANN: **Unsponsored TLD** o definidas por otra organización: **Sponsored TLD**.

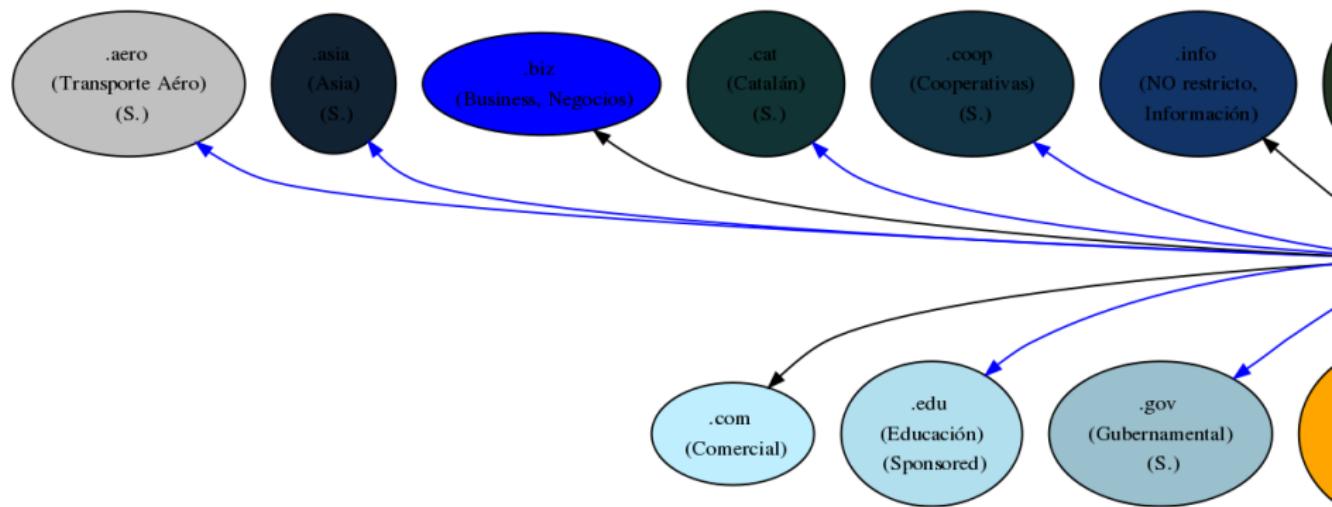
**ccTLD Country-Code TLDs:** contienen dominios delegados a los diferentes países del mundo. ISO 3166-1 alfa-2.

**.ARPA TLD:** es un dominio especial, usado internamente para resolución de reversos.

# Generic TLDs antes de new gTLDs



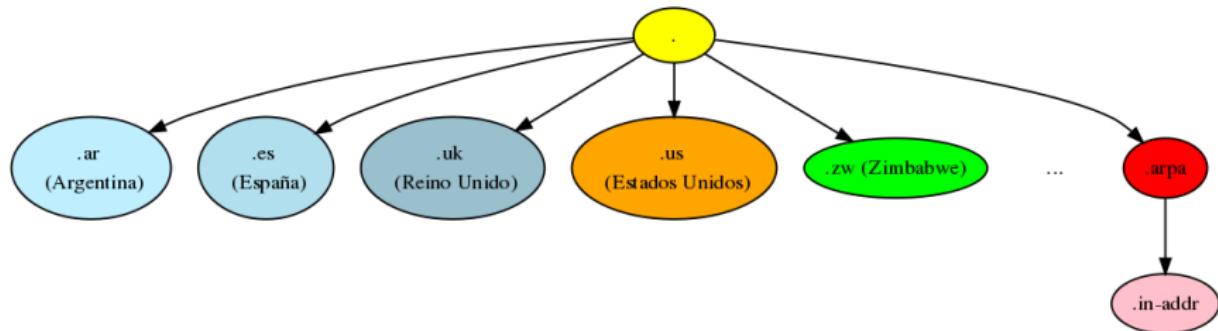
# Generic TLDs antes de new gTLDs (Cont'd)



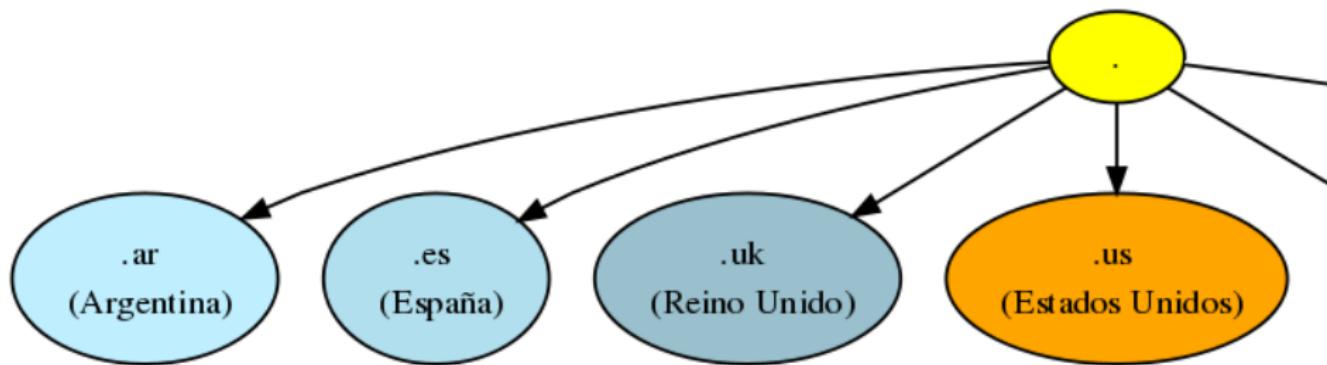
# Generic TLDs actualmente

- A partir de 2012 se comenzó a aceptar nuevas aplicaciones para new gTLDs.
- Proceso de licitación, donde hay en juego grandes sumas de dinero.
- Nuevos dominios registrados (500+):
  - .academy, .casa, ...
  - .beer, .bike, .futbol, ...
  - .pizza, .paris
  - .wiki, .viajes, ...
  - ...

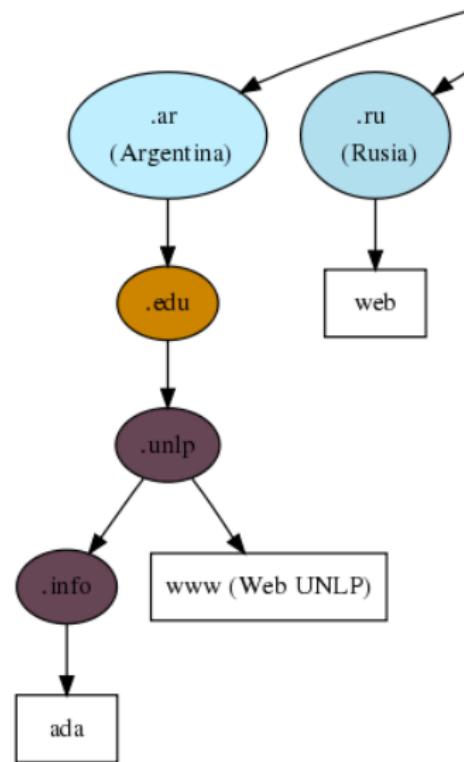
# Country Code y ARPA TLD



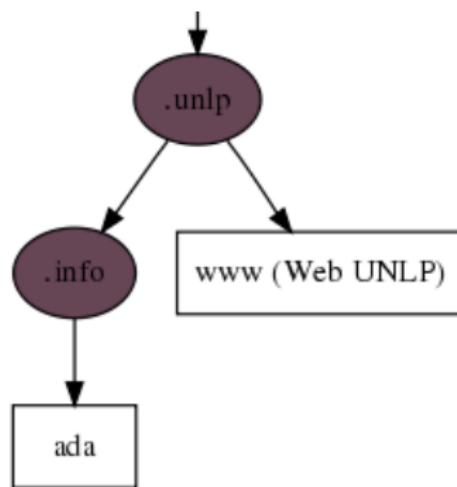
# Country Code TLD



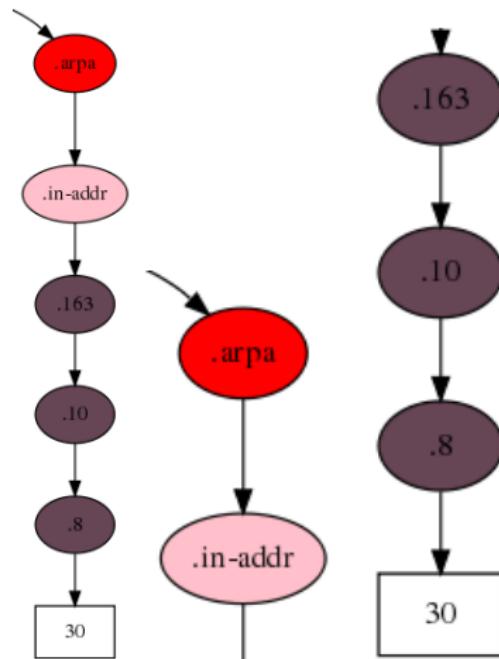
# Esquema de Nombres de DNS (CC)



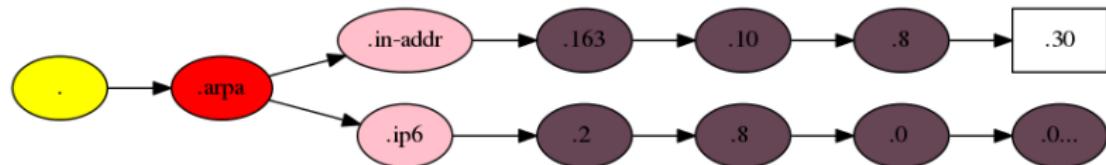
# Esquema de Nombres de DNS (CC Cont'd)



# Esquema de Nombres de DNS (ARPA Cont'd)



## Esquema de Nombres de DNS (ARPA Cont'd)



```
$ host 163.10.8.30 30.8.10.163.in-addr.arpa domain name pointer  
host163-10-8-30.presi.unlp.edu.ar.
```

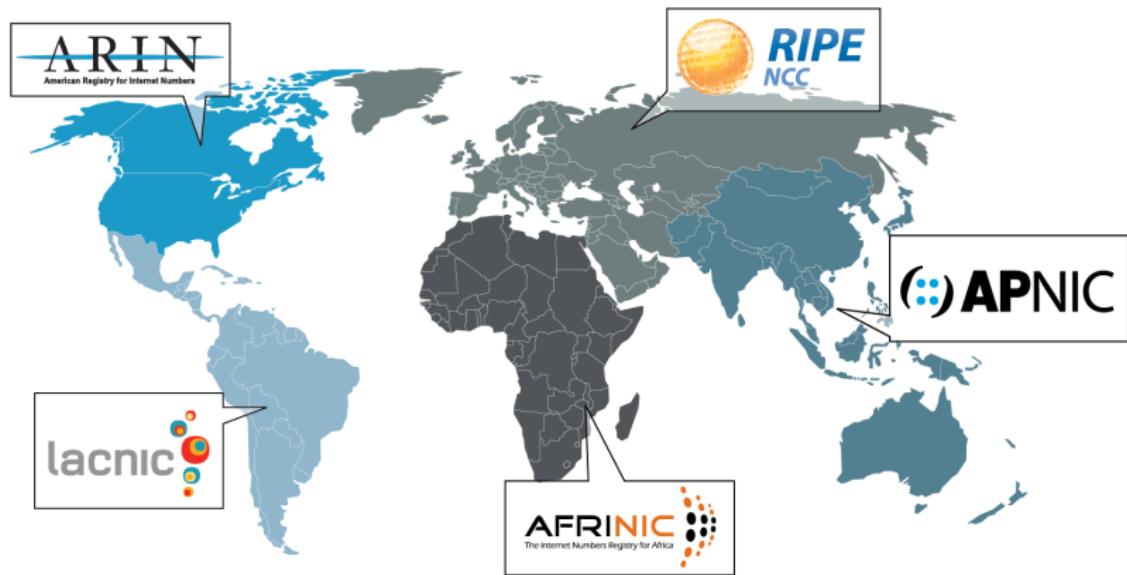
```
$ host 2800:340:0:64::145
```

5.4.1.0.0.0.0.0.0.0.0.0.0.0.0.0.4.6.0.0.0.0.0.0.0.0.4.3.0.0.0.8.2.ip6.arpa  
domain name pointer www.unlp.edu.ar.

# Organización en el DNS

- Recursos de Internet: Sistema distribuido pero regido por organizaciones.
- Organización mediante dominios, sub-dominios y host o servicios (jerárquico).
- IANA a través del ICANN (Internet Corporation for Assigned Names and Numbers) controla el funcionamiento.
- Organizaciones paralelas: Open Root Server Network (ORSN), OpenNIC.
- ICANN homologa y delega gTLD a DNS Registrars y ccTLD a países.
- RIRs (Regional Internet Registers):
  - American Registry for Internet Numbers (ARIN).
  - RIPE NCC -Europa y parte de Asia- (RIPE).
  - Asia-Pacific Network Information Centre (APNIC).
  - Latin American and Caribbean NIC (LACNIC).
  - African Network Information Centre (AfriNIC).
- Nombres se delegan a países y a registrars, direcciones IP no.

# Organización del DNS (Cont'd)



# Ejemplo: Delegación de autoridad

- **ada.info.unlp.edu.ar**
- “Ada” fue registrada por la administración de la red de la Facultad de Informática.
- El administrador de la Facultad obtuvo previamente la autoridad sobre el dominio “info.unlp.edu.ar”. a partir de la administración de la universidad UNLP.
- La Universidad obtuvo autoridad sobre el dominio “unlp.edu.ar” a partir de la administración de “edu.ar”, RIU (Red Inter-universitaria).

# Ejemplo: Delegación de autoridad (Cont'd)

- La RIU obtuvo autoridad sobre “edu.ar” a partir de la delegación de NIC.AR que depende de alguna organización de gobierno, como la Sec. Legal y Técnica u otro ente a cargo de “.AR” (Argentina).
- La administración de nombres en la Argentina, sea la Secretaría Legal y Técnica u otro ente obtuvo la autoridad delegada a partir del IANA o ICANN.

# Base de Datos Distribuida

- Mantener gran cantidad de información.
- Control de la información distribuido (delegación).
- Tolerante a fallos y escalable.
- Modelo de acceso (altamente cacheable):
  - Muchas lecturas.
  - Pocas escrituras.
  - Consistencia relajada.

# Zona “.” (Raíz)

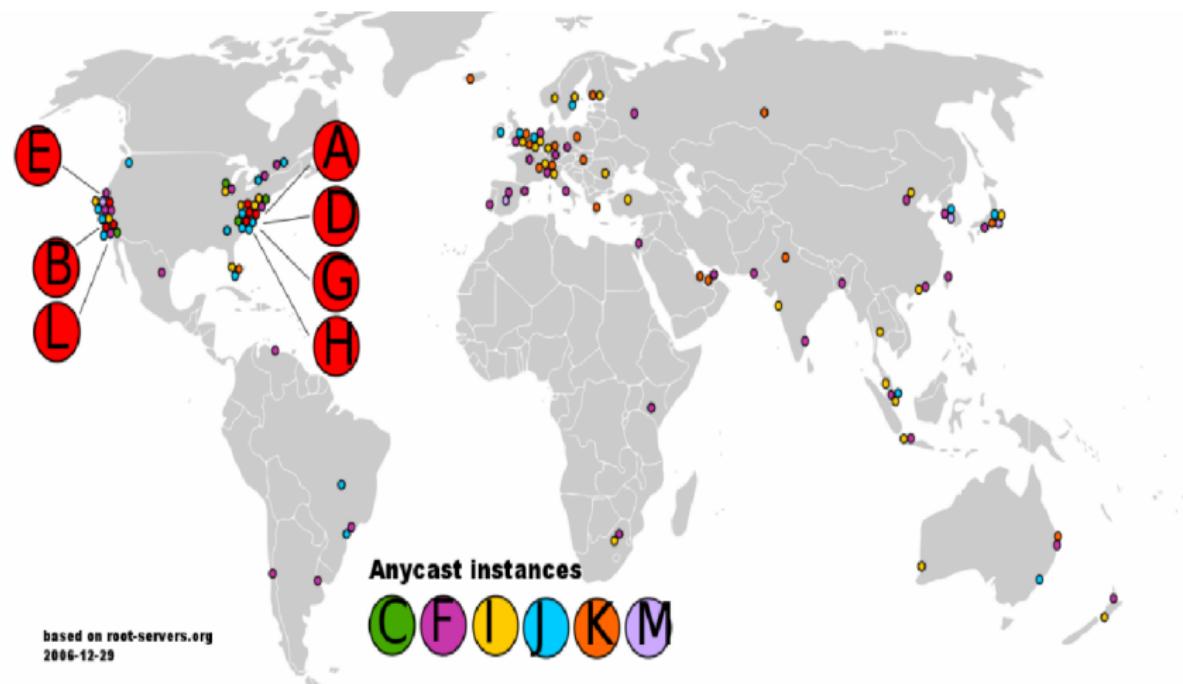
- Punto de inicio (Bootstrap).
- Actualidad: 13 **ROOT Servers** distribuidos en todo el mundo.
- 7 de los cuales trabajan con redundancia y las réplicas están distribuidos geográficamente.
- Redundancia, combinada con **Ruteo Anycast**.

.	518400	IN	NS	A.ROOT-SERVERS.NET. # Versign-grs.com
.	518400	IN	NS	B.ROOT-SERVERS.NET. # ISI.edu
.	518400	IN	NS	C.ROOT-SERVERS.NET. # Cogent.com (ANYCAST)
.	518400	IN	NS	D.ROOT-SERVERS.NET. # UMD.edu (Univ. Maryland)
.	518400	IN	NS	E.ROOT-SERVERS.NET. # NASA.gov
.	518400	IN	NS	F.ROOT-SERVERS.NET. # ISC.org (ANYCAST)
.	518400	IN	NS	G.ROOT-SERVERS.NET. # NIC.mil
.	518400	IN	NS	H.ROOT-SERVERS.NET. # ARMY.mil
.	518400	IN	NS	I.ROOT-SERVERS.NET. # NIC.ddn.mil (ANYCAST)
.	518400	IN	NS	J.ROOT-SERVERS.NET. # Versign-grs.com (ANYCAST)
.	518400	IN	NS	K.ROOT-SERVERS.NET. # RIPE.net (ANYCAST)
.	518400	IN	NS	L.ROOT-SERVERS.NET. # ICANN.org (ANYCAST)
.	518400	IN	NS	M.ROOT-SERVERS.NET. # WIDE.ad.jp (ANYCAST)

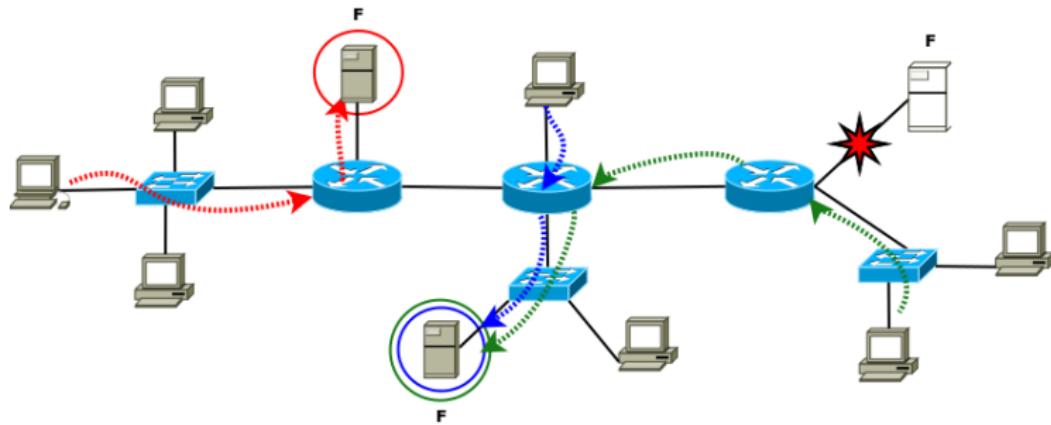
# Distribución de ROOT Servers



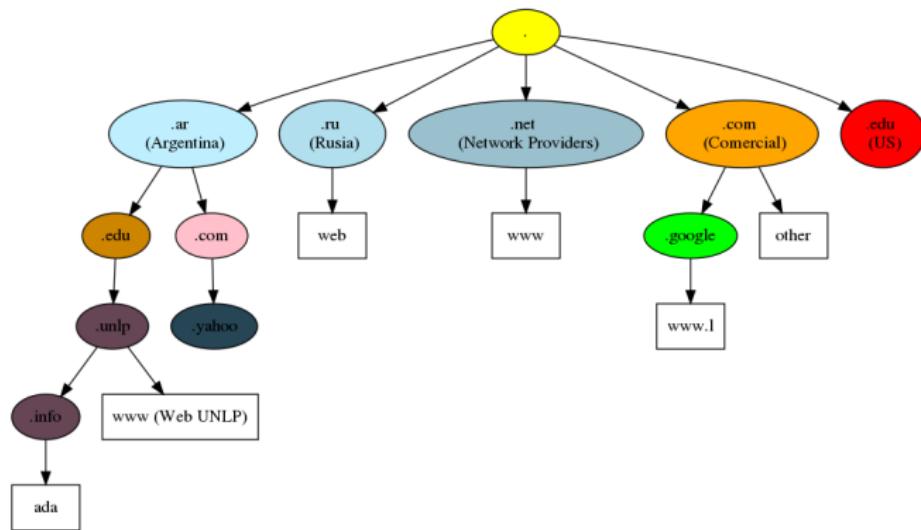
# Distribución de ROOT Servers (Cont'd)



# Distribución de ROOT Servers, Anycast



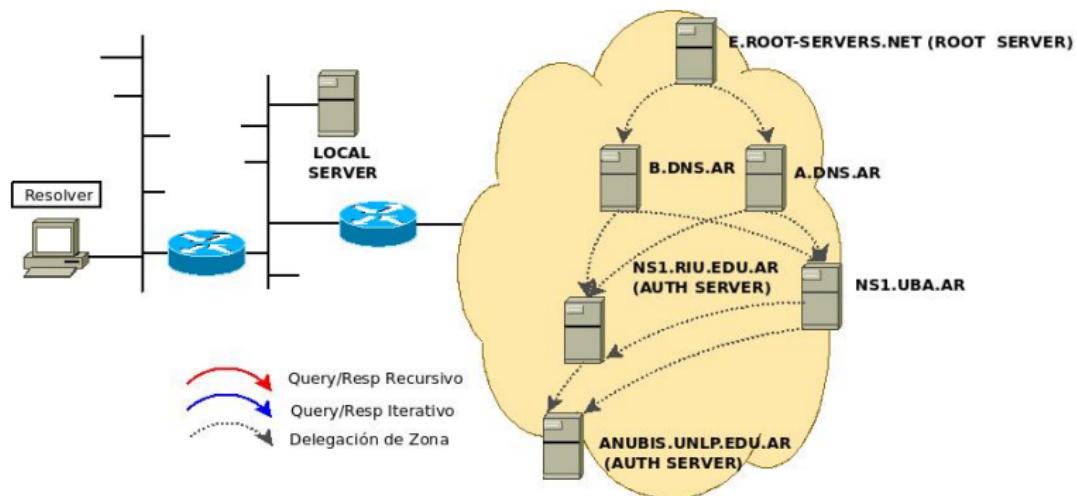
# Arquitectura de DNS



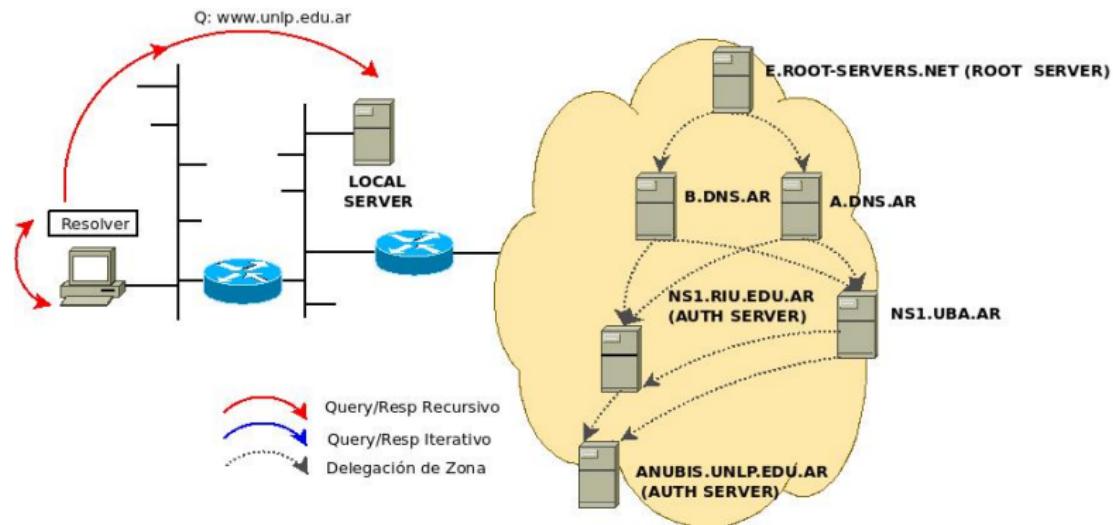
# Funcionamiento de DNS

- Modelo cliente/servidor, Request/Response.
- También hay diálogo entre los servidores.
- Protocolo corre sobre **UDP** y **TCP**, puerto **53**.
- El cliente escoge cualquier puerto no privilegiado.
- No Trabaja sobre texto ASCII.
- Si el mensaje supera los 512 bytes se utiliza TCP, e.g. zone transfer (EDNS permite mayor cantidad de datos).
- Clientes: resolver + cualquier aplicación que requiera la resolución de nombres.
  - Unix el resolver conjunto de funciones **C library (libc)**.
  - Otras implementaciones **Smart Resolver** servidor Local en cada equipo, caching.
- Servidores: BIND (Berkeley Internet Name Domain/Daemon) de ISC; UNBOUND.

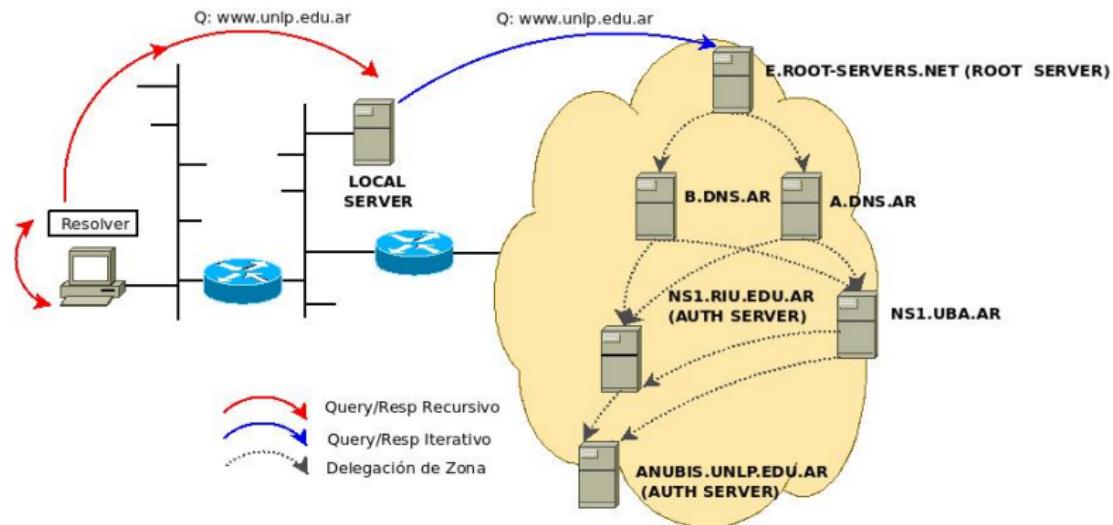
# Resolución de Nombres



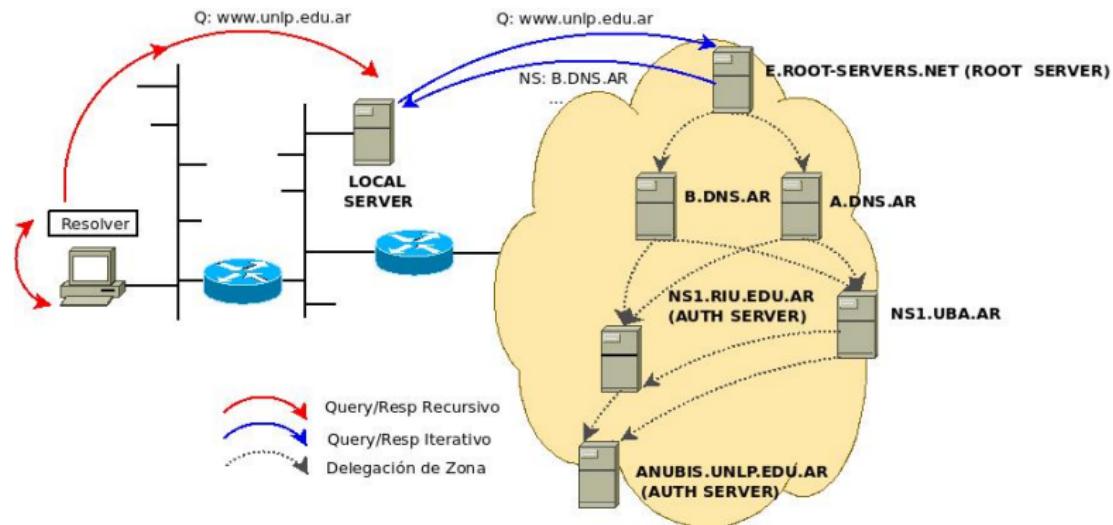
# Resolución de Nombres



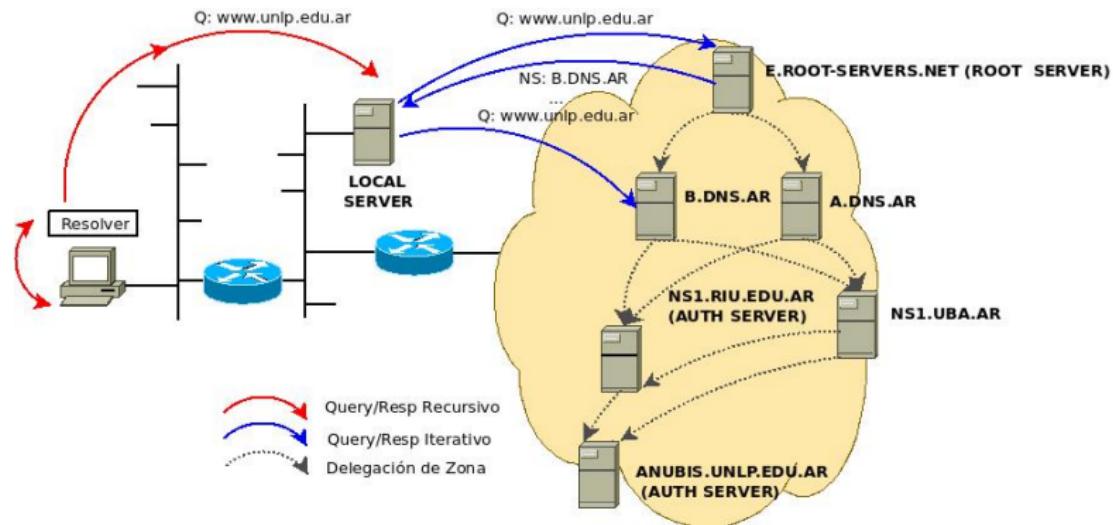
# Resolución de Nombres



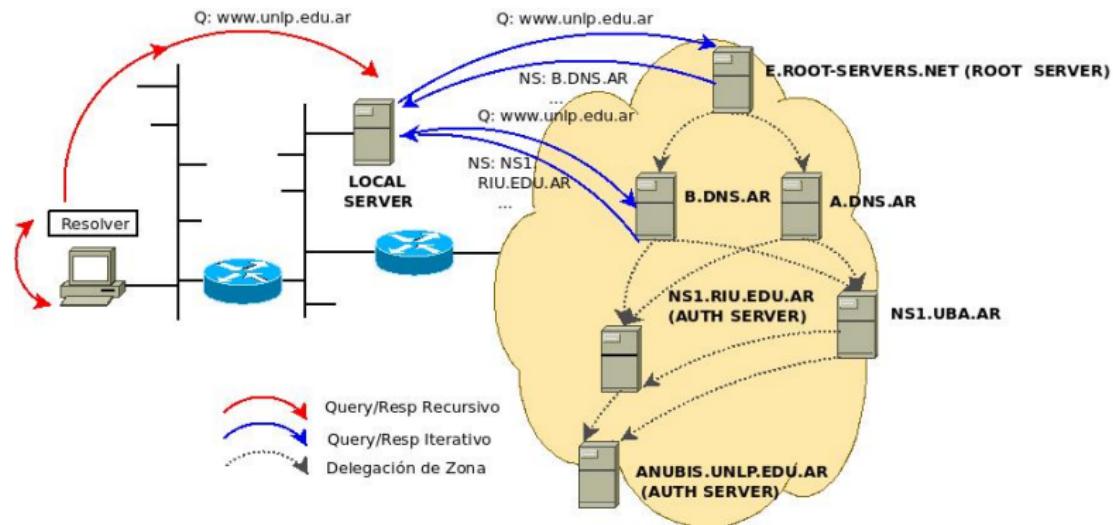
# Resolución de Nombres



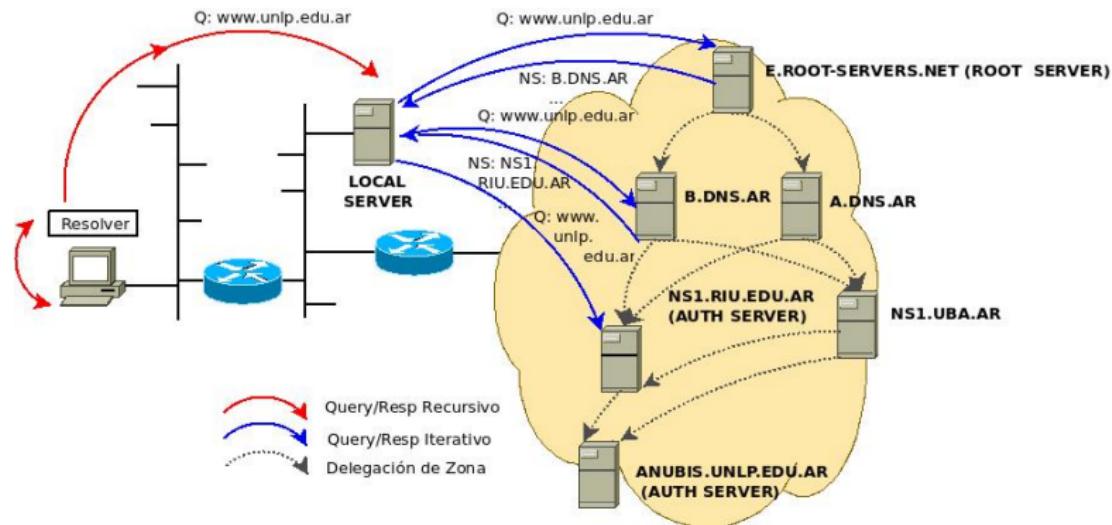
# Resolución de Nombres



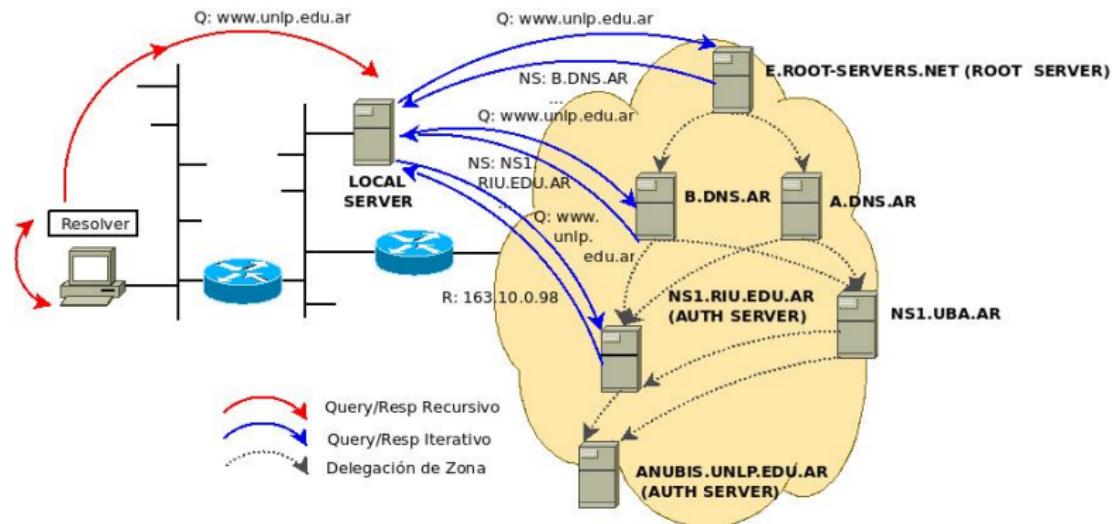
# Resolución de Nombres



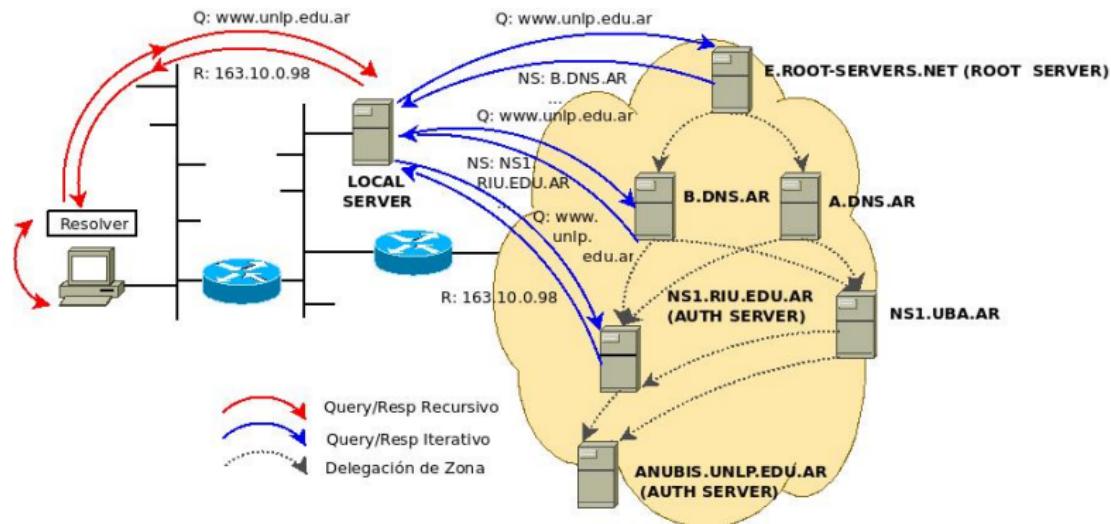
# Resolución de Nombres



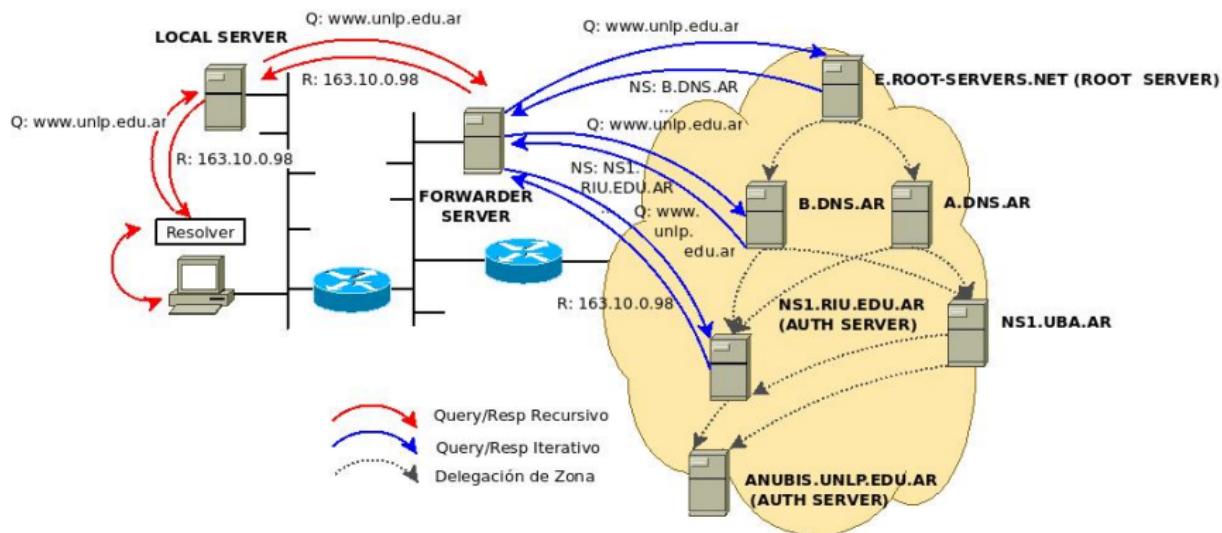
# Resolución de Nombres



# Resolución de Nombres



# Resolución de Nombres, Forwarders



# Tipos de Servidores

**Servidor Raíz:** servidor que delega a todos TLD (Top Level Domains). No debería permitir recursivas.

**Servidor Autoritativo:** servidor con una zona o sub-dominio de nombres a cargo. podría sub-delegar.

**Servidor Local/Resolver Recursivo:** es un servidor que es consultado dentro de una red. mantiene cache. Puede ser **Servidor Autoritativo**. Permite recursivas “internas”. También llamado **Caching Name Server**.

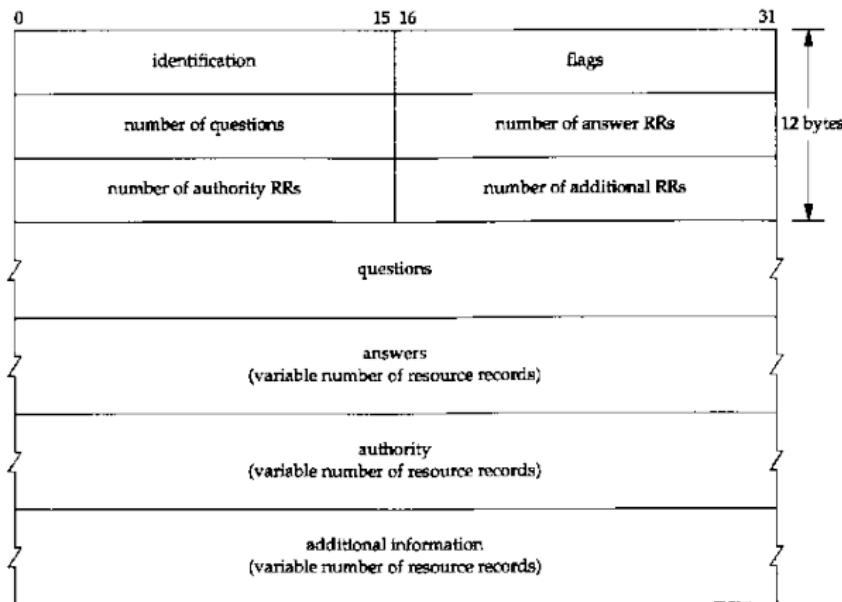
**Open Name Servers:** servidores de DNS que funcionan como locales para cualquier cliente. Por ejemplo 8.8.8.8, 8.8.4.4, 4.2.2.2, 4.2.2.3.

## Tipos de Servidores (Cont'd)

**Forwarder Name Server:** interactúan directamente con el sistema de DNS exterior. Son DNS proxies de otros DNS internos.

**Servidor Primario y Secundario:** solo una cuestión de implementación. donde se modifican los datos realmente.

# Estructura de mensaje de DNS



# Servicios y Registros de DNS

- Servidor de DNS almacena la información formando base de datos (DB) de RR (Resource Records).
- No necesariamente es DB relacional.
- Cada registro diferente tipo de información:

Registros A, AAAA (Address): nombre → IP, IPv6.

Registros PTR (Pointer): IP → nombre.

Registros CNAME (Canonical Name): nombre → nombre.

Registros HINFO (Hardware Info): nombre → info.

Registros TXT (Textual): nombre → info.

Registros MX (Mail Exchanger): nombre-dom → mail exchanger(s).

Registros NS (Name Server): nombre-dom → dns server(s).

Registros SOA (Start Of Authority): params. de dominio.

# Registros A (Address)

```
# less /etc/bind/db.cities.org
...
berlin.cities.org.      IN      A      172.20.1.100
brasilia.cities.org.    IN      A      172.20.1.5
paraguil-br0.cities.org. IN      A      172.20.1.1
...
```

# Registros AAAA (IPv6 Address)

```
# less /etc/bind/db.cities.org
...
berlin.cities.org.    IN AAAA 2001:db8:1234:4567::100
brasilia.cities.org. IN AAAA 2001:db8:1234:4567::5
...
```

# Registros PTR (Pointer)

```
# less /etc/bind/db.172
...
1.1.20 IN PTR paraguil-br0.cities.org.
5.1.20 IN PTR brasilias.cities.org.
100.1.20 IN PTR berlin.cities.org.
...
5.1.19 IN PTR sucre.lat.org.
1.1.19 IN PTR paraguil-tap2.lat.org.
...
```

# Registros CNAME (Canonical Name)

```
# less /etc/bind/db.cities.org
...
ftp.cities.org.    IN      CNAME    berlin.cities.org.
www.cities.org.   IN      CNAME    berlin.cities.org.
...
...
```

# Registros MX (Mail Exchanger)

```
# less /etc/bind/db.cities.org
...
cities.org.      IN      MX 1    brasilia.cities.org.
cities.org.      IN      MX 10   berlin.cities.org.
...

# dig -t mx gmail.com
...
gmail.com  IN      MX      5  gmail-smtp-in.l.google.com.
gmail.com  IN      MX      10  alt1.gmail-smtp-in.l.google.com.
gmail.com  IN      MX      10  alt2.gmail-smtp-in.l.google.com.
gmail.com  IN      MX      50  gsmt147.google.com.
gmail.com  IN      MX      50  gsmt183.google.com.
...
```

# Registros NS (Name Server)

```
# less /etc/bind/db.cities.org
...
; ## ZONA RAIZ
cities.org.          IN  NS  berlin.cities.org.
cities.org.          IN  NS  brasilia.cities.org.
; ## ZONA delegada
trees.cities.org.    IN  NS  brasilia.cities.org.
trees.cities.org.    IN  NS  berlin.cities.org.
trees.cities.org.    IN  NS  oak.trees.cities.org.
; ## GLUE RECORD ##
oak.trees.cities.org. IN  A   192.168.40.1
...
```

# Registros SOA (Start Of Authority)

```
$TTL    604800 ; ### TTL global para todos
cities.org. IN      SOA      berlin.cities.org.
                           root.berlin.cities.org. (
                               2008092901      ; ## Serial
                               604800        ; ## Refresh
                               86400         ; ## Retry
                               2419200       ; ## Expiry
                               604800 )      ; ## Neg Cache TTL
```

...

# Ejemplos con DNS

- Consultas a los DNS.

```
? host -t a berlin.cities.org 127.0.0.1
```

```
? dig +nocomments -t a brasilia.cities.org @127.0.0.1
```

```
? dig +recurse +short www.unlp.edu.ar @192.112.36.4
```

```
? dig +recurse +short www.unlp.edu.ar @127.0.0.1
```

```
? dig +short -t ptr 100.1.20.172.in-addr.arpa @127.0.0.1
```

```
? host -t mx cities.org 127.0.0.1
```

```
? host -t ns cities.org 127.0.0.1
```

# TTL y Registros TXT

```
? dig www.unlp.edu.ar | grep -A1 "ANSWER SECTION"
;; ANSWER SECTION:
www.unlp.edu.ar. 155 IN A 163.10.0.145

? dig -t mx gmail.com | grep -A1 "ANSWER SECTION"
;; ANSWER SECTION:
gmail.com. 3599 IN MX 20 alt2.gmail-smtp-in.l.google.com.
```

# TTL y Registros TXT

```
? dig -t txt gmail.com | grep -A1 "ANSWER SECTION"
;; ANSWER SECTION:
gmail.com. 299 IN TXT "v=spf1 redirect=_spf.google.com"

? dig -t txt _spf.google.com | grep -A1 "ANSWER SECTION"
;; ANSWER SECTION:
_spf.google.com. 299 IN TXT
    "v=spf1 include:_netblocks.google.com
     include:_netblocks2.google.com
     include:_netblocks3.google.com ~all"
```

# Otras Características del DNS

- Transferencia de Zona: AXFR.
  - Entre servidores de DNS primario y secundario.
  - Se realiza sobre TCP de forma periódica.
- Dynamic DNS: DDNS.
  - Actualización dinámica de registros, usada con IP dinámicas.
- Split DNS.
  - Responder de acuerdo de donde proviene la consulta.

- [Stev] TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1 ed. 1994. W. Richard Stevens. 2 ed. K. Fall, W. Stevens.
- [KR] Kurose/Ross: Computer Networking (6th Edition).
- [LX] The Linux Home Page: <http://www.linux.org/>.
- [Siever] Linux in a Nutshell, Fourth Edition June, 2003. O'Reilly. Ellen Siever, Stephen Figgins, Aaron Weber.
- [BIND] DNS and BIND, Fourth Edition By Paul Albitz, Cricket Liu. O'Reilly. La Third Edition de 1998 esta disponible online: <http://www.unix.com.ua/oreilly/networking/dnsbind/index.htm>.
- [LNAG] Linux Network Administrators Guide. Olaf Kirch & Terry Dawson. 2nd Edition June 2000. <http://oreilly.com/catalog/linag2/book/index.html>.
- [RFC-608] <http://www.rfc-editor.org/rfc/rfc608.txt>. Host Names On-line.
- [RFC-768] <http://www.rfc-editor.org/rfc/rfc768.txt>. User Datagram Protocol (Jon Postel 1980 USC-ISI IANA).
- [RFC-793] <http://www.rfc-editor.org/rfc/rfc793.txt>. TCP Transmission Control Protocol (Jon Postel 1981 USC-ISI IANA).
- [RFC-882] <http://www.rfc-editor.org/rfc/rfc882.txt>. DOMAIN NAMES - CONCEPTS and FACILITIES (P. Mockapetris 1983 ISI).
- [RFC-883] <http://www.rfc-editor.org/rfc/rfc883.txt> DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION (P. Mockapetris 1983 ISI).
- [RFC-1034] <http://www.rfc-editor.org/rfc/rfc1034.txt>. DOMAIN NAMES - CONCEPTS AND FACILITIES (P. Mockapetris 1987 ISI).
- [RFC-1035] <http://www.rfc-editor.org/rfc/rfc1035.txt>. DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION (P. Mockapetris 1987 ISI).
- [COM05] Ethereal, Wireshark. Autor original Gerald Combs, 2005. <http://www.ethereal.com/>. <http://www.wireshark.org/>.

# Protocolo FTP

Redes y Comunicaciones

# Historia de FTP

- FTP (File Transfer Protocol) conforma el grupo de los protocolos más viejos de la Internet aún utilizados,
- Propuesta original RFC-114, año 1971, MIT.
- Existió antes de TCP/IP, ejecutaba sobre NCP.
- El protocolo original ha sufrido varias, adaptado a IP, la esencia es la misma.
- Tuvo gran auge con a Internet comercial, en 1992 era el protocolo que más volumen transportaba (En la actualidad, ha sido superado por HTTP).
- FTP es un protocolo para copiar archivos completos, a diferencia de otros protocolos que brindan acceso a archivos: NFS, CIFS.

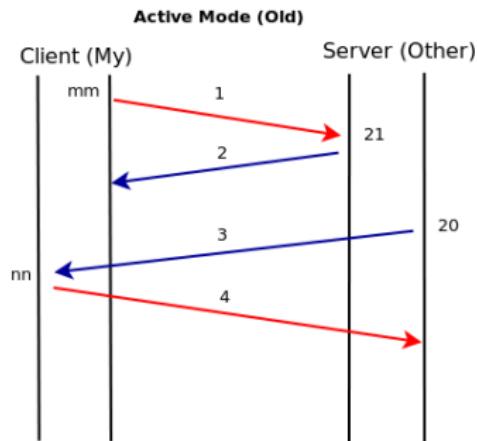
# Características del Protocolo

- Estandarizado por RFC-765, luego convertido en obsoleto por RFC-959. Actualizado por RFC-2228 y RFC-3659.
- Mensajes se codifican en ASCII estándar (de 7 bits codificados en 8). (terminal ASCII NVT -Network Virtual Terminal- CRLF).
- Modelo cliente/servidor, command/response.
- Protocolo corre sobre TCP (requiere protocolo de transporte confiable).
- Los clientes FTP, no requieren interfaz gráfica.
- Soportado por los browsers/clientes Web mediante la URI:  
`ftp://....` Ejemplos de Clientes: WS\_FTP de Ipswitch, ftp, gFTP, FileZilla.
- Ejemplos de Servidores: BSD `ftpd(8)` de Unix BSD, WU-FTP, Pure-FTPd, vsFTPD, FileZilla-Server.

# Funcionamiento de FTP

- Usa 2 (dos) conexiones TCP:
  - Conexión de Control (Out-Of-Band Control) port 21.
  - Conexión para la transferencia de datos.
- Cada conexión requiere servicios diferentes:
  - Conexión de Control: min delay.
  - Conexión de Datos: max throughput.
- El cliente escoge cualquier puerto no privilegiado, ( $n > 1023$ ) y genera conexión de control contra el puerto 21 del servidor.
- El servidor recibe los comandos por dicha conexión y responde/recibe por la conexión de datos aquellos que lo requieran.
- La conexión de datos se crea y se cierra bajo demanda.
- El estado de cada operación se transmite por el canal de control.

# Esquema de FTP



# Algunos Comandos FTP

**RETR:** obtener un archivo desde el servidor. A nivel de interfaz de usuario el comando que lo inicia es el `get`.

**STOR:** envía un archivo al servidor. A nivel de interfaz de usuario el comando que lo inicia es el `put`.

**LIST:** envía un petición de listar los archivos del directorio actual en el servidor. A nivel de interfaz de usuario el comando que lo inicia es el `ls` o `dir`.

**DELE:** comando para borrar un archivo en el servidor.

**SIZE, STAT:** obtiene información de un archivo en el servidor.

**CD, PWD, RMD, MKD:** cambia de directorio, obtiene el dir. actual, borra y crea dir.

# Ejemplos de respuestas FTP

\*\* Positive preliminary reply:

150 Here comes the directory listing.

\*\* Positive completion reply:

200 Switching to ASCII mode.

200 Switching to Binary mode.

200 NOOP ok.

200 PORT command successful. Consider using PASV.

214 The following commands are recognized

226 Directory send OK.

230 Login successful.

250 Rename successful.

## Ejemplos de respuestas FTP (Cont'd)

\*\* Positive intermediate reply:

331 Please specify the password.

350 Ready for RNTO.

\*\* Transient negative completion reply:

426 Failure writing network stream.

\*\* Permanent negative completion reply:

500 Command not understood.

504 Bad MODE command.

530 Please login with USER and PASS.

550 RNFR command failed.

550 Failed to open file.

# Sesión FTP de ejemplo

```
? ftp 127.0.0.1
Connected to 127.0.0.1.
220 (vsFTPD 2.0.5)
Name (127.0.0.1:andres): andres2
331 Please specify the password.
Password: ****
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
```

# Sesión FTP de ejemplo (Cont'd)

```
ftp> help
Commands may be abbreviated. Commands are:
```

!	debug	mdir	qc	send
\$	dir	mget	sendport	site
account	disconnect	mkdir	put	size
append	exit	mls	pwd	status
ascii	form	mode	quit	struct
bell	get	modtime	quote	system
binary	glob	mput	recv	sunique
bye	hash	newer	reget	tenex
case	help	nmap	rstatus	tick
cd	idle	nlist	rhelp	trace
cdup	image	ntrans	rename	type
chmod	lcd	open	reset	user
close	ls	prompt	restart	umask
cr	macdef	passive	rmdir	verbose
delete	mdelete	proxy	runique	?
ftp>	quit			

# Modalidades de FTP

- **FTP Activo (modalidad vieja):**
  - Conexión de control: port 21.
  - Conexión de datos: port 20.
  - Se diferencia como maneja la conexión de datos.
  - El servidor de forma activa se conecta al cliente para generar la conexión de datos.

PORT h1,h2,h3,h4,p1,p2

PORT 127,0,0,1,4,3 == 127.0.0.1:1027 (4\*256)+3

- **FTP Pasivo**

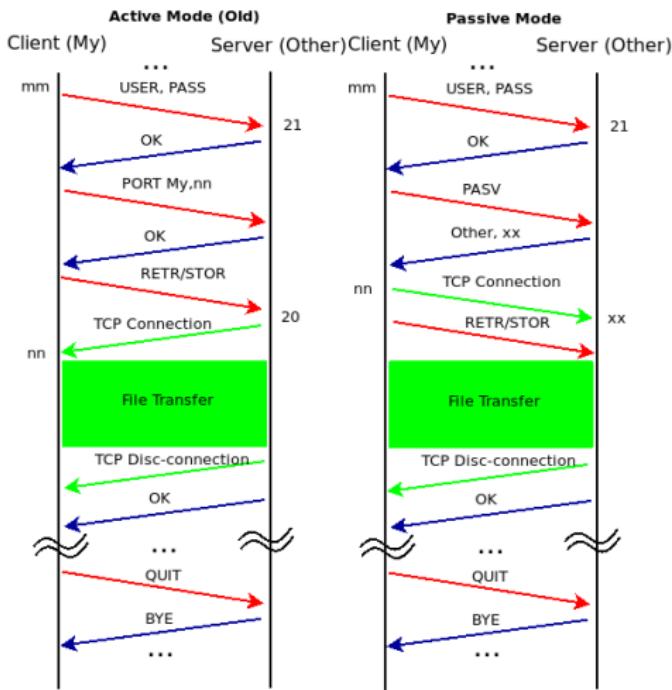
- Conexión de control: port 21.
- Conexión de datos: port no privilegiado.
- El servidor de forma pasiva indica al cliente a que nuevo puerto debe conectarse.

PASV

227 h1,h2,h3,h4,p1,p2 e.g. 227 127,0,0,1,4,3



# FTP Activo vs. Pasivo



# Algunos Comandos FTP que requieren Conexión de Datos

- Enviar un archivo al servidor (STOR).
- Traer un archivo desde el servidor (RETR).

```
ftp> put uno
local: uno remote: uno
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 File receive OK.
78257 bytes sent in 0.01 secs (15067.6 kB/s)
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 1006      1007  26 May 07 18:47 otro
-rw-r--r--    1 0          0      9 May 07 18:48 uno
226 Directory send OK.
```

# Algunos Comandos FTP que requieren Conexión de Datos

- Traer un archivo desde el servidor (RETR).

```
ftp> passive
ftp> get otro
local: otro remote: otro
227 Entering Passive Mode (127,0,0,1,120,149)
150 Opening BINARY mode data connection
      otro (1624334 bytes).
226 File send OK.
```

# Algunos Comandos FTP que NO requieren Conexión de Datos

- Ver en el directorio donde está parado (CWD, PWD).
- Ver información de un archivo en particular (STAT, SIZE).
- Cambiar el modo (MODE), Obtener ayuda (HELP).
- Borrar un archivo (DELE).
- Crear/Borrar un directorio (MKD, RMD).
- Moverse entre directorios (CD).

```
ftp> cd Public
250 Directory successfully changed.
ftp> mkdir OTRO
257 '/home/andres2/OTRO' created
ftp> rmdir OTRO
250 Remove directory operation successful.
```

# FTP, Formato de Datos (Bytes)

- FTP tiene funcionalidad de la capa ISO L6( representación).
- Debido a los diferentes tipos de plataformas, los archivos pueden ser convertidos a diferentes representaciones.
- Es responsabilidad del cliente indicarle al servidor el tipo/formato, sino el default es ASCII, aunque hoy es más común encontrar image.
- Los tipos son:
  - ASCII A NVT-ASCII.
  - EBCDIC E EBCDIC Text.
  - IMAGE I Raw binary, serie de bytes.
  - LOCAL L Raw binary, serie de bytes, usando var. byte size.

# FTP, Formato de Archivos

- Las plataformas (OS) pueden almacenar los archivos en diferentes estructuras.
- FTP define estructuras para transportar datos.
- Formato default File (F).
- Se especifica el formato para transferencia con el comando STRU.
  - File F Unstructured, sequence of bytes.
  - Record R Series of records.
  - Page P Series of data blocks (pages).

# FTP, Modo de Transferencia

- MODE se usa para especificar una codificación adicional aplicada sobre los datos transmitidos, de forma independiente del Formato del Archivo.
  - Stream S stream of bytes: Si es R el formato EOF se pone como registro Si es F el formato EOF indica el cierre del stream.
  - Block B Archivo se envía como header más secuencia de bloques. Permite interrumpir y reiniciar.
  - Compressed C Datos comprimidos usando RLE: Run Length Encoding. BBBBBBNN == 6B2N.
- Habitualmente no soportado: :-(

ftp> mode C

We only support stream mode, sorry.

# Alternativas a FTP

- Versiones de FTP seguras:
  - FTPS, FTP over SSL/TLS.
- Versiones integradas con la suite de Open-SSH:
  - SCP (Secure remote Copy).
  - SFTP (Secure FTP).
- Aplicación para compartir recursos:
  - NFS.
  - SMB/CIFS.
  - iSCSI, ...
- TFTP.

# FTP vs. HTTP

FTP	HTTP
Diseñado para Upload	Se puede con PUT, POST
Soporte de Download	Soporte de Download
Formato ASCII/binary cliente selecciona	meta-data with files, Content-Type, más flexible
No maneja Headers	Manjea Headers, mas información
FTP Command/Response, archivos pequeños más lento	Pipelining
Más complejo con Firewalls y NAT	Más amigable con Firewalls y NATs
Dos conexiones, y modo Activo o Pasivo	Una conexión, más sencillo
Soportan Ranges/resume	Range/resume HTTP opciones más avanzadas
Soporte de Autenticación	Soporte de Autenticación
Soporte de Cifrado (problemas fwall)	Soporte de Cifrado
Soporte de Compresión RLE	Soporte de Compresión Deflate: LZ77+Huffman

-  [TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994. W. Richard Stevens.](#)
-  [TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, \(2nd. Ed\). 2011. Kevin R. Fall, W. Richard Stevens.](#)
-  [Computer Networking: A Top-Down Approach, Addison-Wesley, \(6th Edition\). 2012. Kurose/Ross.](#)
-  [The Linux Home Page: <http://www.linux.org/>.](#)
-  [Linux in a Nutshell, Fourth Edition June, 2003. O'Reilly. Ellen Siever, Stephen Figgins, Aaron Weber.](#)
-  [FileZilla Project. <http://filezilla-project.org/>. vsftpd <http://vsftpd.beasts.org/>.](#)
-  [http://www.rfc-editor.org/rfc/rfc793.txt. TCP Transmission Control Protocol \(Jon Postel 1981 USC-ISI IANA\).](#)
-  [A FILE TRANSFER PROTOCOL. A. Bhushan \(MIT\), 1971.  
<http://tools.ietf.org/html/rfc114>.](#)
-  [FILE TRANSFER PROTOCOL. J. Postel \(ISI\), 1980. <http://tools.ietf.org/html/rfc765>.](#)
-  [FILE TRANSFER PROTOCOL. J. Postel, J. Reynolds \(ISI\), 1985.  
<http://tools.ietf.org/html/rfc959>.](#)
-  [Firewall-Friendly FTP, S. Bellovin \(AT&T\), 1995. <http://tools.ietf.org/html/rfc1579>.](#)
-  [FTP Security Extensions. M. Horowitz \(Cygus Solutions\), S. Lunt \(Bellcore\), 1997.  
<http://tools.ietf.org/html/rfc2228>.](#)
-  [OpenSSL project: <http://www.openssl.org/>.](#)



Ethereal, Wireshark. Autor original Gerald Combs, 2005.

<http://www.ethereal.com/>.

<http://www.wireshark.org/>.

# Mensajería en Internet (E-Mail)

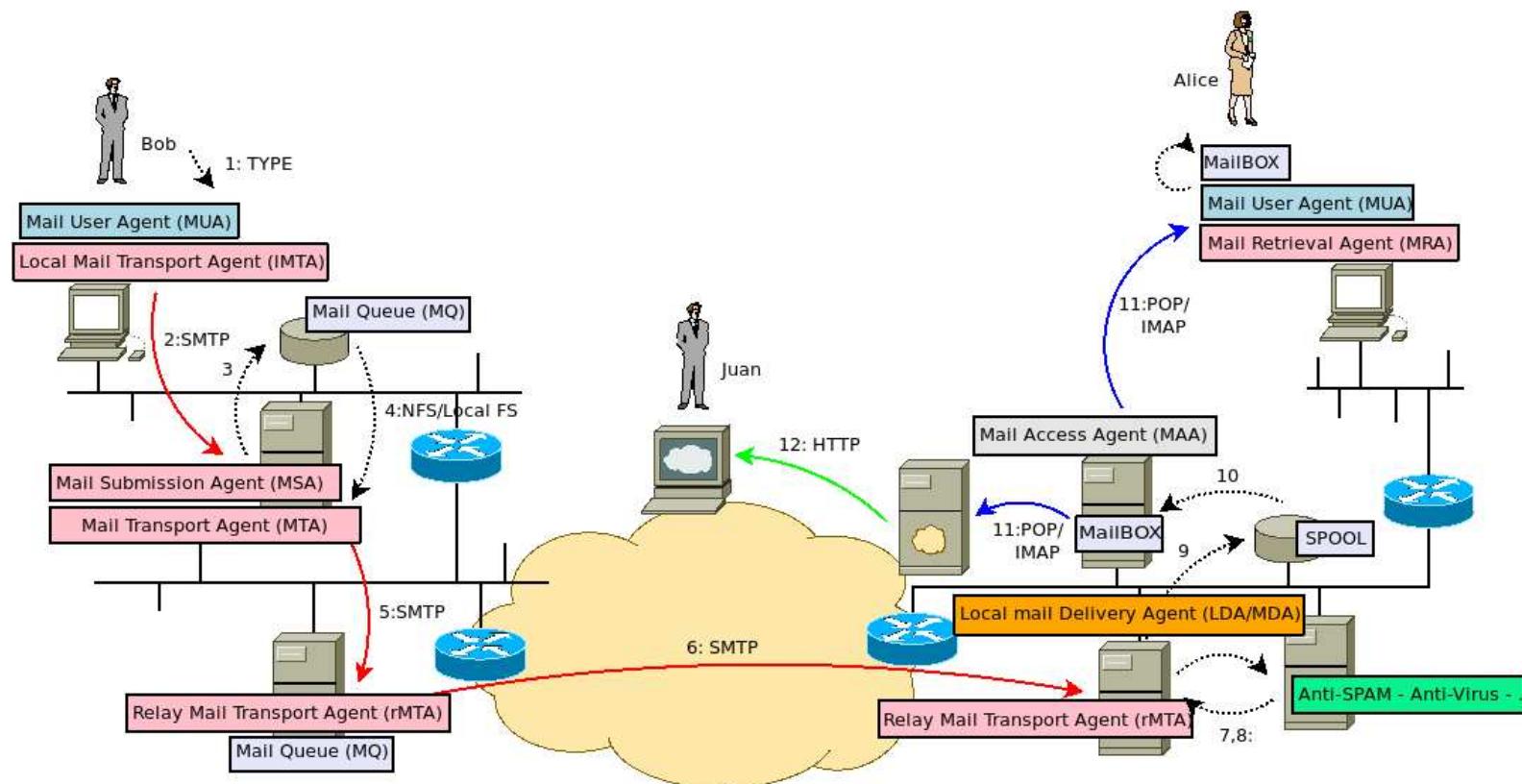
Redes y Comunicaciones



## Introducción

- Uno de los primeros servicios de Internet.
- Creado por Ray Tomlinson en 1971.
- Primeros protocolos de transporte UUCP (Unix-to-Unix Copy).
- Conexiones vía módems.
- Formato de E-mail: MACH2!MACH1!USER.
- En 1972: Se escoge el símbolo “@” para denotar “en”, “at”.
- Casilla de E-mail: USER@FQDN-DOMAIN o  
USER@FQDN-SERVER.
- En 1982: surge SMTP.
- En 1988: Gusano Morris, primer Virus, usaba el Sendmail.

# Arquitectura



## Arquitectura (Cont'd)

- Componentes Principales:
  - MUA (Mail User Agent).
  - MTA (Mail Transport Agent).
  - MDA (Mail Delivery Agent) o LDA (Local Delivery Agent).
  - MAA (Mail Access Agent).
  - Otros: MRA (Mail Retrieval Agent), MSA (Mail Submission Agent).
- Componentes Secundarias:
  - Servidor de Autenticación externo.
  - Web Mail (Front-End WWW).
  - Servidor de Anti-Virus, Anti-SPAM, Servidores de Listas, ...

## Arquitectura (MUA - Mail User Agent)

- Cliente.
- Interfaz con usuario.
- Lector/Editor/Emisor local de correos (e-mails).
- Posee integrado un local MTA para comunicarse con el Servidor de Mail Saliente, MTA que hace relay.
- El MTA que hace relay para el usuario pre-procesa el e-mail recibido desde el MUA con el agente MSA (Mail Submission Agent).
- Agrega la mayoría de los campos del header: Message-ID, To:, From:, Date:, Subject ...
- Posee integrado un MRA para comunicarse con el Servidor de Mail Entrante, MAA (Mail Access Agent).

- Utilizan protocolos SMTP o ESMTP, POP o IMAP, habla con MTA y con MAA propio.
- Ejemplos: Eudora, MS Outlook, Mozilla Thunderbird, elm, pine, mutt.
- Web-mailers: Horde/IMP, Squirrel, GMAIL, Yahoo, Hotmail.

## Arquitectura (MSA - Mail Submission Agent)

- Servidor.
- Agente habitualmente integrado en el MTA.
- Recibe el mensaje del MUA y lo pre-procesa antes de pasarlo al MTA para que haga el transporte.
- Agrega campos que pueden faltar, formato del header:  
Message-ID, To:, From:, Date: ...
- Termina de dar formato al header del e-mail.
- Utiliza protocolo SMTP, ESMTP.
- Habitualmente usaba el port 25, se recomienda usar 587(submission), RFC-6409, ex:RFC-4409.
- Ejemplos: componentes de Postfix (postdrop, pickup), Sendmail-MSA.

## Arquitectura (MTA - Mail Transport Agent)

- Cliente y Servidor.
- Toma el e-mail desde el MSA o directamente desde el MUA (MSA integrado).
- Se encarga de enviar el mensaje de e-mail al servidor donde está la casilla de mensaje destino, comunicación de MTA a MTA.
- Almacena temporalmente el correo saliente.
- Se encarga de recibir y almacenar temporalmente los mensajes para las casillas que sirve desde el MTA remoto.
- Utiliza protocolo SMTP o ESMTP, entre servidores MTA.
- Ejemplos: Postfix, Sendmail, MS Exchange, Exim, Qmail.

## Arquitectura (MDA/LDA Mail Delivery Agent)

- Servicio interno o Servidor.
- Se encarga de tomar los e-mails recibidos por el MTA (acepta mensajes del MTA) y llevarlos al mailbox del usuario local.
- Se lo llama también LDA (Local Delivery Agent).
- Habitualmente define el formato del mailbox.
- Servicio de MDA puede hacer delivery remoto, cumple rol de MAA.
- Se integra con los protocolos POP y/o IMAP dejando los recursos disponibles al MAA o directamente al usuario.
- Ejemplos: procmail, postfix local, Sendmail, courier, cyrus-IMAP, dovecot, sieve, fetchmail(más un MRA), getmail(más un MRA).

## Arquitectura (MAA - Mail Access Agent)

- Servidor.
- Integrado o separado del MDA.
- Autentica al MUA/usuario y lee los e-mails del mailbox local dejados por el MDA/LDA.
- Transpora los e-mails hacia el MUA o los hace accesibles a este.
- Se integra con el MDA. Implementa los protocolos POP y/o IMAP dejando acceder a los recursos y dialogando con el MUA.
- Ejemplos de MDA/LDA: courier, cyrus-IMAP, dovecot, sieve.

## Protocolo SMTP

- Simple Mail Transport Protocol.
- Protocolo Cliente/Servidor.
- Utiliza formato ASCII 7 bits en 8 NVT.
- Usa TCP puerto servidor: 25.
- Los LMTA de los MUA hablan SMTP con su servidor SMTP saliente.
- Los servidores SMTP hablan entre sí este protocolo.
- RFC-821 (SMTP), extendida por RFC-1869 (ESMTP) y Redefinida por RFC-2821.

## Protocolo SMTP (Ejemplo)

```
? telnet 172.20.1.5 25
...
220 brasilia.cities.org SMTP Sendmail 8.12.8/8.12.8 ...
HELO server.other.test
250 brasilia.cities.test Hello server... pleased to meet you
MAIL FROM: <bob@other.test>
250 2.1.0 bob@other.test... Sender ok
RCPT TO: <alice@cities.org>
250 2.1.5 alice@cities.org ... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Esto es una prueba
.
250 2.0.0 k5HF1B0d005647 Message accepted for delivery
QUIT
221 2.0.0 brasilia.cities.org closing connection
```

## Protocolo SMTP (Cont'd)

- Requiere finalizar con CRLF .CRLF.
- Usa conexiones persistentes.
- Trabaja de forma Interactiva (Requerimiento/Respuesta) o Pipeline.
- Puede o no requerir Autenticación.
- Puede o no trabajar de forma segura: SSL/TLS.

## Formato del Mensaje

- El concepto de Envelope fue definido en RFC-821.
- Definido el Cuerpo y el Encabezado en RFC-822, Redefinido RFC-2822.
  - Envelope (Envoltorio), el usuario no lo ve, usado por MTAs, MAIL FROM: , RCPT TO:
  - Header (Encabezado), meta información del mail:  
Subject: , From: , To: , Return-Path: X-Mailer: , X-....:
  - Body (Cuerpo), separado por línea en blanco del header:  
Contenido del e-mail.

## Formato del Mensaje Extendido

- Extensiones para enviar datos binarios.
- MIME (Multipurpose Internet Mail Extensions): Definido en RFC-1521 y RFC-1522. Redefinidas en RFC-2045 y RFC-2046.
- Con tags especiales indica el tipo al sistema destino.
- Luego codifica el mensaje binario en formato que no viole US-ASCII (NVT).
- Algunos Ejemplos: uunecode, base64, quotes-strings, etc.

## Formato del Mensaje (Ejemplo)

```
From: bob@other.test
To: alice@cities.org
Subject: Test Mail MIME
Date: Thu, 11 Aug 2005 20:20:56 -0300 (ART)
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded image .....
.....
.....base64 encoded image
```

## Mensajes Multipart (Ejemplo)

```
From: bob@other.test
To: alice@cities.org
Subject: Test Mail MIME Multipart
Date: Thu, 11 Aug 2005 20:20:56 -0300 (ART)
MIME-version: 1.0
Content-type: multipart/mixed; boundary="DDDIIVVVIISSSOORR"

This is a message with multiple parts in MIME format.
--DDDIIVVVIISSSOORR
Content-type: text/plain

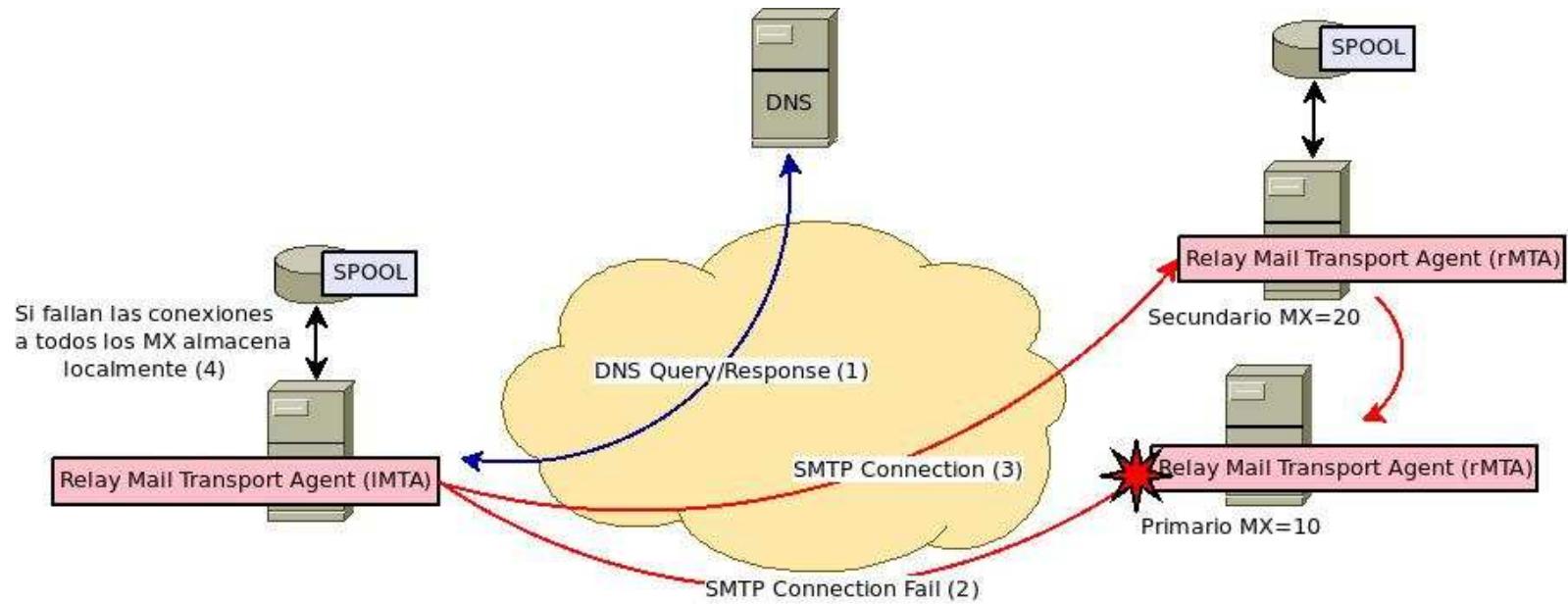
Mensaje en Texto
--DDDIIVVVIISSSOORR
Content-type: application/octet-stream
Content-transfer-encoding: base64

base64 encoded stream .....
.....
.....base64 encoded stream
--DDDIIVVVIISSSOORR --
```

## Protocolo ESMTP (SMTP Extendido)

```
220 brasilia.cities.org ESMTP Sendmail 8.12.8/8.12.8 ...
EHLO server.other.test
250 brasilia.cities.test Hello server..., pleased ...
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-STARTTLS
250-AUTH
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-VRFY
250-EXPN
250 HELP
```

## SMTP y DNS



## Protocolos Acceso a Correo

- POP: Post Office Protocol, RFC-1939: POPv3.
- IMAP: Internet Mail Access Protocol, RFC-1730: IMAPv4.
- Requieren Autenticación.
- Utiliza formato ASCII 7 bits en 8 NVT.
- Usan TCP puertos servidor: 110 y 143.
- Permiten correr de forma segura sobre SSL/TLS.
- IMAP más flexible permite uso de carpetas y manipulación de mensajes en el servidor.

## POP (Ejemplo)

```
? telnet 127.0.0.1 110
...
+OK POP3 localhost.localdomain v2001.78rh server ready
user alice
+OK User name accepted, password please
pass 123456
+OK Mailbox open, 2 messages
NOEXIST
-ERR Unknown TRANSACTION state command
LIST
+OK Mailbox scan listing follows
1 620
2 622
.
TOP 1 1 100
+OK Top of message follows
```

```
Return-Path: <juan@...>
...
TOP 2 1 100
+OK Top of message follows
Return-Path: <root@...>
...
QUIT
+OK Sayonara
```

## IMAP (Ejemplo)

```
? telnet 127.0.0.1 143
...
* OK brasilia.cities.org IMAP4rev1 v12.264 server ready
A0001 USER "alice" "123456"
* OK User alice authenticated
A0002 SELECT INBOX
* 1 EXISTS
* 1 RECENT ...
A0004 FETCH 1 BODY[HEADER]    ### Get first message header
...
A0005 FETCH 1 BODY[TEXT]      ### Get first message body
...
A0005 OK FETCH completed
A0006 LOGOUT
* BYE example.com IMAP4rev1 server terminating connection
A0006 OK LOGOUT completed
```



SPAM

- Open Relay.
- SPAMBots.
- Métodos para Contrarrestarlo.
  - RBL (Realtime Blackhole List) Blacklist.
  - Greylist, Whitelists.
  - Filtros de Contenido.
  - Reglas Dirigidas/Entrenadas por el usuario.

## Fuentes de Información

- Kurose/Ross: Computer Networking (6th Edition).
- TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens.
- RFCs: <http://www.isi.edu/in-notes/rfc821>, rfc822, ...
- Wikipedia: <http://www.wikipedia.org>.
- Internet ...

# Protocolos de Transporte: UDP, TCP

(parte I) 2020

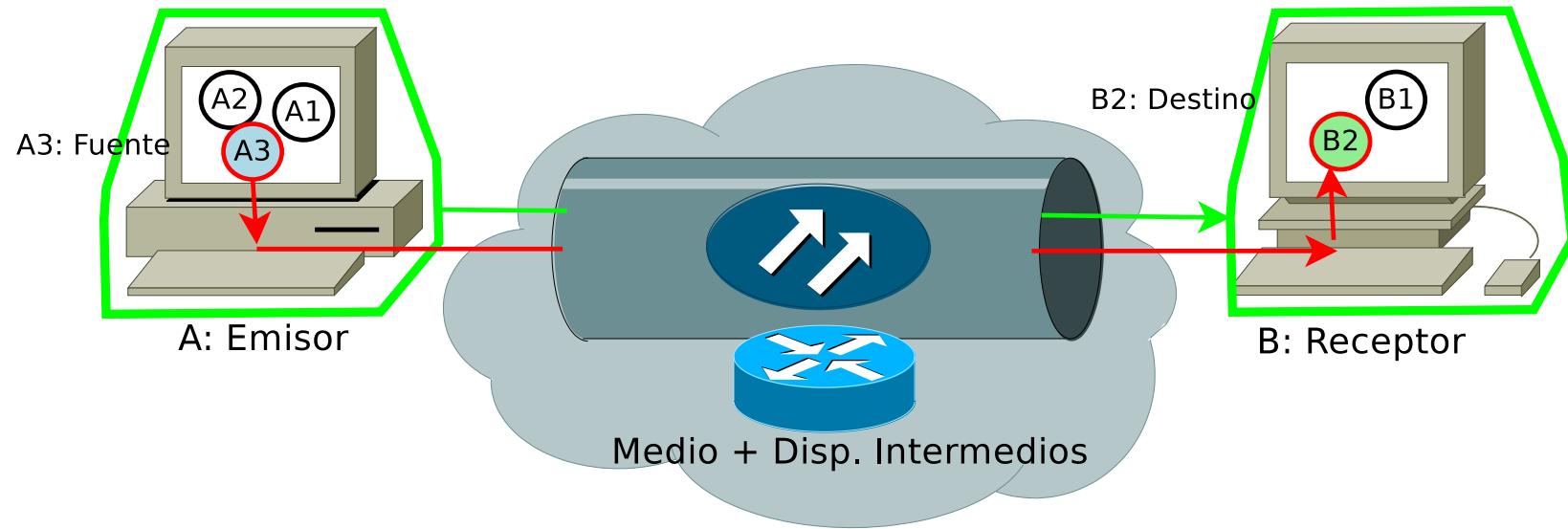


UNIVERSIDAD  
NACIONAL  
DE LA PLATA

## Introducción

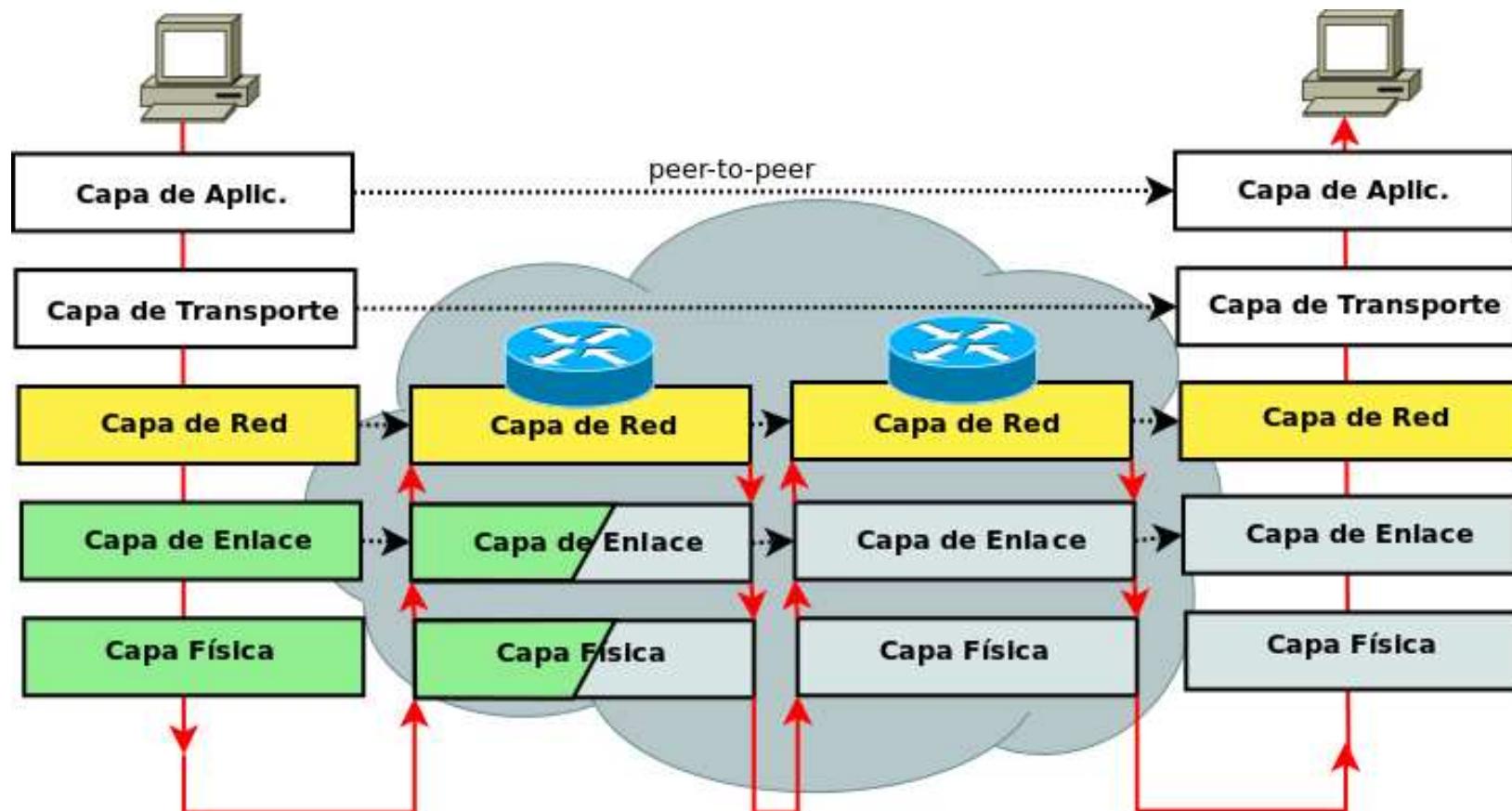
- IP provee un servicio “débil”, pero eficiente: “Best-effort”.
- En IP los paquetes pueden ser descartados, des-ordenados, retardados duplicados o corrompidos.
- Paquetes IP solo dirección DST y SRC, ¿Como elegir la App.?
- IP corre en **todos los nodos de la red** (routers y host), protocolos de transporte solo necesario en **end-points** (hosts).
- IP: comunicación lógica **HOP-BY-HOP**, comunica **hosts**.
- Transporte: comunicación lógica **HOST-TO-HOST (END-TO-END)**, comunica **procesos**.
- Aplicación: comunicación lógica **PROCESS-TO-PROCESS (END-TO-END)** comunica usuarios, agentes, etc.

# Direccionamiento a nivel de Transporte



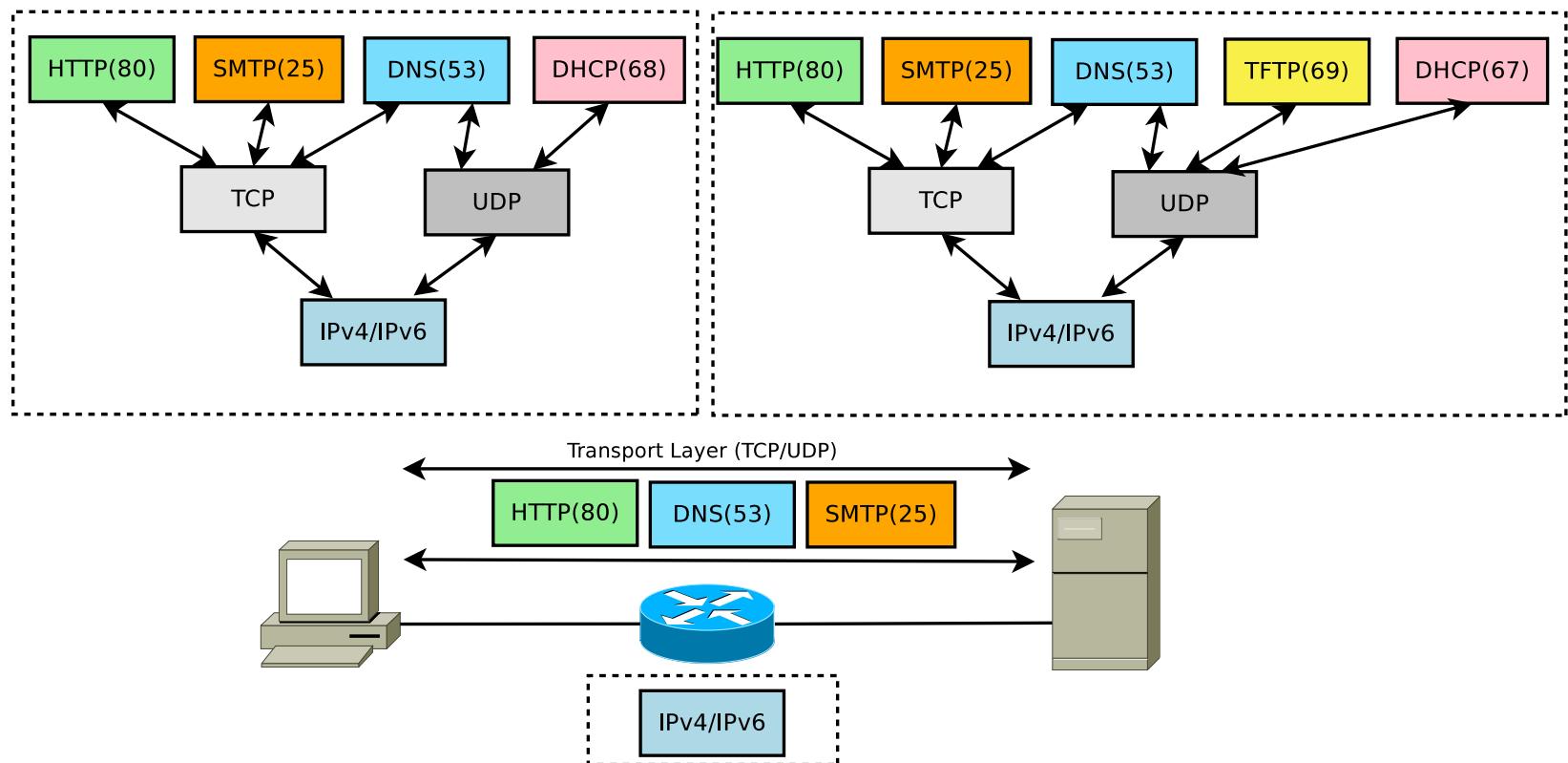
- Servicio Internet, IP, puerta a puerta, host-to-host.
- Servicio Transporte, persona a persona, process-to-process.
- Identificador de proceso independiente de plataforma, nro. de puerto.

# Comunicación en Capas

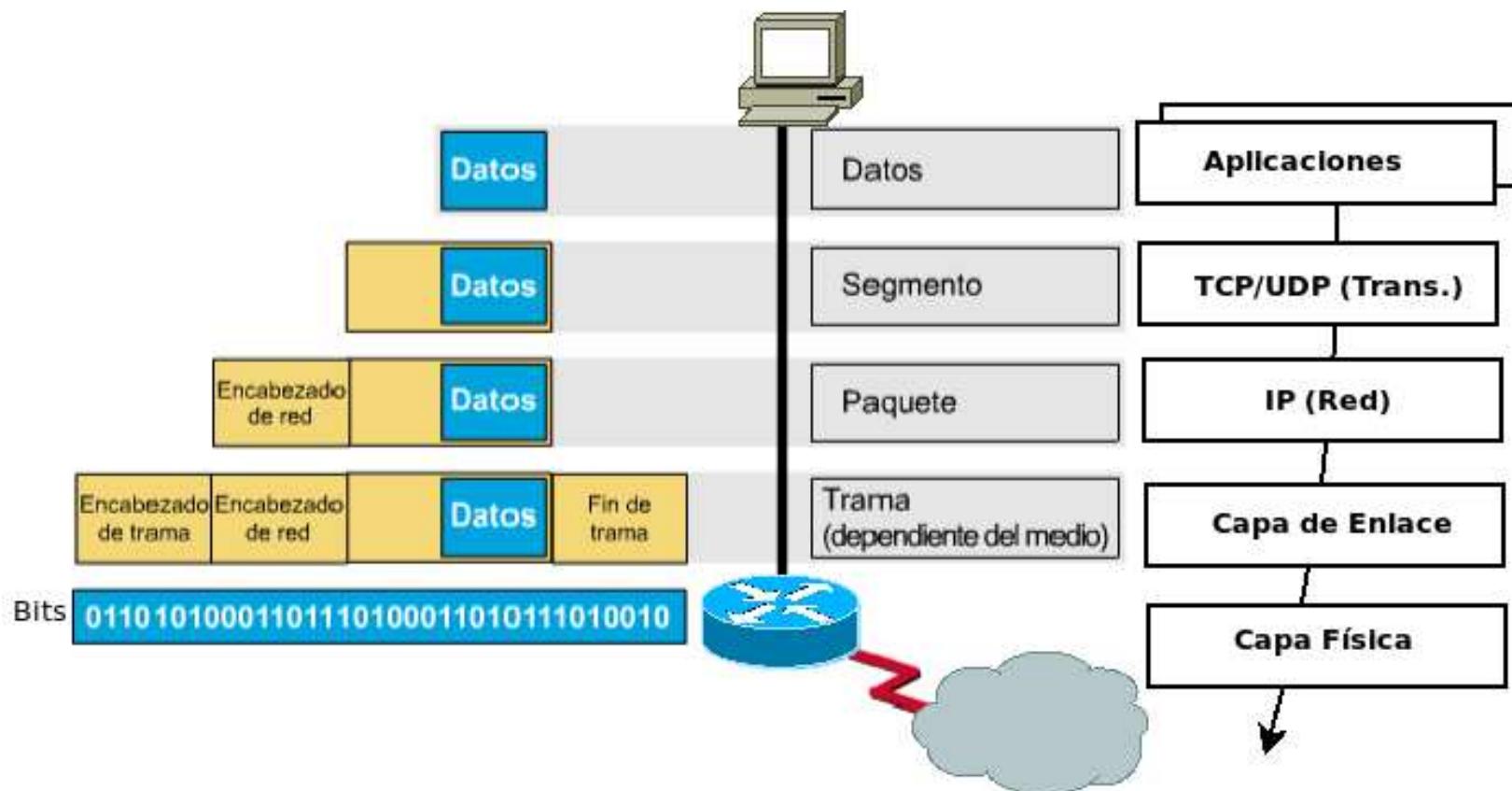


## Funcionalidad de Capa de Transporte

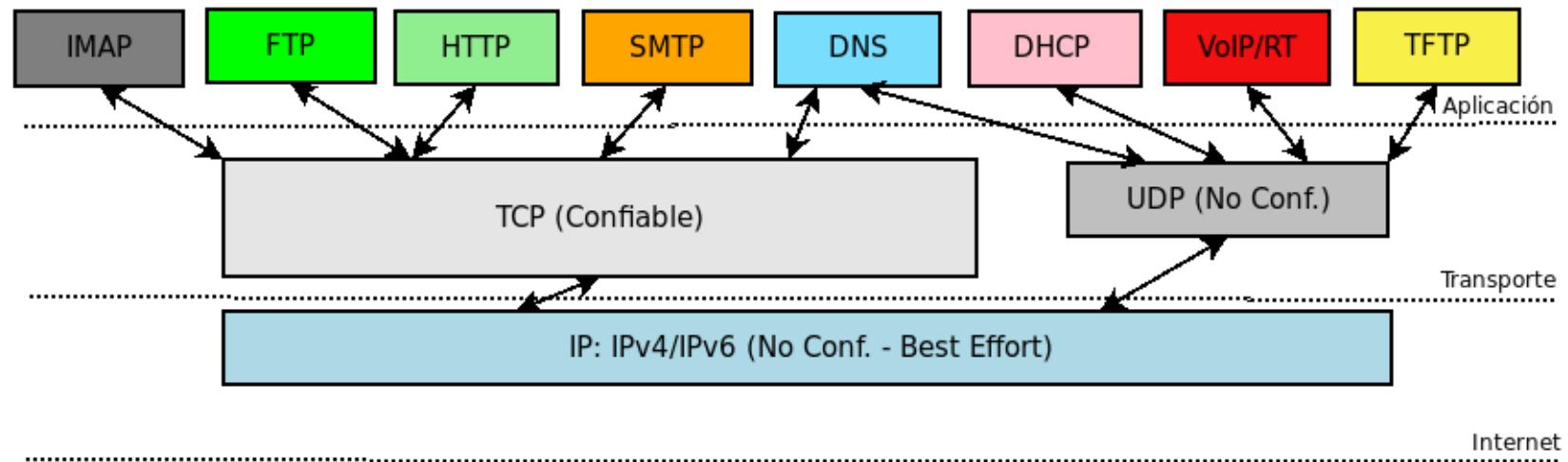
- MUX/DEMUX App. to App. (puertos, Ports).
- Soporte de datos de tamaños arbitrarios.
- Control y Detección de Errores, pérdida, duplicación, se corrompen.
- ¿Cómo enviar info sobre la red de acuerdo al estado de la misma? ¿Cuándo y Cómo una App. debe enviar datos?
  - Control de Flujo.
  - Control de Congestión.
- Dos modelos:
  - Modelo Confiable: TCP.
  - Modelo NO Confiable: UDP.



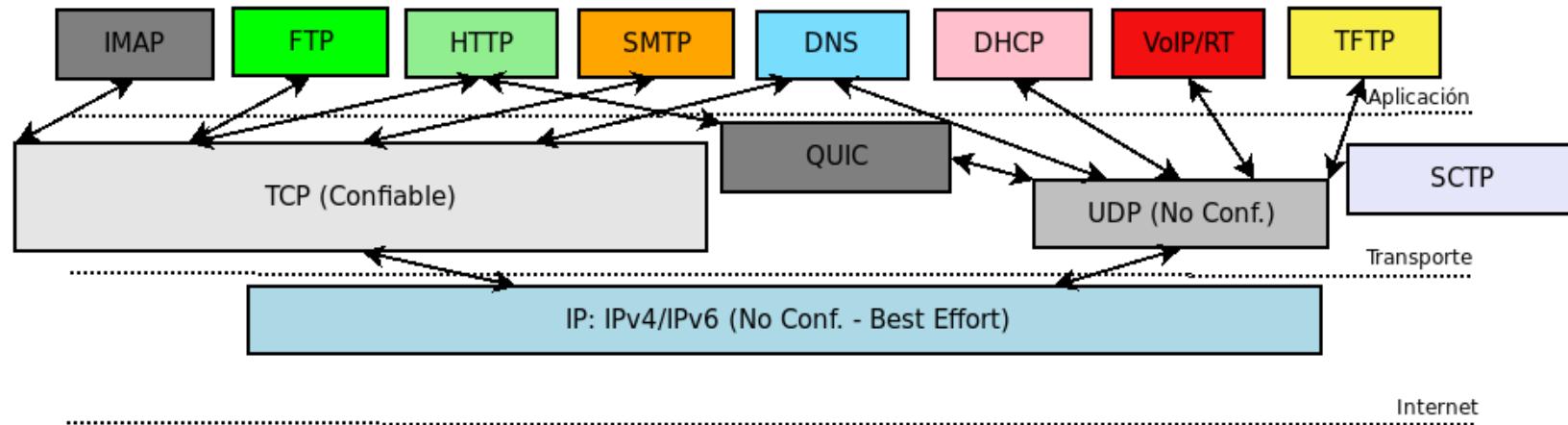
# Protocolos de Transporte, Encapsulación



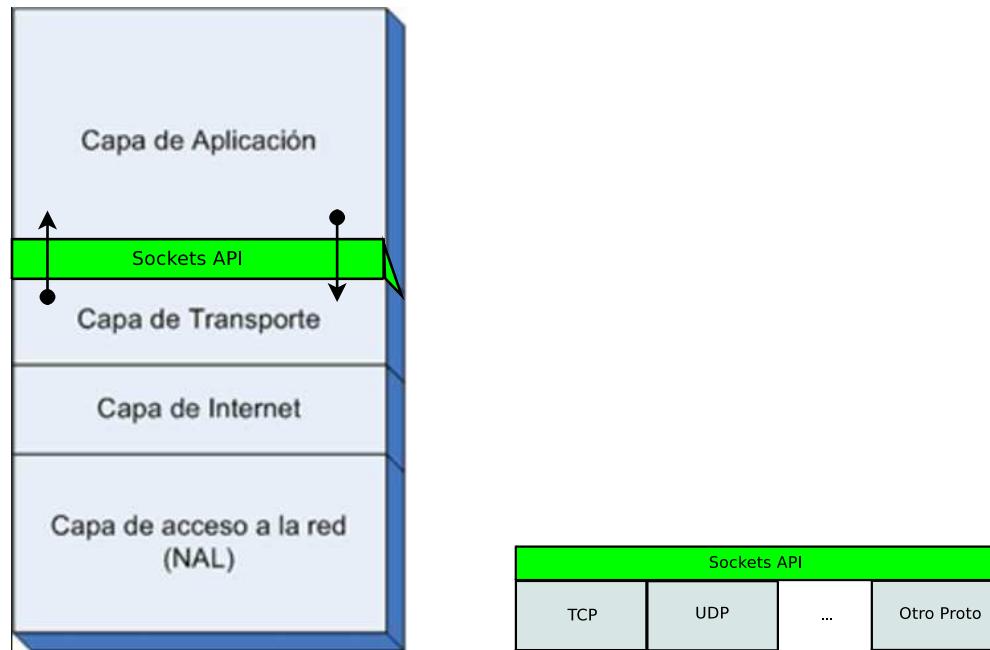
# Protocolos de Transporte



## Protocolos de Transporte, Alt.



## Selección Protocolo de Transporte



- La aplicación de acuerdo a como esta programada selecciona el transporte.
- El acceso a los servicios de transporte se hace mediante **API:Network socket**.

## UDP

- User Datagram Protocol (RFC-768).
- Protocolo Minimalista. Menor Overhead.
- Características de IP: best-effort.
- Orientado a Packets/Datagramas (mensajes auto-contenidos).
- PDU: Datagrama (Por coherencia con nivel Transporte se suele llamar Segmento).
- Solo provee MUX/DEMUX.
- No incrementa Overhead end-to-end.
- No requiere establecimiento de conexión.
- Servicio FDX.
- Aplicaciones: video/voz streaming/TFTP/DNS/Bcast/Mcast.

## TCP

- Transport Control Protocol (RFC-793)
- Protocolo confiable, ordenado, buffering, control de flujo y de congestión.
- Orientado a Streams (secuencia de bytes,  $\equiv$  archivo).
- PDU: Segmento.
- Provee MUX/DEMUX.
- Incrementa Overhead end-to-end para ofrecer confiabilidad.
- Requiere establecimiento de conexión (y cierre).
- Servicio FDX.
- Aplicaciones: FTP/HTTP/SMTP/acceso remoto(SSH, telnet,...)/Unicast.

## SCTP

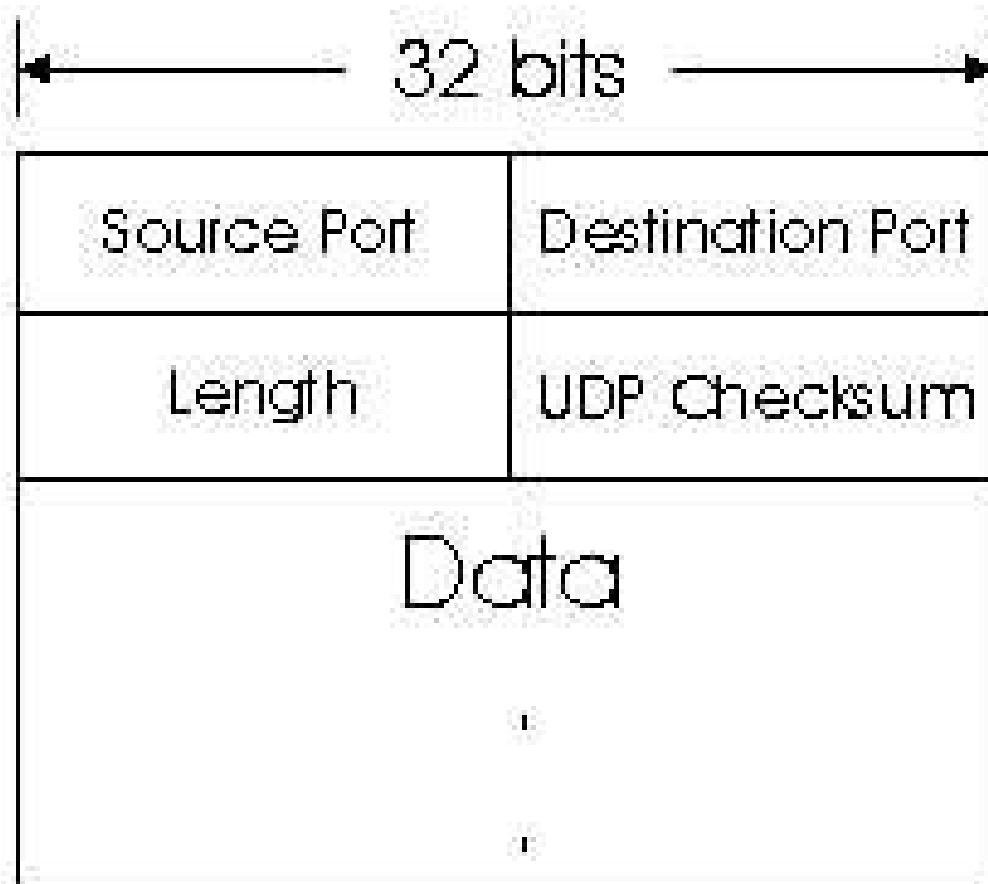
- Stream Control Transmission Protocol (RFC-4960).
- Protocolo con menor Overhead que TCP y más servicios que UDP.
- Orientado a Packets/Datagramas/Mensajes.
- Incrementa Overhead end-to-end.
- Asegura orden, secuencia con control de congestión.
- Servicio FDX.
- Aplicaciones: WebRTC y ssh lo podría utilizar (no es lo más común).

## Headers/Encabezados Servicios

- El encabezado IP provee: Ruteo, Fragmentación, Detección de algunos errores.
- El encabezado UDP provee: MUX/DEMUX de aplic. , Detección de errores (no obligatorio).
- El encabezado TCP provee: MUX/DEMUX, Detección de errores, Sesiones, Control de Errores, Control de Flujo y Control de Congestión.

```
? grep udp /etc/protocols  
udp      17      UDP # user datagram protocol  
? grep tcp /etc/protocols  
tcp      6      TCP # transmission control protocol
```

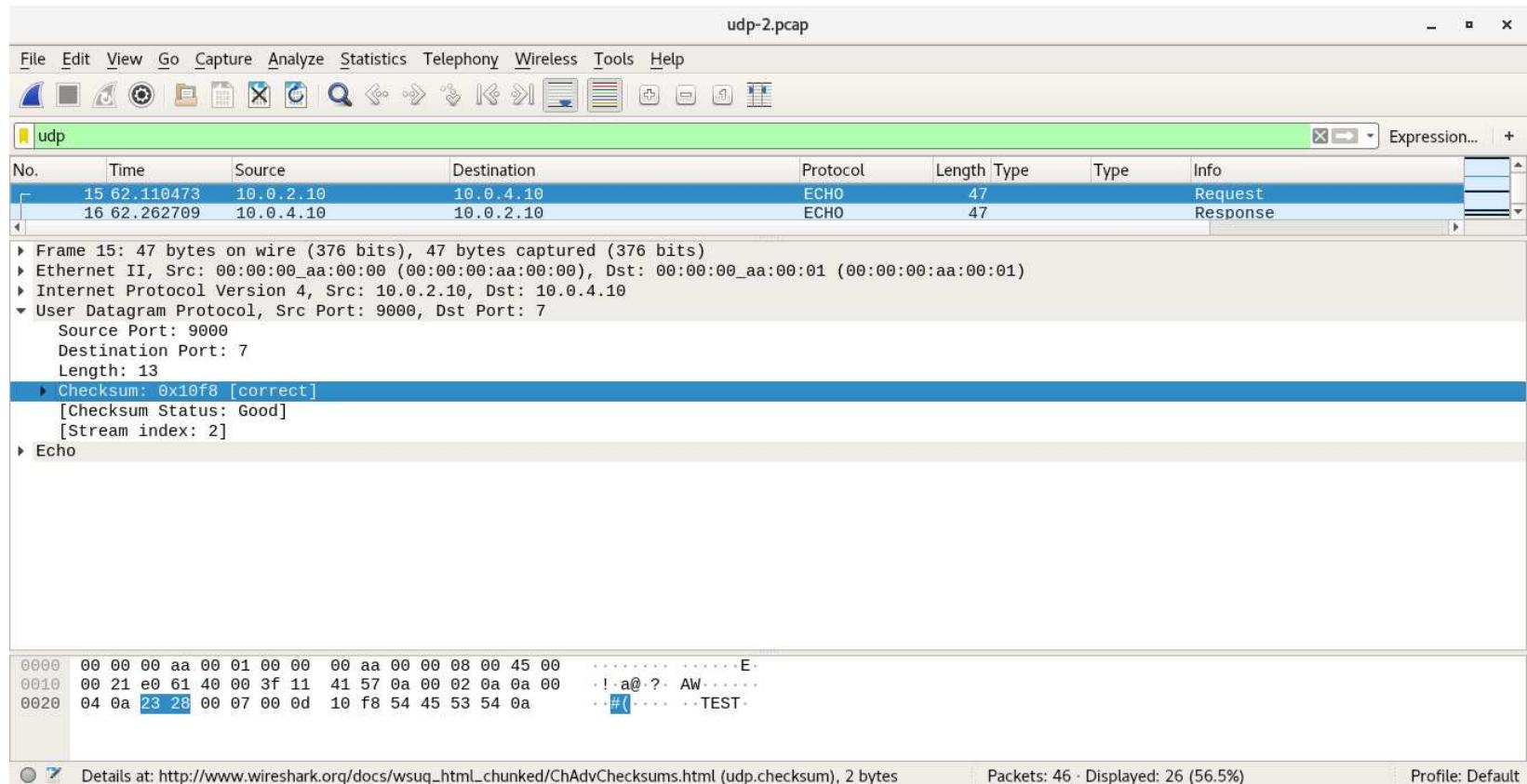
## Datagrama UDP



## Datagrama (UDP)

- Puertos: MUX/DEMUX.
- Longitud: UDP HDR + Payload.
- Checksum
  - Cálculo Ca1, Opcional. 0 = Sin checksum.
  - Calculado HDR + PseudoHDR + Payload.
  - PseudoHDR: IP.SRC + IP.DST + Zero + IP.PROTO + UDP.LENGTH.
  - PseudoHDR: protección contra paquetes mal enrutados.
  - Aplicaciones de LAN por eficiencia lo podrían deshabilitar.
  - Si tiene error se descarta silenciosamente.

# Cálculo de Checksum



## Cálculo de Checksum (Cont.)

Pseudo header SRC=10.0.2.10, DST=10.0.4.10, PROTO=17  
0A00 020A 0A00 040A 0011

UDP header SRCP=9000, DSTP=7, LEN=13, LEN-PH=13, CKSUM=0  
2328 0007 000D 000D

DATA 5445 5354 0A00

00001010 00000000 = 10.0  
00000010 00001010 = 2.10  
00001010 00000000 = 10.0  
00000100 00001010 = 4.10  
00000000 00010001 = 17  
00100011 00101000 = 9000  
00000000 00000111 = 7  
00000000 00001101 = 13  
00000000 00001101 = 13  
00101010 01000101  
00101001 01010100  
00001010 00000000

-----  
~11101111 00000111 EF07 ~EF07 = 10F8  
00010000 11111000

## Cálculo de Checksum (ejemplo con Carry(C))

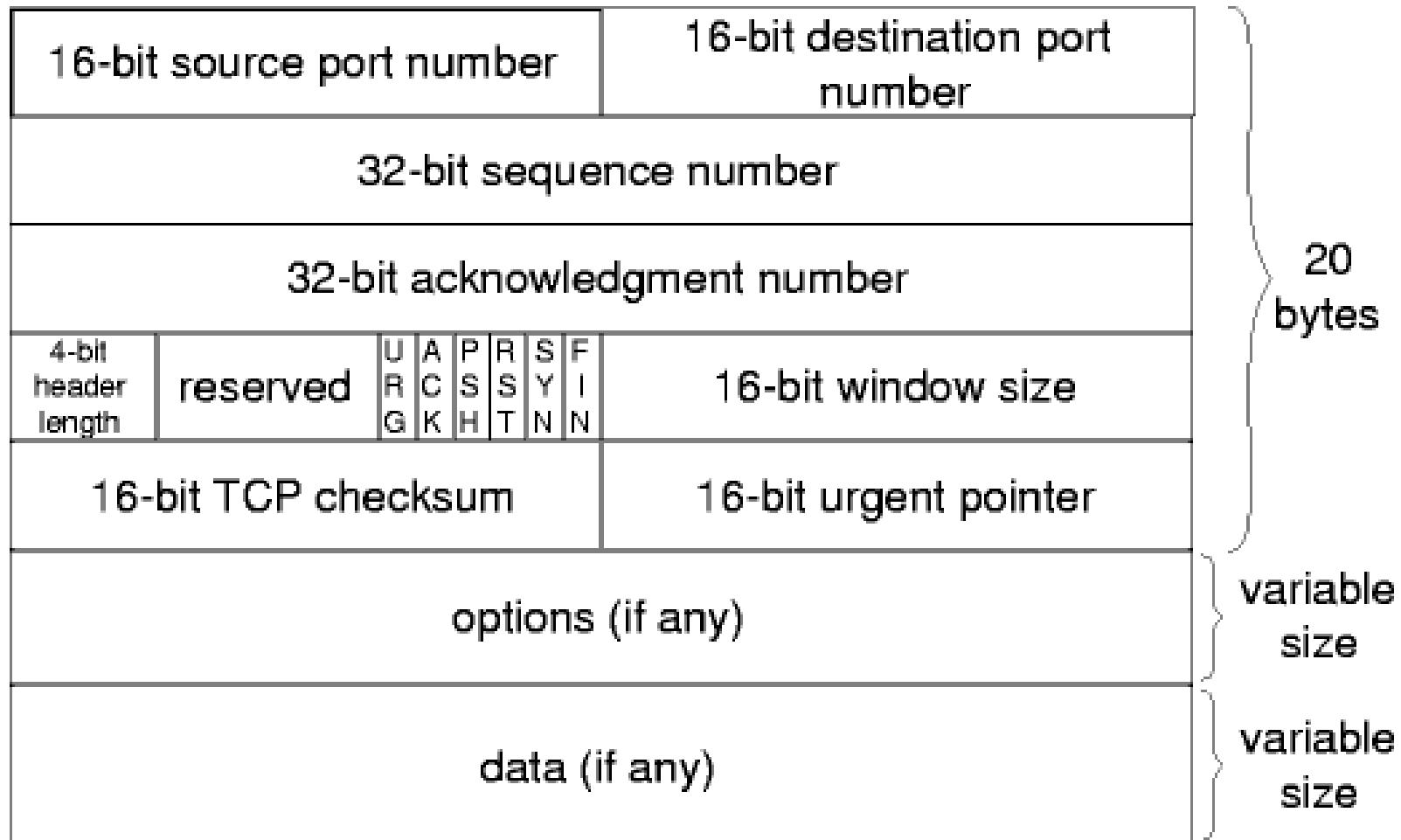
Pseudo header SRC=10.0.2.10, DST=10.0.4.10, PROTO=17  
0A00 020A 0A00 040A 0011

UDP header SRCP=9000, DSTP=7, LEN=15, LEN-PH=15, CKSUM=0  
2328 0007 000F 000F

DATA 5445 5354 0A00 FF[00] [00] = v. padding

00001010	00000000	= 10.0
00000010	00001010	= 2.10
00001010	00000000	= 10.0
00000100	00001010	= 4.10
00000000	00010001	= 17
00100011	00101000	= 9000
00000000	00000111	= 7
00000000	00001111	= 15
00000000	00001111	= 15
01010100	01000101	
01010011	01010100	
00001010	00000000	
11111111	00000000	
-----		
{1}11101110	00010111	
{1}		
-----		
~11101110	00011000	
00010001	11100111	EE18 ~EE18 = 11E7

## Segmento TCP



## Segmento TCP (Cont.)

- Puertos: MUX/DEMUX.
- No tiene Longitud total, si de HDR LEN (variable, max 60B Unit=4B).
- Total LEN se computa para PseudoHDR, no viaja en el segmento.
- Checksum:
  - Cálculo Ca1. Obligatorio, calculado, igual que UDP.
  - Si tiene error podría pedir retransmisión, implementación de TCP descarta y espera RTO (Retransmission Timer).
- Necesidad de manejar Timers, RTO (tmout. por cada segmento). (implementaciones lo manejan más eficiente).

## Segmento TCP (Cont.)

- Campos de Sesiones: Flags: SYN, FIN.
- Campo de Detección de Errores: Checksum.
- Campos de Control de Errores: ACK, Nro. Sec (#Seq), Nro. Ack (#Ack).
- Campo de Control de Flujo: a los de errores se agrega, Win.
- Campos de Congestión: agrega flags si participa la red.
- Máquina de estado finita por cada conexión.

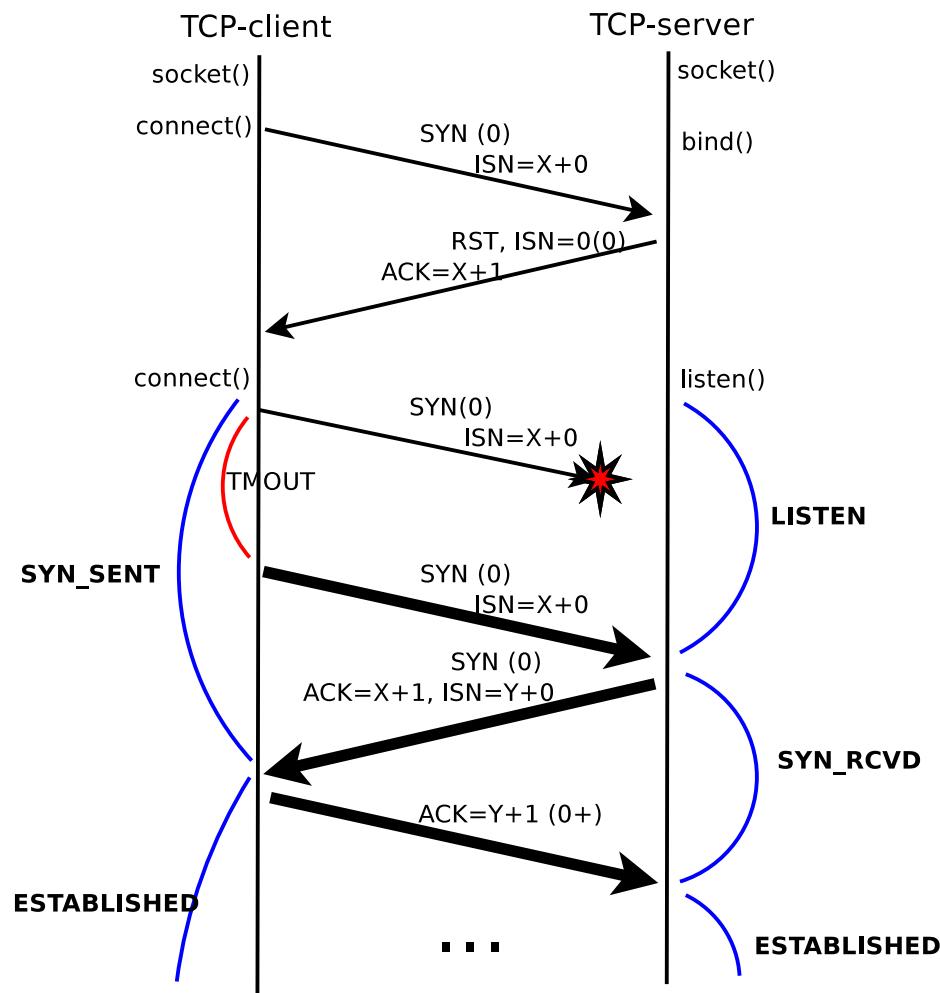
## Segmento TCP (Cont.)

- Permite Opciones y Negociación.
- TCP entrega y envía los datos agrupados o separados de forma dis-asociada de la aplicación:
  - La aplicación puede enviar 300 bytes en un write y TCP lo podría enviar en 3 segmentos separados de 100 bytes c/u.
  - La aplicación puede enviar 100 bytes y luego otros 200 y TCP esperar para enviarlos todos juntos.
  - La aplicación puede intentar leer 200 bytes del buffer y TCP solo entregar 150 bytes y luego el resto.

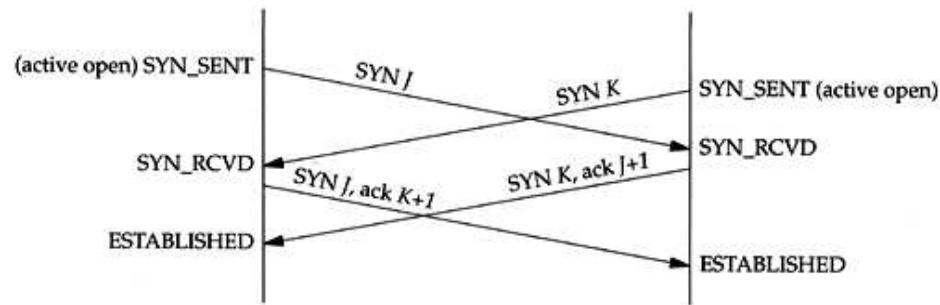
## TCP Establecimiento de Conexión

- Flags: SYN (Synchronize), ACK (Acknowledge) y RST (Reset).
- 3Way-Handshake (3WH).
- En el 3 segmento se puede enviar info.
- el ISN (Initial Sequence Number), se utiliza un contador que se incrementa cada 4 mseg.
- RST si no hay proceso en estado LISTEN.
- Open Pasivo (servidor) y Activo (cliente).
- Open simultáneo.

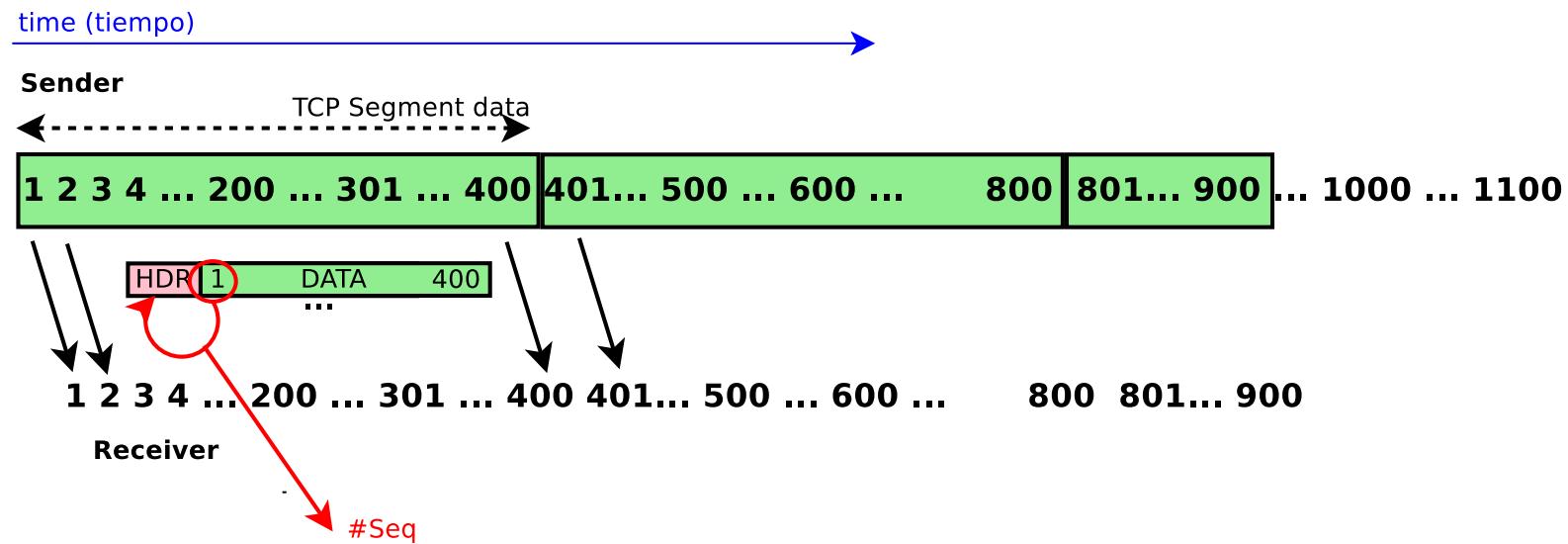
# 3 Way Handshake



## 3 Way Handshake (Open Simultâneo)



## Orientado a streams (secuencia en orden de bytes)



## Nros. de Secuencia/ACK TCP

Time	172.20.1.1	172.20.1.100	Comment
0.000	(41749) SYN	→ (11111)	Seq = 0
0.001	(41749) ← SYN, ACK	(11111)	Seq = 0 Ack = 1
0.001	(41749) ACK	→ (11111)	Seq = 1 Ack = 1
90.730	(41749) PSH, ACK - Len: 5	→ (11111)	Seq = 1 Ack = 1
90.730	(41749) ACK	→ (11111)	Seq = 1 Ack = 6
100.15!	(41749) PSH, ACK - Len: 16	→ (11111)	Seq = 1 Ack = 6
100.150	(41749) ← ACK	(11111)	Seq = 6 Ack = 17
104.58:	(41749) PSH, ACK - Len: 5	→ (11111)	Seq = 6 Ack = 17
104.58:	(41749) ACK	→ (11111)	Seq = 17 Ack = 11
112.29:	(41749) PSH, ACK - Len: 6	→ (11111)	Seq = 17 Ack = 11
112.29:	(41749) ← ACK	(11111)	Seq = 11 Ack = 23
114.89:	(41749) PSH, ACK - Len: 6	→ (11111)	Seq = 23 Ack = 11
114.89:	(41749) ← ACK	(11111)	Seq = 11 Ack = 29
120.620	(41749) FIN, ACK	→ (11111)	Seq = 29 Ack = 11
120.620	(41749) ← FIN, ACK	(11111)	Seq = 11 Ack = 30
120.620	(41749) ACK	→ (11111)	Seq = 30 Ack = 12

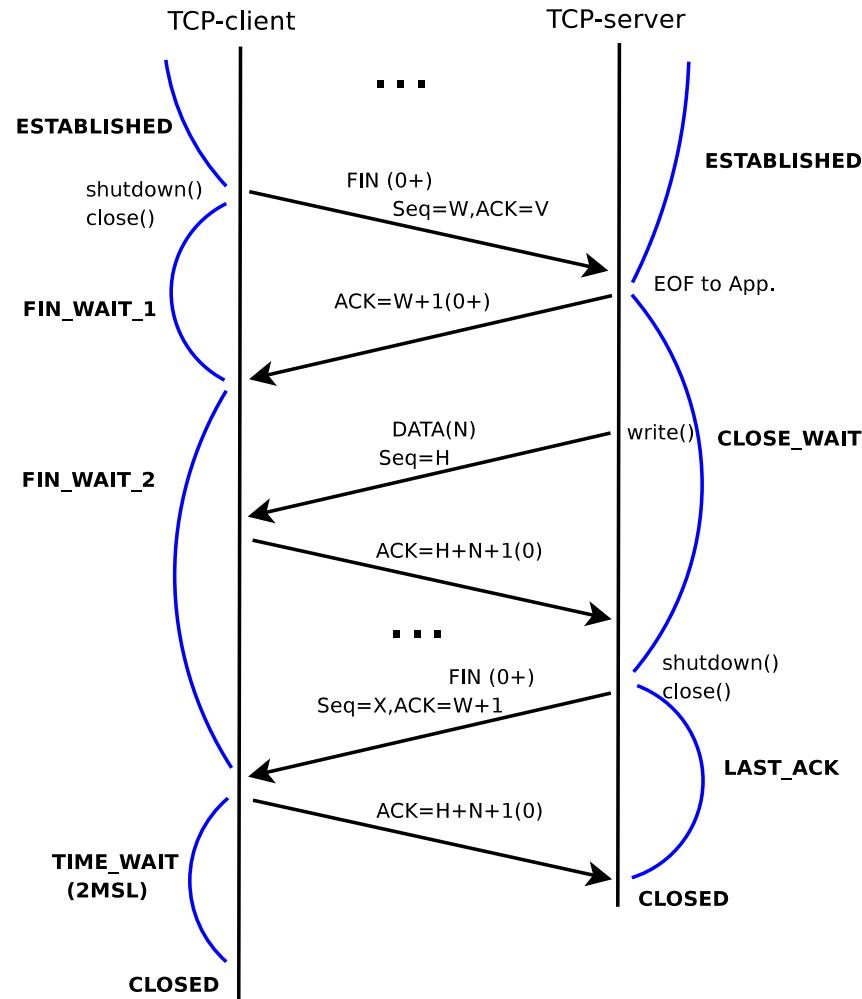
## Control de Errores TCP

- Errores que pueden existir en IP:
  - Pérdida de paquetes: descartados.
  - Des-ordenados, retardados.
  - Duplicados.
  - Corrompidos.
- TCP intenta solucionarlos con el control de Errores que implementa.

## TCP Cierre de Conexión

- Flags: FIN (Finish), ACK y RST.
- 4Way-Close (4WC).
- Posibilidad de Half-Close.
- Podría cerrarse en 3WC.
- Espera en TIME\_WAIT, 2MSL (aprox.  $2^*2\text{min}$ ).
- Evitar con SO\_REUSEADDR.
- Cierre incorrecto con RST.
- Close simultáneo.

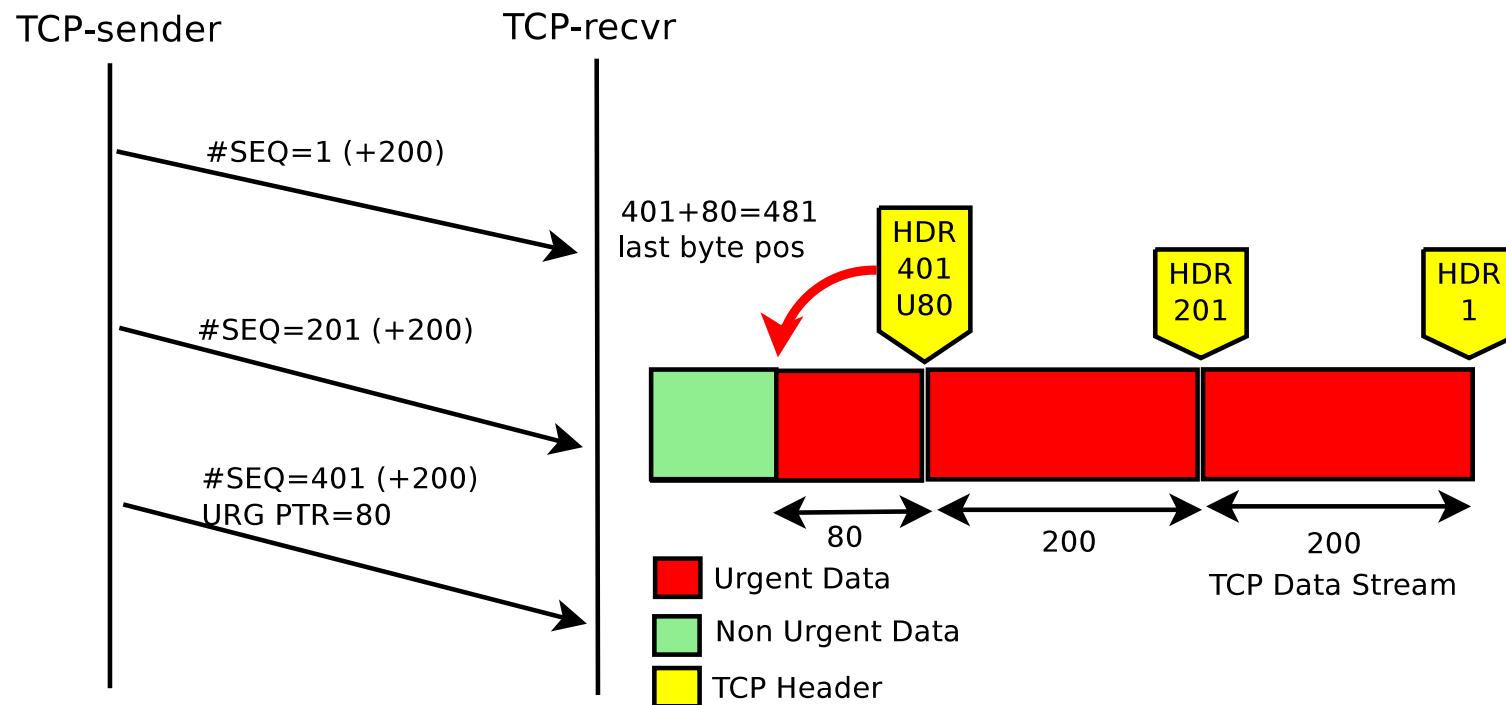
# TCP Close



## Otros campos TCP

- TCP entrega y envía los datos agrupados o separados de forma dis-asociada de la aplicación.
- Datos Urgentes: URG.
  - Urgent Pointer válido si URG=1.
  - Indica: offset positivo + #Seq = last Data Urgent byte.
  - Indicar a la App. datos urgentes, debe leer.
  - Debería combinarse con PSH. Habitualmente llamado OOB data (TCP no soporta OOB!!!).
- Pushear datos: PSH.
  - Fuerza a TCP a pasar datos a la App.
  - No lo deja “Buffear” los datos recibidos (Input).

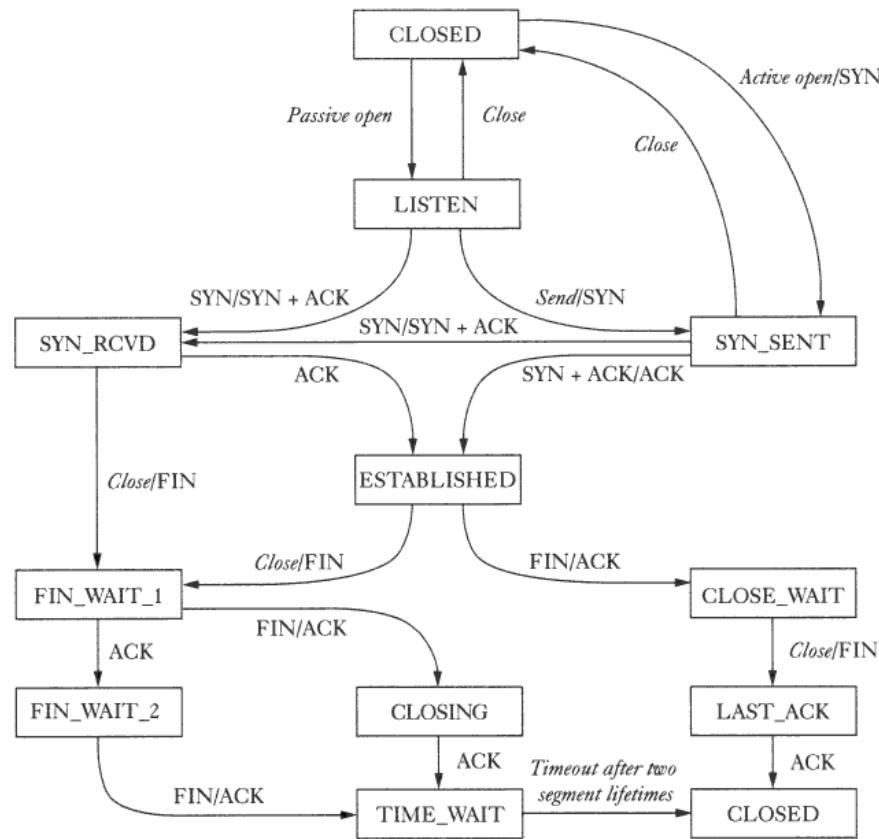
## Urgent Pointer



## Opciones TCP

- Maximum Segment Size (MSS), min. recomendado 536B, RFC-879 (basado en MTU=576, TCP+IP=40). Aclaraciones en RFC-6691.
- Window Scaling.
- Selective Acknowledgements (SACK).
- Timestamps.
- NOP.
- Otras.

# TCP Diagrama de Estados, Reducido



## Fuentes de Información

- Kurose/Ross: Computer Networking (6th Edition).
- TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens, K. Fall (2nd. ed).
- RFCs: <http://www.faqs.org/rfcs/rfc793>, rfc798, ...
- Wikipedia <http://www.wikipedia.org>.
- CS 144: Introduction to Computer Networking, Stanford Course.
- TCP/IP Guide: <http://www.tcpipguide.com/>.
- Internet ...

# Transporte: Control de Errores(S&W)

2018



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Contenidos

## 1 Introducción a Control de Errores

### 2 Canal Confiable

- Algoritmo Base

### 3 Canal con Errores

- Canal con algunos errores: Se corrompen datos
- Algoritmo Retrans (Deteción de Errores)
- Canal con algunos errores: Se corrompen datos y ACK/NAK
- Canal con pérdida de Datos
- Algoritmo: Stop&Wait (S&W) v2.1.1
- Algoritmo Free NAK
- Algoritmo S&W + timer, Free NAK
- Algoritmo: Stop&Wait + Bit Counter para Data/ACK
- Canal con Mensajes fuera de Orden y DUPs

## 4 Análisis y Conclusiones

## 5 Referencias

# Contenidos

- 1 Introducción a Control de Errores
- 2 Canal Confiable
  - Algoritmo Base
- 3 Canal con Errores
  - Canal con algunos errores: Se corrompen datos
  - Algoritmo Retrans (Deteción de Errores)
  - Canal con algunos errores: Se corrompen datos y ACK/NAK
  - Canal con pérdida de Datos
  - Algoritmo: Stop&Wait (S&W) v2.1.1
  - Algoritmo Free NAK
  - Algoritmo S&W + timer, Free NAK
  - Algoritmo: Stop&Wait + Bit Counter para Data/ACK
  - Canal con Mensajes fuera de Orden y DUPs
- 4 Análisis y Conclusiones
- 5 Referencias

# Contenidos

- 1 Introducción a Control de Errores
- 2 Canal Confiable
  - Algoritmo Base
- 3 Canal con Errores
  - Canal con algunos errores: Se corrompen datos
  - Algoritmo Retrans (Detección de Errores)
  - Canal con algunos errores: Se corrompen datos y ACK/NAK
  - Canal con pérdida de Datos
  - Algoritmo: Stop&Wait (S&W) v2.1.1
  - Algoritmo Free NAK
  - Algoritmo S&W + timer, Free NAK
  - Algoritmo: Stop&Wait + Bit Counter para Data/ACK
  - Canal con Mensajes fuera de Orden y DUPs
- 4 Análisis y Conclusiones
- 5 Referencias

# Contenidos

- 1 Introducción a Control de Errores
- 2 Canal Confiable
  - Algoritmo Base
- 3 Canal con Errores
  - Canal con algunos errores: Se corrompen datos
  - Algoritmo Retrans (Detección de Errores)
  - Canal con algunos errores: Se corrompen datos y ACK/NAK
  - Canal con pérdida de Datos
  - Algoritmo: Stop&Wait (S&W) v2.1.1
  - Algoritmo Free NAK
  - Algoritmo S&W + timer, Free NAK
  - Algoritmo: Stop&Wait + Bit Counter para Data/ACK
  - Canal con Mensajes fuera de Orden y DUPs
- 4 Análisis y Conclusiones
- 5 Referencias

# Contenidos

- 1 Introducción a Control de Errores
- 2 Canal Confiable
  - Algoritmo Base
- 3 Canal con Errores
  - Canal con algunos errores: Se corrompen datos
  - Algoritmo Retrans (Detección de Errores)
  - Canal con algunos errores: Se corrompen datos y ACK/NAK
  - Canal con pérdida de Datos
  - Algoritmo: Stop&Wait (S&W) v2.1.1
  - Algoritmo Free NAK
  - Algoritmo S&W + timer, Free NAK
  - Algoritmo: Stop&Wait + Bit Counter para Data/ACK
  - Canal con Mensajes fuera de Orden y DUPs
- 4 Análisis y Conclusiones
- 5 Referencias

# Control de Errores

- Se requiere un mecanismo de control sobre un canal no confiable.
- Se realiza con ARQ: Automatic Repeat reQuest/Automatic Repeat Query.
- Utiliza confirmaciones para validar que los datos se recibieron OK.
- Estudiar que otros mecanismos se necesitan para agregar confiabilidad.

**Notas:** Análisis basado en slides "Supplements: Powerpoint Slides Computer Networking: A Top-Down Approach 6th ed. J.F. Kurose and K.W. Ross", aunque no utiliza exactamente las mismas funciones y llamadas.

Los algoritmos están escritos en pseudo-código a la "C" y solo están como referencia, pueden contener errores ya que no fueron compilados ni testeados.

Este texto no explica como funciona TCP, sino es un enfoque simplificado y progresivo para entender los problemas que TCP debe intentar solucionar.

# Control de Errores, Implementación

- Para simplificar, se supone solo un emisor y un receptor, solo se transmite en un sentido datos y no se requiere elegir origen ni destino.
- Interfaz/API pública de la capa que ofrece los servicios confiables:

```
rdt_rcv(byte_t * data); // asociada a la func. (non-block/block)
                        // deliver_data(byte_t *data);
```

```
rdt_snd(byte_t *data); // asociada a la func. (block)
                      // rdt_send(byte_t *data);
```

- Interfaz/API pública de la capa inferior no confiable:

```
int udt_send(packet_t *pkt); // Non-block
```

```
int udt_recv(packet_t *pkt); // Block
```

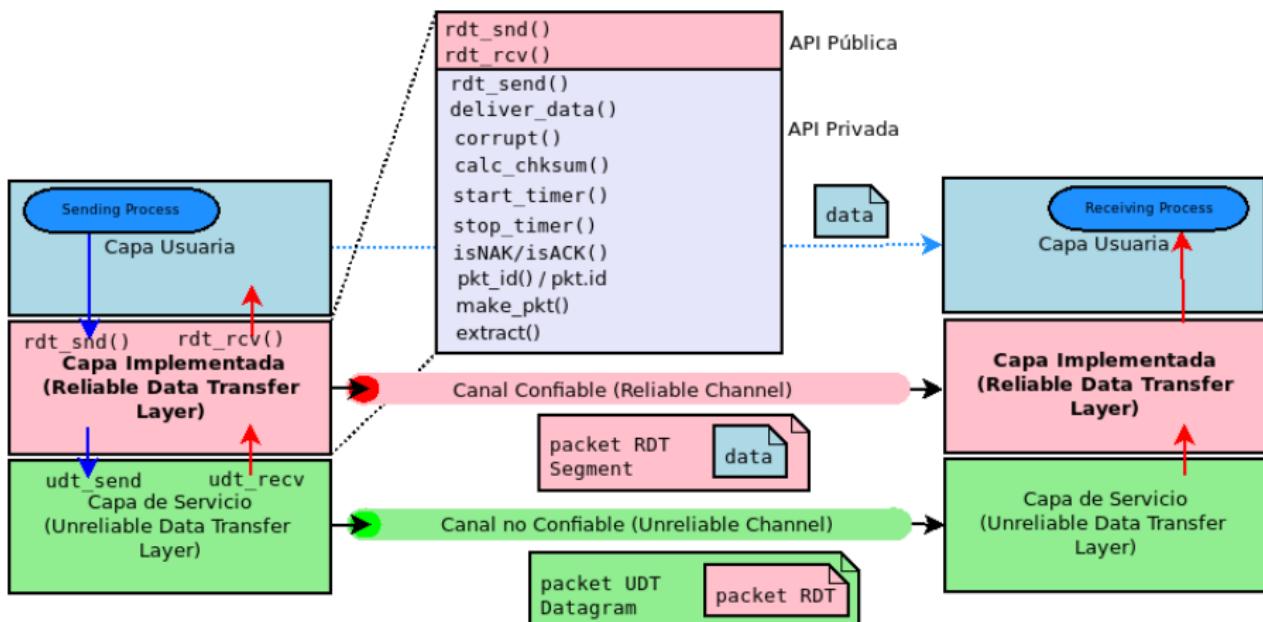
# Control de Errores, Procesos

- Se implementa con 2 procesos:
  - **Sender**, para procesar los requerimientos de enviar datos de la capa superior mediante la llamada `rdt_snd()`.
  - **Receiver**, para procesar los requerimientos de recibir datos de la capa superior mediante la llamada `rdt_rcv()`.
- El **Sender** invoca a `rdt_send()` y se bloquea hasta que la capa superior lo llame mediante `rdt_snd()` dejando datos. En ese momento los datos de la capa superior se copian al buffer/espacio de la capa que implementa el transporte confiable y se desbloquea **Sender**.
- El **Sender** para pasar los datos a la capa inferior y enviarlos invoca a `udt_send()`, se copian los datos y se desbloquea. Asincrónico.

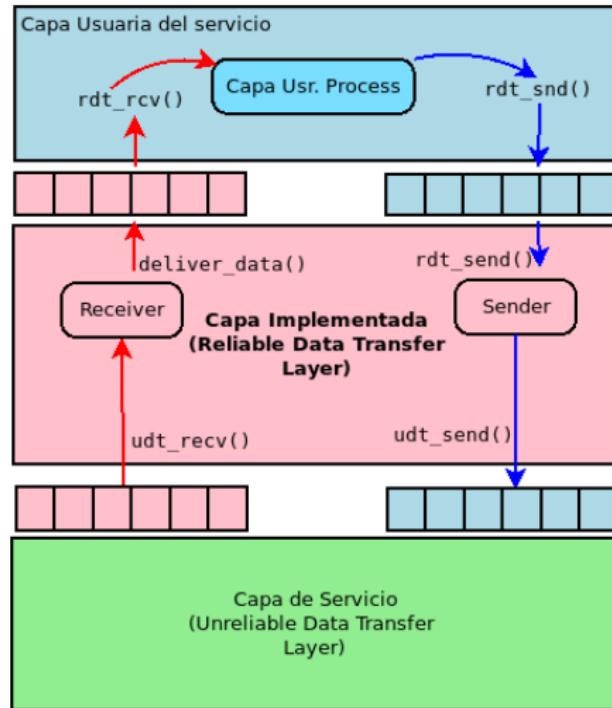
# Control de Errores, Procesos (Cont.)

- El **Receiver** invoca a `udt_recv()` y se bloquea hasta que la capa inferior tenga paquetes en el buffer y los pueda entregar.
- El **Receiver** una vez que proceso los datos recibidos los pasa a la capa superior mediante `deliver_data()`. En ese momento se copia al espacio de buffer de la capa superior y el proceso **Receiver** puede seguir su curso sin bloquearse. La capa superior los puede recibir de forma asíncrona mediante `rdt_rcv()` si tiene buffering, sino será bloqueante.
- Se considera que el envío desde el transporte a implementar a la capa inferior no es bloqueante (Buffer infinito).

# Control de Errores, API



# Control de Errores, API, Comunicación local



# Canal Confiable/Errores Posibles

- No se pierden datos.
- No se duplican.
- No se desordenan.
- No se corrompen.
- No existe delay mayor de 1 RTT.

En conclusión, no hay errores, No hay problemas. Para diferenciar que estamos en presencia de un canal confiable las operaciones utilizadas sobre la capa de servicio son llamadas:

```
int R_dt_send(packet_t *pkt); // en lugar de udt_send()  
int R_dt_recv(packet_t *pkt); // en lugar de udt_recv()
```

```

void sender1()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;

    // Mientras no se termine la comunicación
    while (result != EOT)
    {
        O0: // Emisor recibe datos de capa superior
            // y arma PDU desde data
            result = rdt_send(data);

            // Arma paquete
            pkt = make_pkt(data)
            //pkt->payload = data;

        O1: // Emisor envía datos
            R_dtt_send(pkt); // udt_send(pkt);

        O2: // Emisor vuelve a 0
            // goto O0 — esta en el while
    }
}

```

```

void receiver1()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;

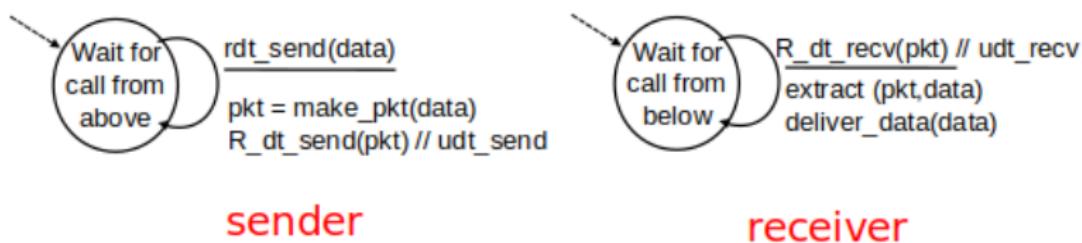
    // Mientras no se termine la com.
    while (result != EOT)
    {
        O0: // Receptor recibe datos de capa inf.
            R_dt_recv(pkt); // udt_recv(pkt);

        O1: // Receptor entrega datos a
            // capa superior data = pkt->payload
            extract(pkt, data);
            result = deliver_data(data);

        O2: // Receptor vuelve a 0
            // goto O0 — esta en el while
    }
}

```

# Algoritmo Base - FSM Canal Confiable



# Análisis de Algoritmo Base

- Un solo loop por cada proceso.
- Se supone que los datos siempre llegan y en orden.
- No hay sincronismo entre emisor y receptor, buffer ilimitado.
- Escenario irreal. Qué sucede si se cambia ?
- Se reemplaza:

```
R_dt_send(data); por udt_send(data);  
R_dt_recv(data); por udt_recv(data);
```

El algoritmo deja de ser seguro, los errores del canal aparecen en la transmisión y la solución no funciona.

## Canal con algunos errores, corrompen Datos

- No se pierden datos.
- No se duplican.
- No se desordenan.
- Sí se corrompen datos, se chequean con Checksum/CRC o mecanismo similar, se avisa cuando no se reciben de forma correcta.
- Utilización de confirmaciones ACK/NAK.
- No se corrompen las confirmaciones ACK/NAK.
- No hay delay mayor de 1 RTT.

# Algoritmo Retrans (Detección de Errores)

- Se debe cambiar la estructura de la PDU de la capa que hace el control de errores.
- Se agrega campo para checksum/CRC y se agrega un flag para indicar si es paquete de datos o de confirmación.
- Se implementa cálculo de código de detección de errores en Sender2:01
- Se implementa chequeo de errores y confirmación Receive2:01.
- Se implementa chequeo de feedback Sender2:03.
- Si los datos llegan corruptos no se pasan a la capa superior.

# Estructura de Datos para Algoritmo Retrans

- Estructura tentativa del paquete/segmento.

```
typedef enum { ack , nak , dat } type_t ;  
  
typedef struct pk {  
    // Header  
    int      len ;  
    type_t   ack;   // ACK | NAK | DATA  
    ck_t     chksum ;  
    // Data  
    byte_t   payload [...] ;  
} packet_t ;
```

```

void sender2()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;
    packet_t *answer = NULL;

    // Mientras no se termine la comunicación
    while (result != EOT)
    {
        O0: // Emisor recibe datos de capa
            // superior y arma PDU desde data
            result = rdt_send(data);

            // Arma paquete
            pkt = make_pkt(data);

        O1: //pkt->header = ...
            calc_chksum(pkt);

            // answer == NAK
            while (isNAK(answer))
            {
                ...
            }
    }
}

```

```

void receiver2()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;
    int ok = 0;

    // Mientras no se termine la com.
    while (result != EOT)
    {
        while (!ok)
        {
            O0: // Receptor recibe datos
                udt_recv(pkt);

            O1: // Receptor chequea datos
                // y confirma con ACK/NAK
                if (!corrupt(pkt))
                {
                    udt_send(ACK);
                    ok = 1;
                }
                else
                {
                    udt_send(NAK);
                    ok = 0;
                }
        }
        ...
    }
}

```

```

...
while (isNAK(answer))
{
O2:    // Emisor envía datos:
        udt_send(pkt);

O3:    // Emisor espera respuesta
        // answer ::= ACK | NAK
        udt_recv(answer);

    } // Emisor se queda
    // retransmitiendo el mismo dato
    // isNAK(answer)

O4: // Emisor vuelve a 0
    // goto O0 — esta en el while
}
} // isACK(answer)
}

```

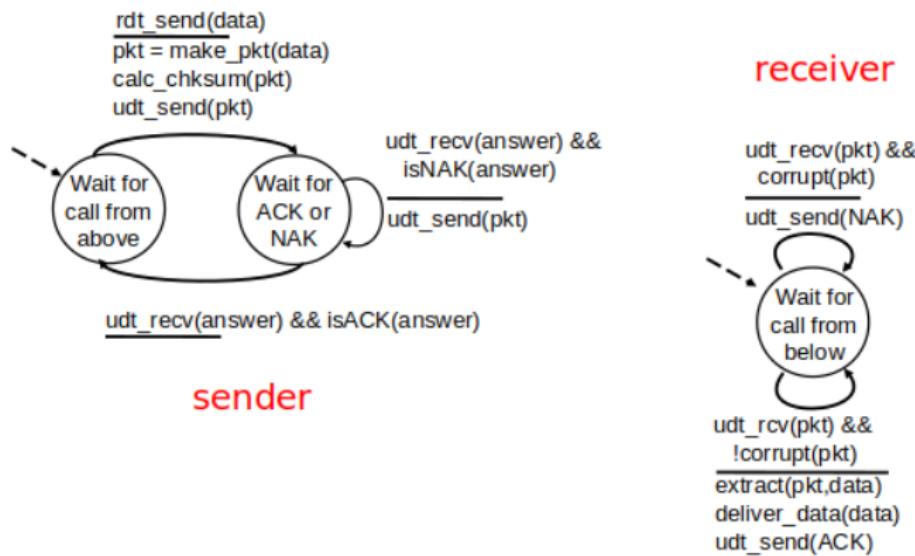
```

...
O2: // Receptor entrega datos a
    // capa superior data=pkt->payload
    extract(pkt, data);
    result = deliver_data(data);

O3: // Receptor vuelve a 0
    // goto O0 — esta en el while
}
}

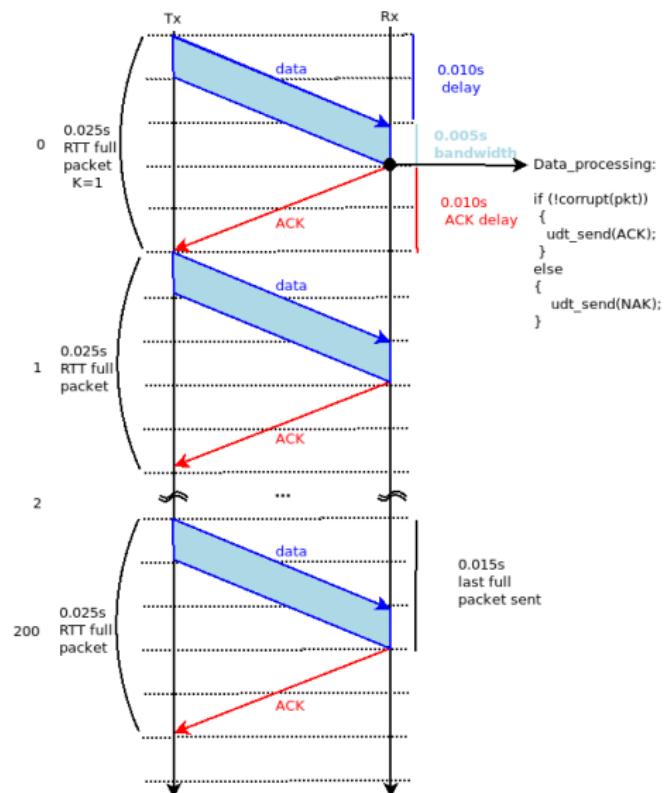
```

# FSM Canal Errores Algoritmo Retrans, data checksum



# Análisis de Algoritmo Retrans

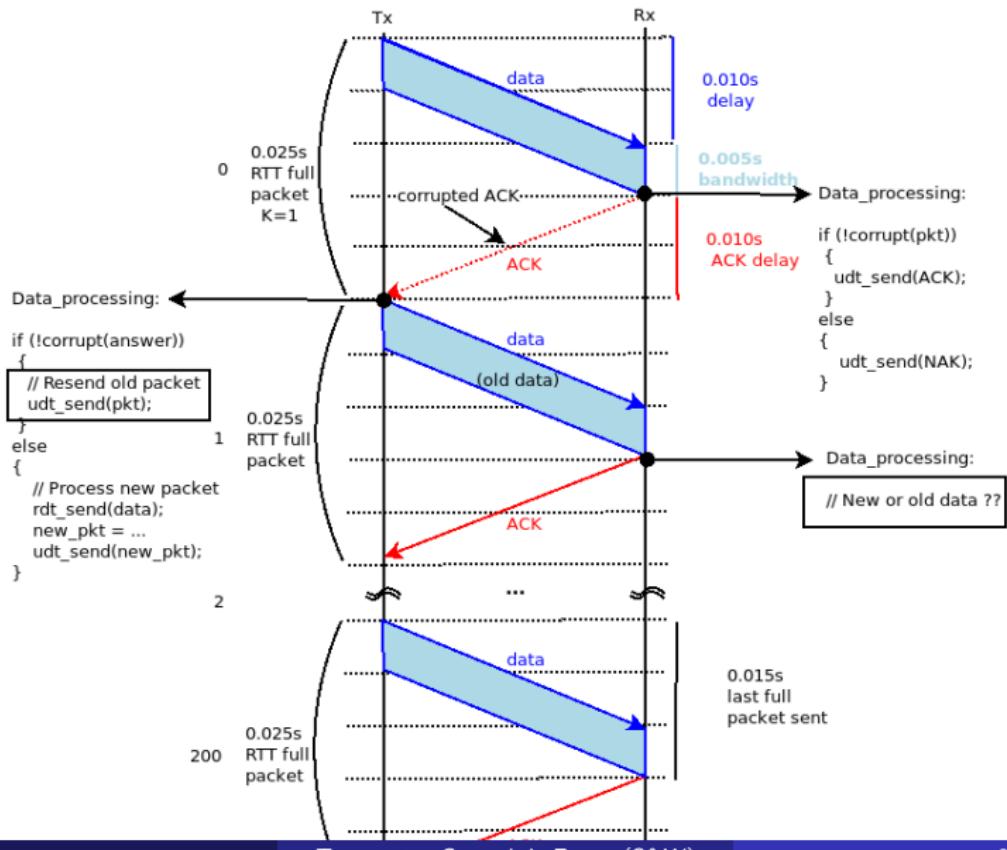
- Ahora emisor y receptor sincronizados, no se requiere buffering entre proceso usuario y RDT. Max=1 dato.
- De a un paquete/segmento por vez.
- Hasta que no se confirma no se pasa al siguiente segmento.
- Para ACK/NAK no requiere calcular el chksum porque se supone seguro.
- Conocido como mecanismo Stop & Wait (S&W), v2.0.



# Problemas de ACK/NAK corruptos

- Se pre-suponía entrega de datos NO confiable en cierto grado, se pueden corromper pero siempre llegan. Se suponía además que ACK/NAK llegaban OK.
- Problema, si cambiamos a un modelo aún más real, donde la capa subyacente puede corromper los NAK/ACK.
- Cálculo de checksum ACK. Qué hacer ante un NAK/ACK corrupto ?
- Se debería retransmitir.
  - Si el corrupto fue un NAK, no hay problema, pero
  - Si fue un ACK el receptor recibirá un mensaje duplicado y no podrá distinguirlo (DUP).
- Se debe identificar el mensaje de dato para que el receptor sepa si es un duplicado, ya que no sabe que el ACK se corrompió, se utiliza ID binario: (0,1).

# Problemas de ACK/NAK corruptos (gráfico)



# Estructura de Datos para Algoritmo S&W con Bit-Counter

- Estructura tentativa del paquete/segmento.

```
typedef enum {ack , nak , dat } type_t ;  
  
typedef struct pk {  
    // Header  
    int      len ;  
    type_t   ack ;    // ACK | NAK | DATA  
    int      id ;    // seq 0,1  
    ck_t     cksum ;  
    // Data  
    byte_t   payload [ ... ] ;  
} packet_t ;
```

```

void sender2.1()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;
    packet_t *answer = NULL;

    // Mientras no se termine la comunicación
    while (result != EOT)
    {
        O0: // Emisor recibe datos de capa
            // superior y arma PDU desde data
            result = rdt_send(data);
            // Arma paquete
            // pkt->header.id = 0
            pkt = make_pkt(0,data); // ID==0
        O1: //pkt->header = ...
            calc_chksum(pkt);
            udt_send(pkt);
        O2: // recv && (NAK || corrupt)
            while ((udt_resv(answer)&&(!isACK(answer)))
            {
                udt_send(pkt);
            } // isACK ...
    }
}

```

```

void receiver2.1()
{
    packet_t *pkt;
    packet_t *answer;
    byte_t *data;
    result_t result = OK;
    int ok = 0;
    int wait_id = 0;

    // Mientras no se termine la com.
    while (result != EOT)
    {
        O0: // Receptor recibe datos
            udt_recv(pkt);

        O1: // Receptor chequea datos
            // y confirma con ACK/NAK
            if (!corrupt(pkt)) {
                answer = make_pkt(ACK);
            }
            else {
                answer = make_pkt(NAK);
            }
            calc_chksum(answer);
            udt_send(answer);
            ...
    }
}

```

```

O3: ...
    // Emisor recibe nuevos datos de capa
    // superior y arma PDU desde new data
    result = rdt_send(data);

    // Arma paquete con nuevo id
    // pkt->header.id = 1
    pkt = make_pkt(1,data); // ID==1

O4: //pkt->header = ...
    calc_cksum(pkt);
    udt_send(pkt);

O5: // recv && (NAK || corrupt)
    while ((udt_resv(answer)&&(!isACK(answer)))
    {
        udt_send(pkt);
    }

O6: // Emisor vuelve a 0
    // goto O0 — esta en el while
}
}

```

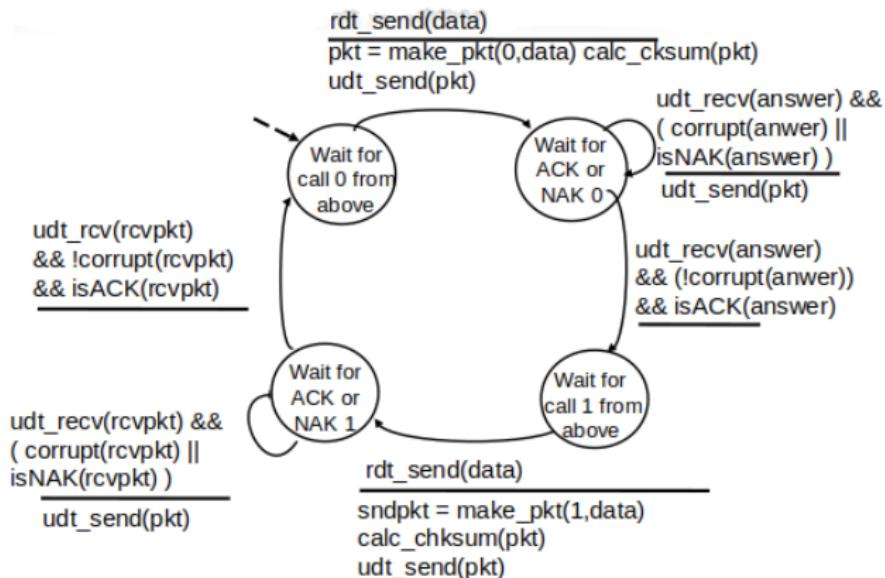
```

...
O2: ...
    // Receptor entrega datos a
    // capa superior data=pkt->payload
    // Solo si el id es el que esperaba
    if (pkt->id == wait_id)
    {
        extract(pkt, data);
        result = deliver_data(data);
        wait_id = (wait_id + 1) % 2;
    }

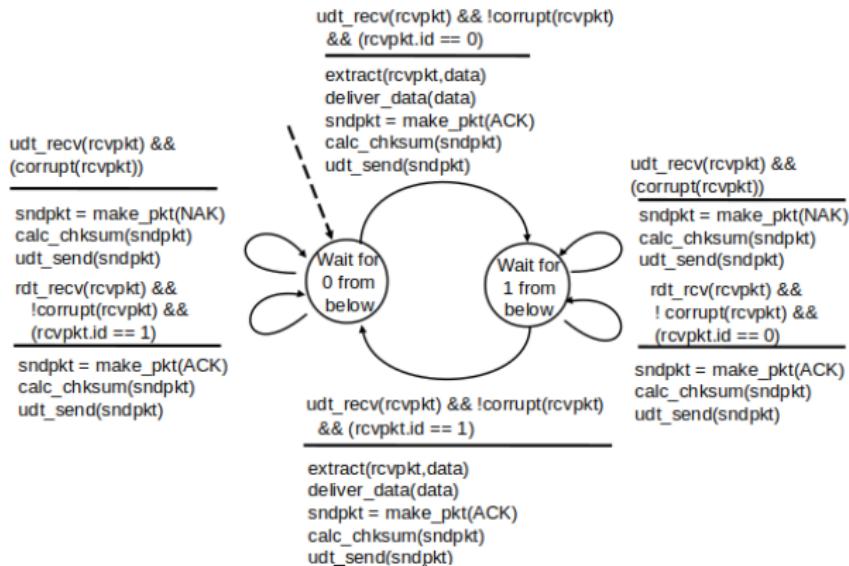
O3: ...
    // Receptor vuelve a 0
    // goto O0 — esta en el while
}
}

```

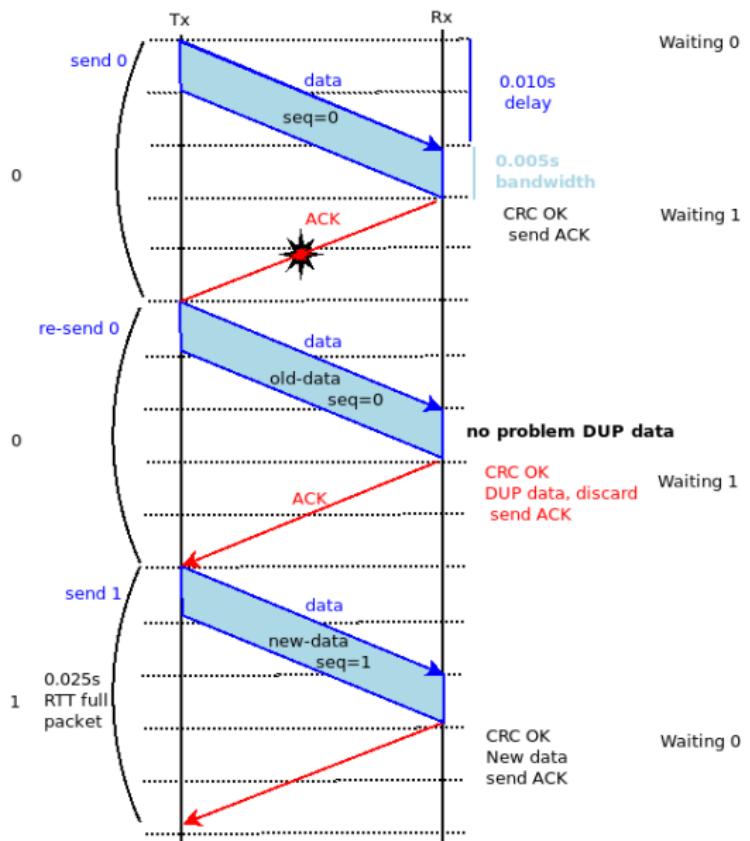
# FSM Canal Errores data/ack cksum, Send



# FSM Canal Errores data/ack chksum, Recv



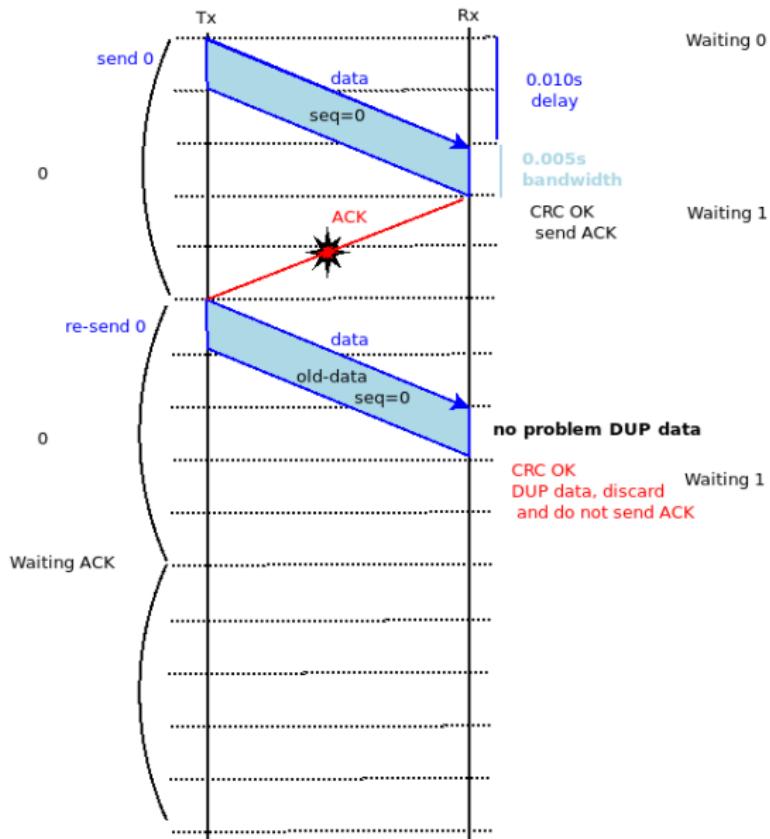
# ACK/NAK corruptos con Bit Counter en Data



# Análisis de Algoritmo para ACK/NAK corruptos

- Valores binarios (0,1) alcanza, por qué?
- Sender debe chequear si el ACK/NAK esta corrupto.
- Sender se queda en el estado hasta que se asegura que el ACK llegó OK.
- El emisor se puede simplificar con loop y un módulo 2 para los nros. de secuencia y no copiar el código.
- El receptor debe chequear que el paquete recibido no sea duplicado.
- Siempre debe confirmar, no importa si recibe un duplicado. No sabe si se recibió OK en el emisor. Si no confirma, el emisor se queda bloqueado esperando el ACK/NAK.

# Problema si no confirma DUP



# Canal con pérdida de Datos

- Se presuponía entrega de datos NO confiable en cierto grado, se pueden corromper pero siempre llegan.
- Problema, si cambia a un modelo aún más real, donde la capa subyacente no es confiable en el grado en el que se pueden perder los datos, no solo corromper.
- Para el primer análisis se vuelve a relajar la corrupción de los ACK/NAK. No hay ACK/NAK corruptos.
- Al perderse los datos, con los algoritmos anteriores, se quedarían bloqueados los procesos en `udt_recv()`, uno esperando por el dato y el otro por la confirmación.

## Canal con nuevos errores, pérdida de Datos

- Sí se pierden datos, no los ACK/NAK.
- No se duplican.
- No se desordenan.
- Sí se corrompen datos:
  - Se chequean con Checksum/CRC o mecanismo similar.
  - No se pasan a la capa superior si tienen errores.
  - Requiere mecanismo para avisar al emisor del error.
- No se corrompen las confirmaciones, ACK/NAK.
- No hay delay mayor de 1 RTT.

# Algoritmo: Stop&Wait (S&W) v2.1.1

- Se agregar timeout en Sender2.1.1:01 y Sender2.1.1:04.
- Se debe indicar donde se detiene el timer.
- Ante un error de checksum/CRC el receptor podría omitir el NAK ?
- Si los ACK/NAK son seguros no se requiere identificar los mensajes de datos, ID (0,1).
- Cómo calcular el timer? En base al RTT.
- Siempre es el mismo el RTT?

```

void sender2.1.1()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;
    packet_t *answer = NULL;

    // Mientras no se termine la comunicación
    while (result != EOT)
    {
O0: // Emisor recibe datos de capa
        // superior y arma PDU en data:
        result = rdt_send(data);

        // Arma paquete
        pkt data = make_pkt(data);
        calc_chksum(pkt);
        // TMOUT | NAK
        while ( ! isACK(answer) )
        {
O1:    // Antes de enviar genera timer,
            // si habia uno corriendo, reset
            start_timer(RTT+delta);
...
    }
}

```

```

void receiver2.1.1()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;
    int ok = 0;

    // Mientras no se termine la com.
    while (result != EOT)
    {
        while (!ok)
        {
O0:    // Receptor recibe datos
            // desde capa inferior
            udt_recv(pkt);

O1:    // Receptor chequea datos
            // y confirma con ACK/NAK
            if (!corrupt(pkt)) {
                udt_send(ACK); ok = 1;
            }
            else {
                udt_send(NAK); ok = 0;
            }
        }
...
    }
}

```

```

...
O2:    // Emisor envía datos:
        udt_send(pkt);

O3:    // Emisor espera respuesta
        // o timeout, valor de salida
        // answer ::= ACK | NAK | TMOUT
        udt_recv(answer);

    } // Emisor se queda
    // retransmitiendo el mismo dato

O4: // Se detiene timer
    stop_timer();
    // Emisor vuelve a 0
    // goto O0 — esta en el while
}

}

```

```

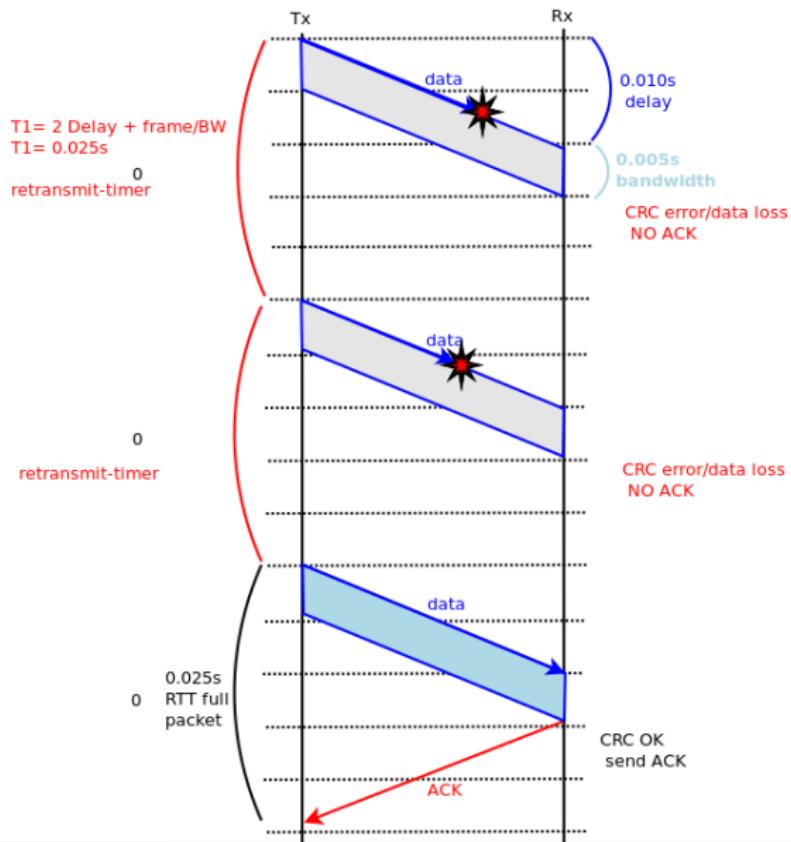
...
O2:    // Receptor entrega datos
        // a capa superior data=pkt->payload;
        data = extract(pkt, data);
        result = deliver_data(data);

O3:    // Receptor vuelve a 0
        // goto 0 — esta en el while
    }

}

```

# Stop & Wait 2.1.1



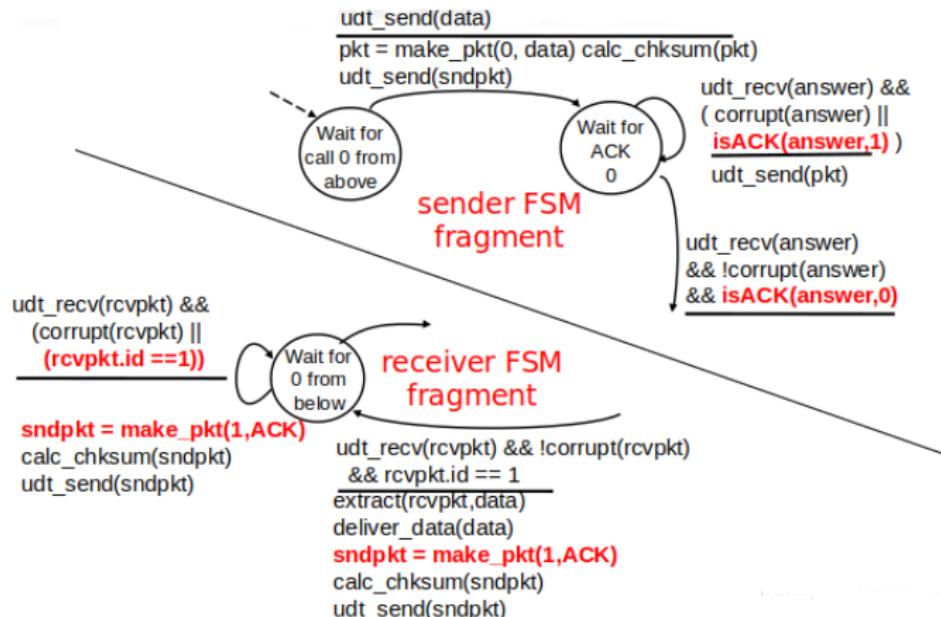
# Problemas de S&W v2.1.1

- Estamos suponiendo que tenemos entrega confiable de confirmaciones, ACK/NAK:
  - No se pierden ACK/NAK.
  - No se corrompen ACK/NAK.
- Problema, si cambiamos a un modelo más real, donde la capa subyacente no es confiable tampoco para las confirmaciones, como v2.1.
- Se podrían corromper los ACK/NAK. Habría que retransmitir.
- Al perderse los datos se retransmiten con el timer, si se pierde el ACK/NAK es como que no llegó el mensaje y se retransmite por el timer también.
- El receptor puede recibir duplicados y no podrá distinguirlos, necesidad de identificar los mensajes, como en 2.1.

# Algoritmo Free NAK

- Tratar de hacer un algoritmo que no necesite NAK.
- Alternativas:
  - Incluir nro. de secuencia explícito en ACK, indicar que dato recibió OK.
  - V2.2: si dato corrupto, se envía el nro. anterior., si dato OK, envío el que se recibió (módulo 2).
  - Se podría utilizar también que dato espera recibir (Confirmación hacia adelante). Si dato corrupto, envía el nro. que espera recibir, si dato OK, envía el siguiente módulo  $N$  (S&W,  $N = 2$ ).
  - V2.3: Usar timer en el emisor, si dato corrupto, no se confirma y se espera que el emisor retransmita como si se hubiese perdido (Como en v2.1.1).

# FSM Errores Algoritmo Free NAK sin timer v2.2



# Estructura de Datos para Algoritmo Free NAK v2.2

- En el ejemplo el id del dato se puede usar para el id del ACK, solo se envían datos en un sentido.
- Si hay envío de datos en los dos sentidos y se confirma en los mismos datos (piggy-backing) se requiere un id para el ACK.

```
typedef enum { ack , nak , dat } type_t ;
```

```
typedef struct pk {  
    // Header  
    int      len ;  
    type_t    ack ;      // ACK | NAK | DATA  
    int      id ;       // seq 0,1  
    int      ack_id : // seq 0,1 Admite piggy-backing  
    ck_t      checksum ;  
    // Data  
    byte_t   payload [ ... ] ;  
} packet_t ;
```

# Algoritmo: Stop&Wait (S&W) v2.3 - Free of NAK

- Con el algoritmo Stop&Wait v2.1.1 si se corrompe el dato podrá, omitir la confirmación y el emisor deberá retransmitir por timeout.
- Con el algoritmo Stop&Wait v2.1.1 si se corrompe el ACK, para el emisor es como si se perdiése el mensaje, dará timeout y retransmitirá.
- **ERROR !!!** el receptor pensará que es un nuevo dato y lo pasará a la capa superior como tal, cuando fue una retransmisión. Aparecen datos duplicados.
- Se debe evitar este problema, por ejemplo identificando con IDs los datos enviados como en v2.2, S&W + Bit Counter (0,1).

# Algoritmo: Stop&Wait (S&W) v2.3 - Free of NAK

- Cambiar Sender2.3:03

```
void sender2.3()
...
O3: // Emisor espera respuesta
    // answer ::= ACK | TMOUT | ERR(corrupt(answer))
    // || NAK (se suprime)
    udt_recv(answer);
...
```

- El envío de ACK, requiere agregar código para detección de errores como en 2.2.
- Se puede simplificar el Receiver, si se corrompe el dato, directamente se descarta y se espera timeout del emisor.

- Se debe agregar el envío de ID/SEQ para los mensajes de datos.
- En principio no requiere ID/SEQ para ACK si se mantiene restricción de RTT.

```
void sender2 .3()
{
    ...
    int      counter = 0;
    ...

    // Arma paquete
    pkt = make_pkt ( counter , data );
    ...

    counter = (counter + 1) % 2;
    ...
}
```

```

void receiver2 .3()
{
    packet_t *pkt;
    packet_t *answer;
    byte_t *data;
    result_t result = OK;
    int ok = 0;
    int wait_id = 0;

    // Mientras no se termine la comunicación
    while (result != EOT)
    {
        ok = 0;
        while (!ok)
        {
            O0: // Receptor recibe datos
            udt_recv(pkt);
            O1: // Receptor chequea datos
                // y confirma con ACK,
                // sino espera tmout
                if ((!corrupt(pkt))&&(pkt->id == wait_id))
                {
                    answer = make_pkt(ACK);
                    calc_cksum(answer);
                    udt_send(answer);
                    ok = 1;
                } // else discard and wait retrans
        }
        ...
        O2: // Receptor entrega datos a
            // capa superior data=pkt->payload
            // Solo si el id es el que esperaba
            // Es la única forma se salir
            // del while
            if (pkt->id == wait_id)
                extract(pkt, data);
                result = deliver_data(data);
                wait_id = (wait_id + 1) % 2;
                ok = 0;
        O3: // Receptor vuelve a 0
            // goto 0 — esta en el while
        }
    }
}

```

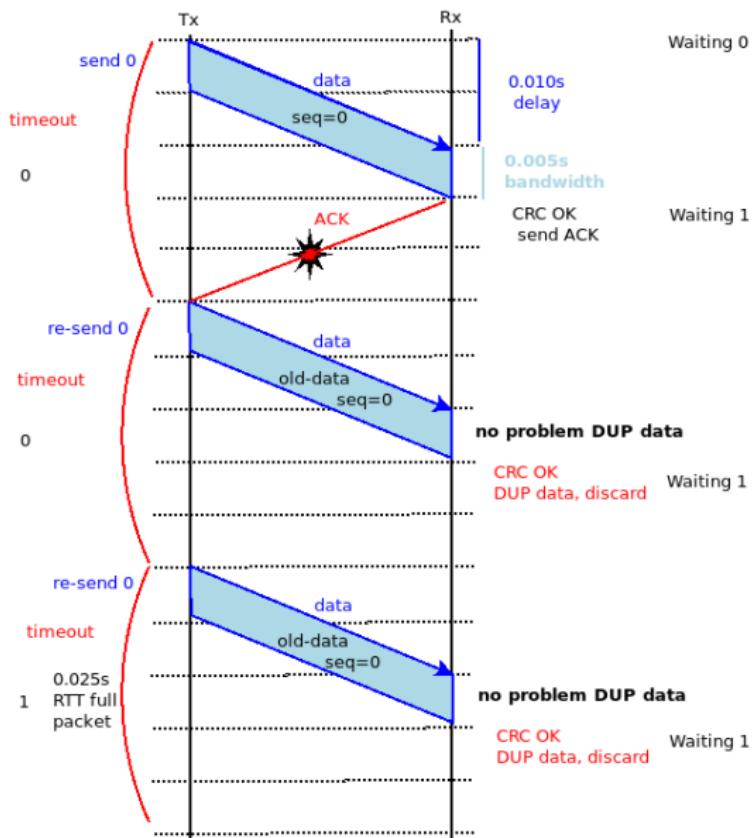
# S&W + Bit Counter, Canal con más errores

- El algoritmo v2.3 S&W + Bit Counter parece ser adecuado para el siguiente entorno:
  - Sí se pierden datos, y Sí se pierden los ACK/NAK.
  - No se duplican.
  - No se desordenan.
  - Sí se corrompen datos, se chequean con Checksum/CRC o mecanismo similar, se avisa cuando no se reciben de forma correcta.
  - Sí se corrompen las confirmaciones, ACK/NAK.
  - No hay delay mayor de 1 RTT.

# Problemas de S&W + Bit Counter

- **ERROR !!!**, el receptor no está confirmando los duplicados.
- Dará timeout en el emisor y volverá a enviar el mismo.
- Si al receptor le llegó bien y confirmo, pero la confirmación se corrompió o perdió, el emisor se queda enviando siempre el mismo, ya que el receptor espera el nuevo dato.
- El Receptor debe confirmar los DUPs.

# Problema Stop & Wait 2.3 (No ACK DUP)

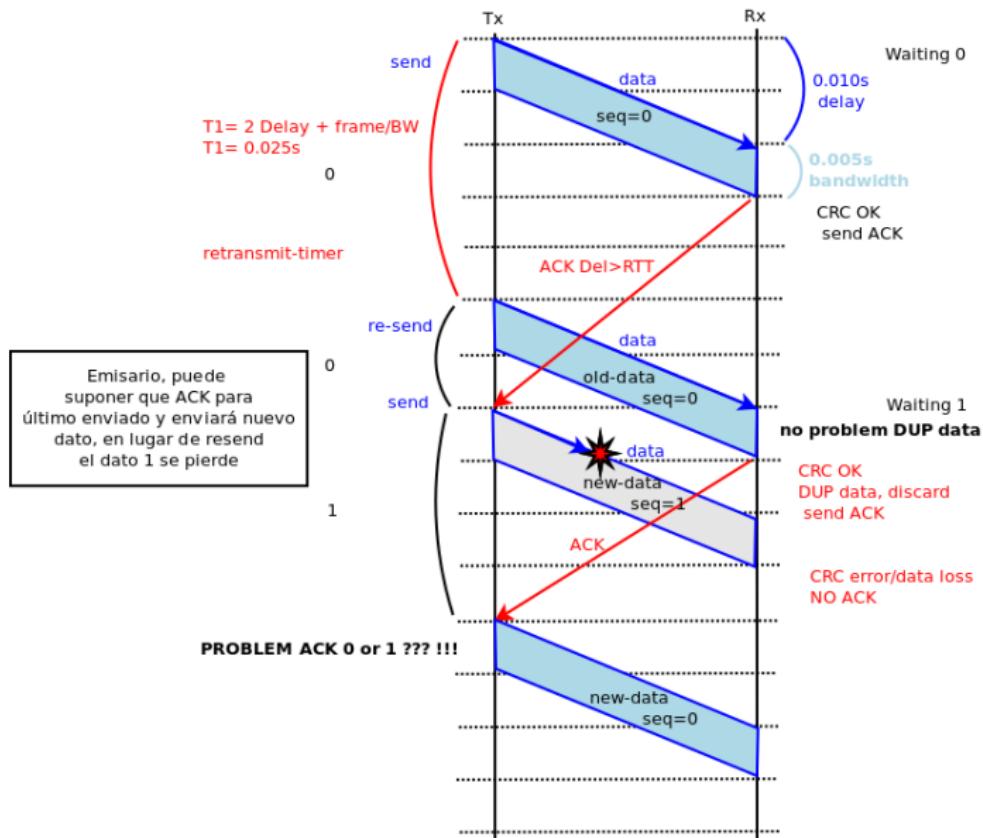


```
void receiver2.4()
{
...
O1:   // Receptor chequea datos
      // y confirma con ACK, incluso DUP
      // pero !corrupt, sino espera tmout
      if (!corrupt(pkt))
      {
          answer = make_pkt(ACK);
          calc_chksum(answer);
          udt_send(answer);
          if (pkt->id == wait_id) ok = 1;
          else ok = 0; // (pkt->id != wait_id)
      }
      else
      {
          // discard and wait retrans
      }
...
}
```

# Problemas de S&W + Bit Counter, v2.4

- No se considera que los Delayed ACK con valores mayores a 1 RTT.
- Los datos siempre van ordenados.
- No hay problema, de datos repetido porque tienen nro. de secuencia.
- Si hay problema de confundir un ACK con otro, ya que no tienen nro. de secuencia.
- El transmisor cree que siempre se confirma el corriente, pero puede suceder:
  - ① Ocurre un timeout, re-envía e inmediatamente le llega el ACK retardado.
  - ② Al recibir este envía segmento nuevo.
  - ③ Segmento nuevo se pierde.
  - ④ Recibe el ACK duplicado del segmento que retransmitió.
  - ⑤ Cree que es del nuevo.
- **ERROR!!!!** con  $ACKDelay > 1RTT$ .

# Problema Stop & Wait 2.4 (No ACK ID)



# Canal con Errores y $1 \text{ RTT} < \text{ACK Delay}$

- Sí se pierden datos, y Sí se pierden los ACK/NAK.
- No se duplican.
- No se desordenan.
- Sí se corrompen datos, se chequean con Checksum/CRC o mecanismo similar, se avisa cuando no se reciben de forma correcta.
- Sí se corrompen las confirmaciones, ACK/NAK.
- Sí  $2\text{RTT} < \text{MAXDelay} < 3\text{RTT}$ , con 2 nros. de secuencia alcanza 0..1.

# Estructura de Datos para Algoritmo S&W v3.0

```
typedef struct pk {
    // Header
    int      len;
    bool     ack;      // 1,ACK | 0,DAT
    int      id;       // seq 0,1
    int      ack_id: // seq 0,1 Admite piggy-backing
    ck_t     checksum;
    // Data
    byte_t   payload [...];
} packet_t;
```

# Algoritmo: Stop&Wait + Bit Counter Data/ACK

- Los ACK también necesitan nro. de secuencia como primer algoritmo Free NAK: v2.2.
- Funciona con  $ACKDelay > 2RTT$ , pero ...
- Deja de funcionar en casos con  $ACKDelay > 3RTT$ .

# Algoritmo: S&W + Bit Counter Data/ACK: Sender

```

void sender3()
{
    packet_t *pkt;
    byte_t *data;
    result_t result = OK;
    packet_t *answer = NULL;
    int counter = 1;

    // Mientras no se termine la comunicación
    while (result != EOT)
    {
        O0: // Emisor recibe datos de capa
            // superior y arma PDU en data:
            result = rdt_send(data);

        O1: // Arma paquete
            counter = (counter + 1) % 2;
            pkt = make_packet(counter, data)
            calc_chksum(pkt);
    ...
}

```

```

    ...
    // TMOUT | implicit NAK | ERR
    while ( (corrupt(answer)) ||  

        (isACK(answer) &&  

        (answer->ack_id != counter)) )
    {

        O2: // Antes de enviar genera timer,
            // si había uno corriendo, reset
            start_timer(RTT+delta);

            // Emisor envía datos:
            udt_send(pkt);

        O3: // Emisor espera respuesta
            // answer ::= ACK|TMOUT|ERR
            udt_recv(answer);

        } // Emisor se queda
            // retransmitiendo el mismo dato

        O4: // Se detiene timer
            stop_timer();
            // Emisor vuelve a 0
            // goto O0 — esta en el while
        }
    }
}

```

# Algoritmo: S&W + Bit Counter Data/ACK: Receiver

```

void receiver3()
{
    packet_t *pkt;
    packet_t *ack;
    byte_t *data;
    result_t result = OK;
    int ok = 0;
    int wait_id = 0;

    // Mientras no se termine la comunicación
    while (result != EOT)
    {
        ok = 0;
        while (!ok)
        {
O0:       // Receptor recibe datos
            udt_recv(pkt);
O1:       // Receptor chequea datos
            // y confirma con ACK,
            // sino espera tmout
            if (!corrupt(pkt))
            {
                // Arma ACK
                // ack->ack      = 1; //true
                // ack->ack_id   = pkt->id;
                ack = make_pkt(pkt->id, ACK);
            }
        ...
    }
}

```

...

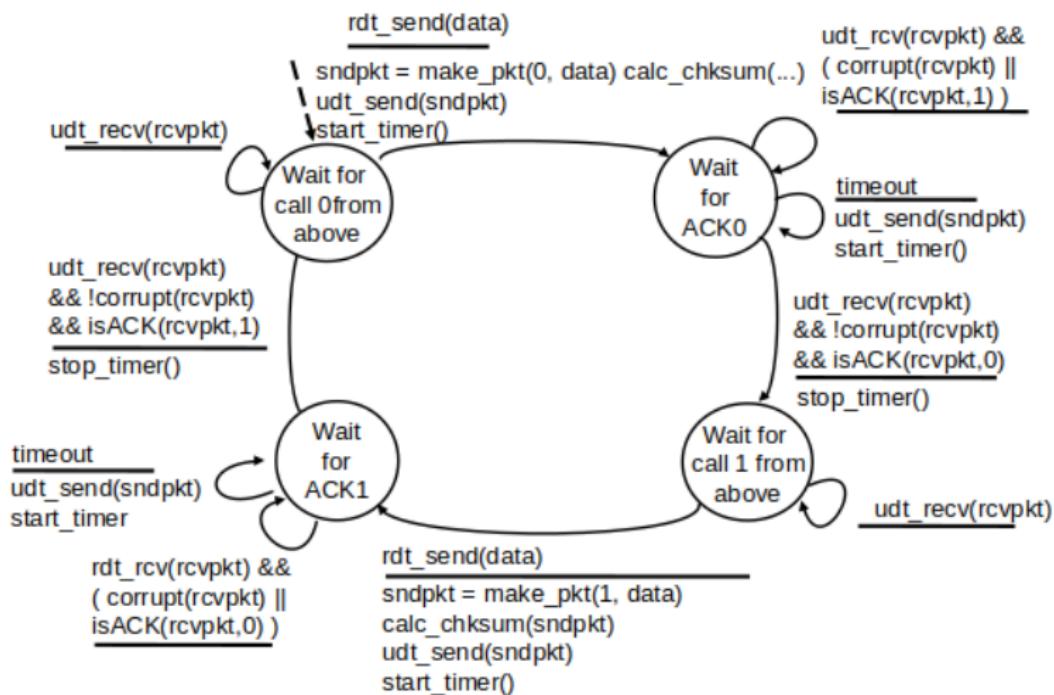
```

    if (pkt->id == wait_id)
    {
        ok = 1;
        wait_seq = (wait_seq+1)%2;
    } // else discard
} // else discard , ERR data , ok=0
} // while (!ok)

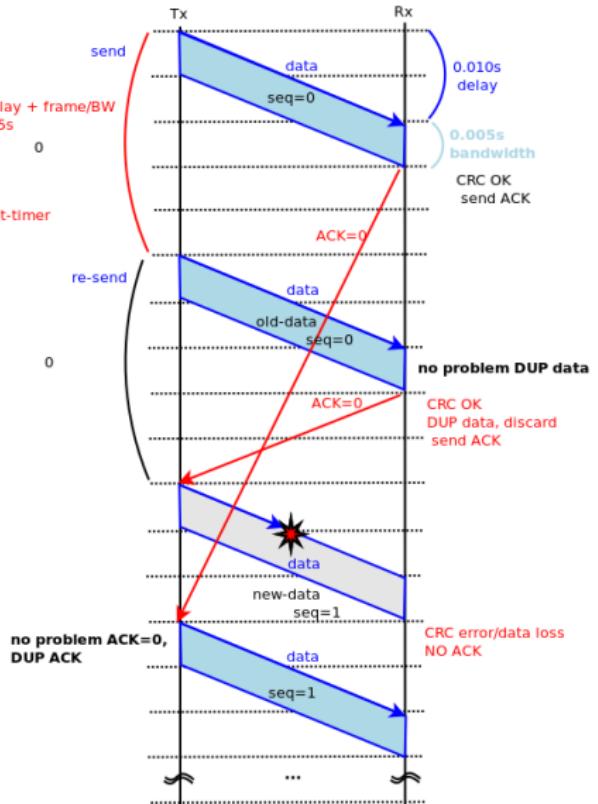
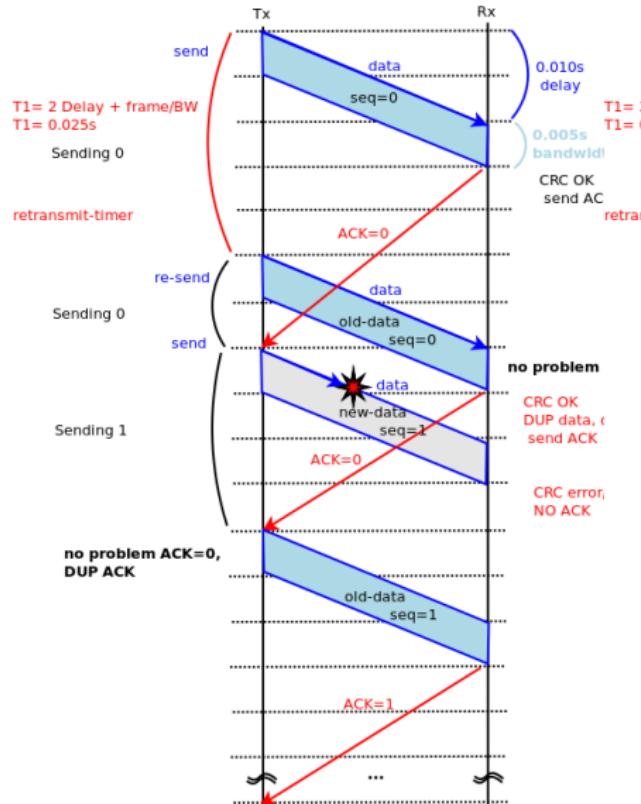
```

O2:     // Receptor entrega datos  
       // a capa superior  
       // data=pkt->payload  
       data = extract(pkt);  
       result = deliver\_data(data);  
O3:     // Receptor vuelve a 0  
       // goto 0 — esta en el while
} // (result != EOT)

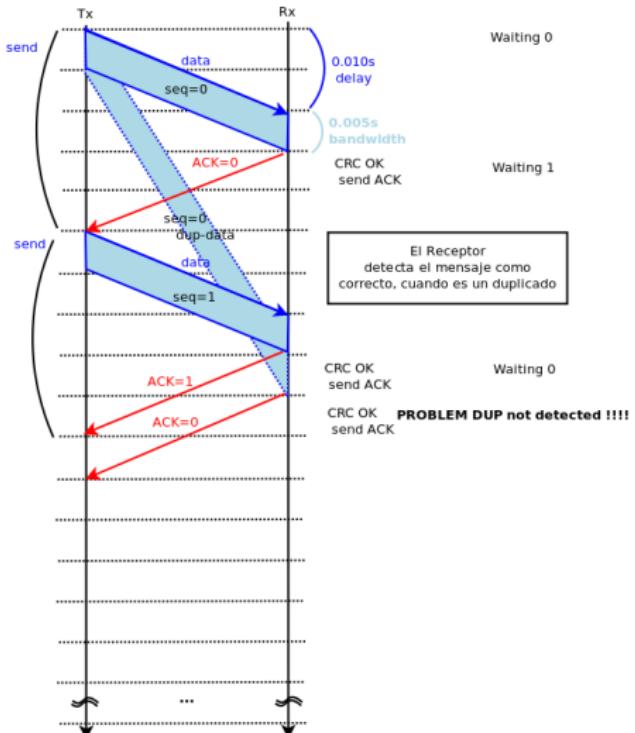
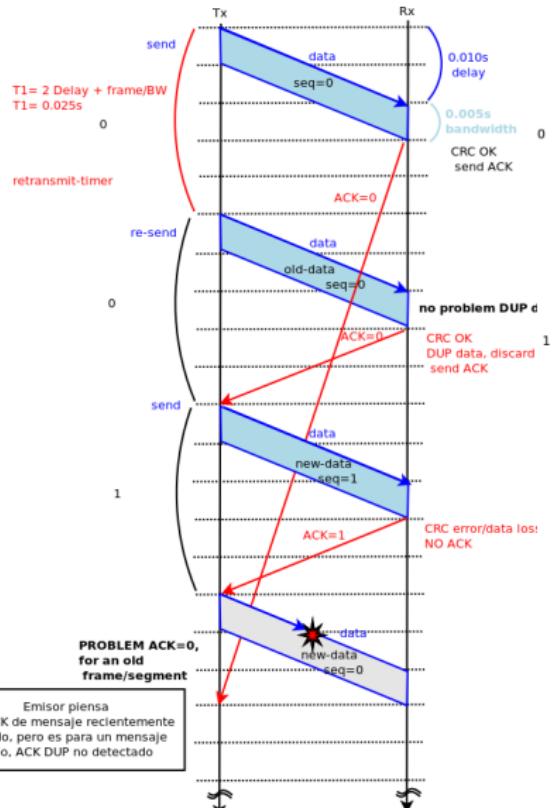
# Algoritmo S&W v3 - FSM - Send



# S&W v3 , ACKDelay > 1RTT, ACKDelay > 2RTT



# Problemas S&W v3 , ACKDelay > 3RTT, DUPmsg



# Canal con Mensajes fuera de Orden y DUPs

- Sí se pierden datos, y Sí se pierden los ACK/NAK.
- Sí se duplican.
- Sí se desordenan.
- Sí se corrompen datos, se chequean con Checksum/CRC o mecanismo similar, se avisa cuando no se reciben de forma correcta.
- Sí se corrompen las confirmaciones, ACK/NAK.
- Sí hay delay mayores:  $ACKDelay > N \times RTT$ .

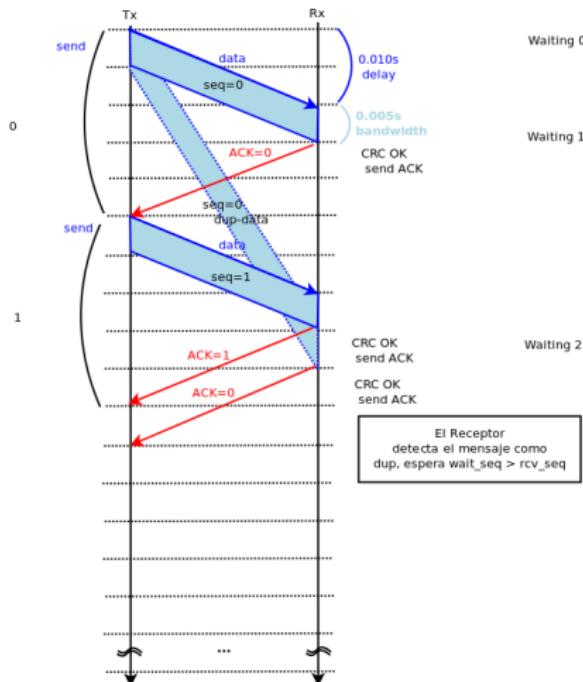
# S&W + Seq Num Data/ACK > 1

- Si se considera que los datos pueden llegar desordenados.
- Además se considera los duplicados.
- Se debe aumentar los nros. de secuencia de ráfagas a valores mayores a los Delays de acuerdo a los RTT.
- Si se considera que los Delayed ACK, con valores mayores a varios RTT.
  - $2RTT < MAXDelay < 3RTT$ , con 2 nros. de secuencia alcanza 0..1.
  - $N \times RTT < MAXDelay < (N + 1) \times RTT$ , entonces con  $N$  nros. alcanza: 0.. $N - 1$ .

# Estructura de Datos para Algoritmo S&W mejorado

```
typedef struct pk {
    // Header
    int      len;
    bool     ack;      // 1,ACK | 0,DAT
    int      id;       // seq 0,1,...N-1 (mod N)
    int      ack_id;   // seq 0,1,...N-1 piggy-backing
    ck_t     checksum;
    // Data
    byte_t   payload [...];
} packet_t;
```

# S&W v4, Nros. de secuencias mayores



# Análisis de Rendimiento

$$S = \text{MaxSgmt}_{bytes} = 1500B$$

$$RTT = \text{Latencia}_{seg} = 0.020s = 0.010s + 0.010s$$

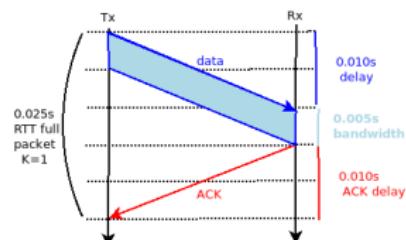
$$L = \text{DelayTransf}_{seg} = 0.005s$$

$$R = BW_{bps} = \frac{S \times 8bits}{L} =$$

$$\frac{1500 \times 8}{0.005} = 2400000 bps = 2.4 Mbps$$

$$U = Utiliz = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} =$$

$$0.005 / (0.020 + 0.005) = 0.2(20\%)$$



Se obtiene:  $2.4 Mbps \times 0.2 = 0.4 Mbps = 400 Kbps$

# Análisis de Rendimiento (Cont.)

- Si  $RTT$  aumenta y  $BW$  aumenta se hace peor.
- Por ejemplo enlace de 1 Gbps y 50ms de latencia ida y vuelta  
 $BDP = 1\text{Gbps} \times 50\text{ms}$ .

$$L = 1\text{Gbps}$$

$$RTT = 0.050\text{s}$$

$$\frac{L}{R} = \frac{1500 \times 8}{(1 \times 1000^3)} = 0.0000120\text{s}$$

$$U = \frac{0.0000120}{(0.050 + 0.0000120)} = 0.00024(0.0024\%)$$

Se obtiene:  $1\text{Gbps} \times 0.00024 = 240\text{Kbps}$

# Conclusiones

- S&W base soluciona pérdida de datos, con restricciones casi ideales sobre la red.
- S&W no soluciona ACK perdidos o dañados, ni ACK delay.
- S&W+bit counter en datos soluciona pérdida de datos y de ACK.
- S&W+bit counter en datos no soluciona  $ACKDelay > 1RTT$ .
- S&W+bit counter datos/ACK soluciona los problemas anteriores.

# Conclusiones (Cont.)

- S&W+bit counter datos/ACK no soluciona:
  - Datos/ACK delayed múltiples RTT ( $> 3RTT$ ).
  - Datos fuera de orden.
  - Duplicados (DUPs).
- S&W+nros de secuencias 0.. $N$  datos/ACK soluciona la mayoría de los problemas en la red.
- Otros problemas:
  - Sequence Number Wrap Around PAWS (puede usarse time-stamping)
  - El sistema es ineficiente, envía un dato por vez, ventana de transmisión/recepción:  $K = 1$ ,  $W = 1$ .

## Referencias

[K-R] Computer Networking International Edition, 6e. James F. Kurose & Keith W. Ross. ISBN: 9780273768968.

# Transporte: Ventana Deslizante

2019

# Contenidos

1

Control de Errores S&W

2

Pipelining/Sliding Window

- Go-Back N
- Selective Repeat
- Control de Errores de TCP

3

Referencias

# Contenidos

1

Control de Errores S&W

2

Pipelining/Sliding Window

- Go-Back N
- Selective Repeat
- Control de Errores de TCP

3

Referencias

# Contenidos

1

Control de Errores S&W

2

Pipelining/Sliding Window

- Go-Back N
- Selective Repeat
- Control de Errores de TCP

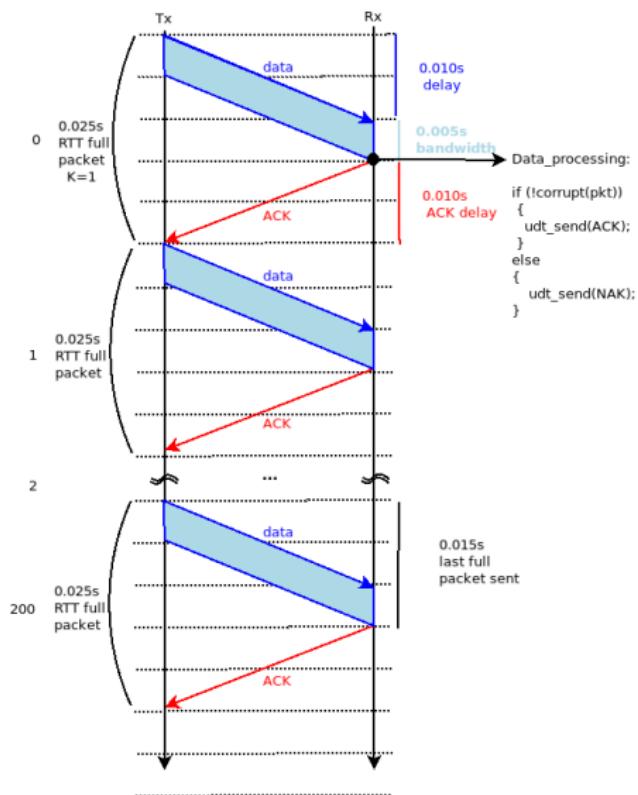
3

Referencias

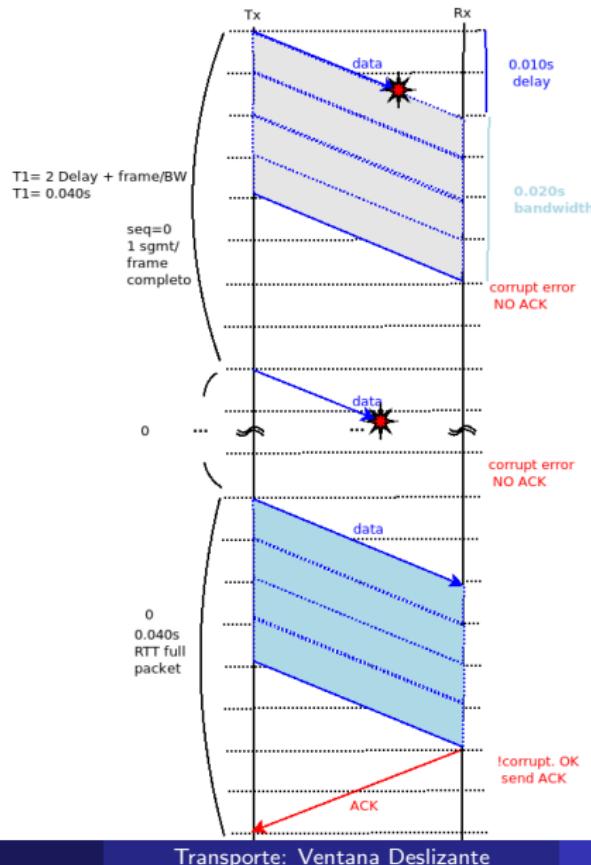
# Control de Errores S&W

- El sistema S&W es ineficiente, envía un dato por vez, ventana de transmisión/recepción:  $K = 1$  (también llamada  $W$ ).
- No se envía el próximo mensaje hasta que no se confirma el que se envió.
- Sistema simple y poco eficiente: no optimiza producto: Delay, Bandwidth:  $BDP = D \times B$ ,  $D = RTT$  o  $D = RTT + L$ .
- Cada vez que envía un segmento requiere arrancar un timer:  $RTO$  o  $T1$ .
- Si no recibe confirmación se vence el timer y retransmite.
- Se optimizaría enviando la mayor cantidad de datos posibles en un segmento/frame, pero se torna aún más deficiente si hay muchas retransmisiones. Limitación del tamaño del segmento/frame con respecto al código de detección de errores.

# Stop & Wait



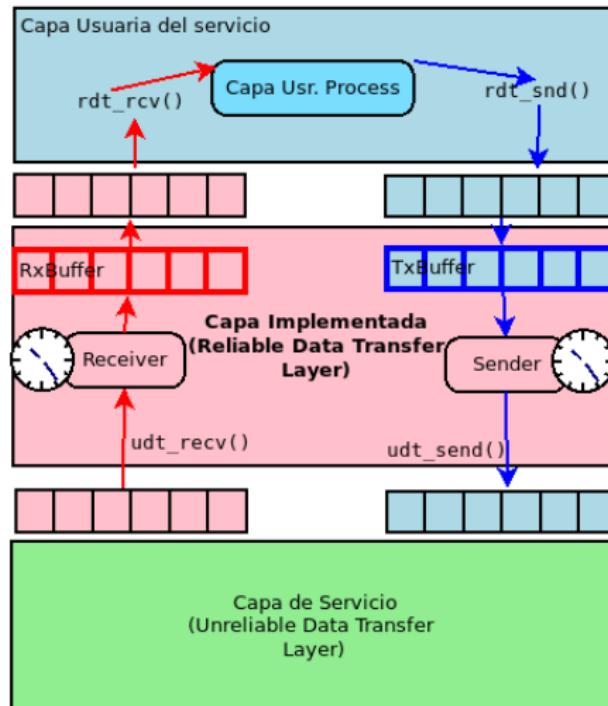
# Stop & Wait - Big Frame/Segmento



# Pipelining/Sliding Window

- Pipelining: permitir enviar múltiples segmentos/paquetes sin aún haber recibido confirmaciones, paquetes “in-flight”.
- La cantidad de segmentos que se puede enviar sin aún recibir confirmación se llama **Ventana**, notado como  $K$  o  $W$ ,  $K = n$ , donde  $n > 1$ .
- Requiere ampliar los números de segmentos y de las confirmaciones además del buffering entre capa usuaria-RDT en ambos procesos: Sender, Receiver.
- Por cada mensaje enviado se inicia un timer de retransmisión:  $T1$  o  $RTO$  (ver más adelante la simplificación).
- Por cada confirmación se descarta/re-inicia el timer  $RTO$ . Si no se recibe confirmación vence  $RTO$  (timeout) y se retransmite, nuevo timer.

# RxBUFFER, TxBUFFER, Timers en Pipelining

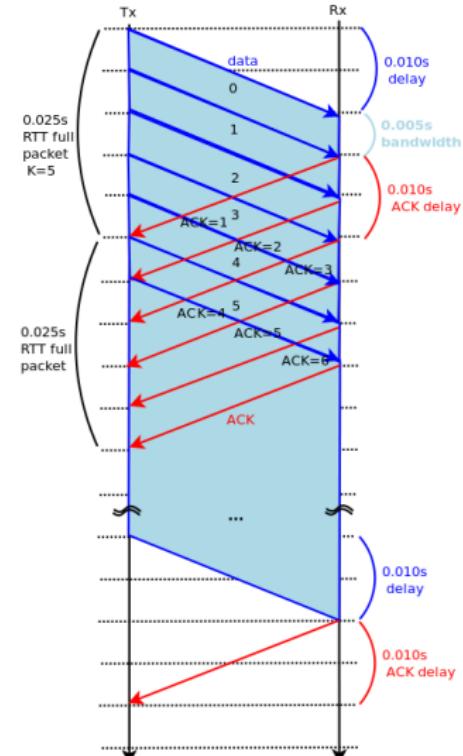
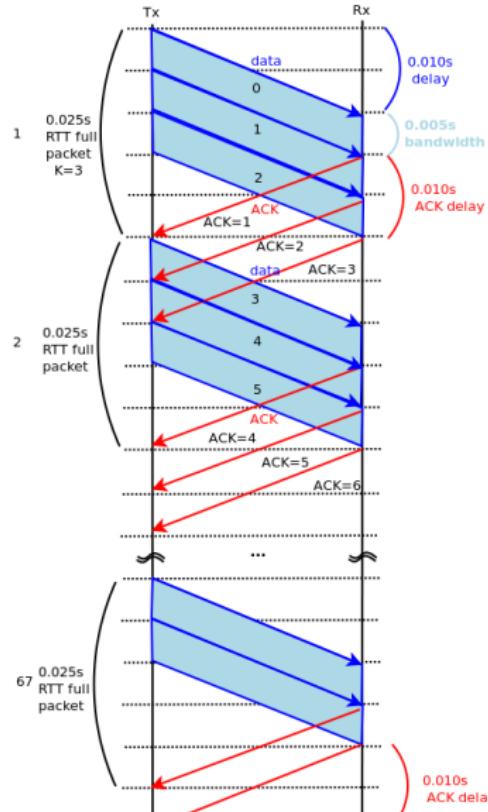


# Pipelining/Sliding Window (Cont.)

- Podría generarse Confirmaciones Negativas: NAK (NO Acknowledge), implícitas o explícitas.
- Algoritmo más eficiente, óptimo, si llena el pipe.
- Los números de secuencia,  $0..M - 1$ , si se numeran en módulo, no necesariamente son  $M = K$ ,  $K < M$ .
- Recordar que aumentar  $M$  permite ser tolerante a fallas de ACK delayed.
- Alternativas:
  - Go-back-N.
  - Selective-Repeat.

**Nota:** Los ejemplos se muestran con confirmaciones por segmento, no por byte e indicando el que se espera

# Pipelining



# Análisis de Rendimiento

$$K = Window = 3, S = MaxSgmt_{bytes} = 1500B$$

$$RTT = Latencia_{seg} = 0.020s = 0.010s + 0.010s$$

$$L = DelayTransf_{seg} = 0.005s$$

$$R = \frac{1500 \times 8}{0.005} = 2400000 bps = 2.4 Mbps$$

$$U = \frac{K \times \frac{L}{R}}{RTT + \frac{L}{R}} =$$

$$(3 \times 0.005) / (0.020 + 0.005) = 0.6(60\%)$$

- Se obtiene:  $2.4 Mbps \times 0.6 = 1.44 Mbps$
- Si  $K = 5$  se obtiene el 100%, si el tráfico es constante.

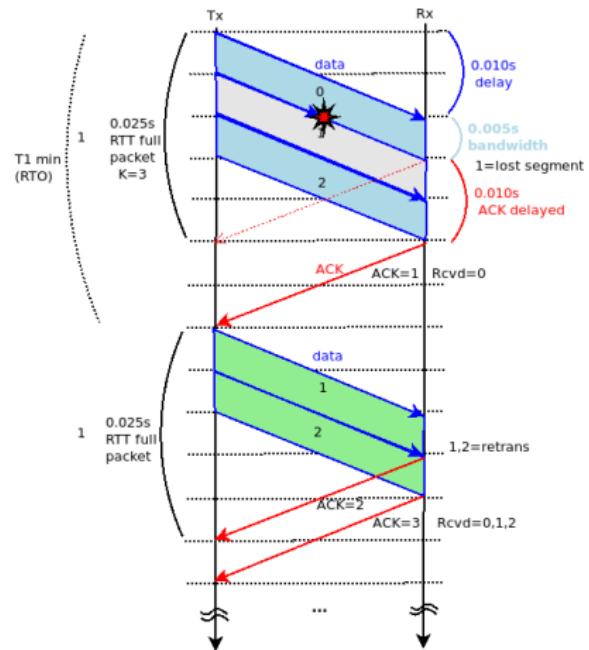
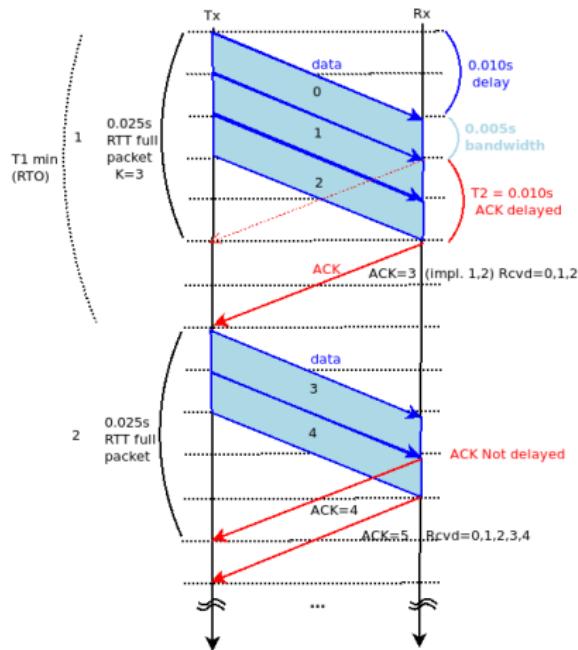
## Go-back N

- Se tiene una ventana "estática" de tamaño  $K = n, n > 1$ . Numeración de segmentos, se realiza en módulo  $M$ ,  $K \leq (M - 1)$ .
- No admite segmentos fuera de orden, ni confirmaciones fuera de orden. Solo se confirman por la positiva los segmentos que se pudieron colocar en el buffer en orden.
- Se puede confirmar desde  $N$  hacia atrás (ACK acumulativos). No necesariamente se confirma cada segmento individualmente.
- Se puede re-transmitir ante un timeout o un NAK. Requiere buffering extra en el emisor, no se pueden descartar del buffer de com. con la capa usuaria.
- Si se re-transmite ante un timeout se hace desde  $N$  hacia adelante, los que ya se enviaron.
- Emisor puede mantener solo un timer para el segmento más viejo enviado y aún no confirmado, si este expira retransmite todos los no confirmados.
- Si llega una confirmación en orden arranca un nuevo timer.

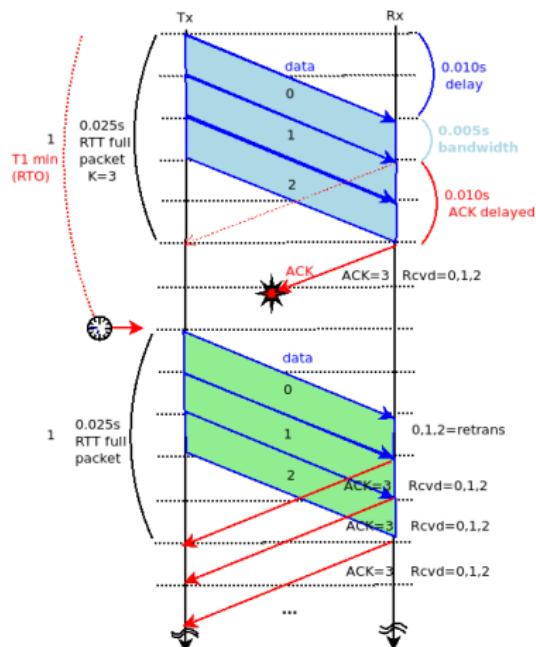
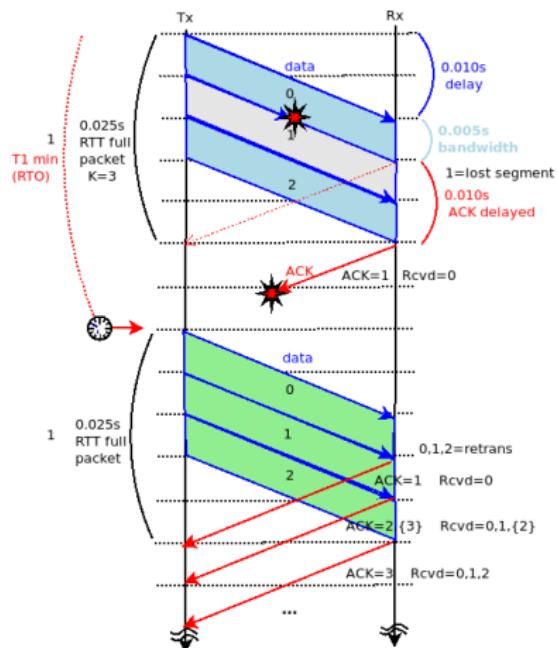
## Go-back N (Cont.)

- Si llega un segmento de datos en orden se puede/debe confirmar por la positiva indicando el próximo.
- Si llega un segmento fuera de orden o esta corrupto se puede confirmar por la negativa indicando que se espera el próximo al último recibido de forma adecuada.
- Ante el error se puede esperar la retransmisión.
- Se pueden aprovechar tramas de datos para confirmar: Piggy-backing.
- El receptor puede usar: timer de ACK,  $T_2$ , para confirmar.  $T_1 > T_2$ ,  $T_1 > T_2 + RTT$  (Delayed ACK).
- $T_2$  debe aprovechar piggy-backing, y confirmaciones acumulativas pero sin demorar demasiado tiempo el flujo de datos.
- Si se pierde uno segmento es como “vaciar” el pipe y volver atrás: Go-Back.

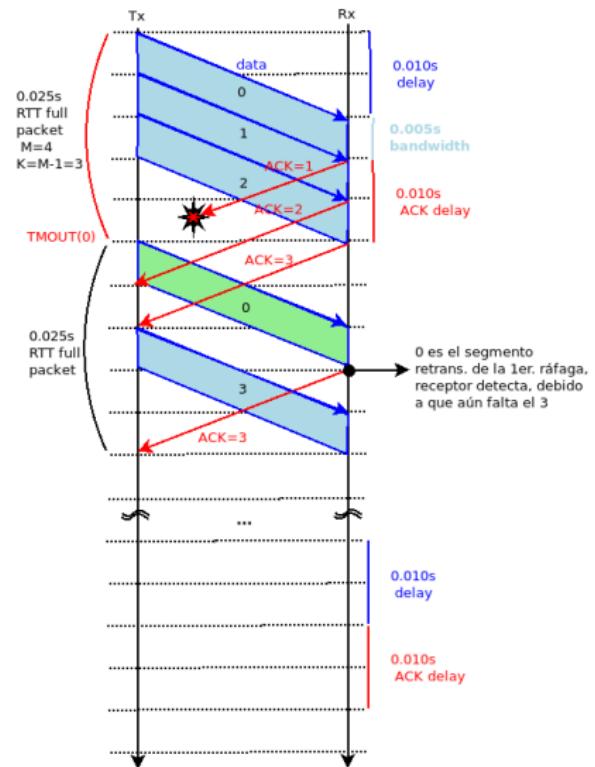
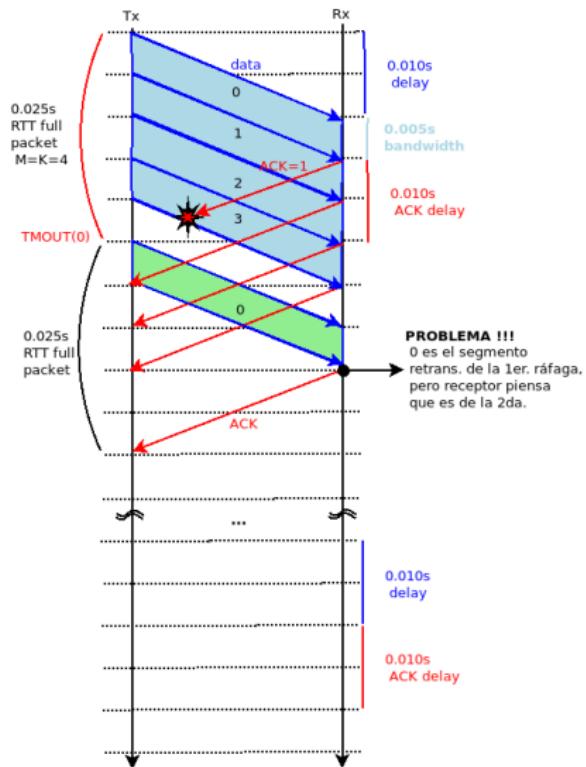
# Go-back N (ejemplo)



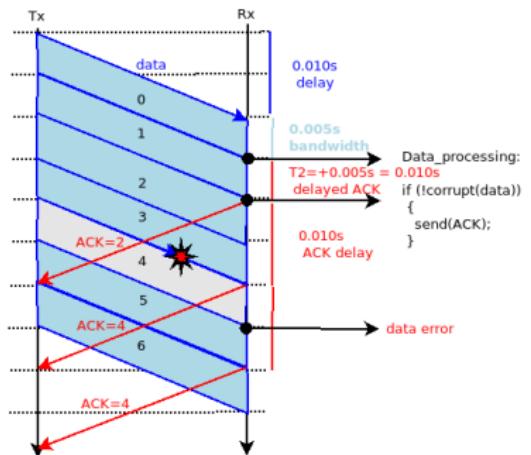
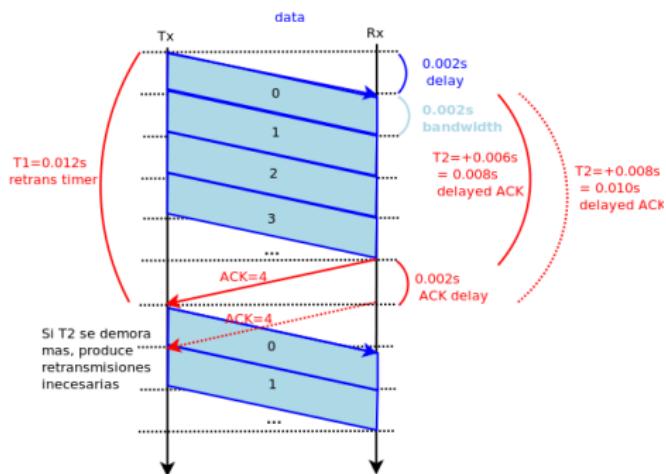
# Go-back N (timer T1 -RTO-)



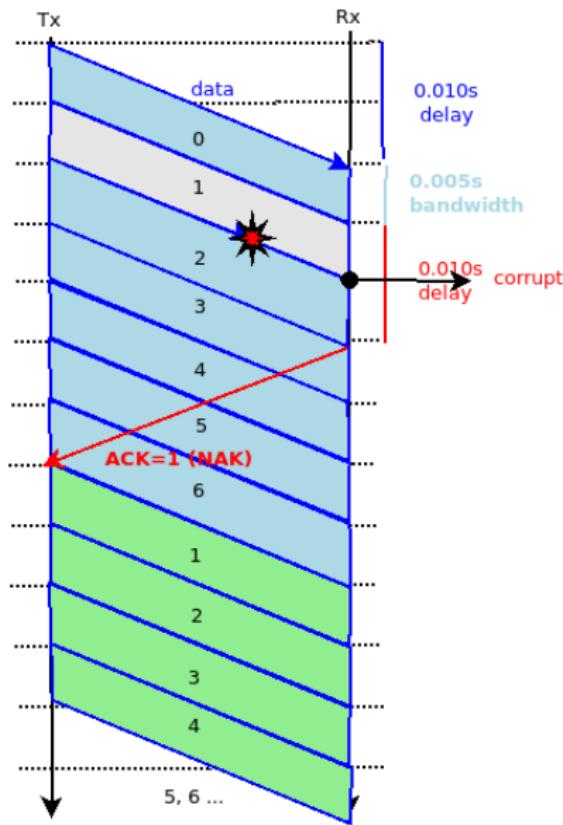
Relación K,M: Ej. M=K=4 ; M=4,K=(M-1)=(4-1)



# Go-Back N (timers T1 y T2)



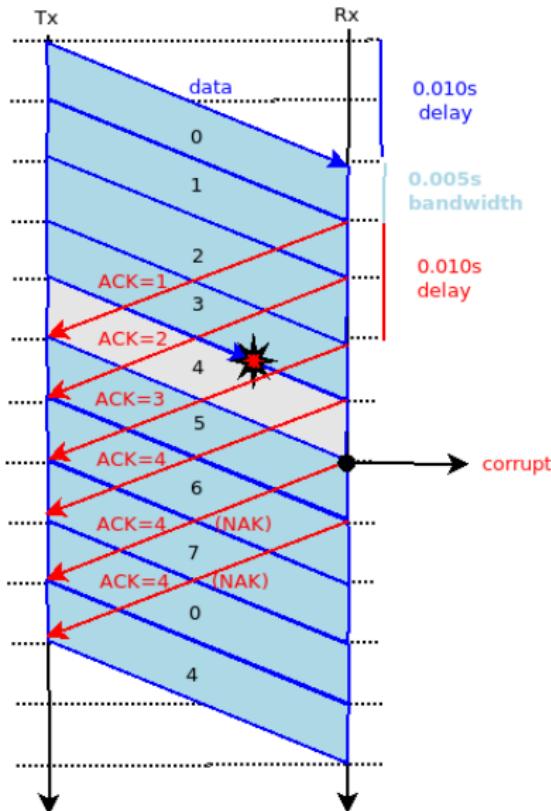
# Go-Back N, Segmento Corrupto



# Go-back N (Cont.)

- Las confirmaciones por la negativa pueden generar ACK duplicados (NAK).
- Mensajes fuera de orden:
  - Buffering hasta recibir los que llenan los huecos. Hasta cuando se mantienen? (requiere buffering de Rx antes de pasarlos a la capa usuaria).
  - Descartarlos y esperar retransmisiones (no requiere buffering del receptor, solo recordar la secuencia que se espera).

# Go-Back N, ACK duplicados, NAK implícitos

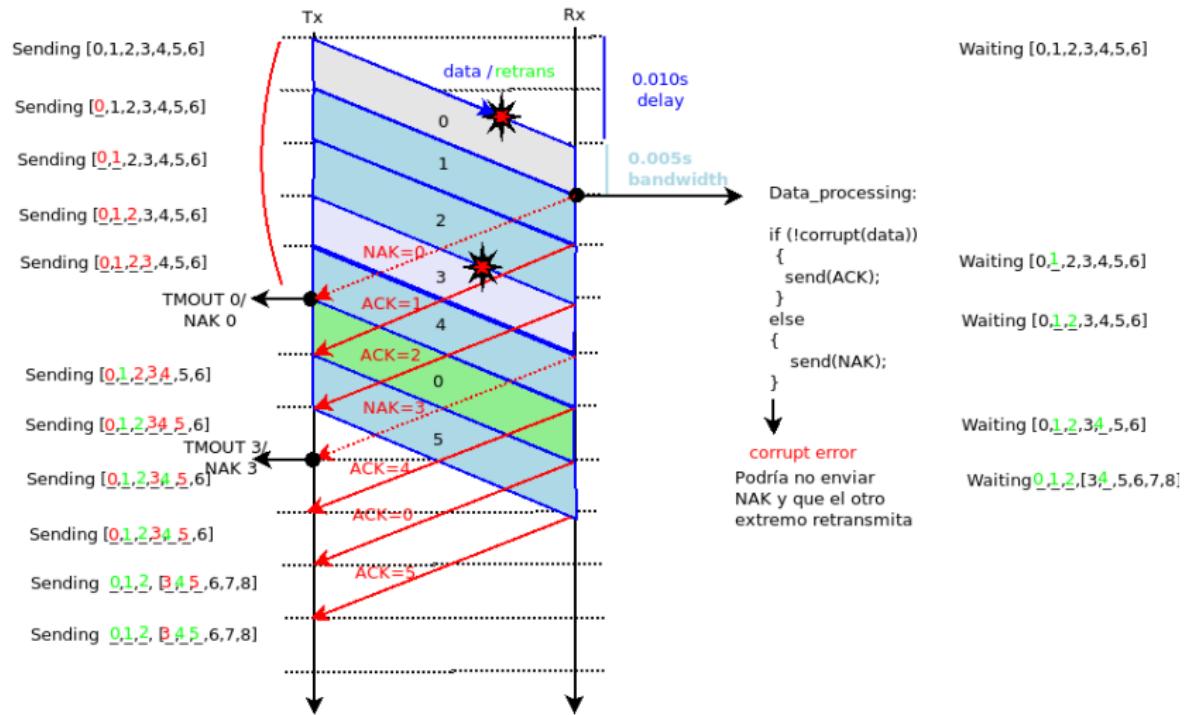


# Selective Repeat (SR)

- Go-Back-N ante pérdidas retransmite segmentos innecesarios si se perdió uno del medio del stream de datos solamente y hubo timeout.
- Selective Sliding Window/Selective Repeat solo retransmite los que no se confirmaron.
- El receptor puede confirmar de a uno o usar bit vectors/intervalos de confirmaciones.
- No se puede usar confirmaciones acumulativas.
- No se deben confundir los segmentos de diferentes ráfagas. No se deben reusar #ID/SEQ hasta asegurarse que tiene todos los mensajes previos o estos no están en la red.
- Se realiza en módulo  $M$ ,  $K \leq \frac{(M-1)}{2}$ , para evitar confundir los ACK de segmentos.
- La ventana se desliza sin dejar huecos, desde los confirmados más viejos.

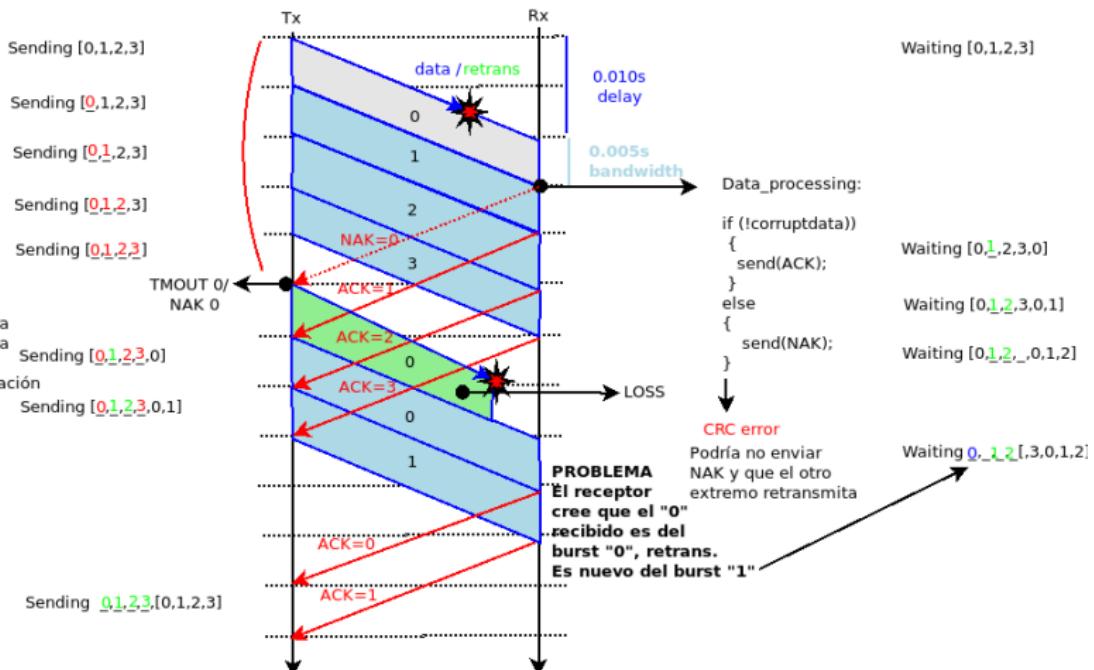


## Selective Repeat Ejemplo

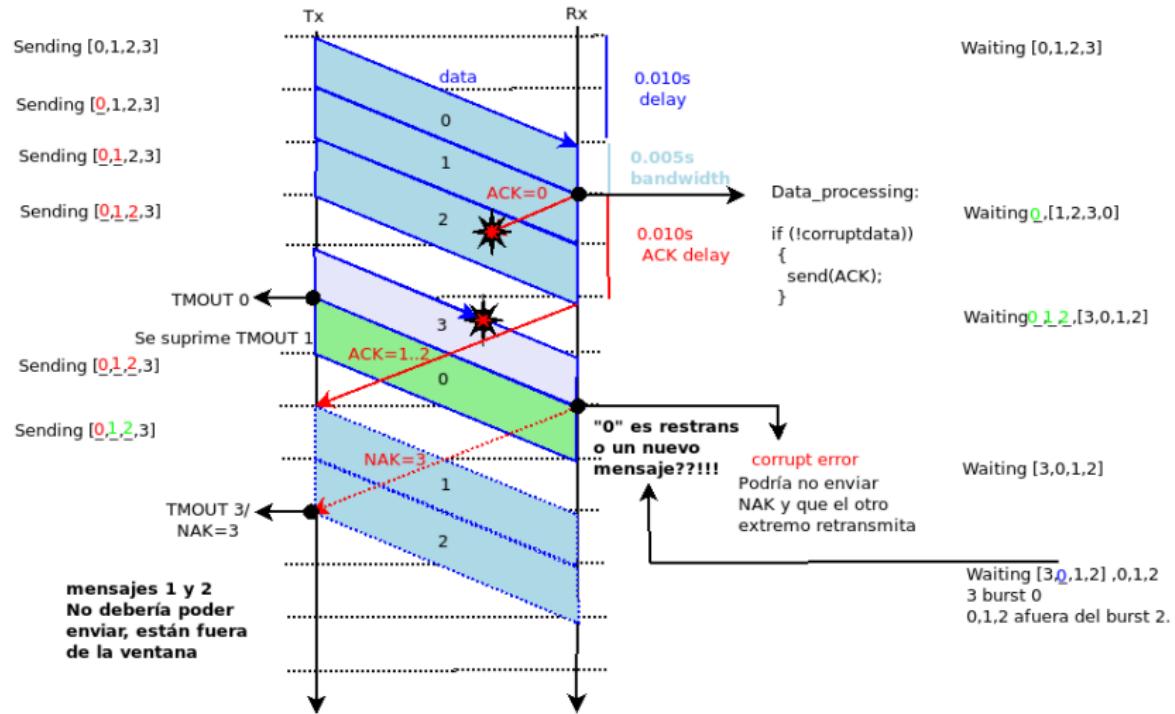


# Selective Repeat, Problemas, K y M

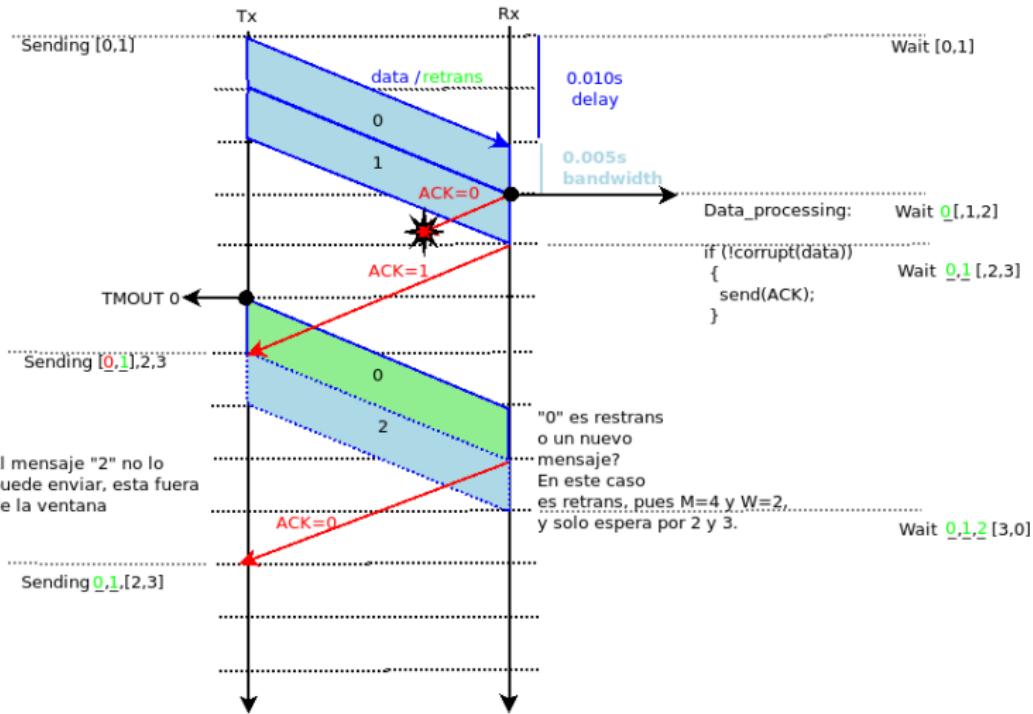
El problema se genera al moverse la ventana teniendo pendientes mensajes de confirmación viejos. Se hace lugar de forma incorrecta



# Selective Repeat, Problemas, K y M



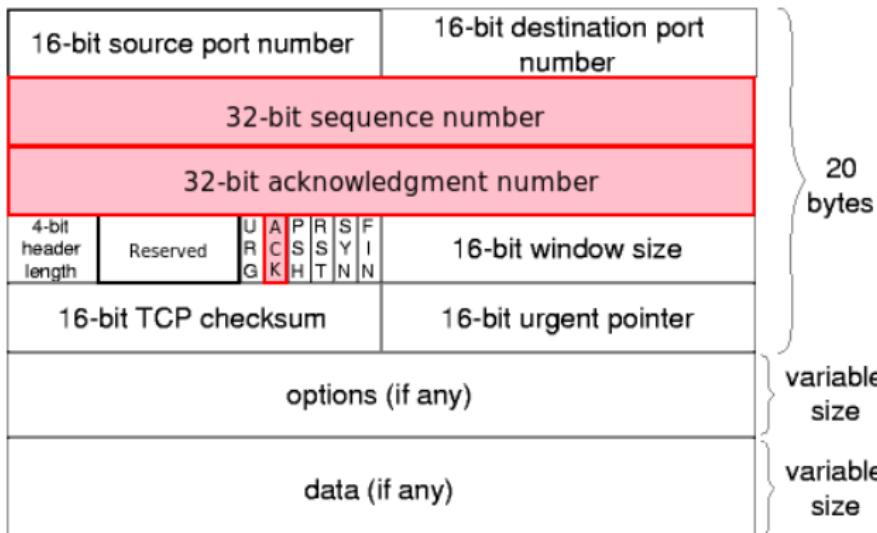
# Selective Repeat, $K$ adecuado



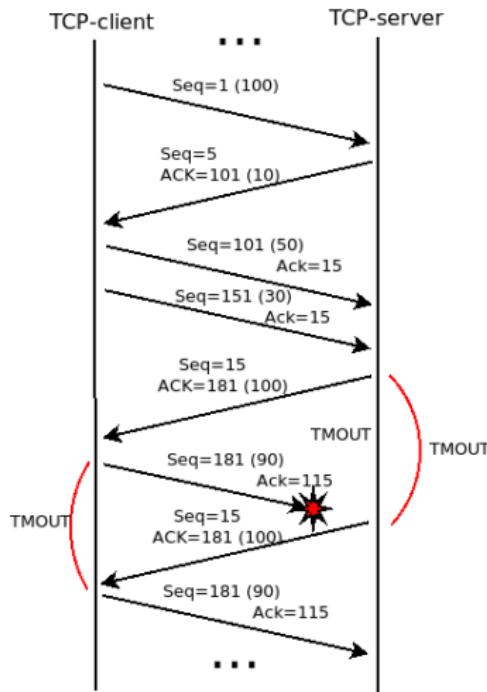
# Control de Errores en TCP

- TCP hace el control de errores por bytes (byte oriented), no por segmentos.
- Los segmentos se numeran de acuerdo a bytes enviados (nro. del primer byte).
- Los números se negocian al establecer la sesión y cada implementación los elige libremente (ISN).
- Las confirmaciones son “anticipativas”, indican el nro. de byte que esperan.
- Utiliza Go-back-N con ventana dinámica (flow-control), utiliza piggy-backing y permite negociar Ventana Selectiva con Opciones.
- Para control de errores TCP utiliza los campos: #SEQ, #ACK, flag ACK más timer y algunas opciones.

# Segmento TCP, Control de Errores



# Ejemplo de Control de Errores de TCP



# Otro Ejemplo de Control de Errores de TCP

Time	172.20.1.1 172.20.1.100	Comment
0.000	(41749)  → (11111)	Seq = 0
0.001	(41749) ←  (11111)	Seq = 0 Ack = 1
0.001	(41749)  → (11111)	Seq = 1 Ack = 1
90.730	(41749)  → (11111)	Seq = 1 Ack = 1
90.730	(41749)  → (11111)	Seq = 1 Ack = 6
100.150	(41749)  → (11111)	Seq = 1 Ack = 6
100.150	(41749) ←  (11111)	Seq = 6 Ack = 17
104.580	(41749)  → (11111)	Seq = 6 Ack = 17
104.580	(41749)  → (11111)	Seq = 17 Ack = 11
112.290	(41749)  → (11111)	Seq = 17 Ack = 11
112.290	(41749) ←  (11111)	Seq = 11 Ack = 23
114.890	(41749)  → (11111)	Seq = 23 Ack = 11
114.890	(41749) ←  (11111)	Seq = 11 Ack = 29
120.620	(41749)  → (11111)	Seq = 29 Ack = 11
120.620	(41749) ←  (11111)	Seq = 11 Ack = 30
120.620	(41749)  → (11111)	Seq = 30 Ack = 12

# Control de Errores en TCP (Cont.)

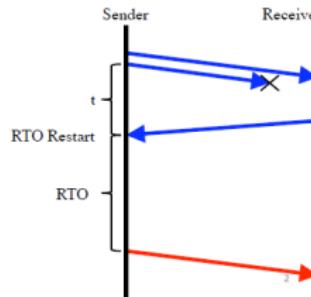
- Por cada segmento (con datos) que envía TCP es como si iniciara un Timer local,  $RTO$  y pone copia del segmento en cola local (RFC-793)  $TxBuf$ .
- Por cada segmento ACKed descarta el timer asociado y descarta la copia del segmento (RFC-793) del  $TxBuf$ . Hace lugar para nuevos segmentos a Tx.
- Si  $RTO$  expira antes que se confirme el segmento TCP lo copia del  $TxBuf$  y retransmite (RFC-793).
- Segmentos ACKed no indica leído por aplicación, sí recibido por TCP (RFC-793) (ubicado en el  $RxBuf$  del receptor).
- Si el receptor detecta error en el segmento simplemente descarta y espera que expire  $RTO$  en el emisor (podría enviar un NAK, re-enviar ACK para el último recibido en orden, forma de solicitar lo que falta).

# Control de Errores en TCP (Cont.)

- Receptor con segmentos fuera de orden descarta directamente y podrá re-enviar ACK (podría dejar en *RxBuf* pero no entregar a la aplicación, tiene huecos).
- Se puede confirmar con ACK acumulativos.
- TCP NO arranca un *RTO* por cada segmento, solo mantiene un por el más viejo enviado y no ACKed y arranca uno nuevo solo si no hay *RTO* activo.
- Si se confirman (ACKed) datos, se inicia un nuevo *RTO* (RFC-6298) recomendado. Si todo confirmado se detiene RTO.

# Control de Errores en TCP (Cont.)

- El nuevo  $RTO$  le esta dando más tiempo al segmento más viejo aún no confirmado. Mejor: RFC-7765:  $RTO_{new} = RTO - T_{earliest}$  (menos el tiempo que pasó del pendiente más viejo).

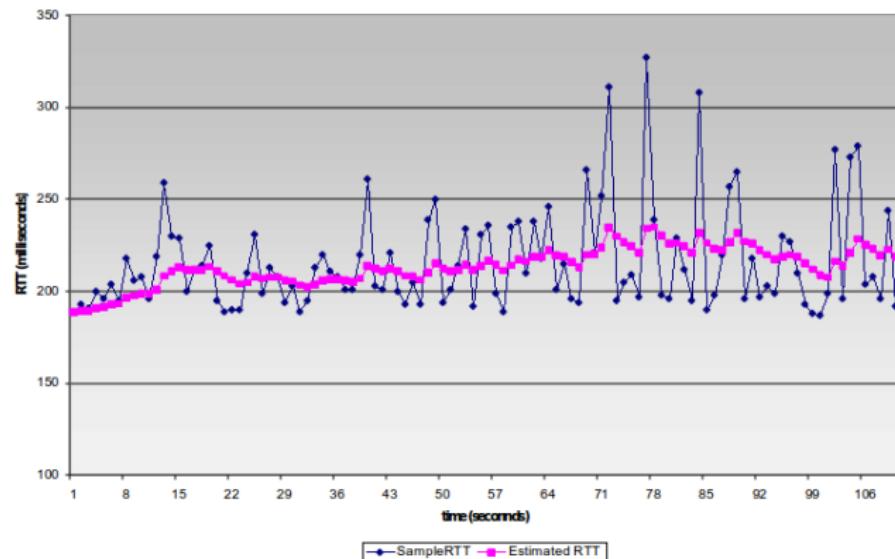


- Si vence un  $RTO$  se debe retransmitir el segmento más viejo no ACKed y se debe duplicar: Back-off timer  $RTO_{new} = RTO * 2$   $RTO_{MAX} = 60s$  (RFC-6298) recomendado.

# Cálculo de RTO

- $RTO$  debe ser dinámico, debe contemplar estado de la red.
- $RTO$  estático solo sirve para L2 (directamente conectados).
- Para calcular  $RTO$  se estima RTT (Round Trip Time). RTT inicial RFC-2988(2000), 3seg - RFC-6298(2011), 1 seg. Cambio en las redes.
- $RTO = SRTT + (4 * DevRTT)$  (RFC-6298).
- $SRTT_i = (1 - \alpha) * SRTT_{i-1} + \alpha * RTT, \alpha = 1/8$
- Influencia de las muestras pasadas decrece exponencialmente.
- $DevRTT_i = (1 - \beta) * DevRTT_{i-1} + \beta * |RTT - SRTT_i|, \beta = 1/4$
- Si hay gran variación en  $SRTT_i$  se usa un mayor margen.
- $RTO < 1\text{seg}$  : redondeado a 1 seg (RFC-6298).
- Se mide por cada RTT. Se puede utilizar la opción TimeStamp.

# SRTT



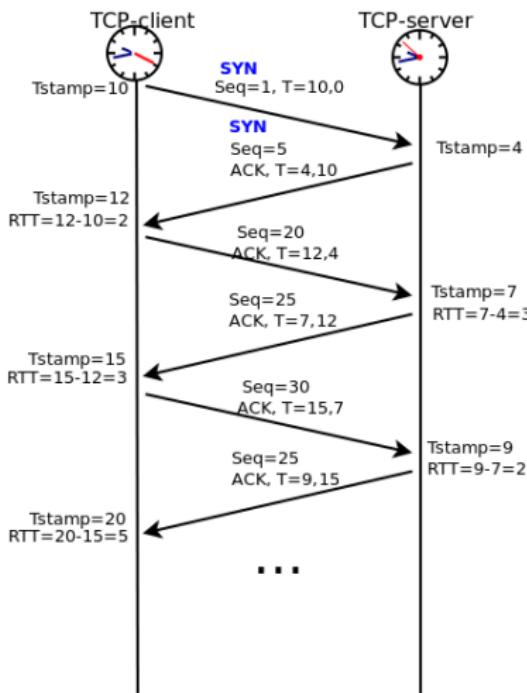
# Mínimo RTO

- $RTO < 1\text{seg}$  : redondeado a 1 seg (RFC-6298).
- Por qué tan conservador?  
(The TCP Minimum RTO Revisited, Ioannis Psaras and Vassilis Tsaoussidis)
  - Considerar sistemas con granularidad de 500ms.
  - Delayed ACK ( $T2 = \text{DelACK} = 200\text{ms}$ ).
- Sistemas reales ignoran esta recomendación:
  - Linux  $RTO \geq 200\text{ms}$ ,  $\text{DelACK} = \text{dyn.}$
  - Windows  $RTO \geq 300\text{ms}$ ,  $\text{DelACK} = 200$ .
  - BSD  $RTO \geq 30\text{ms}$ .
  - Solaris  $\text{DelACK} = 50..100\text{ms}$ .

# TimeStamp

- RFC-1323.
- Se envía en el primer SYN el timeStamp local, opcional.
- En cada mensaje TCP con esta opción, se copia el timeStamp local y se hace echo del último timeStamp recibido desde otro extremo.
- Con el valor recibido como echo y el valor del reloj local se calcula el RTT.
- Si el mensaje no es un ACK válido no se actualiza la estimación del RTT  $SRTT$ .
- Relaja la necesidad de usar timer por cada segmento para estimar RTT.
- Protección contra Wraparounds de num. secuencia (PAWS).

# Ejemplo de TimeStamping



## Referencias

[K-R] Computer Networking International Edition, 6e. James F. Kurose & Keith W. Ross. ISBN: 9780273768968.

# TCP (Transport Control Protocol) Flow Control

2019

# Contenidos

1

TCP

TCP Flow Control/Control de Flujo



# Servicios de TCP

## Control de Errores:

- Mecanismo protocolar, algoritmo, que permite ordenar los segmentos que llegan fuera de orden y recuperarse mediante solicitudes y/o retransmisiones de aquellos segmentos perdidos o con errores.
- Objetivo: recuperarse de los efectos del re-ordenamiento, la pérdida o la corrupción de los paquetes en la red.
- Se realiza por cada conexión: End-to-End, App-to-App.

# Servicios de TCP (Cont.)

## Control de Flujo (Flow-Control):

- Mecanismo protocolar, algoritmo, que permite al receptor controlar la tasa a la que le envía datos el transmisor.
- Control cuanto puede enviar una aplicación sabiendo que la receptora tiene capacidad de recibirla y procesarla.
- Objetivo: prevenir que el emisor sobrecargue al receptor con datos evitando un mal uso de la red.

# Control de Errores y de Flujo

- Para realizar control de errores y control flujo se utilizan técnicas de ARQ (Automatic Repeat reQuest), **Transferencia de Datos Fiables**.
- ARQ **solo** no hace control de flujo, requiere de otros mecanismos como RNR, o Dynamic Window (Ventana Dinámica).
- La capacidad de envío será  $\text{MIN}(\text{Congestion}, \text{Flujo}, \text{Errores})$ .

# Control de Errores TCP (Repaso)

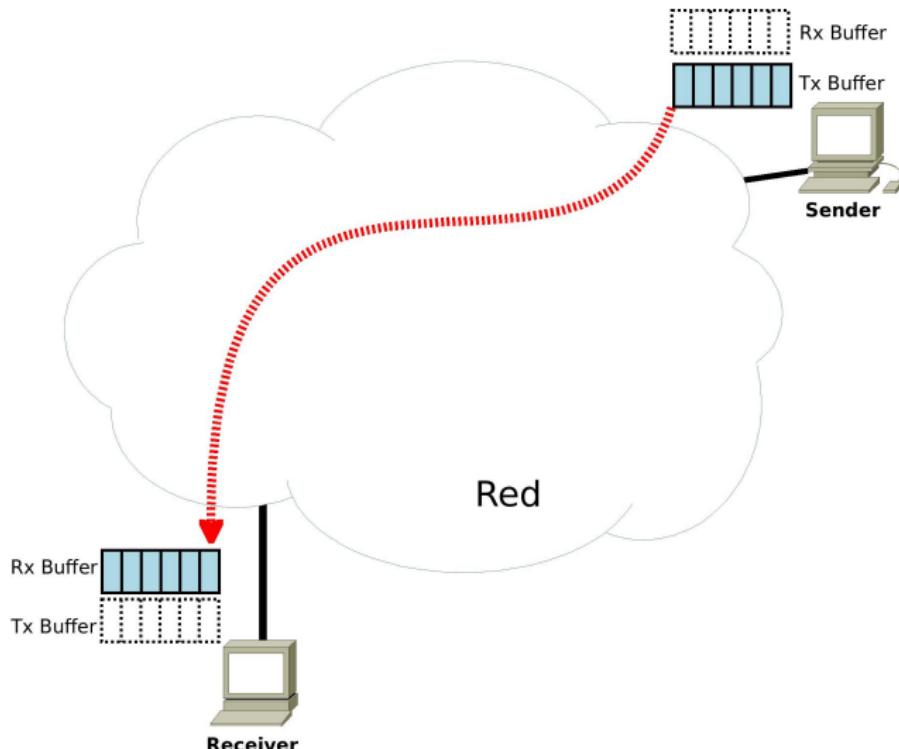
- Segmentos ACKed no indica leído por aplicación, sí recibido por TCP (RFC-793) (ubicado en el **Rx Buffer** del receptor).
- Si el receptor detecta error en el segmento simplemente descarta y espera que expire *RTO* en el emisor (podría enviar un NAK, re-enviar ACK para el último recibido en orden, forma de solicitar lo que falta).
- Receptor con segmentos fuera de orden descarta directamente y podrá re-enviar ACK (podría dejar en **Rx Buffer** pero no entregar a la aplicación, tiene huecos).
- Se puede confirmar con ACK acumulativos.

# Control de Errores TCP (Repasso)

- TCP NO arranca un *RTO* por cada segmento, solo mantiene un por el más viejo enviado y no ACKed y arranca uno nuevo solo si no hay *RTO* activo.
- Si se confirman (ACKed) datos, se inicia un nuevo *RTO* (RFC-6298) recomendado.
- El nuevo *RTO* le esta dando más tiempo al segmento más viejo aún no confirmado.
- Si vence un *RTO* se debe retransmitir el segmento más viejo no ACKed y se debe doblar: Back-off timer  $RTO = RTO * 2$   $RTO_{MAX} = 60s$  (RFC-6298) recomendado.
- TCP calcula el *RTO* de forma dinámica. RFC-6298(2011), ayudado por Timestamp Option.

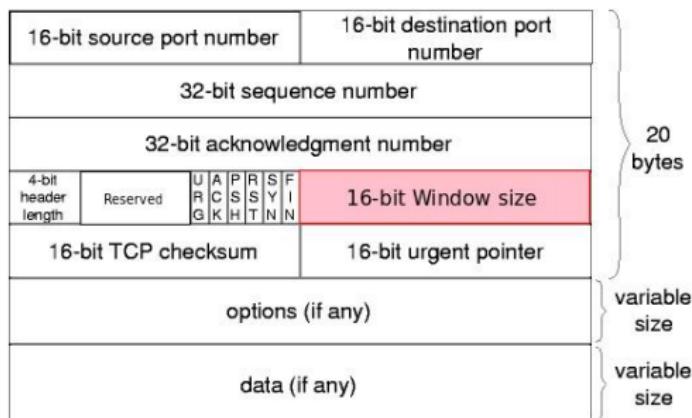
# Control de Flujo TCP

- De Extremo a Extremo, principio end-to-end.



# Control de Flujo TCP

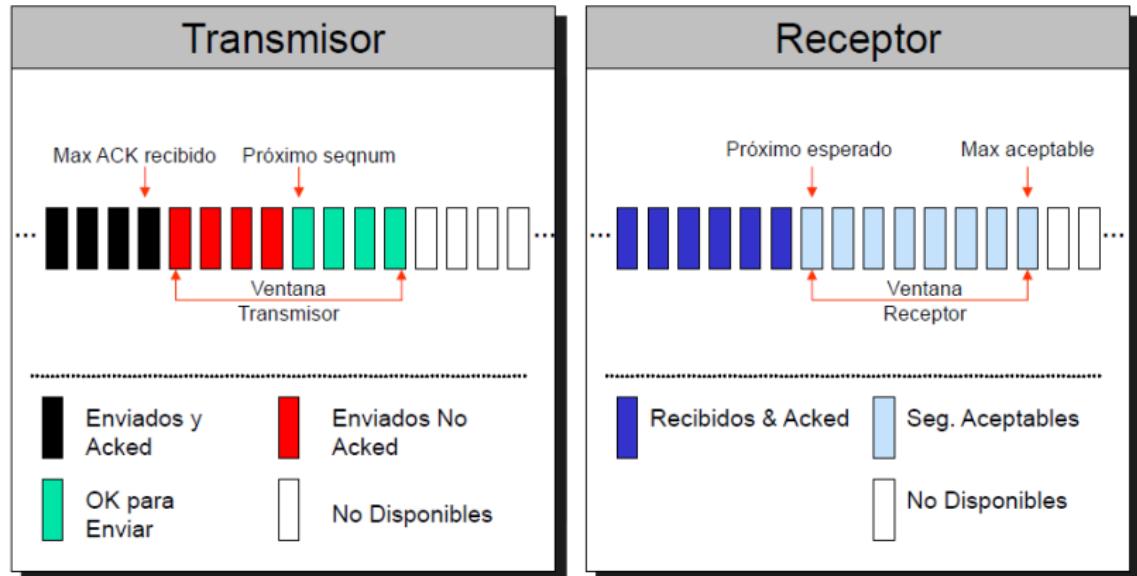
- El receptor (cada extremo puede recibir, es FDX) indica el espacio del buffer de recepción, **Rx Buffer**, en el campo del segmento: **Window** (de datos o ACK) **Advertised Window** (Ventana Anunciada).



# Control de Flujo TCP

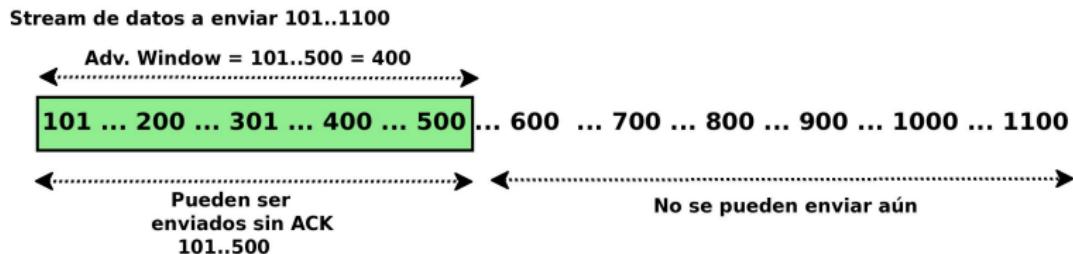
- Por cada segmento que envía indica el tamaño del buffer de recepción **Rx Buffer** (mbufs). (Cada conexión mantiene su propio buffer) en espacio del kernel (TCP).
- Window (Ventana) indica la cantidad de datos en bytes que el emisor le puede enviar sin esperar confirmación (mejora notablemente contra Stop & Wait).
- La ventana de recepción de cada extremo es independiente.
- Cada vez que llega un segmento es puesto por TCP en el **Rx Buffer**, TCP lo debe confirmar.
- Cada vez que la aplicación lee se hace espacio en el **Rx Buffer**. Se va modificando el tamaño de la ventana.
- Cada vez que llega un ACK en orden se mueve la ventana en el Transmisor, se descartan segmento confirmado de **Tx Buffer**.

# Ventana Deslizante TCP



# Ventana Deslizante TCP (Inicial)

- Se establece la conexión, se indica  $WIN = 400$ .

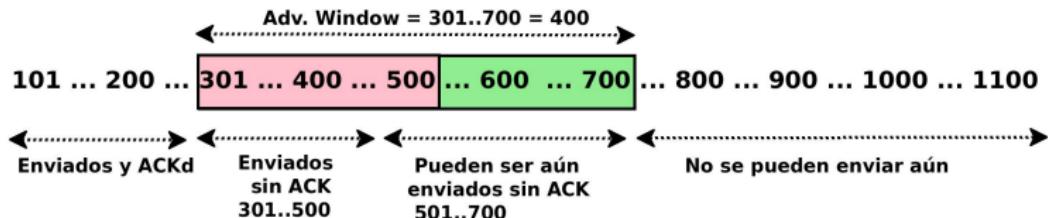


- Luego, la aplicación que envía escribe, `write()`, y se envían 400 bytes (los 400 bytes se pueden enviar en múltiples segmentos).
- Se recibe un segmento con  $ACK = 301$  y  $WIN = 400$ .
- Se desliza ventana.

# Ventana Deslizante TCP I

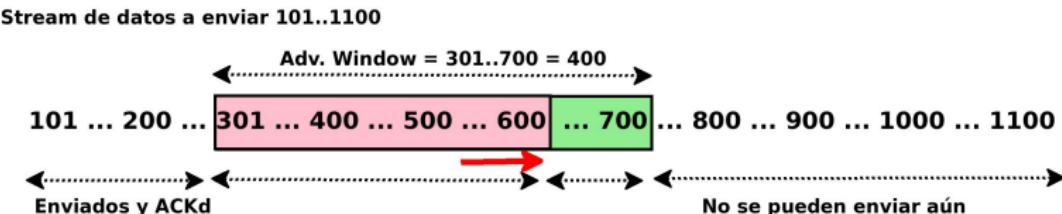
- 101..300 en ningún buffer, enviados y leídos.
- 301..500 en Tx Buffer y “en vuelo” o entrando a Rx Buffer.
- 501..700 en Tx Buffer, aún no han sido enviados.
- 701..1100 en la aplicación que envía, bloquea en caso de `write()`, depende de Tx Buffer.

Stream de datos a enviar 101..1100



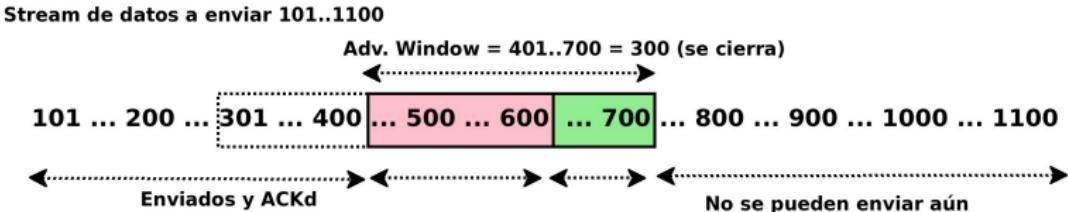
## Ventana Deslizante TCP II

- Se envía un segmento con los bytes 501..600.
  - No se recibe confirmación aún, el último segmento recibido  $W/N = 400$ .
  - 301..600 en Tx Buffer y “en vuelo” o llegando a Rx Buffer.
  - 601..700 en Tx Buffer, aún no han sido enviados.



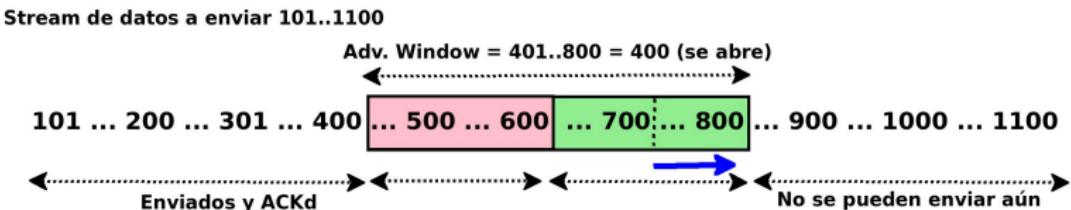
## Ventana Deslizante TCP III

- Se recibe un segmento  $ACK = 401$ ,  $WIN = 300$ .
  - 101..300 ya estaban procesados, 301..400 en Rx Buffer, no se han leído aún.
  - 401..600 en Tx Buffer, “en vuelo”.
  - 601..700 en Tx Buffer, aún no han sido enviados.
  - Ventana se cierra, la aplicación receptora no lee, no llama a `read()`.



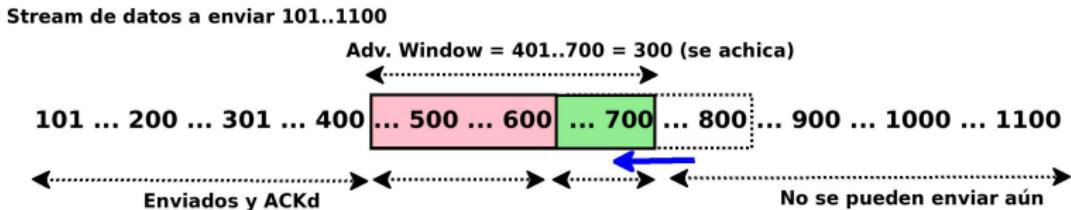
# Ventana Deslizante TCP IV

- Se recibe un segmento  $ACK = 401$ ,  $WIN = 400$ .
- $401..600$  en Tx Buffer, “en vuelo”.
- $601..800$  en Tx Buffer, aún no han sido enviados.
- Ventana se abre, la aplicación receptora lee, llama a `read()`.
- $101..400$  no están más en Rx Buffer, se procesaron.



# Ventana Deslizante TCP V

- Se recibe un segmento  $ACK = 401$ ,  $W/N = 300$ .
- 401..600 en Tx Buffer, “en vuelo”.
- 601..700 en Tx Buffer, aún no han sido enviados.
- Ventana se achica.
- No debería suceder, TCP achica el Rx Buffer.



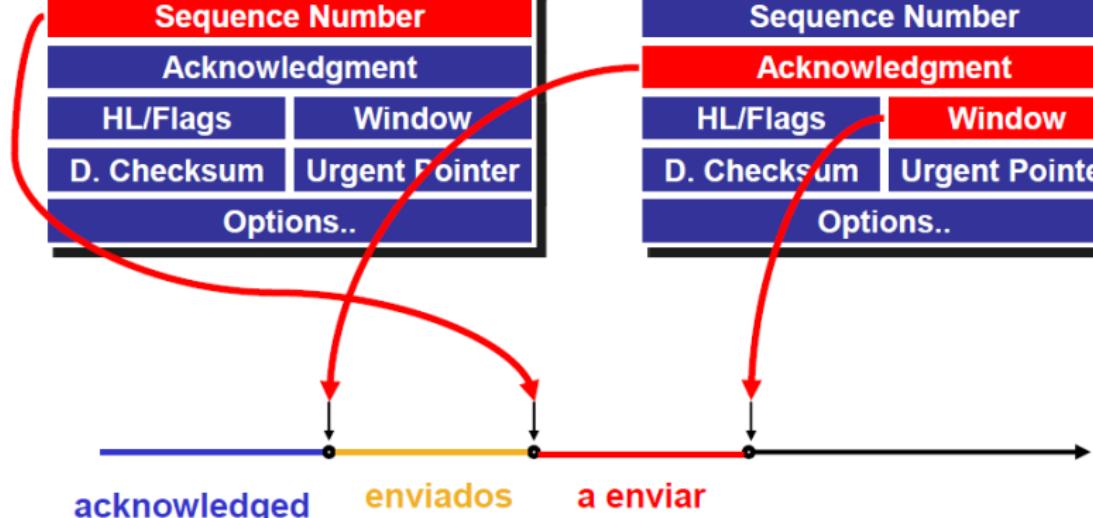
# Ventana Deslizante TCP

Segmento enviado

Source Port	Dest. Port
<b>Sequence Number</b>	
Acknowledgment	
HL/Flags	Window
D. Checksum	Urgent Pointer
Options..	

Segmento recibido

Source Port	Dest. Port
<b>Sequence Number</b>	
Acknowledgment	
HL/Flags	Window
D. Checksum	Urgent Pointer
Options..	



# Control de Flujo TCP

- Ventana de Recepción recibida:  $Win = rwnd$ .
- El receptor “ofrece/publica” la ventana  $Win$  en los segmentos TCP.
- El transmisor no puede enviar más de la cantidad de bytes en:  
 $Win - Sent.No.ACKed$ ,  
 $Effective\_Win = Win - (LastByteSent - LastByteAcked)$   
si no se tiene en cuenta la congestión.
  - Al recibir ACKs de TCP (App. no lee aún) se cierra ventana.
  - Al recibir ACks y  $Win$  fijo desliza ventana (App. lee a rate fijo).
  - Al achicarse  $Win$  se reduce ventana (App, no lee).
  - Al agrandarse  $Win$  tiene posibilidad de enviar más (App. lee más rápido).
  - Tamaño de ventana seleccionado por el kernel o por aplicación `setsockopt()`.

# TCP Bulk y TCP Interactivo

- Delayed ACKs: No enviar ACK sin esperar de enviar datos antes: piggy-back (200ms, MAX=500ms).
- Algoritmo Nagle: No enviar datos en chunks pequeños, esperar juntar información.
- Perjudica aplicaciones interactivas.
- Tinygrams: segmentos chicos: App. interactivas, *Win* casi vacía.
- Silly Window: *Win* casi llena se “ofrecen” pequeños incrementos. Solución: Muestra incrementos de  $\text{Min}(\text{MSS}, \text{RecvBuf})/2$ .

# Referencias

- [KR] Kurose/Ross: Computer Networking (5th Edition).
- [Stev] TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens.
- [StevII] TCP/IP Illustrated, Volume 1: The Protocols, 2nd Ed. W. Richard Stevens, Kevin R. Fall.
- [RFCs] RFCs: <http://www.faqs.org/rfcs/> RFC-793, ... RFC-791, RFC-1323, RFC-2001, RFC-2018, RFC-2581, RFC-5681, RFC-2582, RFC-6582, RFC-3168, RFC-3649, RFC-2988, RFC-6298.
- [TCPIPg] TCP/IP Guide: <http://www.tcpipguide.com/>.
- [Transport Layer] <http://people.westminstercollege.edu/faculty/ggagne/spring2007/352/notes/unit4/index.html>

# TCP: Control de Congestión

2019



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

0-0

## Servicios de TCP

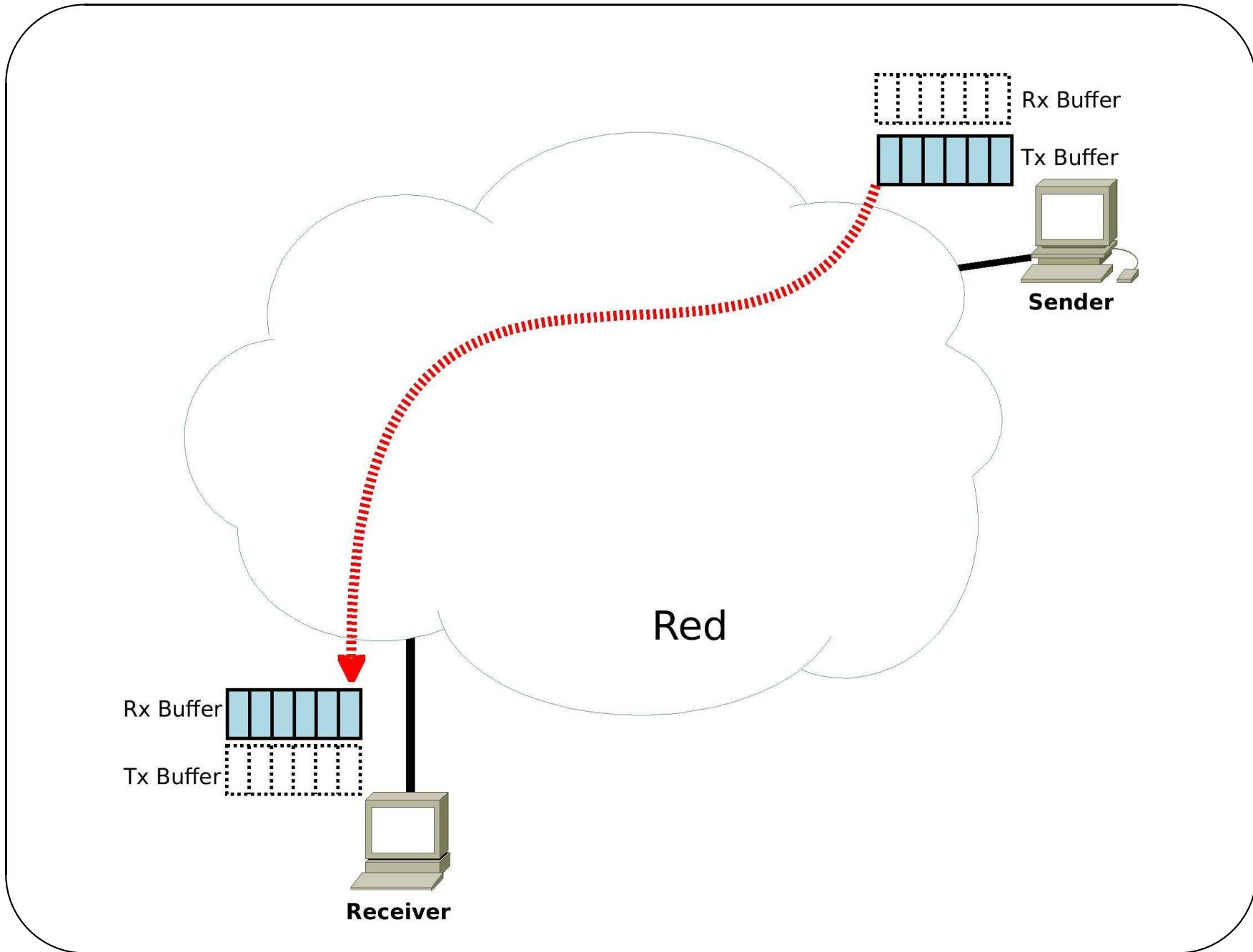
- Control de Errores:
  - Mecanismo protocolar que permite ordenar los segmentos que llegan fuera de orden y recuperarse mediante solicitudes y/o retransmisiones de aquellos segmentos perdidos o con errores.
  - Se realiza por cada conexión: End-to-End, App-to-App.
- Control de Flujo (Flow-Control):
  - Mecanismo protocolar que permite al receptor controlar la tasa a la que le envía datos el transmisor.
  - Controla cuanto puede enviar una aplicación sabiendo que la receptora tiene capacidad de recibirla (espacio en RxBuf).

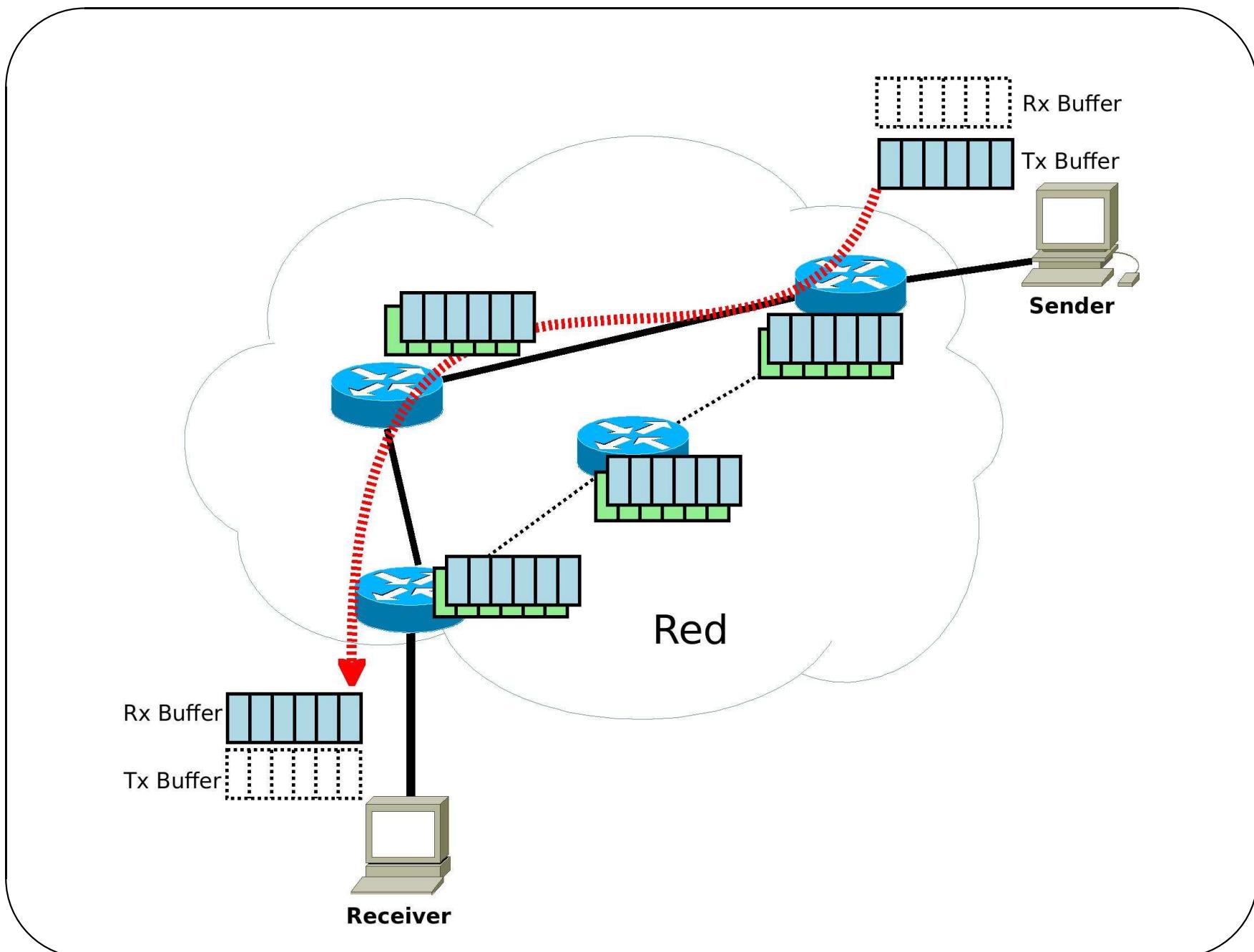
## Servicios de TCP (Cont.)

- Control de Flujo (Cont.):
  - Se realiza por cada conexión: End-to-End, App-to-App.
  - Permite que aplicaciones con diferentes capacidades dialoguen regulando la velocidad, tasa de transmisión.
  - Tiene en cuenta solo el estado del receptor, no el de la red.
- Control de Congestión:
  - Se realiza por cada conexión: End-to-End, App-to-App.
  - Permite que aplicaciones no saturen la capacidad de la red.
  - Tiene en cuenta el estado de la red a diferencia del control de flujo que solo ve el receptor.

## Control de Congestión TCP

- **Objetivo:** Controlar el tráfico que se envía evitando que se colapse la red y se descarte teniendo que retransmitirse.
- Se puede implementar End-to-End (caso TCP, RFC-5681 (ant. RFC-2581)).
- O tomando partida la red. Modelo basado en la Red: IP+TCP:RFC-3168 (similar a mecanismos L2/L3: Frame-Relay, ATM).
- **Congestión:** Problemas de delay en los routers, problemas de overflow y descarte.





## Causas de Congestión en la Red

- Límite de la capacidad de la red:
  - Velocidad de los Routers/Switches (CPU).
  - Capacidad de los Buffers de los Routers/Switches (Memoria).
  - Velocidad de los Enlaces (Interfaces).
- Utilización de la red:
  - Demasiado tráfico en la red (modelo de red compartida).
  - Se detecta por los nodos intermedios (router/switches) por ejemplo: cuando las colas sobrepasan un umbral. Se utiliza simple umbral o doble umbral (min,max).

## Control de Congestión, Modelo End-to-End

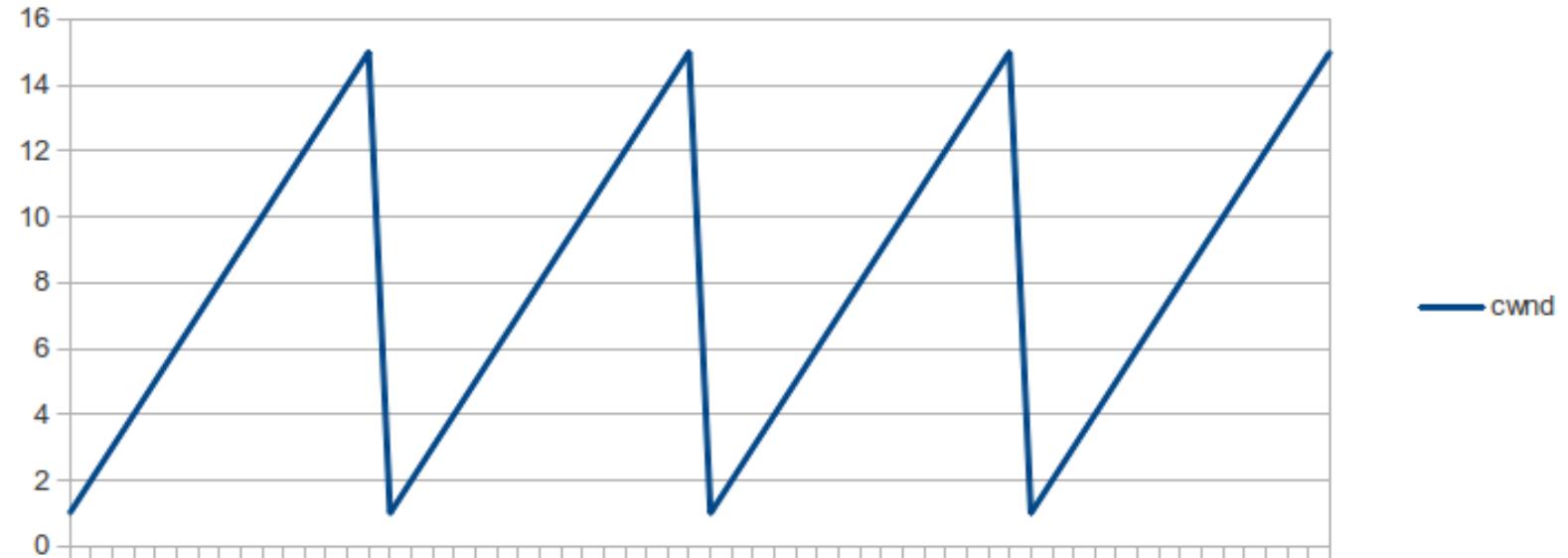
- Modelo en el cual no participa la red (más implementado).
- Se utilizan nuevos parámetros a los de Control de Flujo (variables locales):
  - *cwnd* Ventana de congestión. tiene en cuenta el estado de la red.
  - *ssthresh* Slow Start Threshold (Umbral).
  - Se calcula:  $MaxWin = \min(rwnd, cwnd)$ . *rwnd* era la ventana de recepción, usada para el control de flujo.

- $FlightSize = (LastByteSent - LastByteACKed)$   
segmentos en vuelo, enviados y aún no confirmados  
( $Sent.No.ACKed$ )
  - Puede enviar:  
$$Effective\_Win = MaxWin - FlightSize.$$
  - $MaxWin = Min(rwnd, cwnd)$  se calcula en base a quién  
esta más cargado, el receptor o la red.
- Diferentes Fases:
- Si  $cwnd < ssthresh$ : Fase de crecimiento inicial: SS (Slow Start).
  - Si  $cwnd \geq ssthresh$  Fase de mantenimiento: CA (Congestion Avoidance).

## Evolución Control de Congestión TCP (Old Version)

- 1974, 1era. Versión TCP, Vint Cerf, Bob Kahn, TCP incluía también la capa de red (IP).
- Se divide en capas, TCP/IP, No considera control de Congestión inteligente.
  - “Ventana de congestión: *cwnd*” (no definida), tráfico crece hasta que se resetea, buffer overflow o error.
  - El destino podría limitarla con *rwnd*: válido para LAN.
  - No utiliza Slow Start (SS), se supone: incrementa *cwnd* ++ (en MSS) por cada RTT.
  - ACK perdido o segmento erróneo deriva en arrancar de 0.
  - Congestión/Límite de la red detectada por solo un tipo de evento, expira RTO.
  - Algoritmo no válido para entorno WAN, Internet.

## Control de Congestión TCP (Old Version)



## Control de Congestión TCP (Old Tahoe/Tahoe)

- Old Tahoe (BSD 4.2 - 1984 ?).
- TCP Tahoe (BSD 4.3 - 1988). Van Jacobson.
  - Utiliza Slow Start (SS), ventana crece exponencialmente, inicio  $cwnd = IW = 1 * MSS$ , o similar,  $IW$  (Init Window) puede ser 2 o 3 segmentos.
  - Una vez que se alcanza  $ssthresh$  se trabaja con Congestion Avoidance (CA).
  - Valor inicial  $ssthresh = \infty$  (un valor alto).
  - Old-Tahoe sin Fast Retransmit, solo SS y CA.

- Tahoe agrega Fast Retransmit, aunque no esta bien implementado.
- Congestión detectada por RTO o 3DUP ACKs, derivaba en ambos casos en: Slow Start:  
 $ssthresh = \text{Min}(cwnd}/2, 2) * MSS$  ,  
 $cwnd = LW = 1 * MSS$  (Loss Window). (mejor implementado en TCP Reno).
- Puede suceder que MSS sea diferente entre emisor y receptor, para este caso se considera  $SMSS$  y  $RMSS$ . Los cálculos se hacen en base a  $SMSS$ .

## Control de Congestión TCP (SS)

- Slow Start:

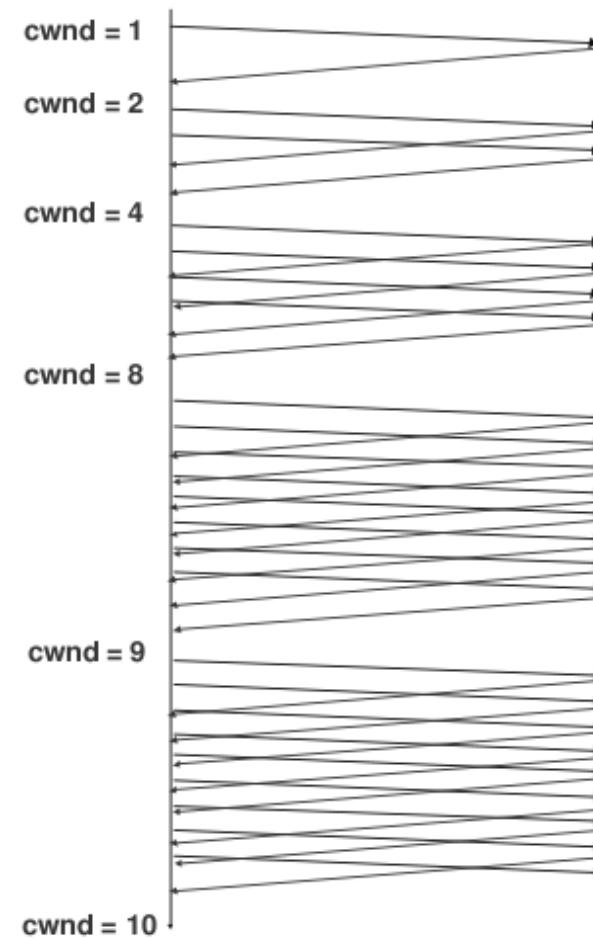
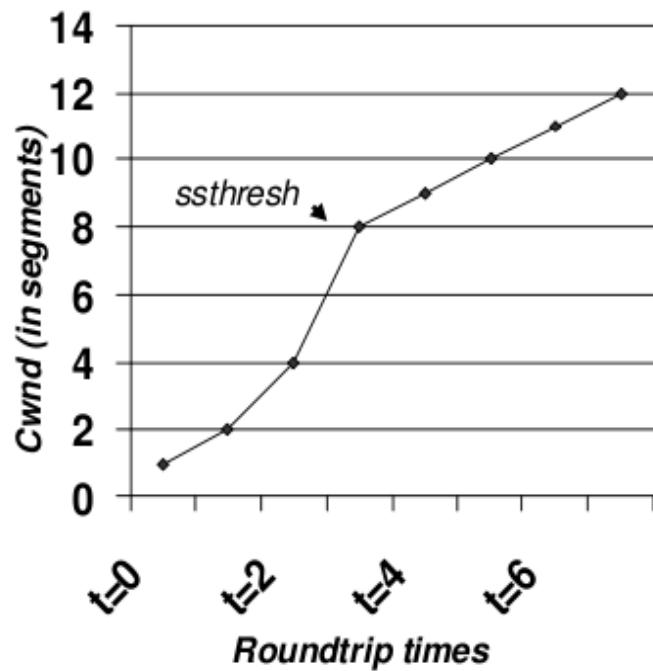
- Crece exponencialmente, de forma rápida, no es slow (lento).
- Se le llama Slow Start porque comienza a probar con pocos paquetes, menos agresivo que el enfoque de enviar tanto como la ventana de recepción permita.
- Inicia  $cwnd = IW = 1 * MSS$  (a veces se usa 2 o 3). Transmite y espera ACK.
- ACK recibido,  $cwnd += cwnd = 2 * MSS$ , nuevos ACKs:  $cwnd = 4 * MSS \dots$
- Si destino retarda ACK no se cumple.
- Incrementa  $cwnd +=$  (en  $MSS$ ) por cada ACK (varios por RTT, ráfaga).

## Control de Congestión TCP (CA)

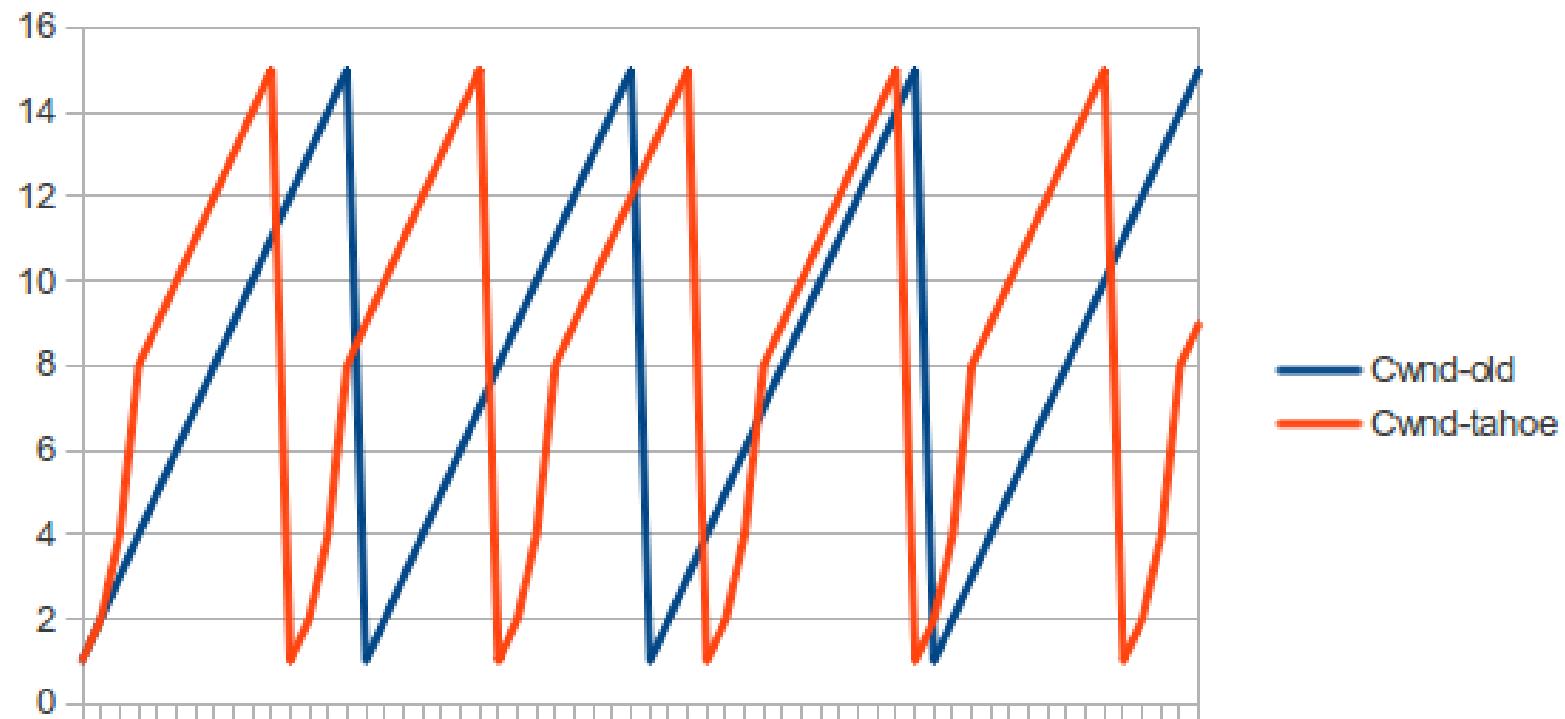
- Congestion Avoidance:
  - Ante el primer evento de congestión se calcula el  $ssthresh$ , primer ráfaga SS puro.
  - Una vez que  $cwnd \geq ssthresh$  crece de forma lineal.
  - Incrementa  $cwnd++$  por cada RTT.
- Fast Retransmit: objetivo, recuperarse más rápido que un timeout. En Tahoe, FRT seguido por Slow Start. Vuelve al inicio  $cwnd = 1 * MSS$ .

## Fases Control de Congestión TCP (SS y CA)

Assume that  $ssthresh = 8$



## Control de Congestión TCP (Old Version vs. Tahoe)



## Control de Congestión TCP (Reno)

- TCP Reno (BSD 4.3 - 1990) Van Jacobson.  
RFC-2001/RFC-2581/RFC-5681.
- Se implementan de forma correcta Fast Retransmit y Fast Recovery.
  - Slow Start (SS)
  - Congestion Avoidance (CA).
  - Fast Retransmit (FRT)
  - Fast Recovery (FR).
- $FlightSize = cwnd$  si siempre tuvo datos para enviar, aunque en general  $cwnd > FlightSize$ , donde  $cwnd$  crece sin enviar realmente datos, por lo tanto no reflejaría en ese caso el límite de la red.

## Control de Congestión TCP (Reno FRT)

- Fast Retransmit:
  - Intenta recuperarse más rápido que un timeout (expira RTO).
  - El receptor inmediatamente al recibir un segmento fuera de orden no debe esperar RTO del emisor sino generar un ACK (DUP ACK). No se debe retardar.
  - Debido que el emisor no sabe si se perdió o llegó fuera de secuencia: espera por más ACK duplicados.
  - El emisor, si recibe 3 ACK duplicados (4 ACK para el mismo segmento) considera que se perdió (no es re-ordenamiento) y re-envía el segmento solicitado sin esperar RTO.

## Control de Congestión TCP (Reno FR)

- Fast Recovery:

- Si recibe 3 ACK duplicados (4 ACK para el mismo segmento) entra en Fast Retransmit y se reenvió el segmento.
- Luego de Fast Retransmit entra en fase Fast Recovery, crece lineal.
- Incrementa de forma lineal la ventana por cada ACK recibido, considera que es un espacio nuevo en la red (incremento inicialmente en 3, por los 3ACK duplicados)
- Luego de Fast Recovery (se ACKed el segmento perdido) se realiza CA(Reno), no SS(Tahoe).

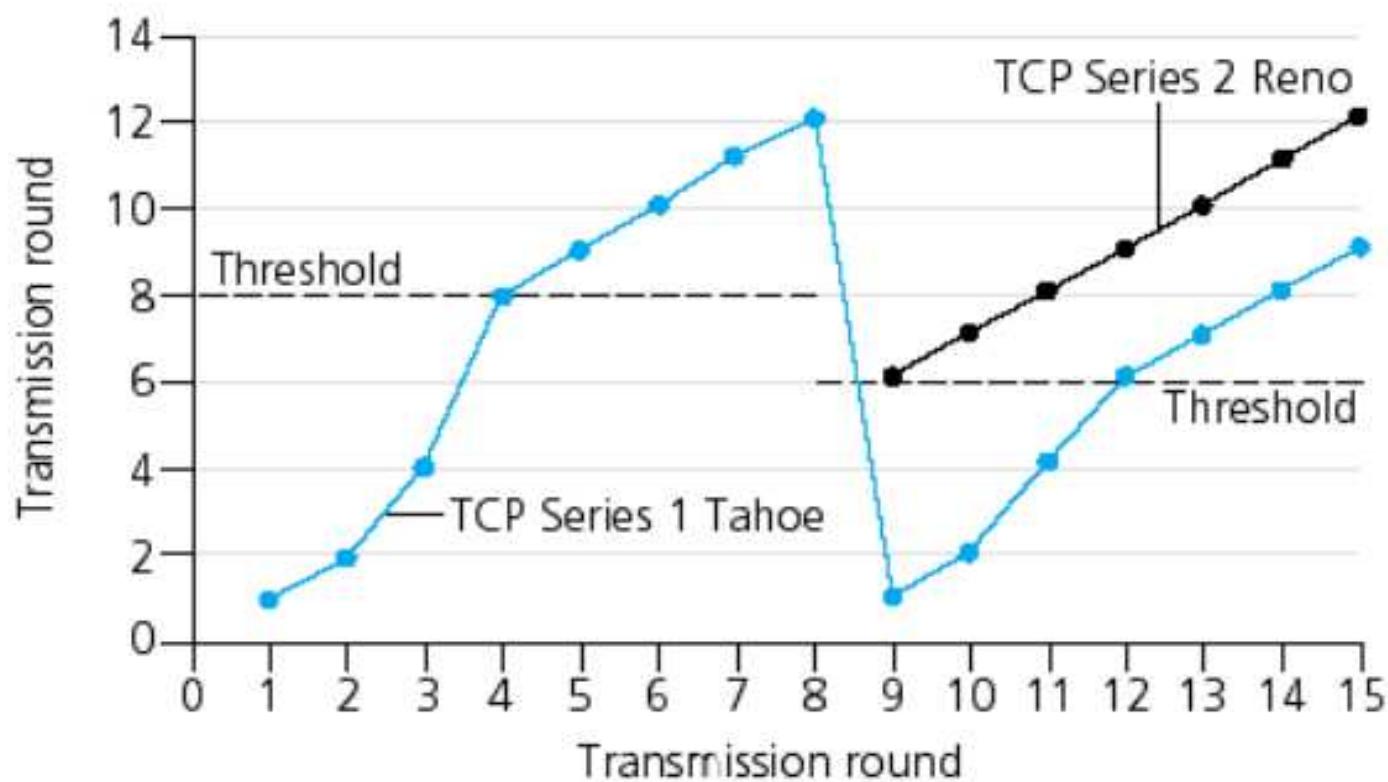
## Control de Congestión TCP (Reno)

- Si se detectan 3DUP ACKs (Fast Retransmit):
  1.  $ssthresh = \text{Min}(cwnd/2, 2) * MSS$ , en RFC  
 $ssthresh = \text{Min}(\text{FlightSize}/2, 2 * SMSS)$
  2. Fast Retransmit.
  3. Fast Recovery:  $cwnd = (ssthresh + 3) * MSS$  (3 segments cached in receiver).
  4. Por cada ACK de segmento distinto que recibe incrementa la ventana  $cwnd ++$  (crece linealmente).
- Una vez recuperado (ACKed segmento perdido) vuelve “a la mitad”,  $cwnd = ssthresh$ , y comienza con CA.
- Los incrementos lineales realizados en FR previos a la recuperación se vuelven atrás (se achica un poco  $cwnd$ ).

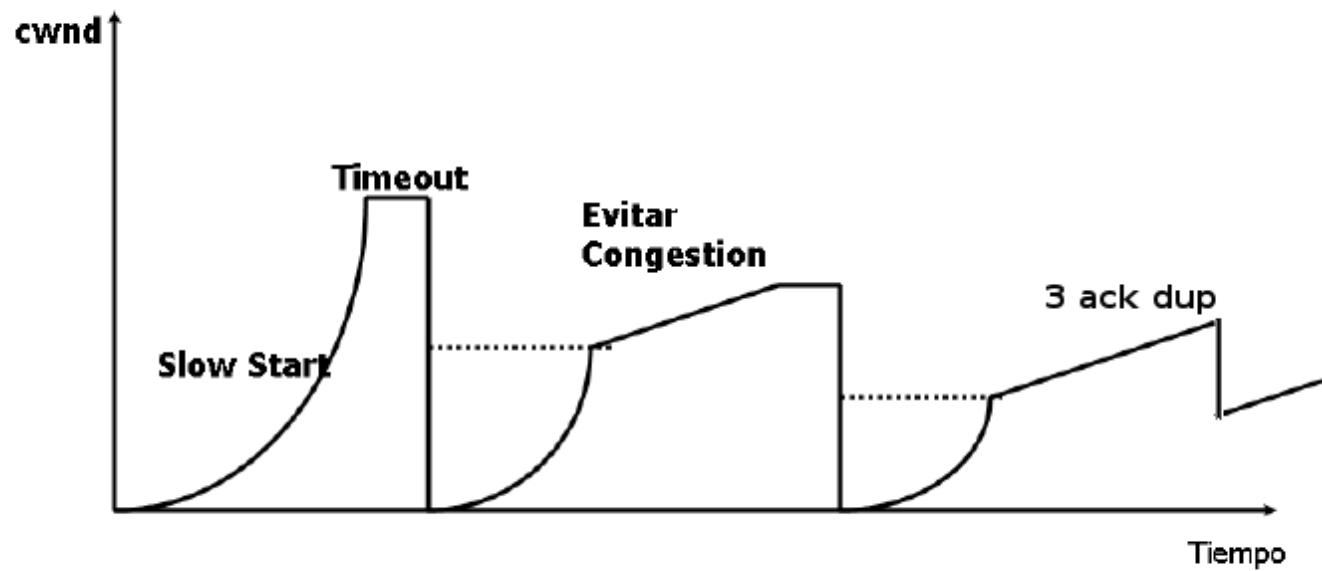
## Control de Congestión TCP (Reno)

- Si se timeout, RTO expira.
  1. Retransmite el segmento perdido.
  2.  $ssthresh = \text{Min}(cwnd}/2, 2) * MSS$ , en RFC  
 $ssthresh = \text{Min}(FlightSize}/2, 2 * SMSS)$
  3.  $cwnd = LW = 1 * MSS$ , en RFC  $LW = 1 * SMSS$ .
  4. Inicia con Slow Start como en los algoritmos anteriores.
  5. Comienza a retransmitir como Go-Back-N a partir del segmento que dio timeout de acuerdo a lo que le permite la ventana.

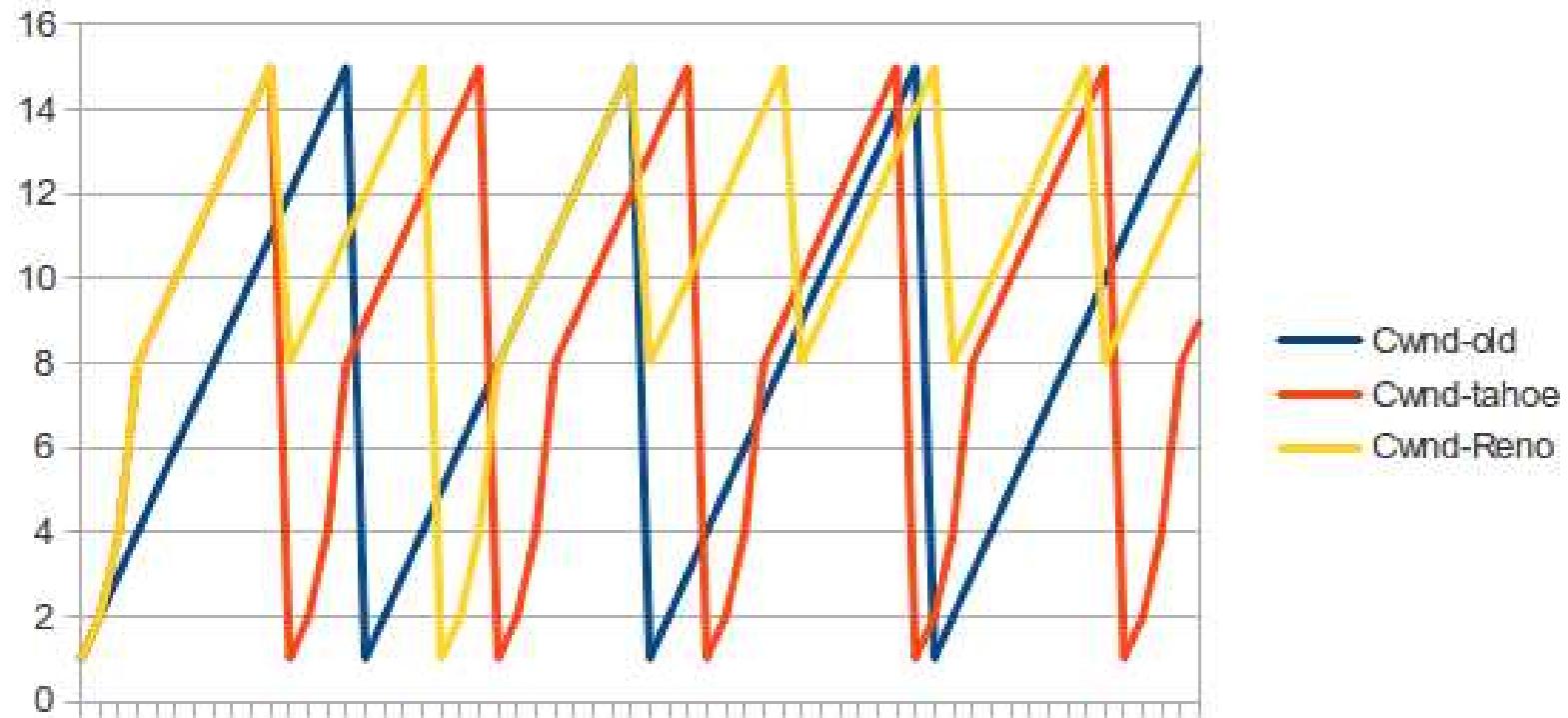
## Control de Congestión TCP (Fases)



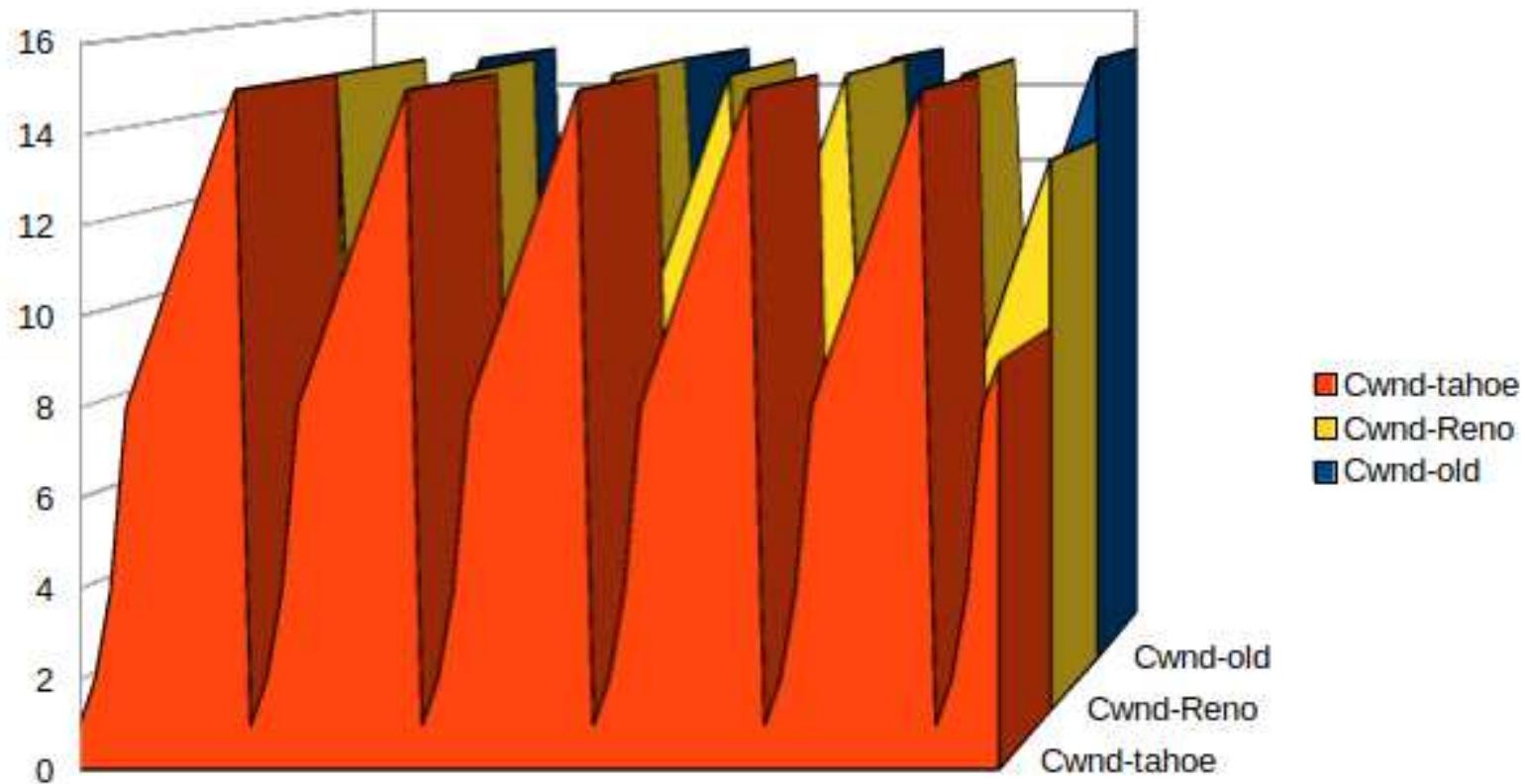
## Ejemplo: Evolución Ventana de Congestión



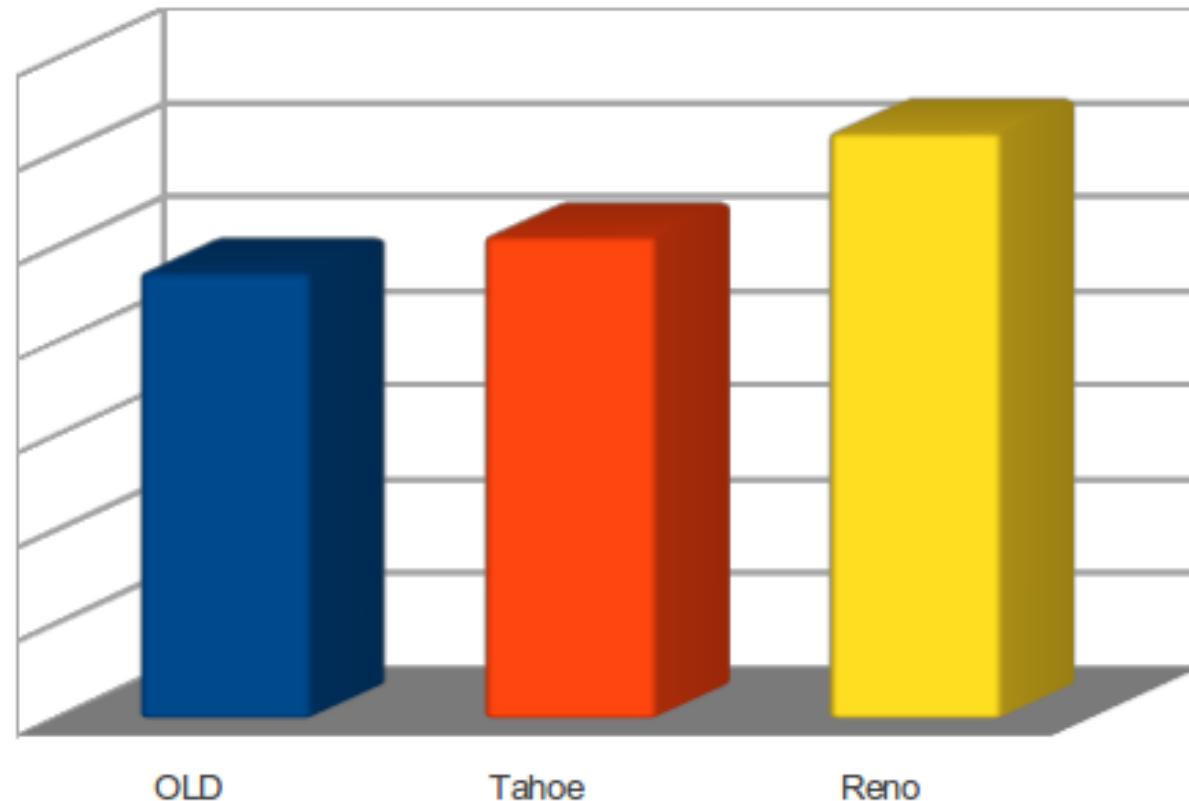
## Control de Congestión TCP (Old vs. Tahoe vs. Reno)



## Control de Congestión TCP (Old vs. Tahoe vs. Reno) 3D



## Control de Congestión TCP (Old vs. Tahoe vs. Reno)



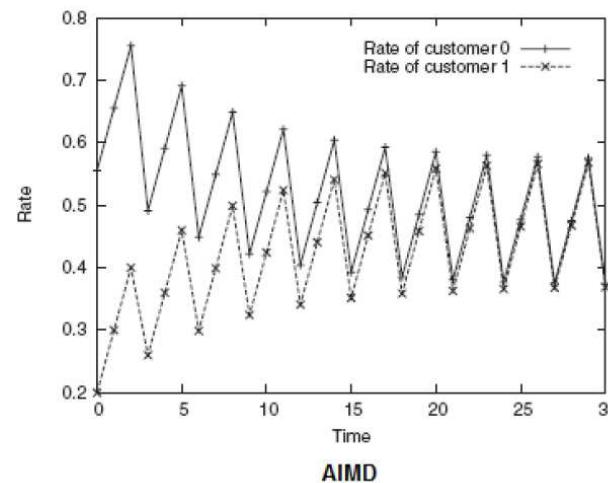
## Modelo de Algoritmo TCP CC

```
void tcp_snd(event_t ev)
{
    switch (ev) {
        case (init):
            cwnd      = 1;
            ssthresh = INF;
            ...
            break;
        case (newack):
            if (cwnd<ssthresh) {
                /* Slow Start */
                /* 1 MSS for each ACK */
                cwnd++; // cwnd = cwnd + 1
            } else {
                /* Congestion Avoidance */
                /* 1 MSS for each RTT */
                cwnd = cwnd + 1/cwnd;
            }
            ...
            break;
        ...
        case (timeout):
            ssthresh = cwnd / 2;
            cwnd = 1;
            ...
            break;
        case (3ackdup):
            fast_retrans_fast_recover();
            ssthresh=cwnd / 2;
            cwnd = ssthresh + 3;
            ...
    }
}
```

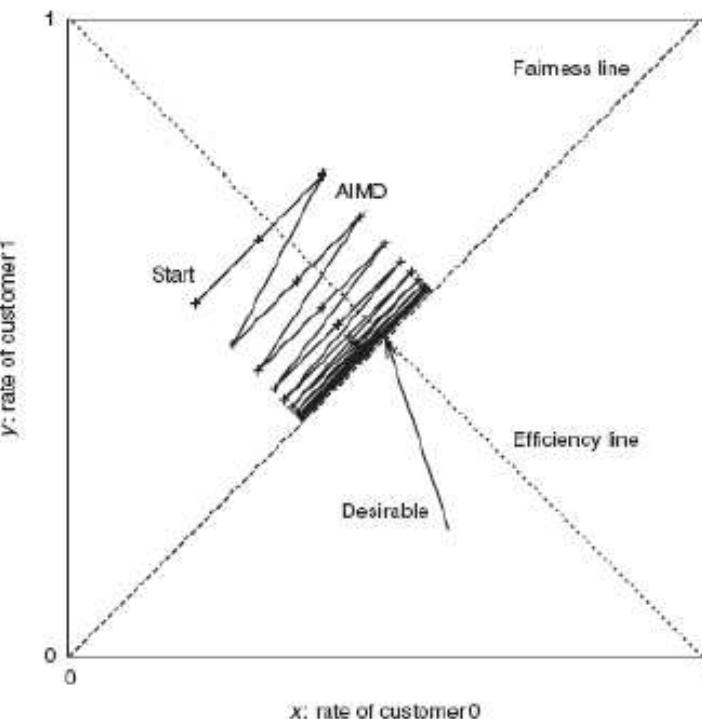
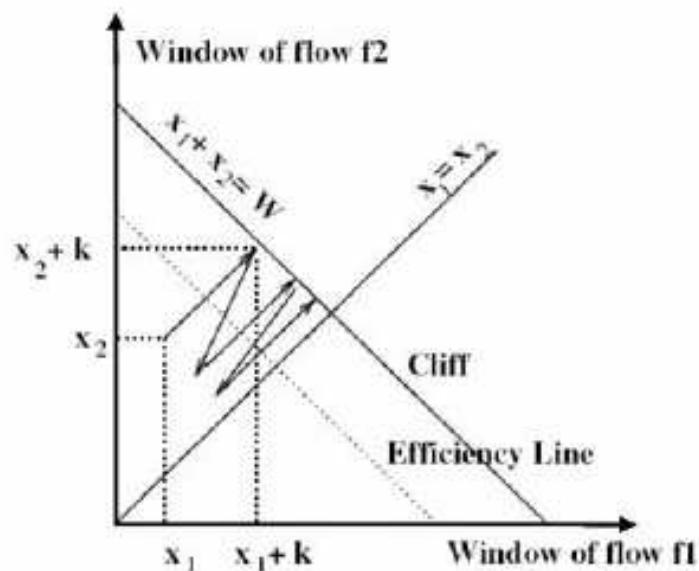
## AIMD (Additive Increase/Multiplicative Decrease)

- De acuerdo al modelo de TCP parece ser estable y justo.
- Utiliza un enfoque AIMD, crece de forma aditiva con ACK positivos y decrece de forma multiplicativa ante 3 ACK DUP (a la mitad).
- Se tratan de autoregular entre flujos.
- $a_i > 0; a_d = 0$
- $b_i = 1; 0 < b_d < 1$

$$w(t+1) = \begin{cases} a_i + b_i w(t) & \text{si } (+) \\ a_d + b_d w(t) & \text{si } (-) \end{cases}$$



## AIMD (Additive Increase/Multiplicative Decrease)



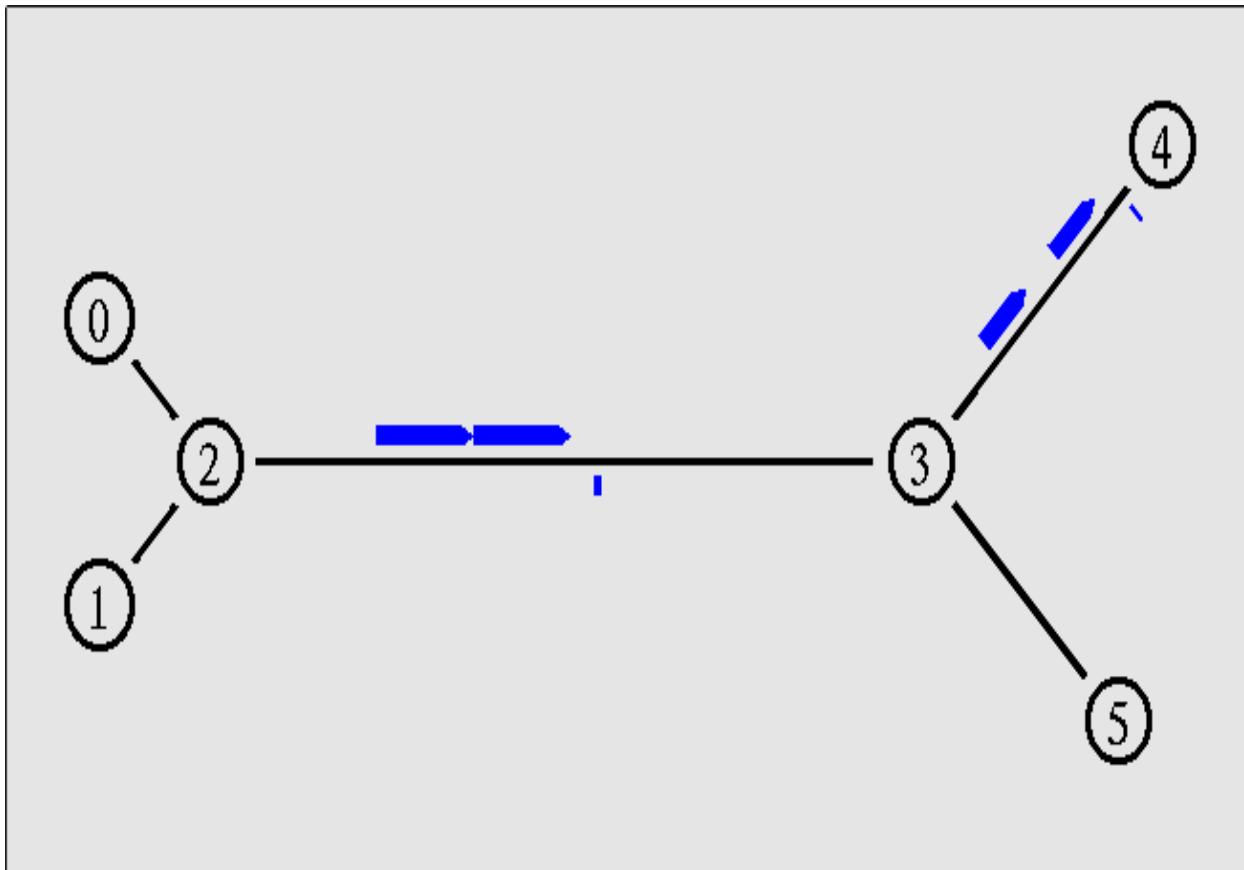
## Control de Congestión TCP (New Reno)

- TCP Reno ante la pérdida de múltiples segmentos probablemente termine en timeout (RTO expira) y vuelve a Slow Start.
- TCP New Reno, última versión RFC 6582(2012) hace obsoletas RFC 3782 (2004), RFC 2582(1999) S. Floyd, T. Henderson.
- Una vez que está en Fast Recovery, permite enviar varios segmentos perdidos y recuperarse de ACK parciales.
- New Reno obtiene buen rendimiento con pérdida en varios segmentos y sin SACK.

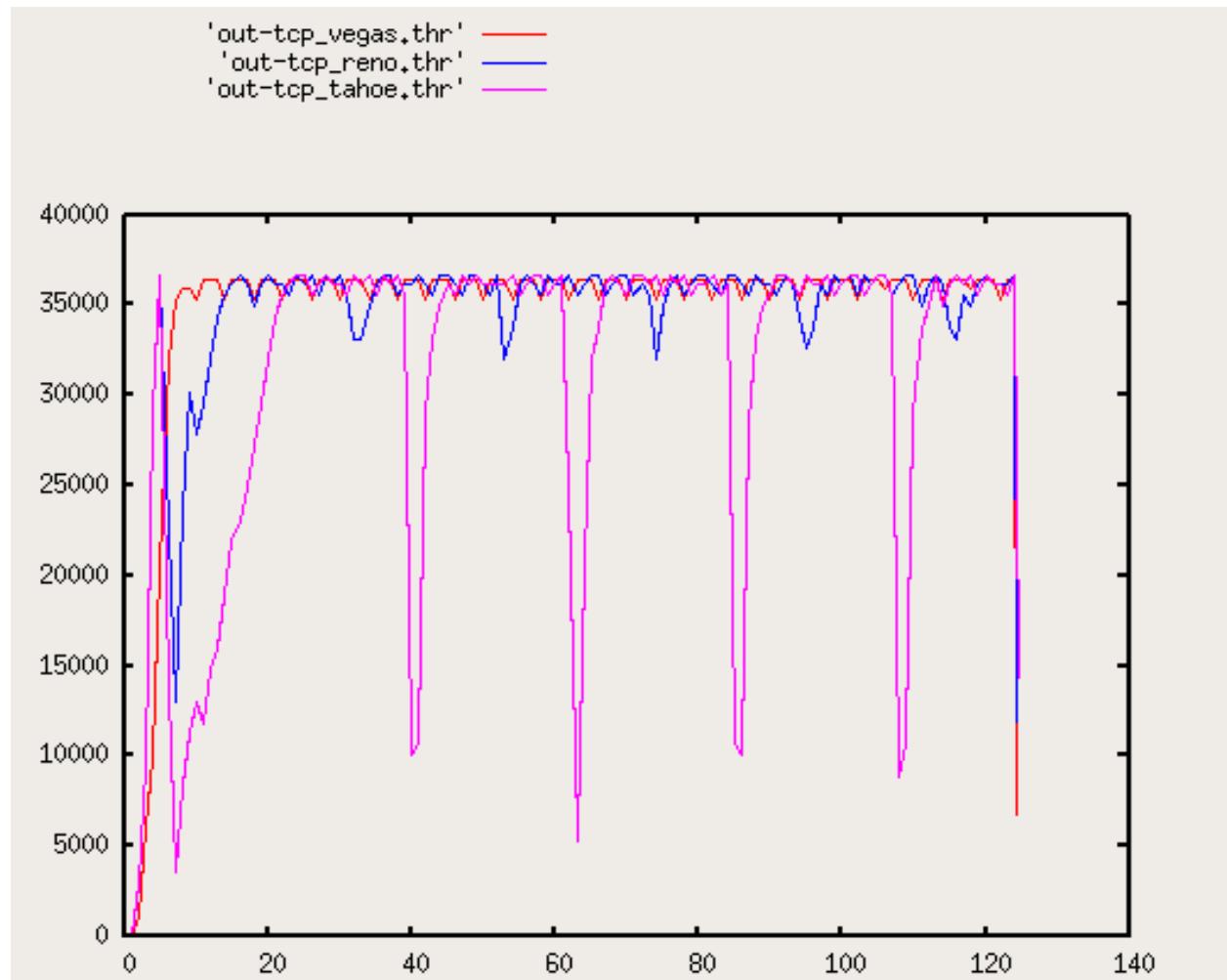
## Otras Implementaciones TCP CC

- TCP New Reno (1996), mejora ante la pérdida de más segmentos.
- SACK (1996).
- TCP Vegas.
- TCP BIC.
- TCP Cubic.
- TCP Westwood.

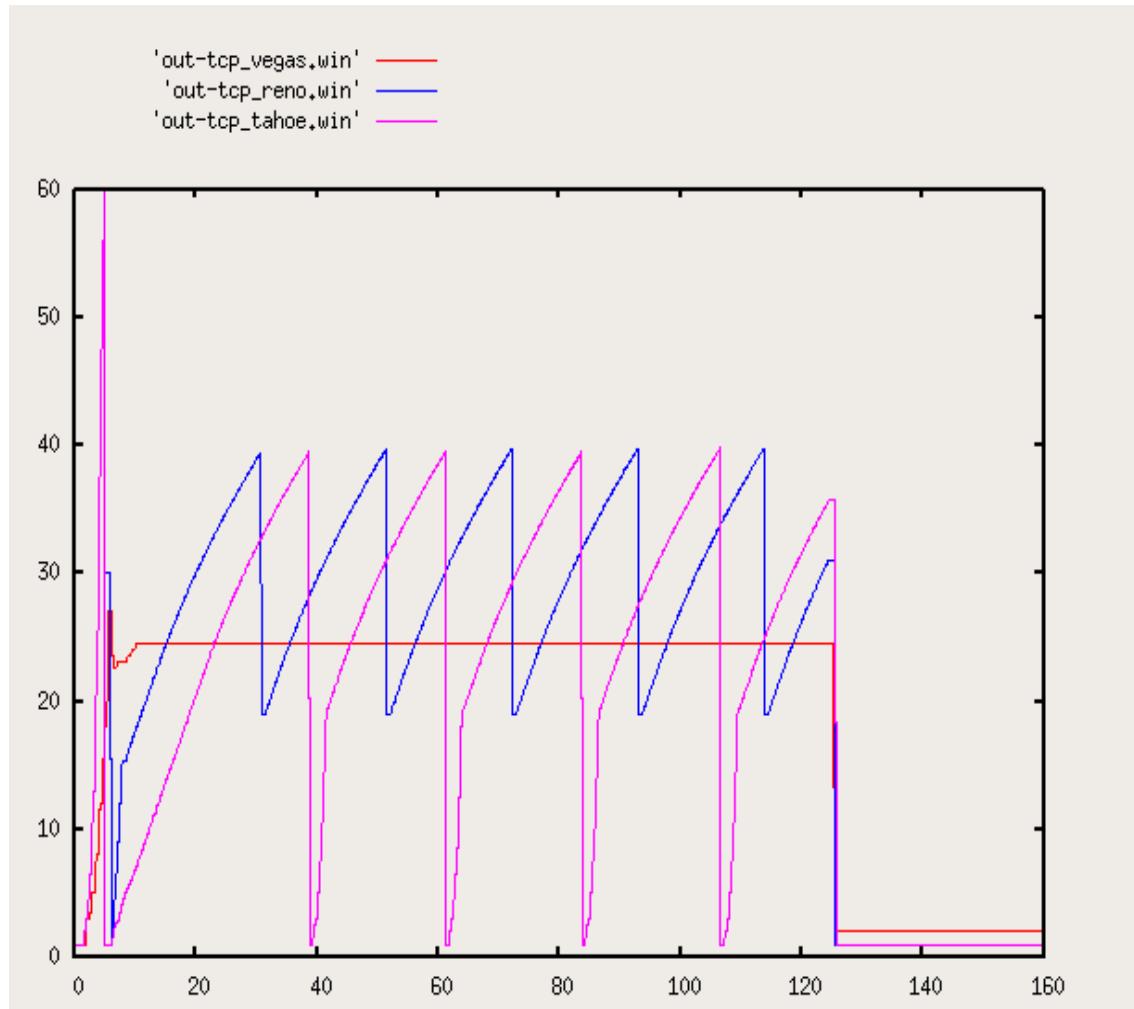
## Implementaciones Modelo



## Implementaciones (Comparar Throughput)



## Implementaciones (Comparar cwnd)



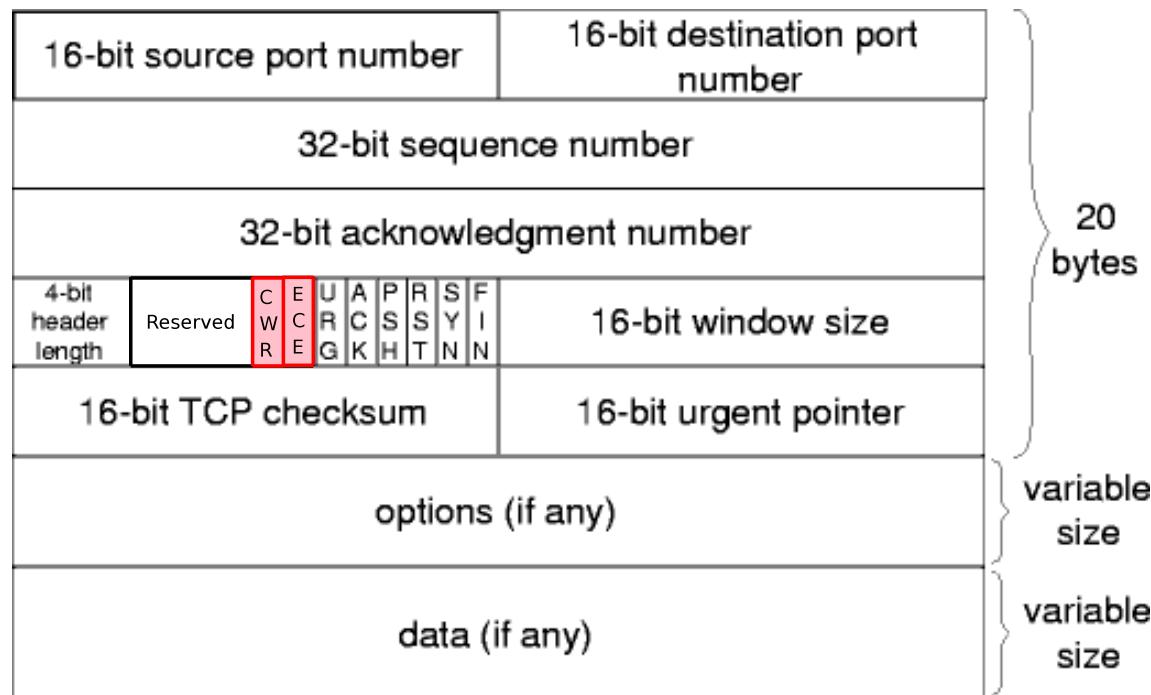
## Control de Congestión por la Red

- TCP se monta sobre IP, best effort protocol, no considera en su origen QoS.
- Luego RFC-1349(ToS), obsoleta por RFC-2474(DSCP).
- Luego con RFC-3168 se agrega funcionalidad de CC por la red.
- Se utilizan campos reservados para marcar tráfico.

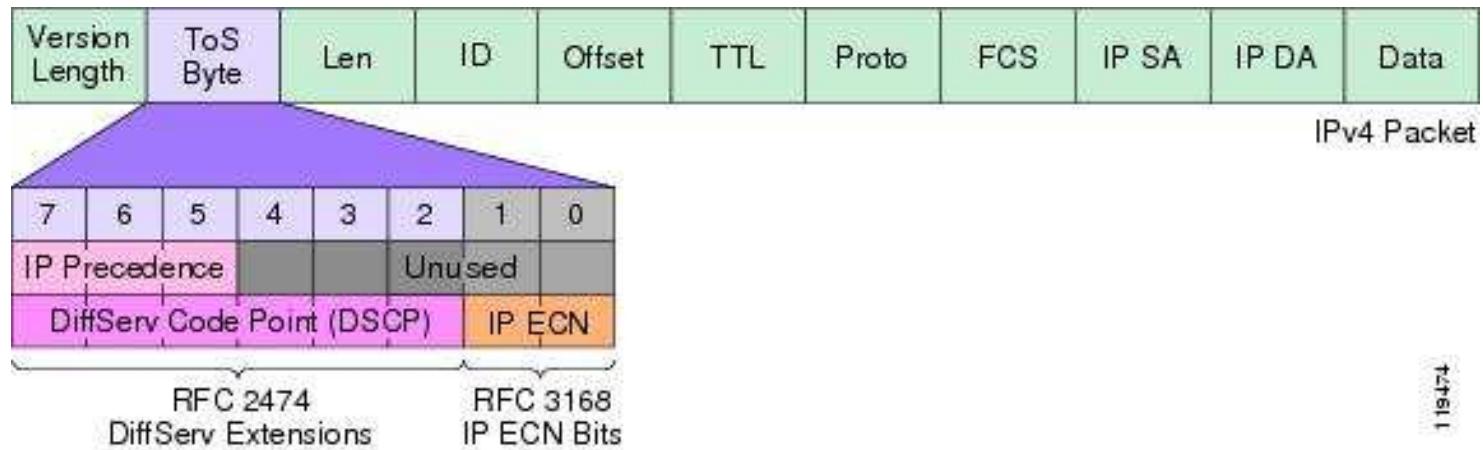
## Control de Congestión por la Red (Cont'd)

- Los destinos congestionados, o routers intermedio pueden generar ICMP: Source Quench.
- Estos mensajes deberían ser considerados y bajar el data rate.
- No solo funcionan con UDP, es a nivel IP.
- Habitualmente son mensajes ignorados.
- TCP+IP con ECN es un mecanismo más adecuado.

## Segmento TCP Marcado con ECE y CWR



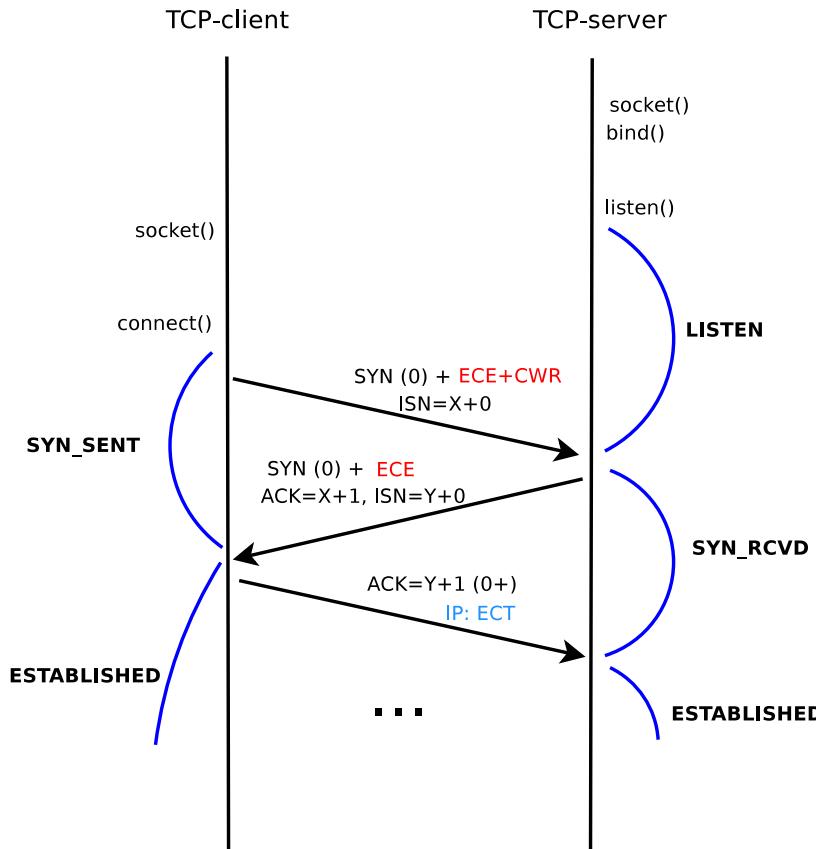
## Datagrama IP Marcado con ECN: ECT, CE



## Control de Congestión TCP+IP (Cont'd)

- Al establecerse la conexión TCP, se negocia capacidad de ECN.
- Configuran en el Setup, SYN el  $ECE = CWR = 1$  (ECN-Echo). “ECN-setup SYN packet”.
- SYN+ACK configuran  $ECE = 1$  y  $CWR = 0$ .
- Luego en datagramas IP se configuran  $ECT(0) = 01$  o  $ECT(1) = 10$  (ECN-Capable Transport).
- Si router (nodo intermedio) detecta congestión, tamaño de cola por encima de umbral aplica RED (Random Early Detection).
- Al aplicar RED, en lugar de descartar marca  $CE = 11$  (Congestion Experienced).

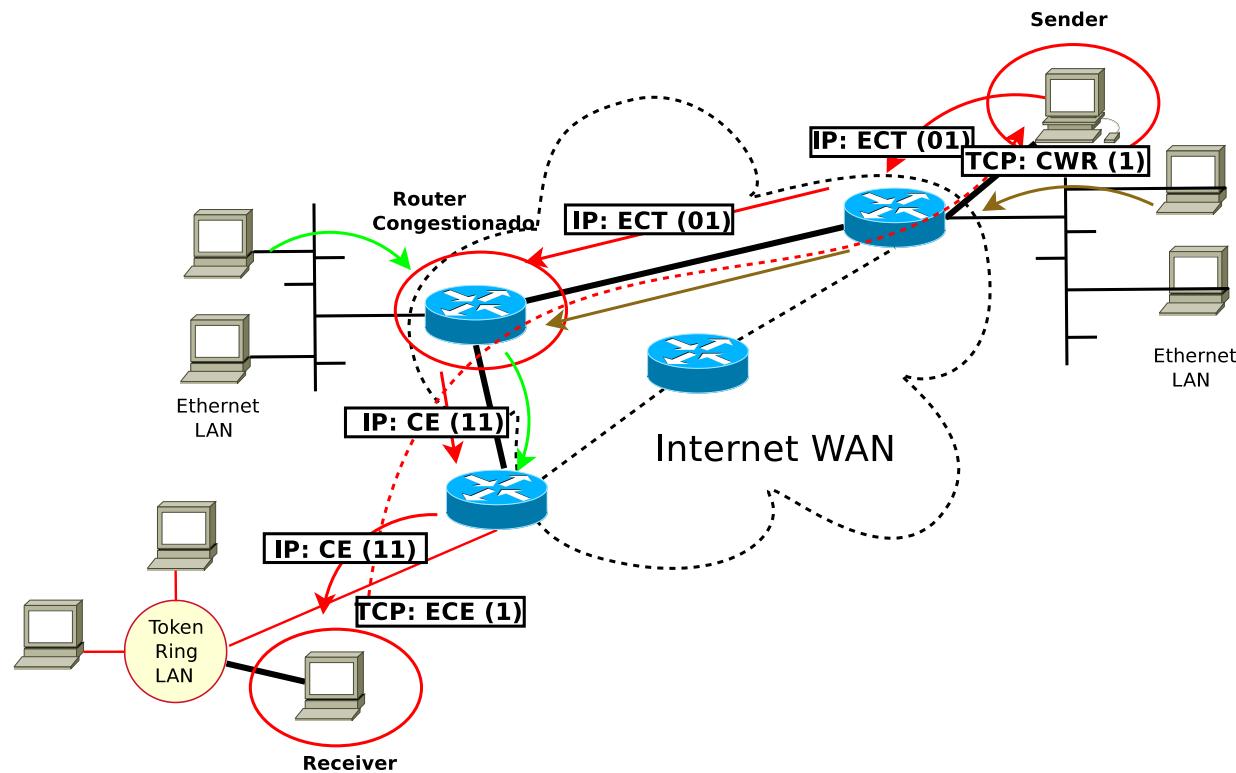
# TCP ECN Setup



## Control de Congestión TCP+IP (Cont'd)

- El receptor al recibir  $CE = 1$ , setea ECE en el próximo segmento de ACK.
- El emisor detecta  $ACK == ECE == 1$  y aplica TCP CC clásico como si el mensaje hubiese sido descartado.
- El emisor también configura  $CWR = 1$  (Congestion Window Reduced) para notificar que reacciono.

## TCP+IP ECT+CE, ECE+CWR



## Fuentes de Información

- Kurose/Ross: Computer Networking (5th Edition).
- TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens.
- TCP/IP Illustrated, Volume 1: The Protocols, 2nd Ed. W. Richard Stevens, Kevin R. Fall.
- A Protocol for Packet Network Intercommunication. Cerf, Vinton G. & Kahn, Robert E. (1974).
- RFC 675, Specification of Internet Transmission Control Protocol, V. Cerf et al. (December 1974).
- RFCs: <http://www.faqs.org/rfcs/> RFC-793, ... RFC-791, RFC-1323, RFC-2001, RFC-2018, RFC-2581, RFC-5681, RFC-2582, RFC-6582, RFC-3168, RFC-3649, RFC-2988, RFC-6298.

- Jaco88: Van Jacobson, “Congestion Avoidance and Control”, ACM SIGCOMM ’88.
- CJ89: Chiu, D. and R. Jain, “Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks”, Journal of Computer Networks and ISDN Systems, vol. 17, no. 1, pp. 1-14, June 1989.
- Jaco90: Van Jacobson, “Modified TCP Congestion Avoidance Algorithm”, email to end2end-interest@ISI.EDU, April 1990.
- FF96: Fall, K. and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", Computer Communication Review, July 1996. <ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z>.
- Wikipedia <http://www.wikipedia.org>.
- Slides del Prof. Luis Marrone del curso Ing. de Tráfico.
- TCP/IP Guide: <http://www.tcpipguide.com/>.

- Transport Layer: <http://people.westminstercollege.edu/faculty/ggagne/spring2007/352/notes/unit4/index.html>
- Internet ...



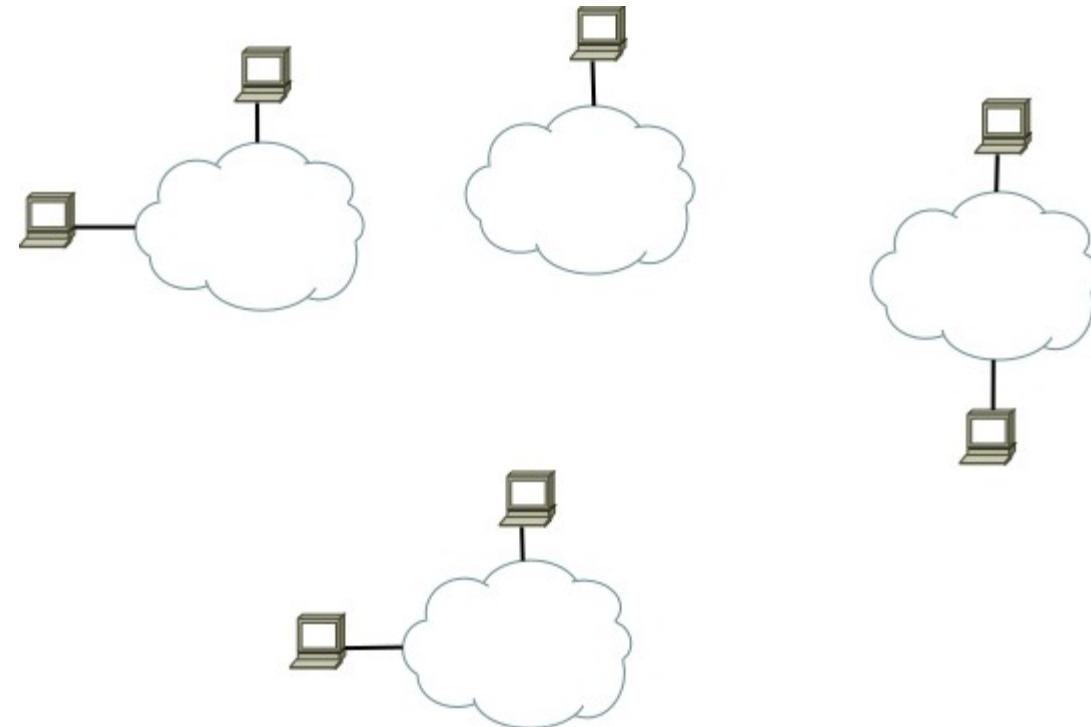
# IP (Internet Protocol)

# Contenido

- Introducción
- Direccionamiento IP
- Protocolo IP, estructura del mensaje
- Ruteo Estático IP
- Fragmentación

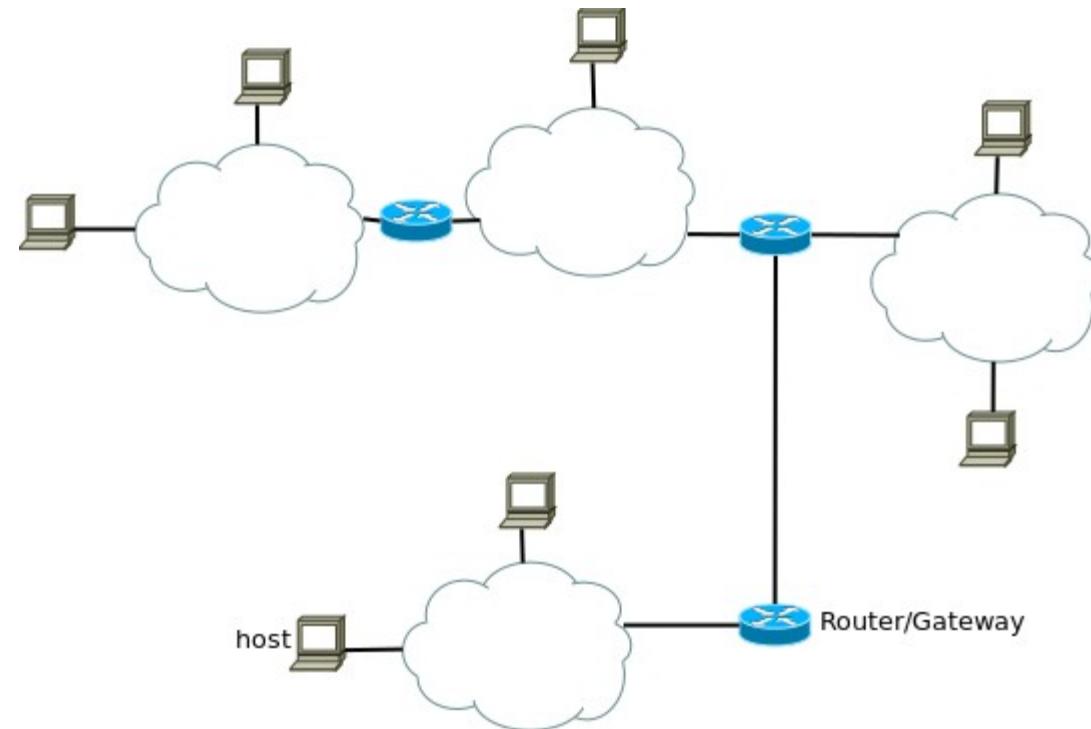
# Estructura de Internet

- Conjunto de redes



# Estructura de Internet

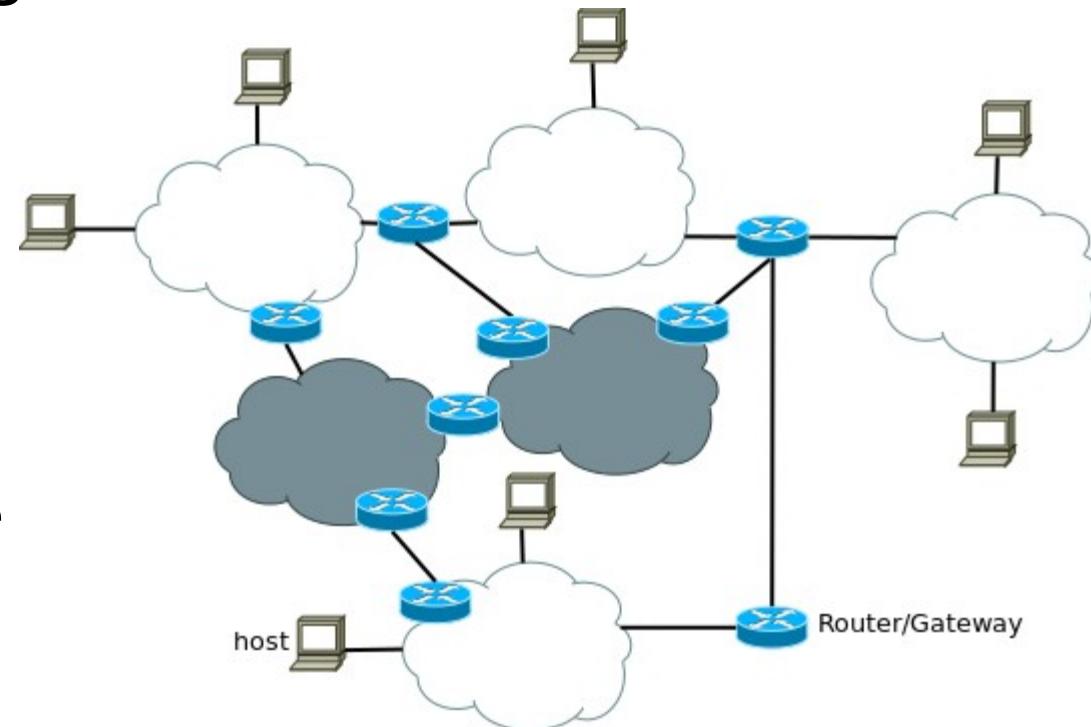
- Conjunto de redes interconectadas



# Estructura de Internet

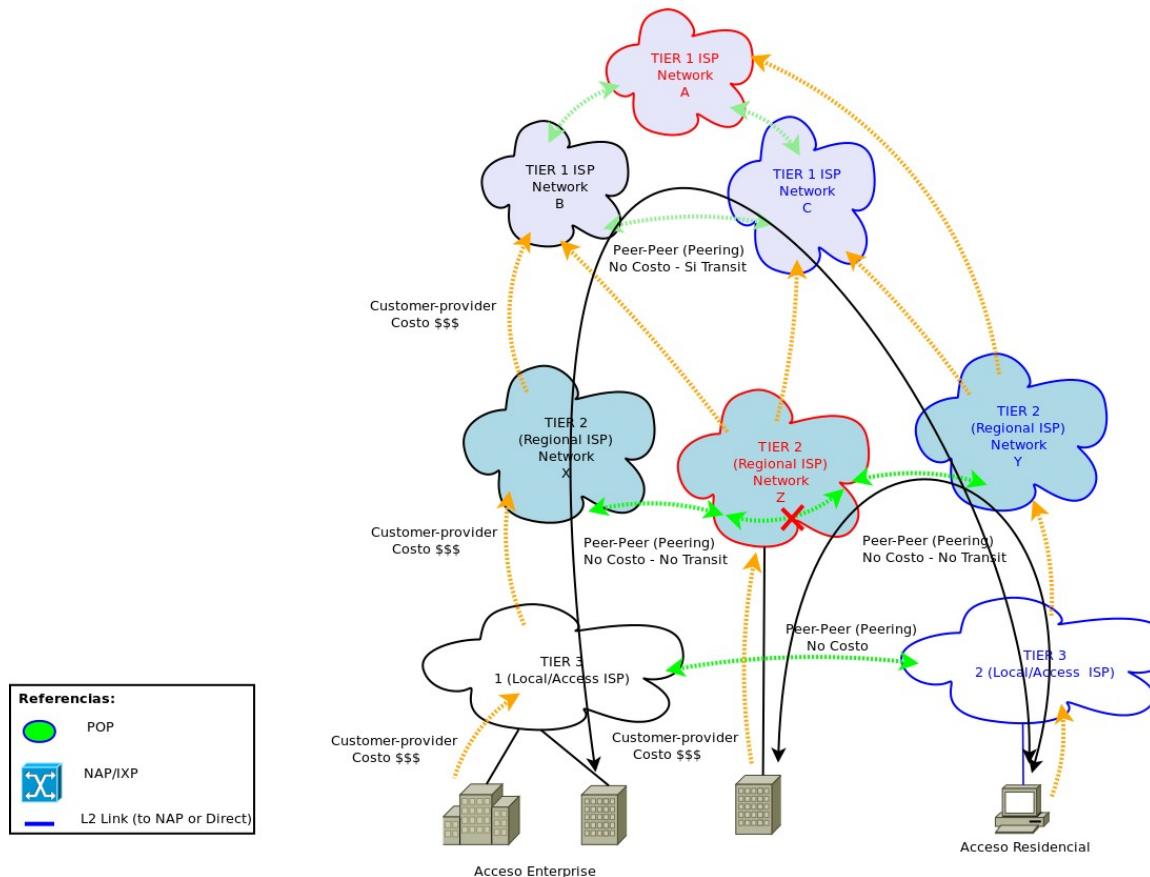
- Conjunto de redes interconectadas y agregadas

- Red de redes
  - Acceso
  - Carrier/Transporte
- Protocolo común:IP



# Estructura de Internet

- Conjunto de redes interconectadas y agregadas de forma “jerárquica”

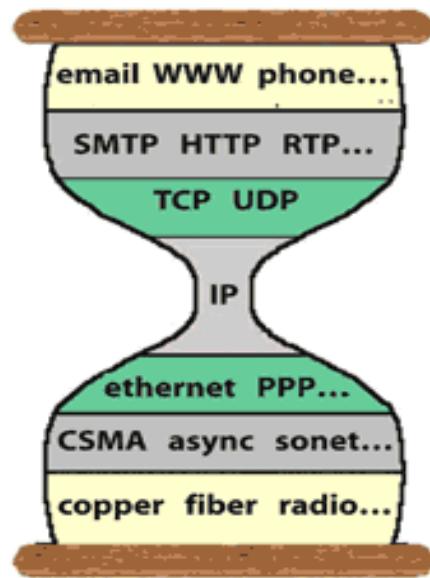


# Estructura de Internet

- Net Neutrality (Neutralidad de la Red)
  - Tratamiento de todo el tráfico de forma equivalente por redes de ISP y carriers.
- Open Internet (Internet abierta)
- Accesibilidad de Internet.

# Modelo de Internet

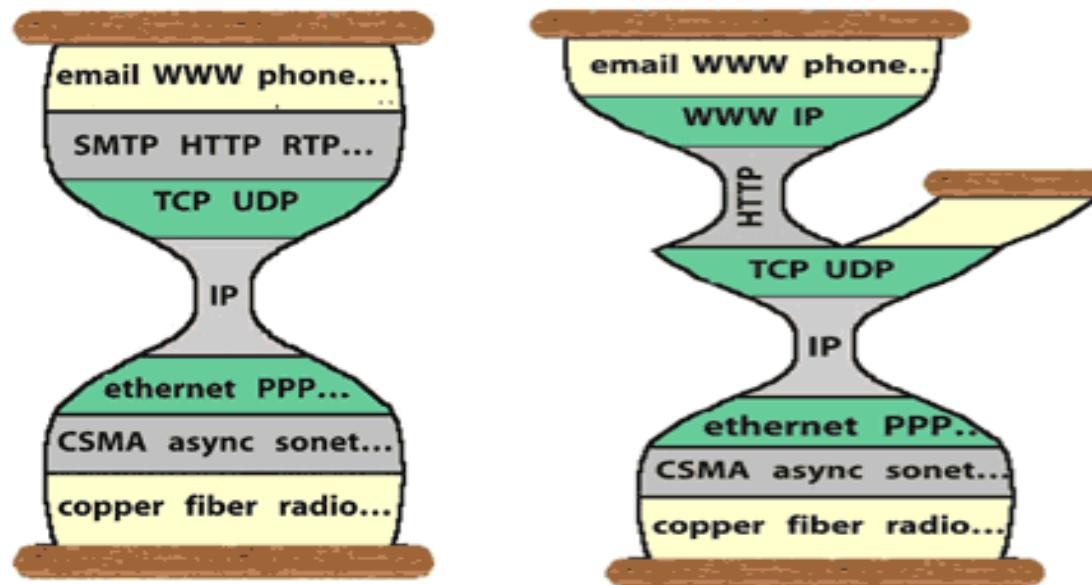
- Modelo de reloj de arena:



Fuente: <http://isoc.org/wp/ietfjournal/?p=454>

# Modelo de Internet

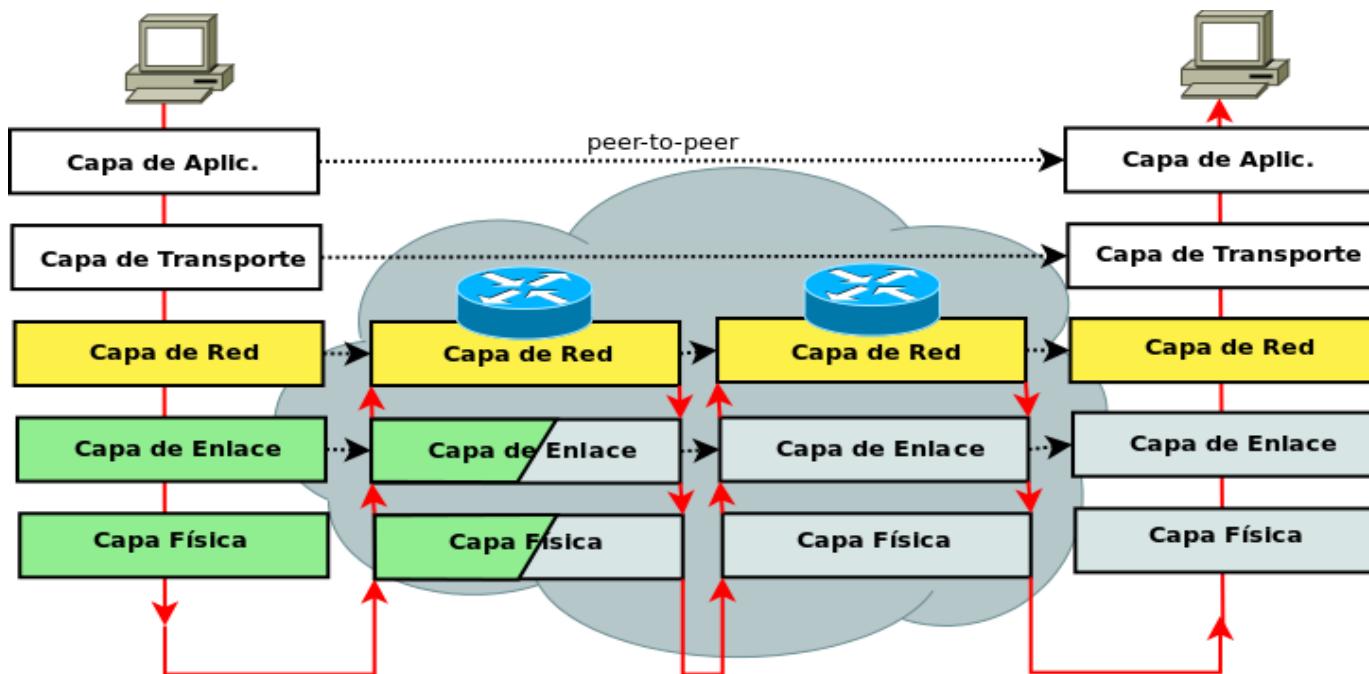
- Modelo de reloj de arena actual ?



Fuente: <http://isoc.org/wp/ietfjournal/?p=454>

# Cómo trabaja IP

- End-to-End (Extremo-a-extremo)
- Ruteo se produce hop-by-hop (salto-a-salto)
- Cada nodo debe implementar IP



# Protocolos IP actuales

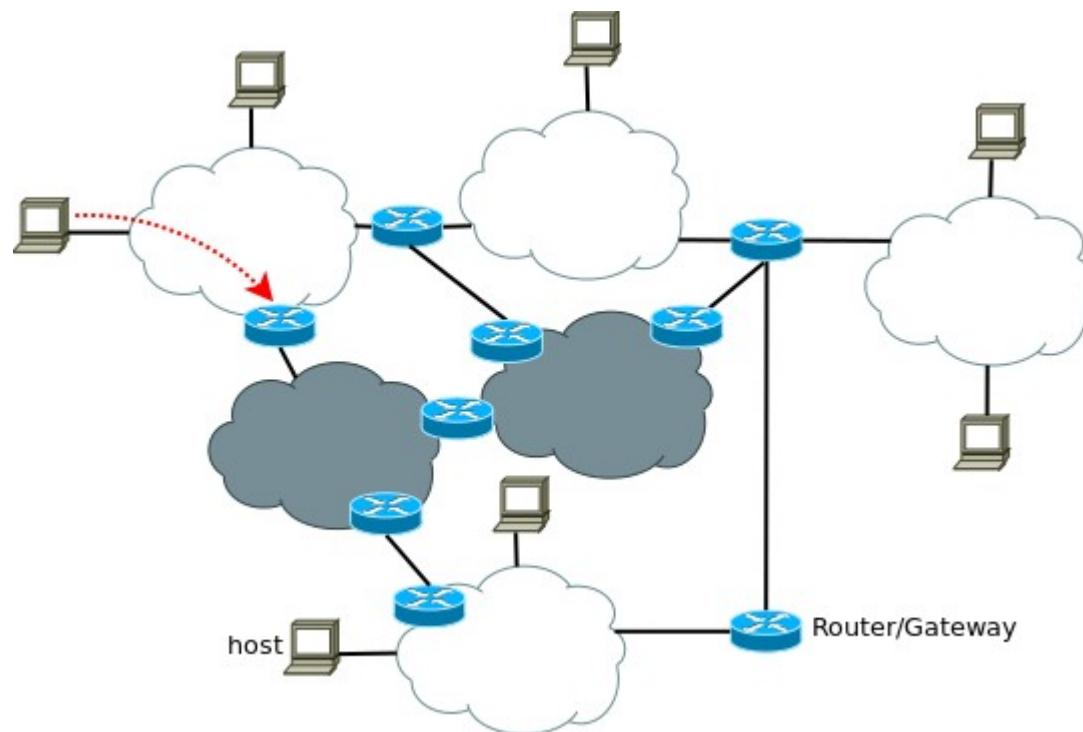
- Brinda servicios a Transporte.
- Usa servicios de Enlace.
- **IPv4**, comúnmente llamado IP.
- IPv6 llamado antiguamente IP-ng.
- No son versiones de uno mismo, no son compatibles.
- En este texto se comenzará a estudiar IPv4.

# Características de IPv4

- Protocolo de Red no orientado a conexión.
- Protocolo de Mejor Esfuerzo: best-effort, no confiable (no asegura el arribo de los mensajes).
- PDU: datagrama o paquete.
- Definido RFC 791 (STD-5).
- Funcionalidad:
  - Direcccionamiento.
  - Ruteo/Forwarding/Switching L3.
  - Mux/Demux de protocolos superiores.
  - Accesorias (Solucionar deficiencias del protocolo)
    - Fragmentación.
    - Otras: como evitar loops (TTL), detección de errores.

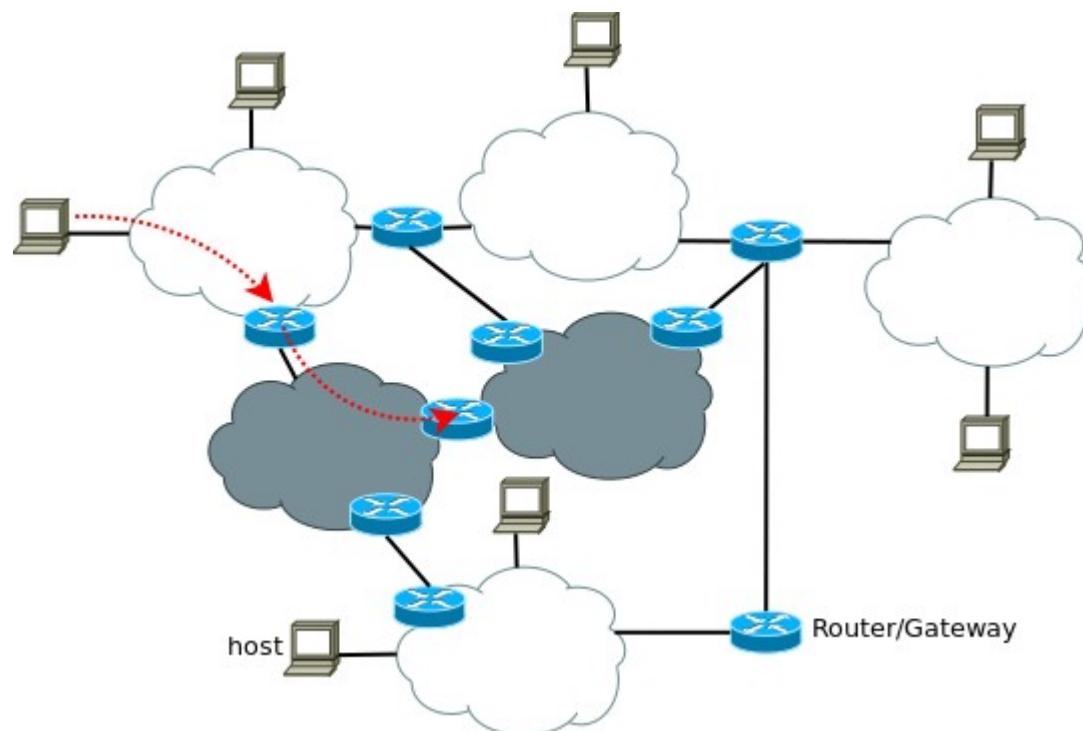
# No orientado a conexión

- Protocolo de Red no orientado a conexión a diferencia de TCP u otros protocolos considerados de red X.25, ATM.



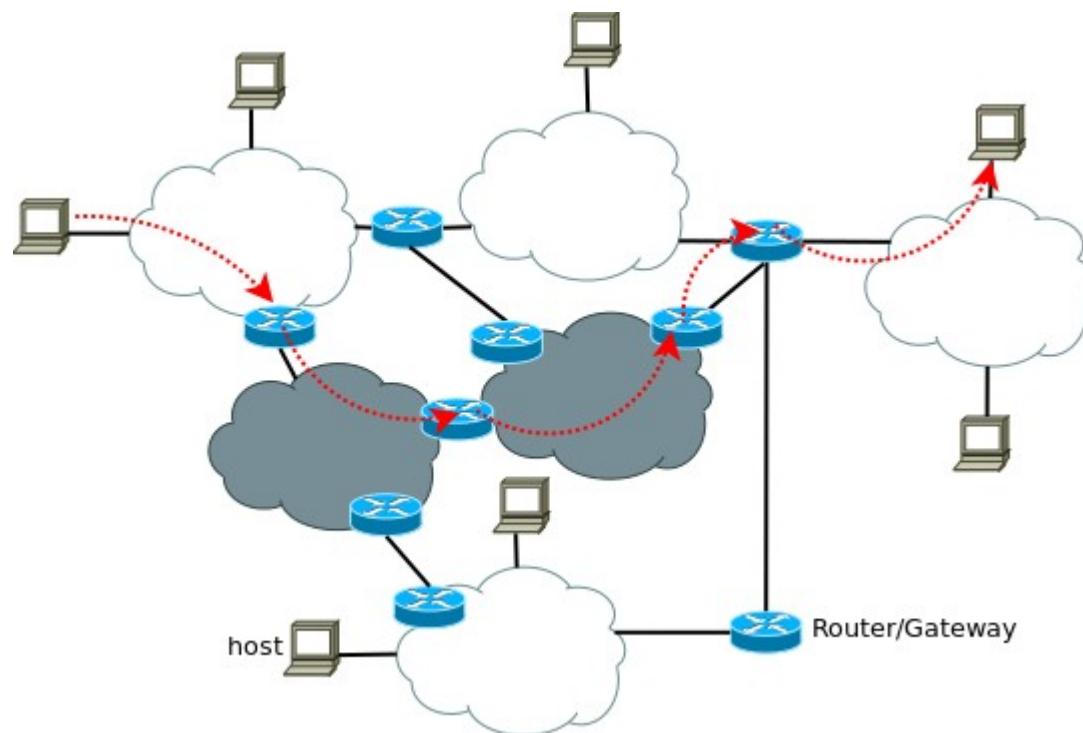
# No orientado a conexión

- Protocolo de Red no orientado a conexión a diferencia de TCP u otros protocolos considerados de red X.25, ATM.



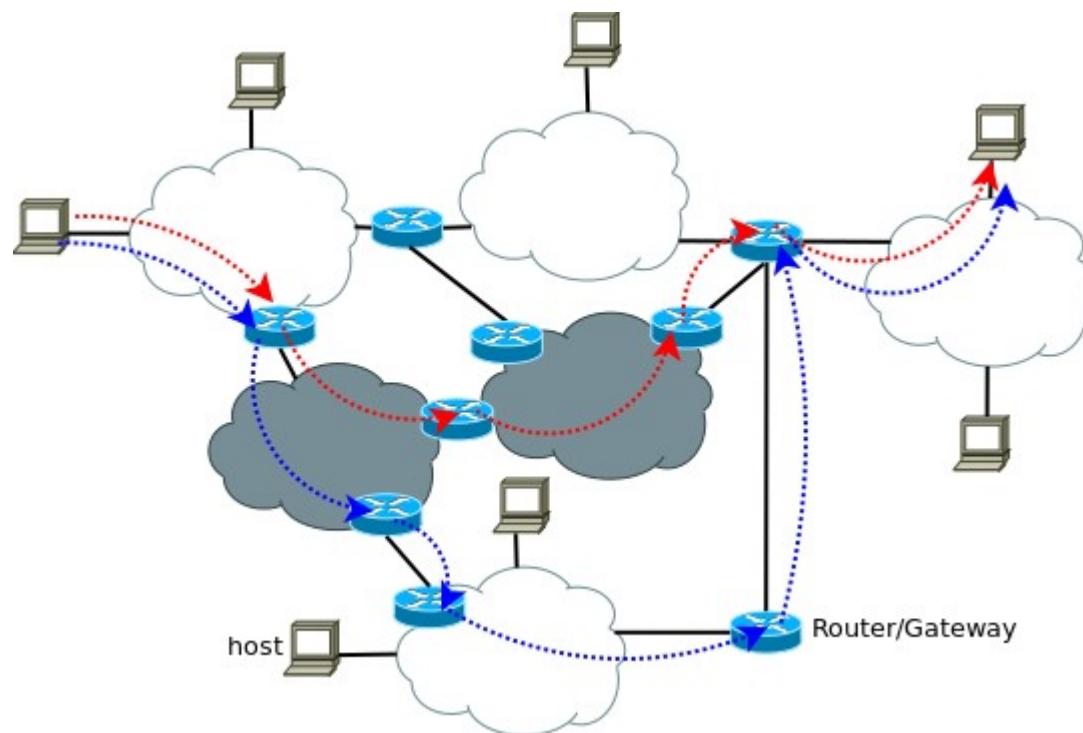
# No orientado a conexión

- Protocolo de Red no orientado a conexión a diferencia de TCP u otros protocolos considerados de red X.25, ATM.



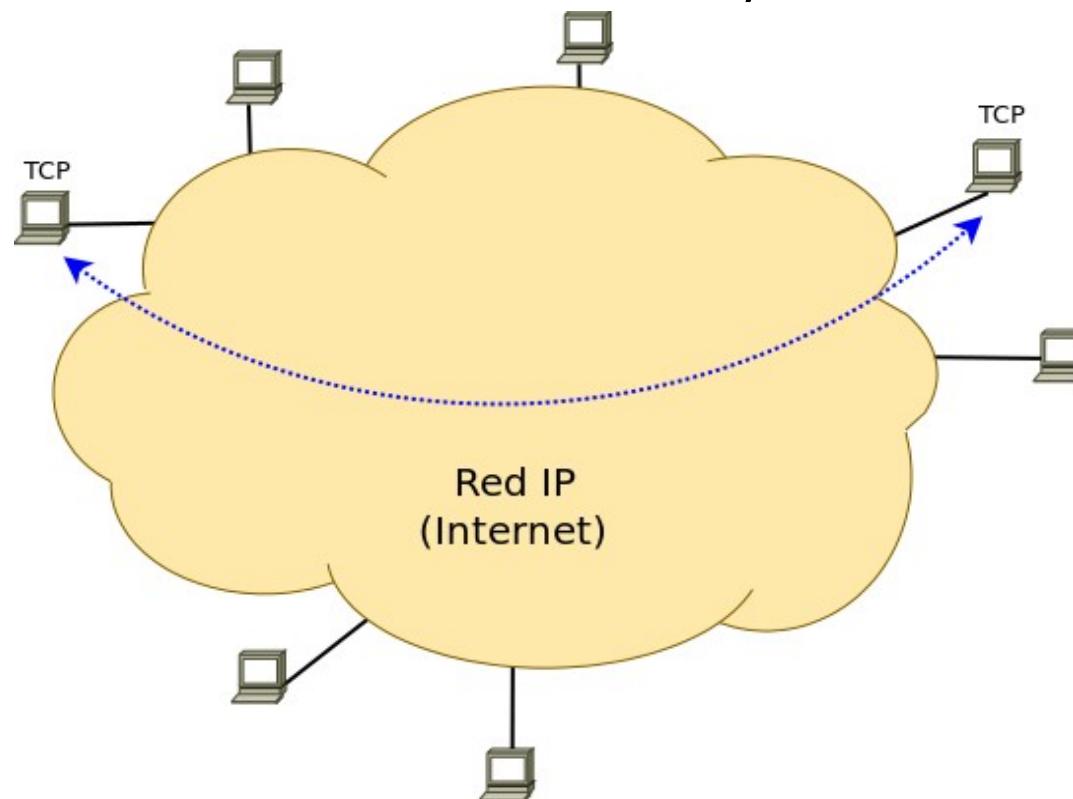
# No orientado a conexión

- Protocolo de Red no orientado a conexión a diferencia de TCP u otros protocolos considerados de red X.25, ATM.



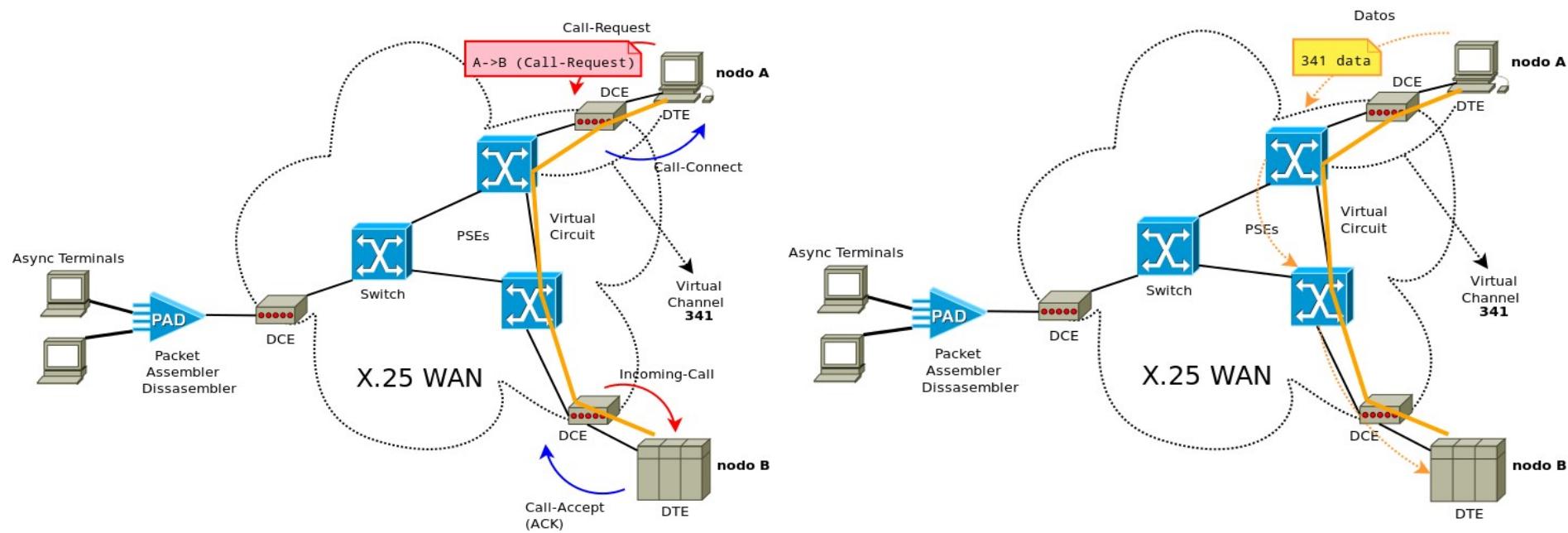
# No orientado a conexión

- Protocolo de Red no orientado a conexión a diferencia de TCP u otros protocolos considerados de red X.25, ATM.

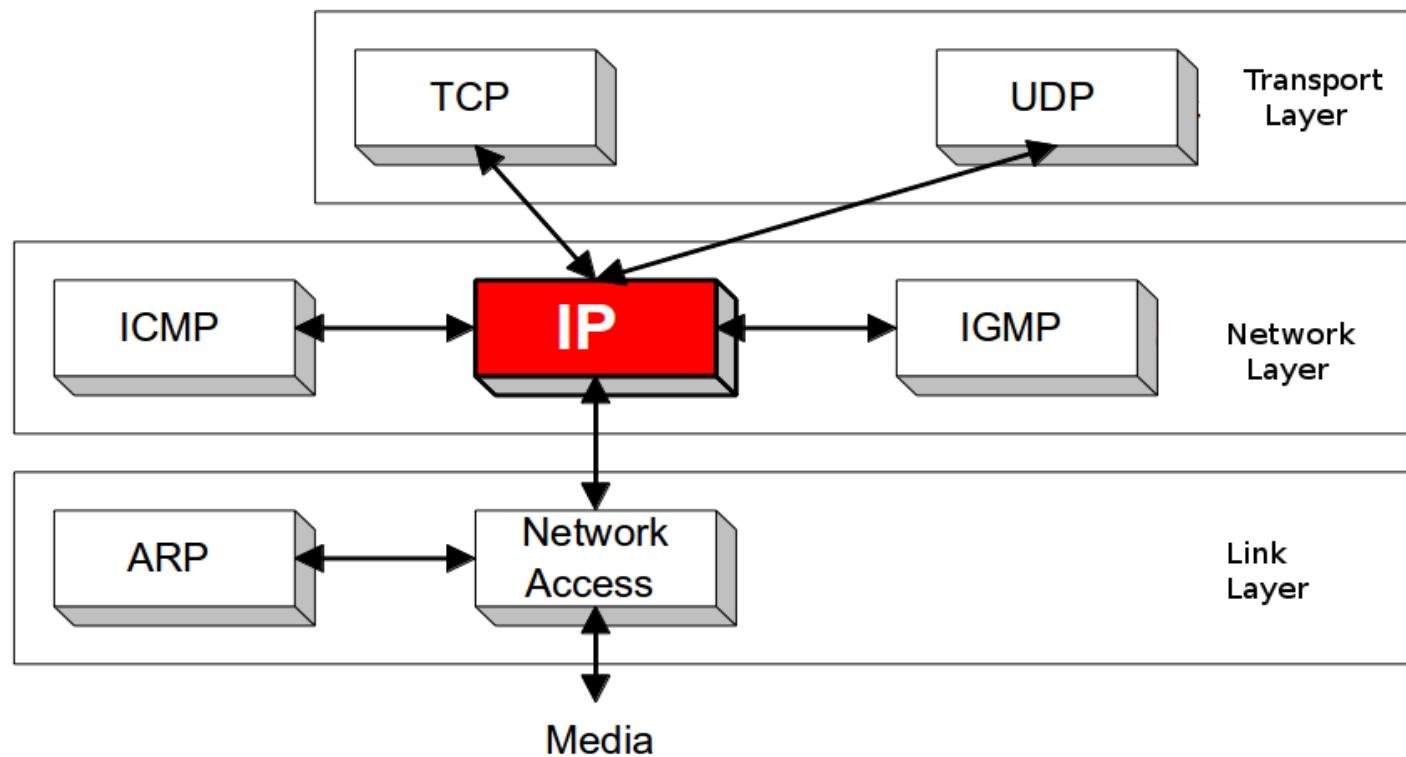


# Versus orientado a conexión

- X.25, ATM usan Virtual Circuits (VC)



# Esquema de IP en TCP/IP



- Es el núcleo de la Internet.
- Requiere protocolos “Helpers”.

# Direccionamiento IP

**Dir IP:** identifica únicamente un punto de acceso (interfaz) a la red.

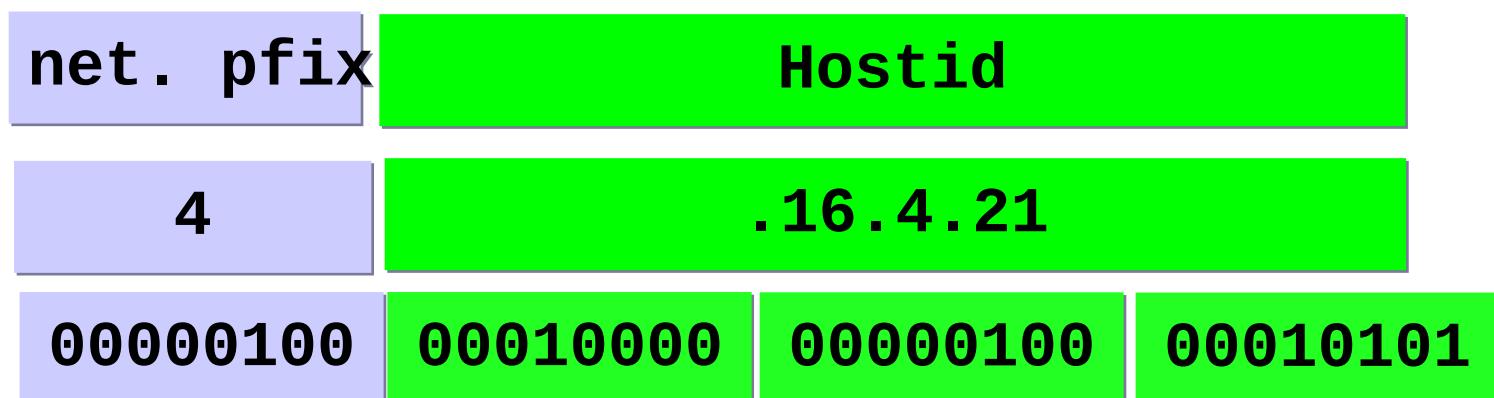
- Un router o un host multi-homed tienen varias IPs. Cada interfaz un valor único.  
Puede tener varias Dir. IP una interfaz.
- Tienen un significado global en la Internet o privado (local).
- Globales: asignadas por autoridad central:
  - Principio: John Postel, InterNIC (Internet Network Information Center).
  - Hoy: el IANA (Internet Assigned Numbers Authority), responsable, el ICANN, delegando la asignación a los RIRs (Regional Internet Registers), siendo para América Latina y parte del Caribe: LACNIC.

# Direcciones IP

- Son números de 32 bits, expresados en notación decimal delimitada por puntos byte a byte (e.g. 163.10.45.77).
- Son 4G de direcciones ( $2^{32}$ ) puras, que organizadas en forma jerárquica se reducen.
- Para facilidad de los usuarios, mapping con nombres de domino (DNS - Domain Name Server).
- Son necesarias para rutear la información por la Internet.
- Son direcciones lógicas.

# Direcciones IP

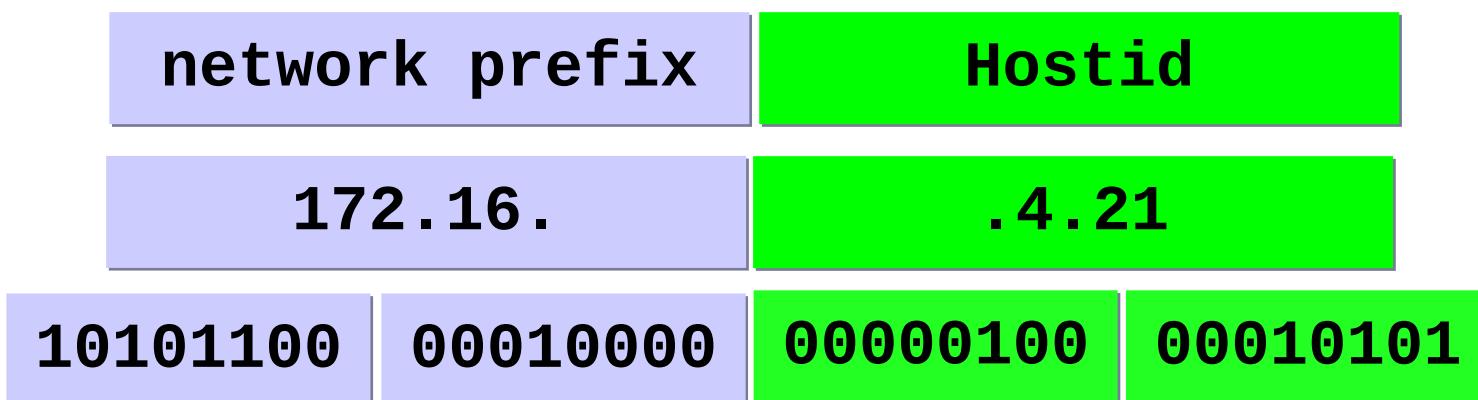
- Codificadas en dos partes:
  - Red (Net).
  - Anfitrión (Host).



- Hasta 1981, solo había pocas redes con mucho hosts disponibles. Sin clases. Redes 8 bits. (1979 RFC-758).
- En 1981: RFC-790 define **clases**.

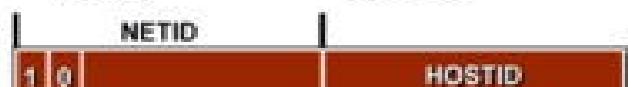
# Direcciones IP

- Cada Clase para diferentes tipos de redes:
  - Clases A, pocas redes grandes.
  - Clases B, más redes medianas.
  - Clases C, muchas redes chicas.



- En 1984 se agrega una tercer parte, **subred** y se requiere un máscara: RFC-917.

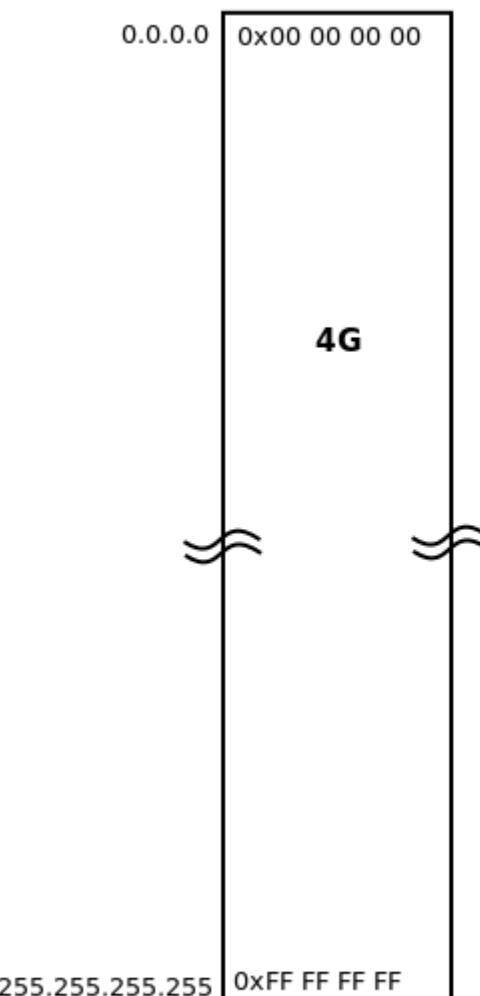
# Clases de Direcciones IP

Class	First Octet Range	Max Hosts	Format
A	1-126	16M	 1 Octet                            3 Octets
B	128-191	64K	 2 Octets                            2 Octets
C	192-223	254	 3 Octets                            1 Octet
D	224-239	N/A	 Multicast Address
E	240-255	N/A	 Experimental

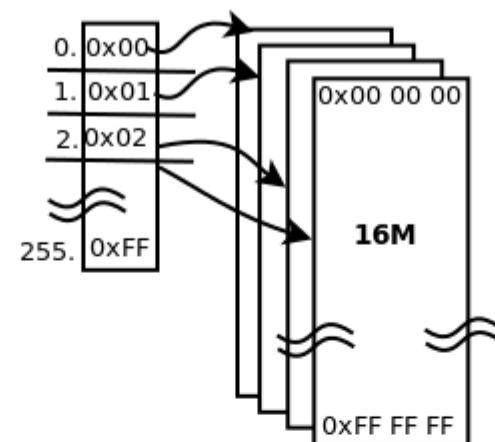
- Definidas en , RFC-790 (Assigned Numbers), RFC-796 (Address mapping).



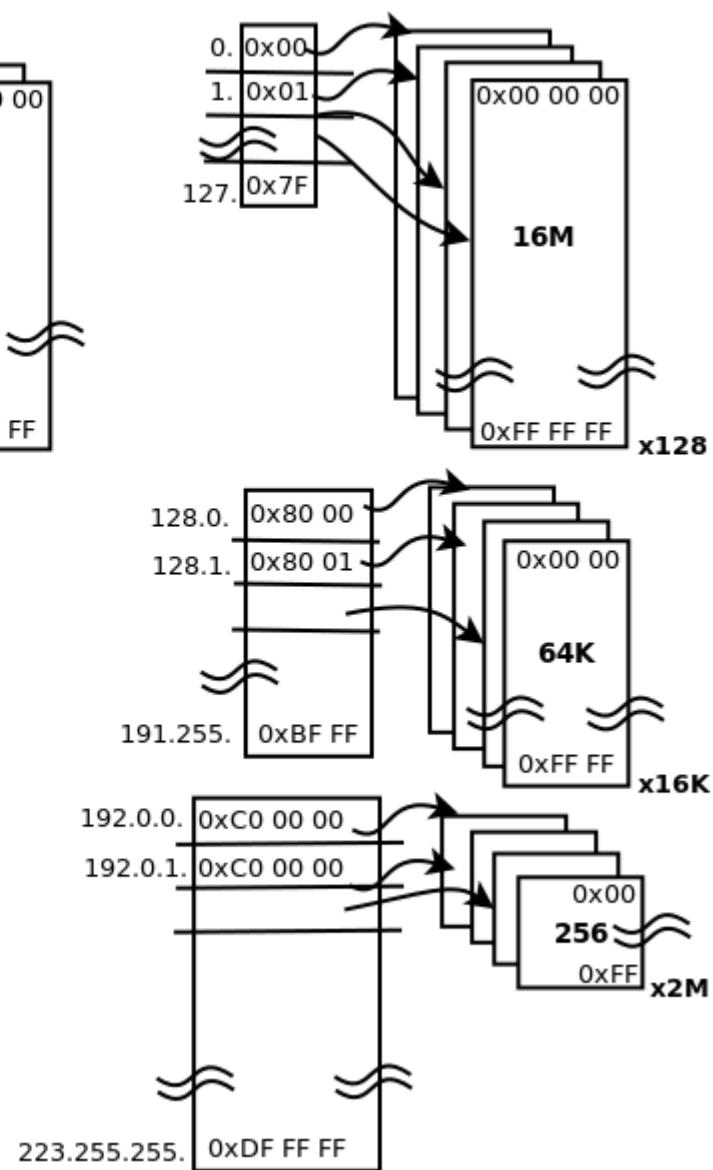
Esquema Plano



Esquema Jerárquico  
1 sola clase



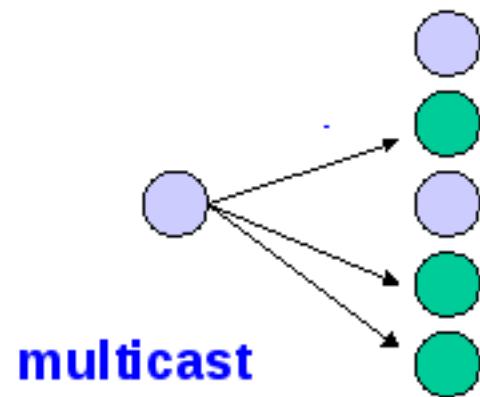
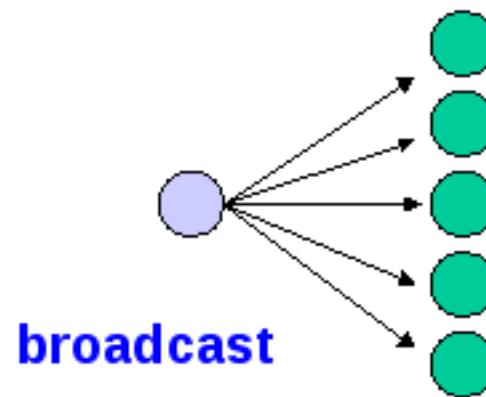
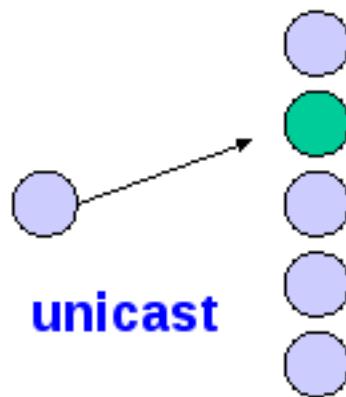
Esquema Jerárquico  
3 clases, dif. tamaño



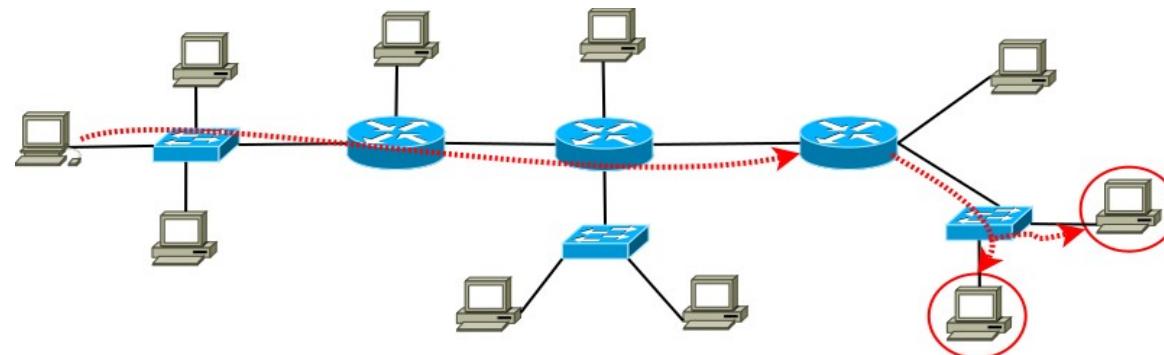
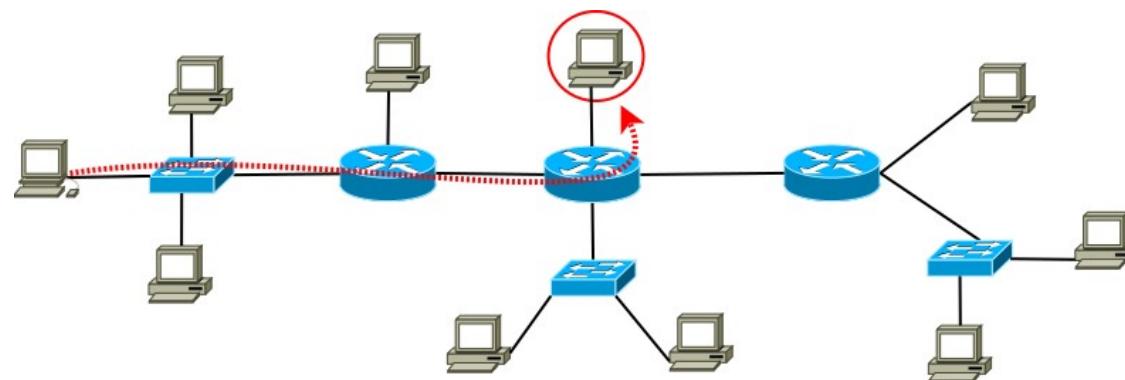
# Tipos de Direcciones IP

- **Unicast**: destino a un host/interfaz en particular, son las más comunes.
  - e.g: 172.16.4.21
- **Broadcast**: destino a todos los hosts en una red.
- **Multicast**: destinada a un grupo de hosts en una red o varias redes. Clase D.
- **Anycast**: destinada al primero que resuelva. IPv4 no hay casos especiales.

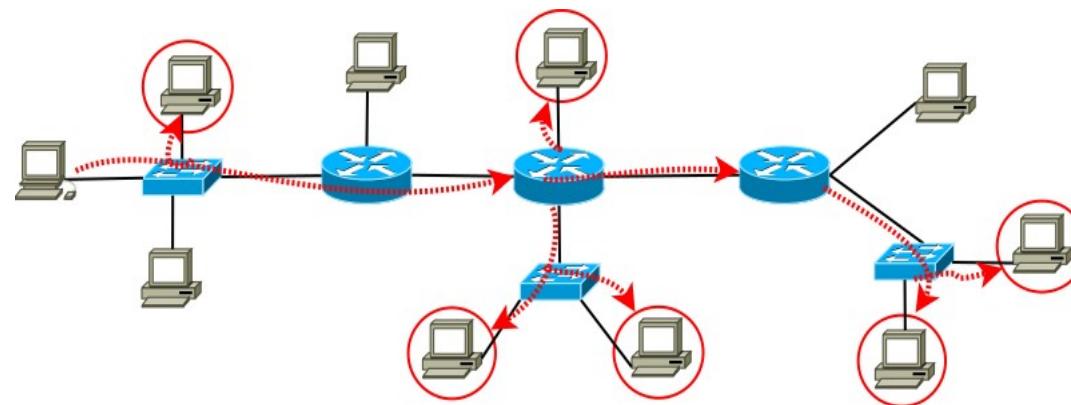
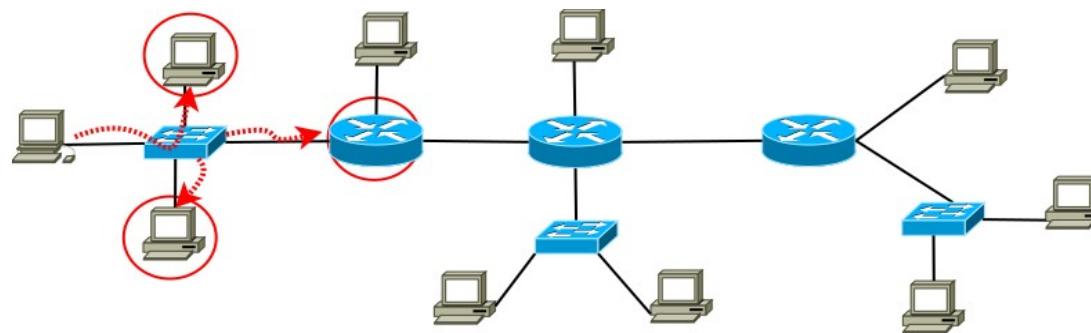
# Tipos de Direcciones IP



# Tipos de Direcciones IP



# Tipos de Direcciones IP



# Direcciones IP especiales

- loopback: unicast, red clase A. 127.0.0.1
  - La más utilizada: 127.0.0.1, localhost.
  - Aunque podría ser cualquier otra:
    - 127.10.0.1
    - 127.34.34.1, etc.
- Dirección de red: la primera (zero).
  - e.g. 172.16.0.0, 192.168.1.0.
- Dirección de broadcast:
  - Directed Broadcast: la última (ones).
  - e.g. 172.16.255.255, 192.168.1.255.

# Direcciones IP especiales

- Limited Broadcast: (all ones).
- 255.255.255.255.
- “Este host”, cuando aún no tiene asignada una dirección:
  - 0.0.0.0  
(Utilizada en BOOTP/DHCP)

# Direcciones Privadas

- No tienen significado global no son únicas.
- Definidas en RFC-1918.
- Se utilizan en Intranets. Redes autónomas sin conexión a Internet.
- Para conectar a Internet requieren un proceso de transformación: NAT, RFC-1631.
- No deberían pasar a la Internet. Filtradas por routers de borde.
  - 10.0.0.0 – 10.255.255.255, 1 Clase A.
  - 172.16.0.0 – 172.31.255.255, 16 Clases B.
  - 192.168.0.0 – 192.168.255.255, 256 Clases C.

# Direccionamiento Fijo (Ejemplo)

## ■ 4 Redes físicas, requieren 4 redes IP:

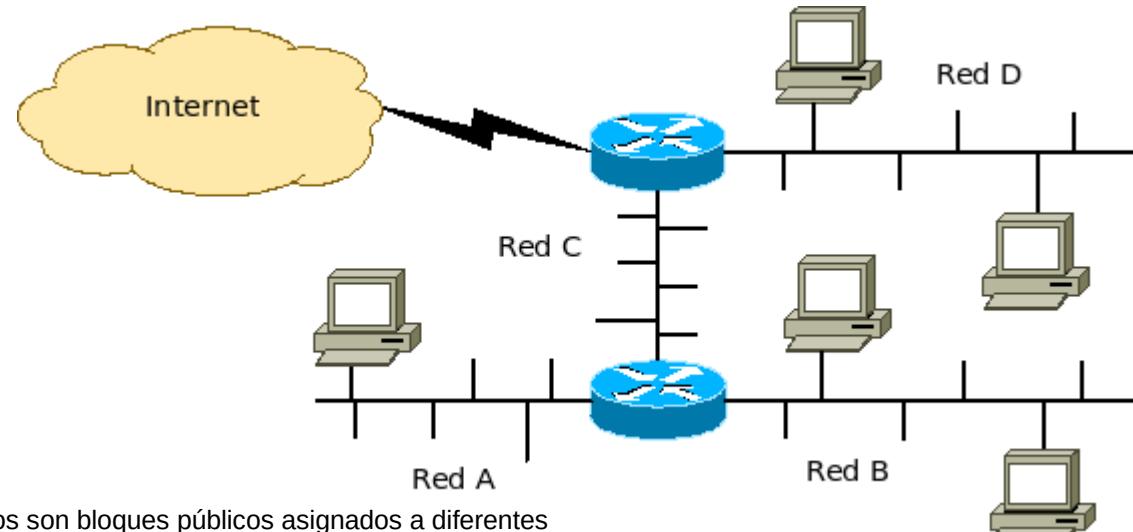
- Si cada red menos de 254 hosts, por ejemplo 25 c/red.  
se pueden utilizar 4 clases C:

Red A: 193.168.1.0\*

Red B: 193.168.2.0

Red C: 193.168.3.0

Red D: 193.168.4.0



\* Los bloques utilizados en los ejemplos son bloques públicos asignados a diferentes Instituciones por RIPE. Solo se muestran a modo ilustrativo.

# Problemas con Dir. IP Fijo

- Prefijos de longitud fija por clase, provoca un uso ineficiente en el espacio de direcciones.
- Muchos equipos, produce escasez de direcciones.
- Crecimiento acelerado de la Internet, evidencia la falta de escalabilidad del esquema. Crecimiento de tablas de ruteo en el núcleo de la red.
- Codificar la red en la dirección IP implica que si un host cambia de red, cambiará su dirección (IP Mobility). Problema atacado en IPv4, mejor resuelto en IPv6.
- Soluciones IPv4: subnetting, CIDR, NAT, DHCP.
- Definitivamente solucionados en IPv6.

# Subnetting IP

- Se toma una parte del **hostid**.
- Se utiliza para generar redes dentro de la red.
- Se agrega una “máscara” de bits.
- Para saber la subred se aplica un “AND” lógico.

network prefix	Subnet	Hostid
172.16.	.4	.21
10101100	00010000	00000100
11111111	11111111	11111111
172.16.4.	.0	

# Subnetting IP

- En 1984 se agrega una tercer parte y se requiere una “máscara” de subred: RFC-917, RFC-940, RFC-950.
- Agregar un nivel más en la estructura:
  - Red, Subred, Host.
  - Ejemplo usar un bloque clase B como 256 clases C:

network prefix	Subnet	Hostid
172.16 .	.4	.21
10101100	00010000	00000100
11111111	11111111	11111111
		00000000

# Subnetting

- Las máscaras se escriben en notación decimal o hex.
  - 255.255.255.0 o 0xff ff ff 00.
- También pueden escribirse como longitud de prefijo: /24.
- Otros ejemplos:
  - 255.255.255.192 /26.
  - 255.224.0.0 /11.
  - 255.255.255.252 /30.
- Las máscaras defaults:
  - Clase A: 255.0.0.0.
  - Clase B: 255.255.0.0.
  - Clase C: 255.255.255.0.

# Subnetting

- Valen los mismos conceptos para redes completas.
- Ejemplo para 172.16.4.21:
  - Dirección de broadcast: 172.16.4.255.
  - Dirección de red: 172.16.4.0.
  - Redes y hosts:  $(2^n)$ ,  $(2^{(32-(m+n))})$ .
  - Ejemplo Clase B con /24: n=8, m=16.
    - Cantidad de hosts:  $(2^8)$ .
    - Cantidad de hosts útiles:  $(2^8)-2$ .
    - Cantidad de subredes:  $(2^8)$ .
    - Cantidad de subredes útiles:  $(2^8)-2$ .
    - Las 2 que se restan a las subredes se pueden utilizar: dando:  $2^8$  redes útiles.

# Ejemplo Subnetting Fijo

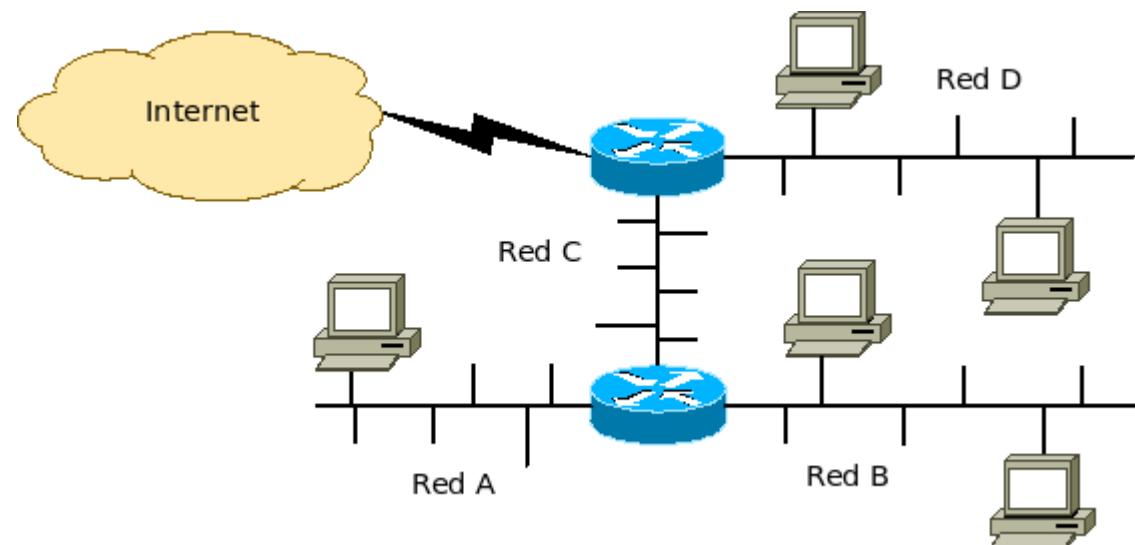
- Si cada red menos de 254 hosts, por ejemplo 25 c/red. Se pueden utilizar 1 clase C dividida en 4:

Red A: 193.168.4.0    255.255.255.192 o /26

Red B: 193.168.4.64    “

Red C: 193.168.4.128    “

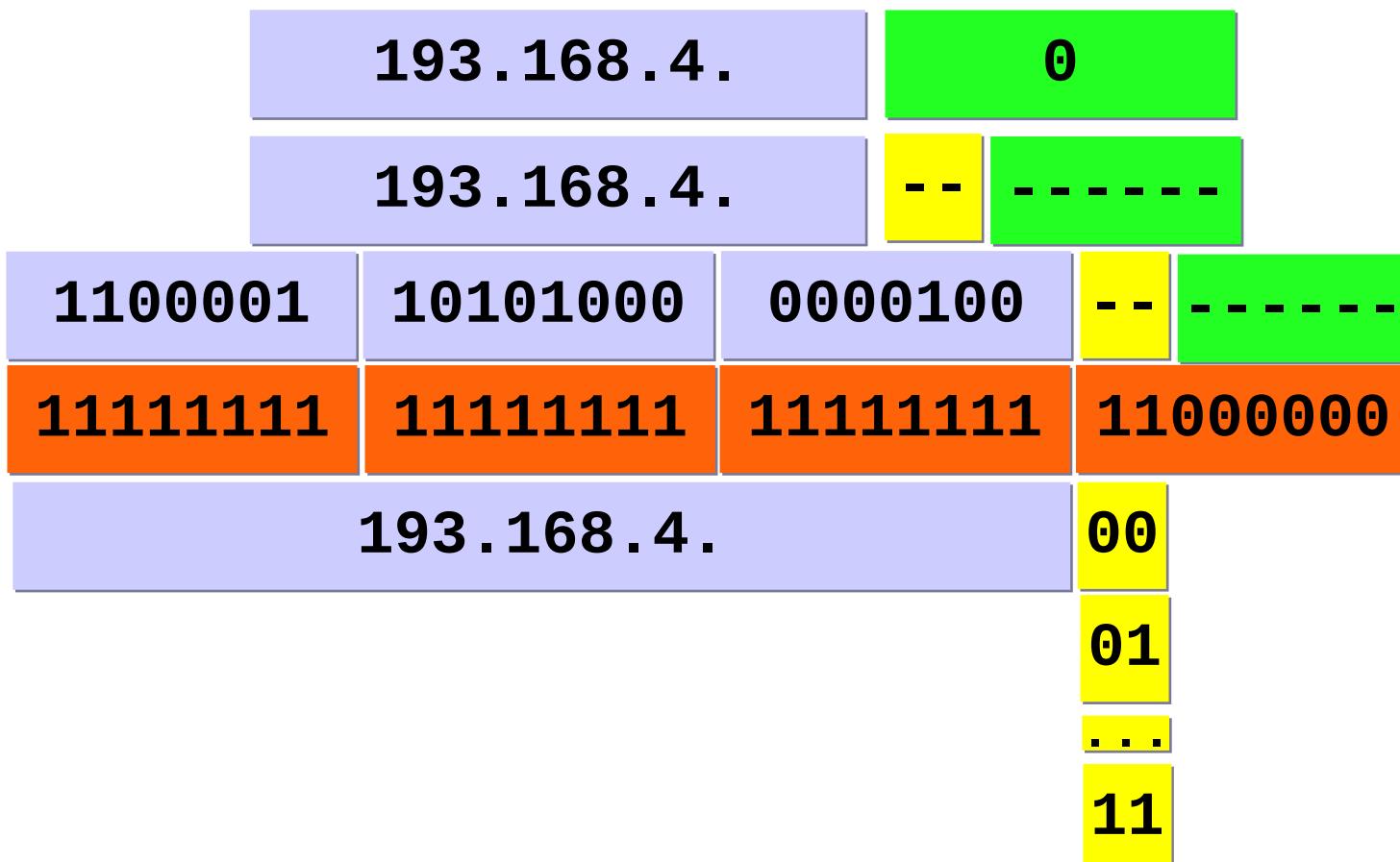
Red D: 193.168.4.192    “



# Ejemplo Subnetting Fijo

- 4 Redes físicas, pueden direccionarse con una red IP:

- 4 redes requieren 2 bits  $2^2 = 4$ .
- Si fuesen 6, se requieren  $2^3 = 8 \sim 6$ . Siempre potencias de 2.



# Subnetting Fijo

- En un principio, por cuestiones de compatibilidad con viejos sistemas no se permitía utilizar la primera ni la última sub-red:

Red A: 193.168.4.0      Dir. de sub-red = Dir. De red.

Red D: 193.168.4.192      Dir. de “sub-bcast” = Dir de bcast de red.

- Se genera mucho desperdicio, en este caso el 50% de las direcciones.
- RFCs subsiguientes lo permiten, hoy completamente difundido.

# VLSM Subnetting

- Variable Length Subnet Mask. RFC-1009, RFC-1878.
- La longitud de la máscara no tiene necesidad de ser para todas las subredes igual.
  - 193.168.4.0 /26
    - /26 00 193.168.4.0,
    - /26 01 193.168.4.64,
    - /26 10 193.168.4.128,
    - /26 11 193.168.4.192. 62 hosts max. c/u.
- ¿Qué sucede si se tienen diferentes cantidades de hosts en las subredes?
  - Por ejemplo la Red A: tiene 70 hosts, la Red B tiene 40 host y la C,D tienen 25 hosts.

# VLSM Subnetting

- El siguiente esquema no sirve:
  - 193.168.4.0 /26
    - /26 00 193.168.4.0,
    - /26 01 193.168.4.64,
    - /26 10 193.168.4.128,
    - /26 11 193.168.4.192. 62 host c/u.
- Se deben agrupar o dividir redes del esquema fijo:
  - /25 000 193.168.4.0,
  - /26 100 193.168.4.128,
  - /27 110 193.168.4.192,
  - /27 111 193.168.4.224. 126, 62, 30, 30 hosts respectivamente.

# VLSM Subnetting

- Subredes iguales: /26

255.255.255.192

255.255.255.192

255.255.255.192

255.255.255.192

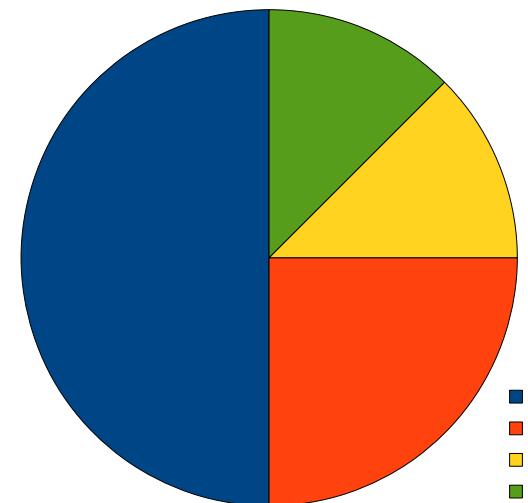
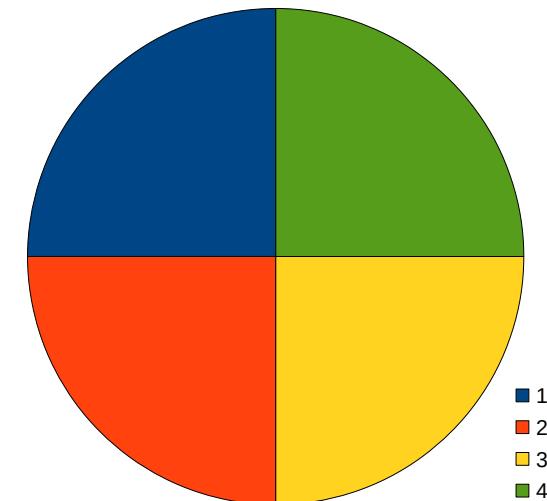
- VLSM: /25, /26, /27, /27:

255.255.255.128

255.255.255.192

255.255.255.224

255.255.255.224



# VLSM Subnetting

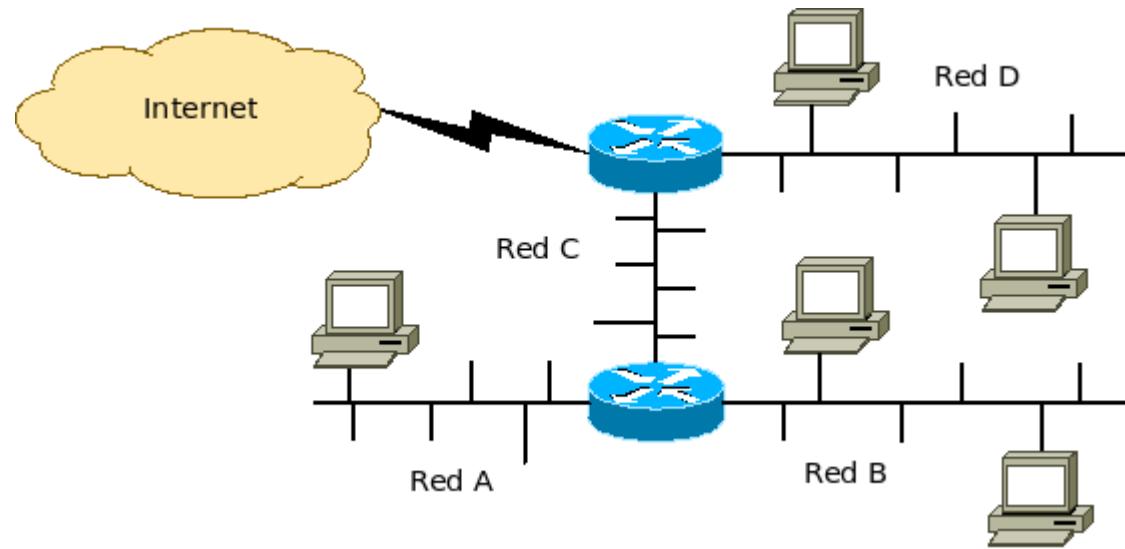
- VLSM: /25, /26, /27, /27:

Red A: 255.255.255.128

Red B: 255.255.255.192

Red C: 255.255.255.224

Red D: 255.255.255.224

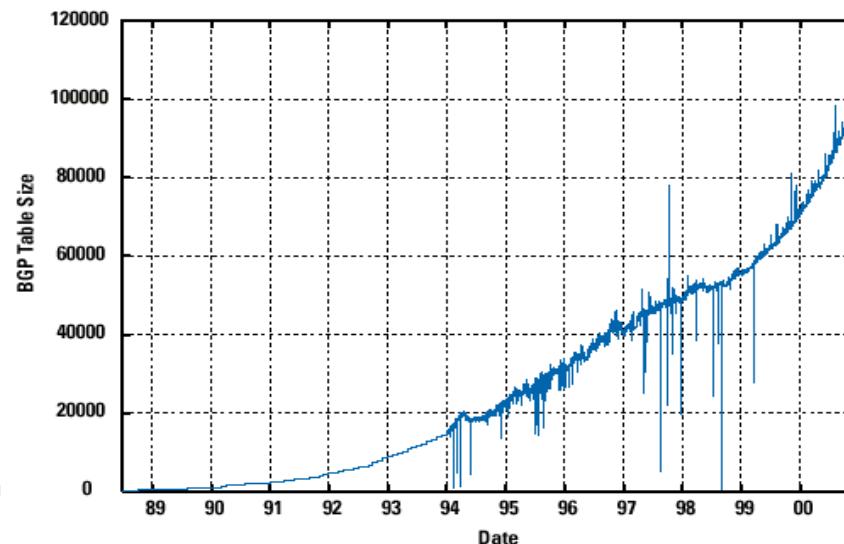


# CIDR - Supernetting

- Classless Inter Domain Routing.
- Hasta 1993, se asumía, de acuerdo a la clase de la Dir. IP la máscara default.
- Los bits de la red definida por la clase eran fijos.
- El direccionamiento era Classful.
- Con CIDR, se sacan las clases: Classless y siempre debe haber una máscara o long. de Pref.
- RFC-1338, RFC-1517, RFC-1518, RFC-1519.
- Permite agrupar, reducen long. tablas de ruteo:
  - e.g. 193.168.0.0, 193.168.1.0, ... 193.168.3.0 /24
  - En 193.168.0.0/22 se agrupan 4 redes.

# CIDR - Supernetting

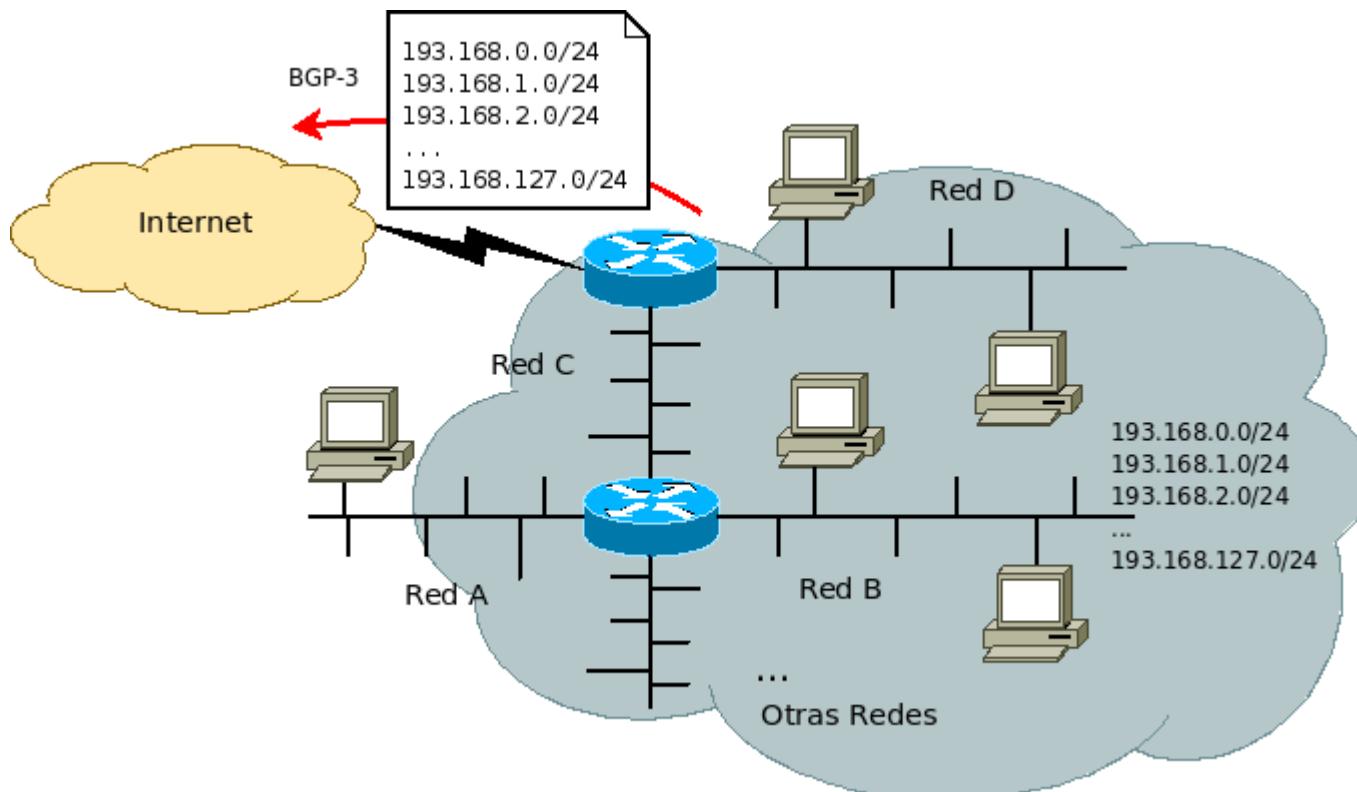
- En 1992 la RFC-1338 marca los problemas del crecimiento de las tablas de ruteo:
  - Las clases A y B el 50% asignadas, clases C solo el 2%.
  - Las clases C:  $2^{21}$  redes aumentarían las tablas de ruteo notablemente.
  - Crecimiento de 1988 a 2000 de tablas de ruteo:



Fuente: <http://www.cisco.com>

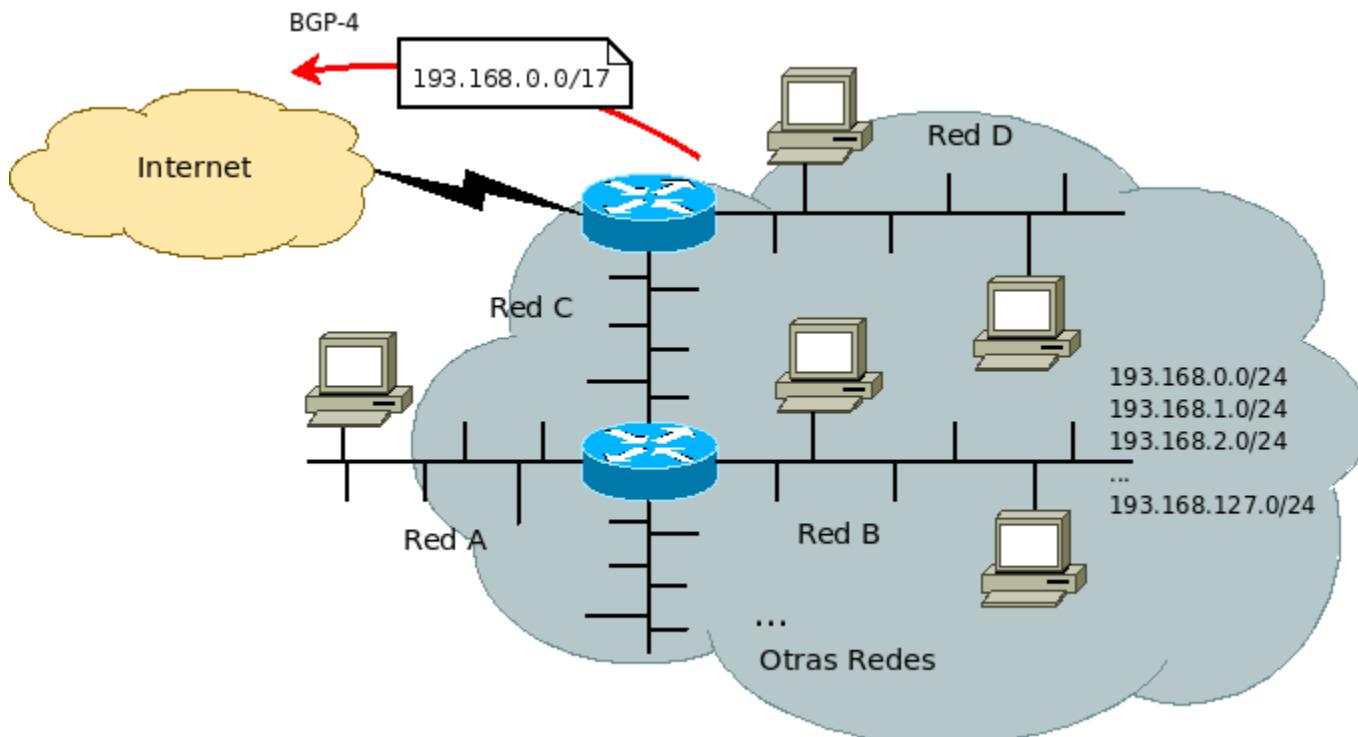
# CIDR - Supernetting

- BGP update classful:



# CIDR - Supernetting

- BGP update classless:

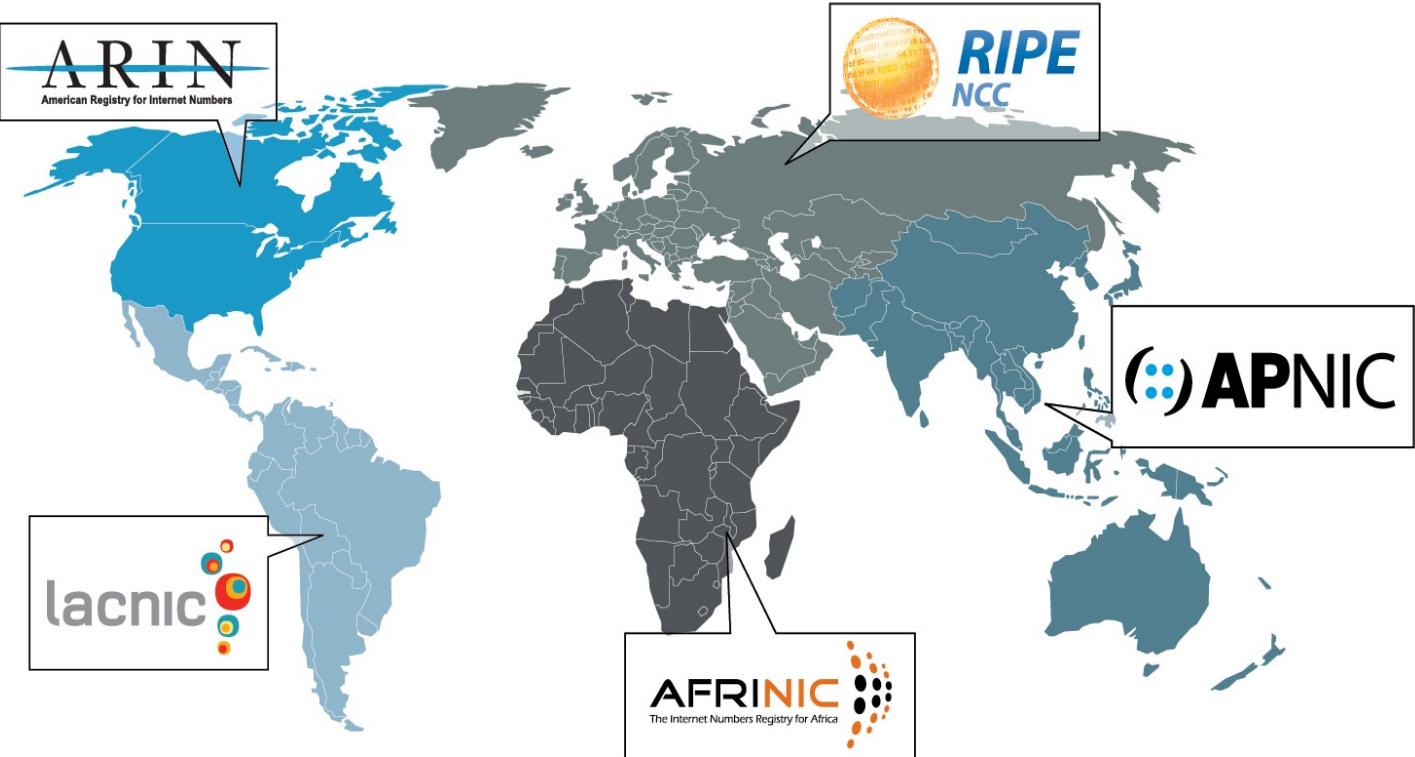


# CIDR - Supernetting

- Agrupación de bloques de forma contigua por ISP.
- Asignación por regiones geográficas.
- El IANA crea los RIRs (Regional Internet Registers):
  - RIPE (Europa)
  - ARIN (Estados Unidos)
  - LACNIC (América Latina y Caribe)
  - APNIC (Asia y Pacífico)
  - AfriNIC (africa)

# CIDR - Supernetting

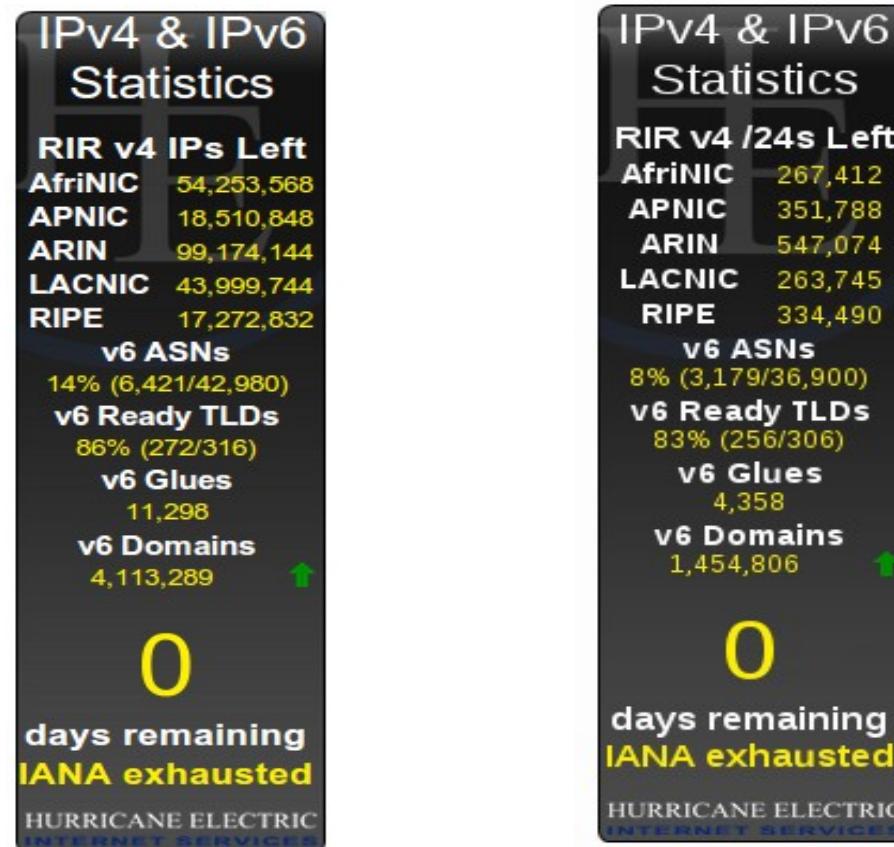
- RIRs:



Fuente: <http://www.arin.net>

# CIDR - Supernetting

- Evolución  
(2013-2014):



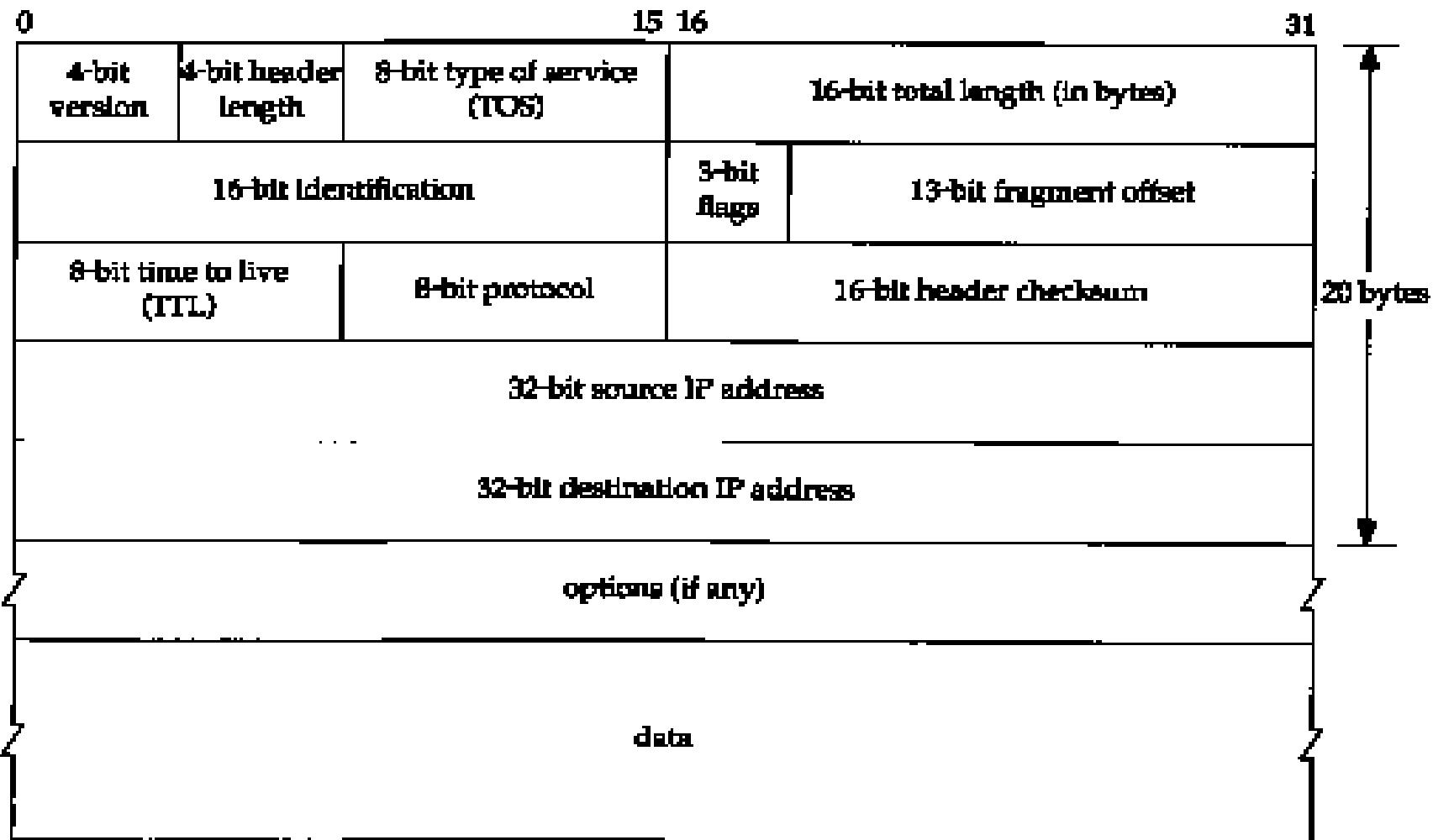
Fuente:<http://ipv6.he.net/statistics/>

# CIDR - Supernetting

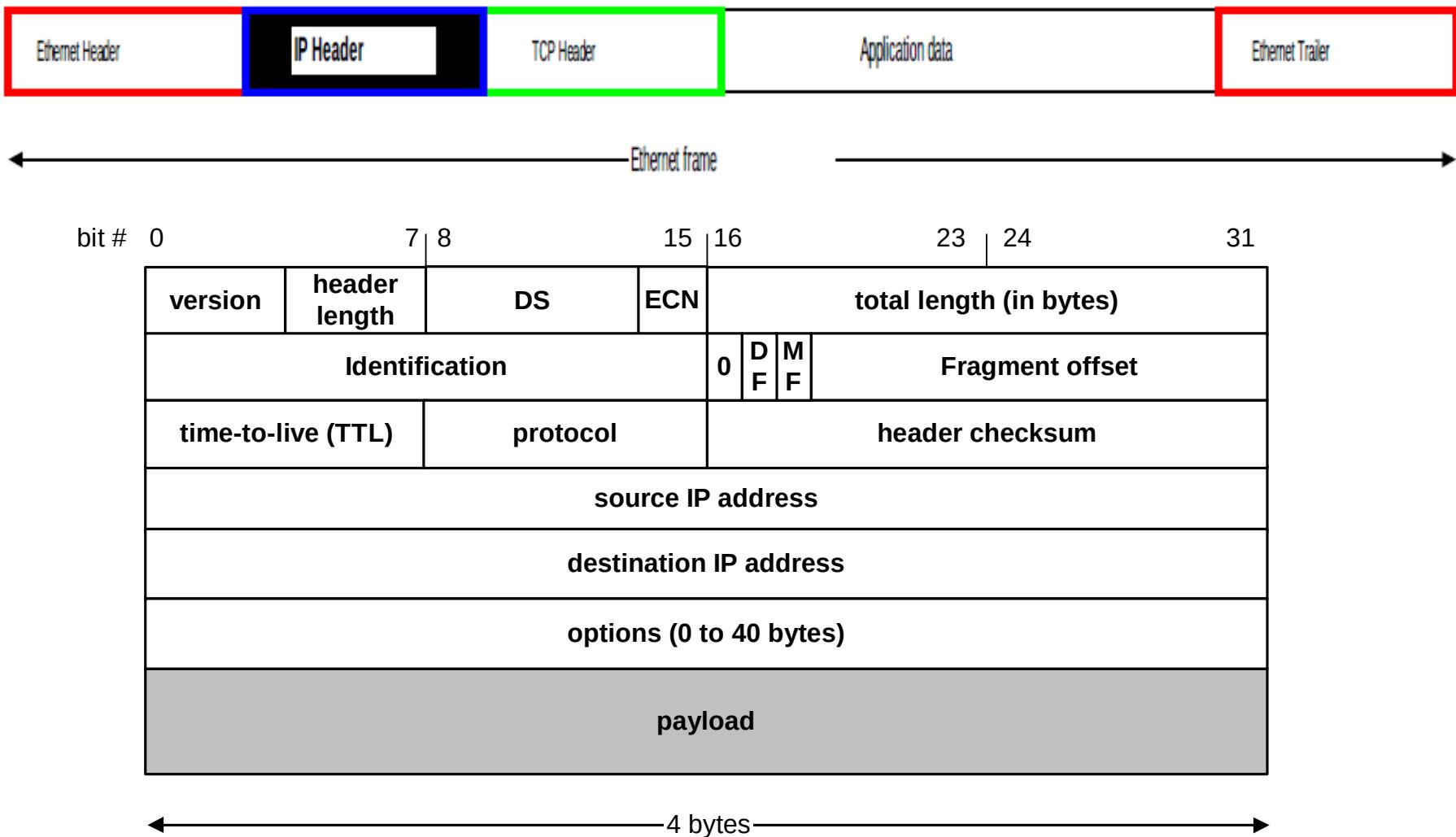
- Evolución:



# Datagrama IPv4



# Datagrama IPv4



# Campos de Datagrama IP

**Version (4 bits)**: versión actual 4, la nueva 6.

**Header length (4 bits)**: longitud en múltiplos de 4B.

## **DS/ECN field (1 byte)**

TOS (Type of Service), DSCP DiffService Codepoint.

### **Differentiated Service (DS) (6 bits)**:

Usado para marcar QoS

### **Explicit Congestion Notification (ECN) (2 bits)**:

Usado en control de congestión con TCP.

# Campos de Datagrama IP

**Identification (16 bits):** identificador único.  
Utilizado para la fragmentación.

## Flags (3 bits):

- Primero es 0.
- DF bit (Do not fragment)
- MF bit (More fragments)

Utilizados para la fragmentación.

# Campos de Datagrama IP

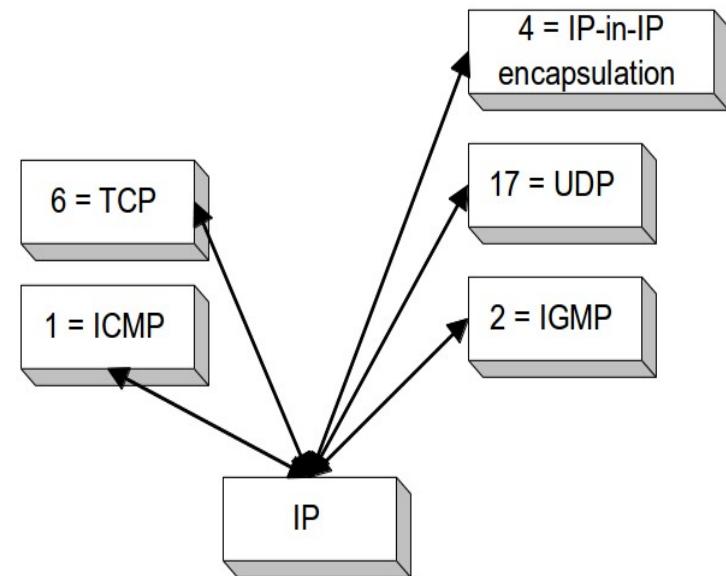
## Time To Live (TTL) (1 byte):

- Cuantos saltos puede dar el datagrama.
- Evita loops.
- Emisor lo pone a un valor, e.g. 128 o 64.
- Cada router por el que pasa lo decrementa en 1.
- Si esta más de un segundo también.
- Si llega a un router que no esta en la red destino y TTL=0, se descarta.

# Campos de Datagrama IP

## Protocol (1 byte):

Para mux/demux.



## Header checksum (2 bytes):

16 bit checksum del header solamente.

# Campos de Datagrama IP

## Options:

- Security restrictions.
- Record Route
- Timestamp.
- (loose) Source Routing
- (strict) Source Routing.

## Padding:

agregado para ser múltiplo de 4B.

# Ruteo

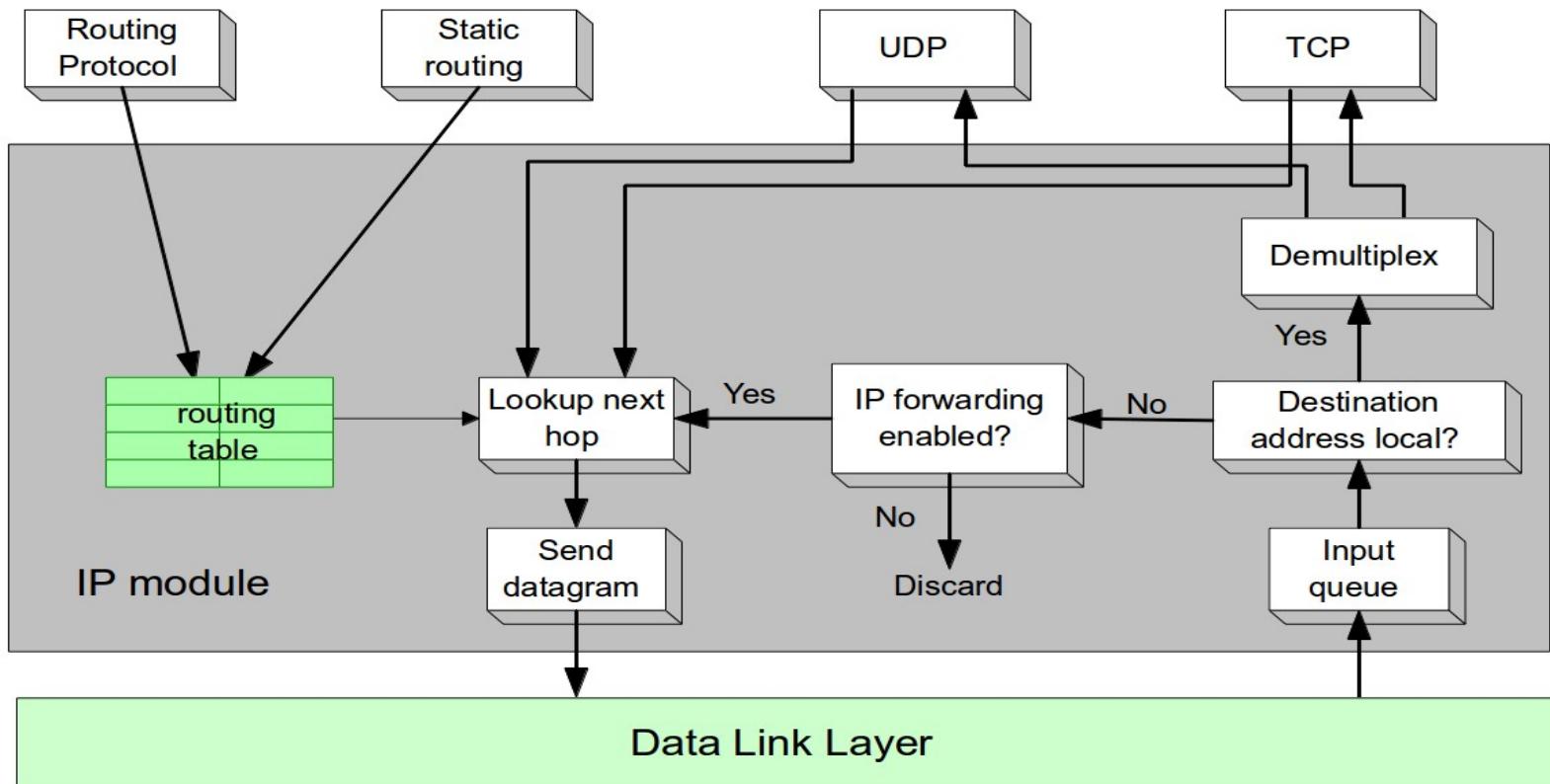
- **Tabla de ruteo:** estructura en hosts y routers (gateways) que indica como despachar un mensaje. Perspectiva del vecino, siguiente salto.
- **Host:** no despacha mensajes que recibe que no son para él. Despacha solo sus mensajes mirando su tabla de ruteo.
- **Router:** Nodos intermedios, más de una interfaz, despacha mensajes mirando tabla de ruteo, desde cualquier interfaz.
- **Host multihome:** tiene varias interfaces, no rutea.

# Ruteo

- **Ruteo:** seleccionar la interfaz de salida y el próximo salto. Routers y Hosts.
- **Forwarding/Despacho:** pasar el paquete desde una interfaz de entrada hacia una de salida. Solo routers.
- El forwarding es más intensivo.
- El ruteo es de control, alimentado por protocolos de enrutamiento (routing).
- El forwarding es de datos, envía protocolos enrutados (routed).

# Ruteo

- Routers tienen el forwarding habilitado, los hosts no.



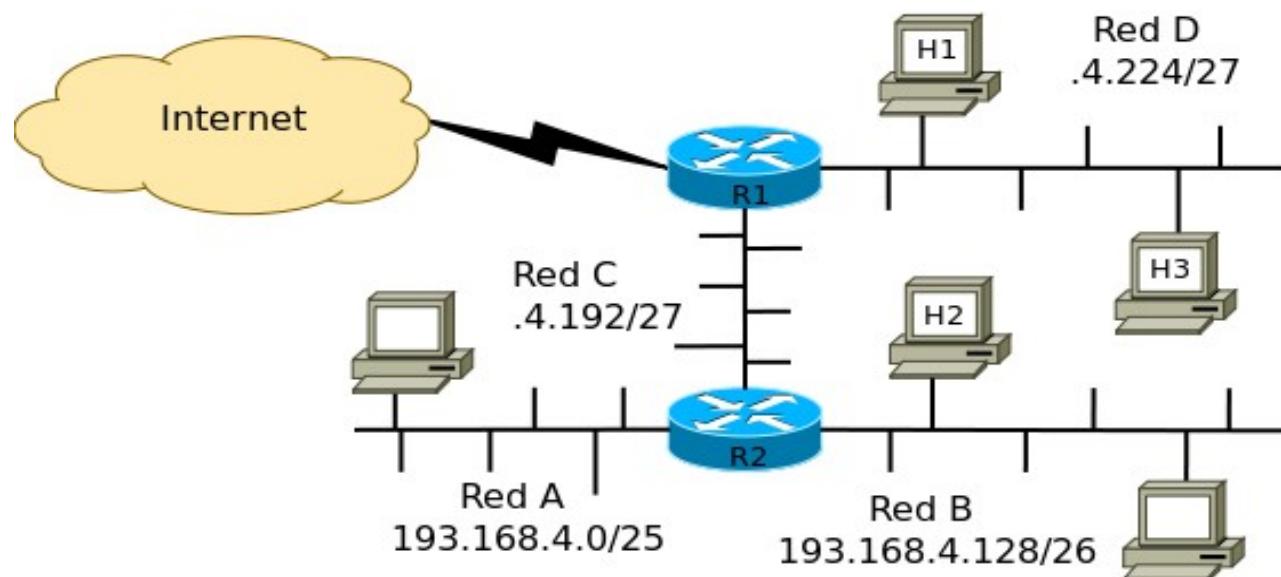
# Tabla de Ruteo

- Estructura de tabla de ruteo:
  - Red Destino (Net/Mask).
  - Next Hop (Próximo salto).
  - Interfaz de salida.
- En un Host es más simple:

```
andres@h1:~$ netstat -nr
```

Destination	Gateway	Genmask	Metric	Iface
193.168.4.224	0.0.0.0	255.255.255.224	0	e0
<b>193.168.4.128</b>	<b>193.168.4.225</b>	<b>255.255.255.192</b>	<b>2</b>	<b>e0</b>
0.0.0.0	193.168.4.225	0.0.0.0	-	e0

# Tabla de Ruteo (H1)



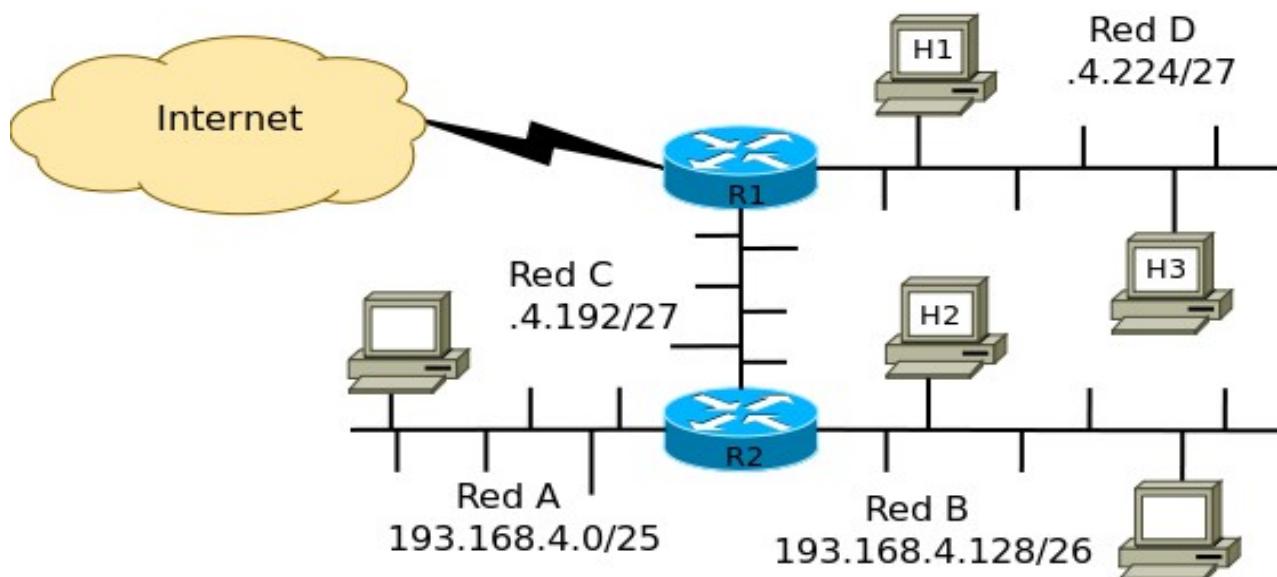
```
root@h1:~# ifconfig e0 193.168.4.226 netmask 255.255.255.224
```

```
root@h1:~# route add default gw 193.168.4.225
```

```
root@h1:~# netstat -nr
```

Destination	Gateway	Genmask	Metric	Iface
193.168.4.224	0.0.0.0	255.255.255.224	0	e0
0.0.0.0	193.168.4.225	0.0.0.0	-	e0 66

# Tabla de Ruteo (R2)

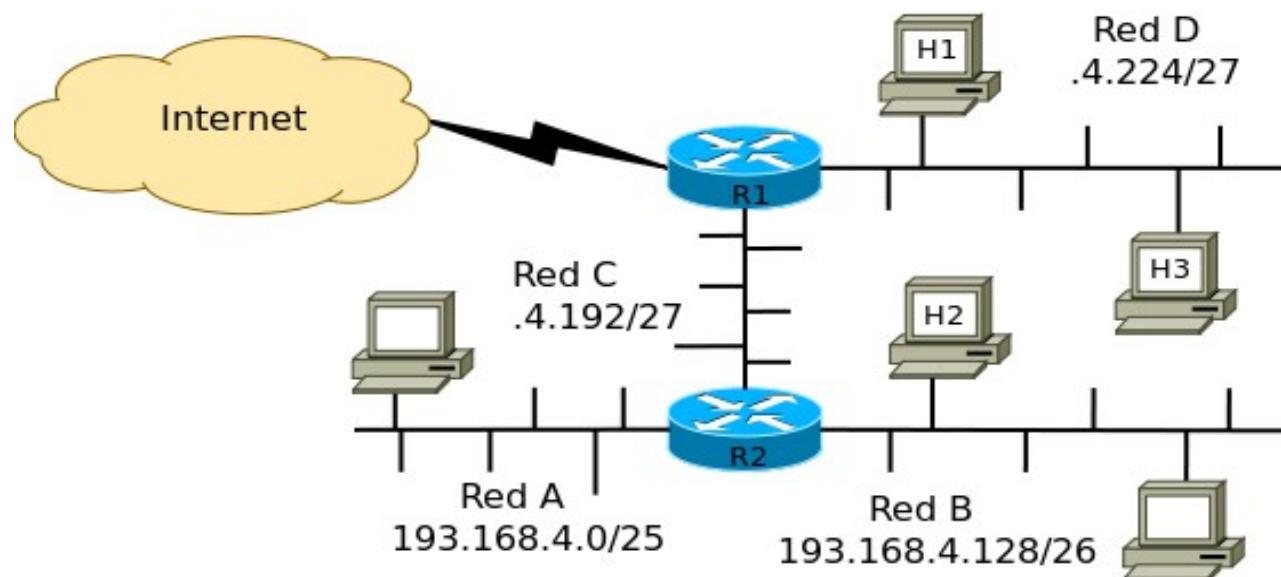


```
root@r2:~# netstat -nr
```

Destination	Gateway	Genmask	Metric	Iface
193.168.4.0	0.0.0.0	255.255.255.128	0	e0
193.168.4.128	0.0.0.0	255.255.255.192	0	e1
193.168.4.192	0.0.0.0	255.255.255.224	0	e2
0.0.0.0	193.168.4.193	0.0.0.0	-	e2

```
root@r2:~# sysctl net.ipv4.ip_forward=1
```

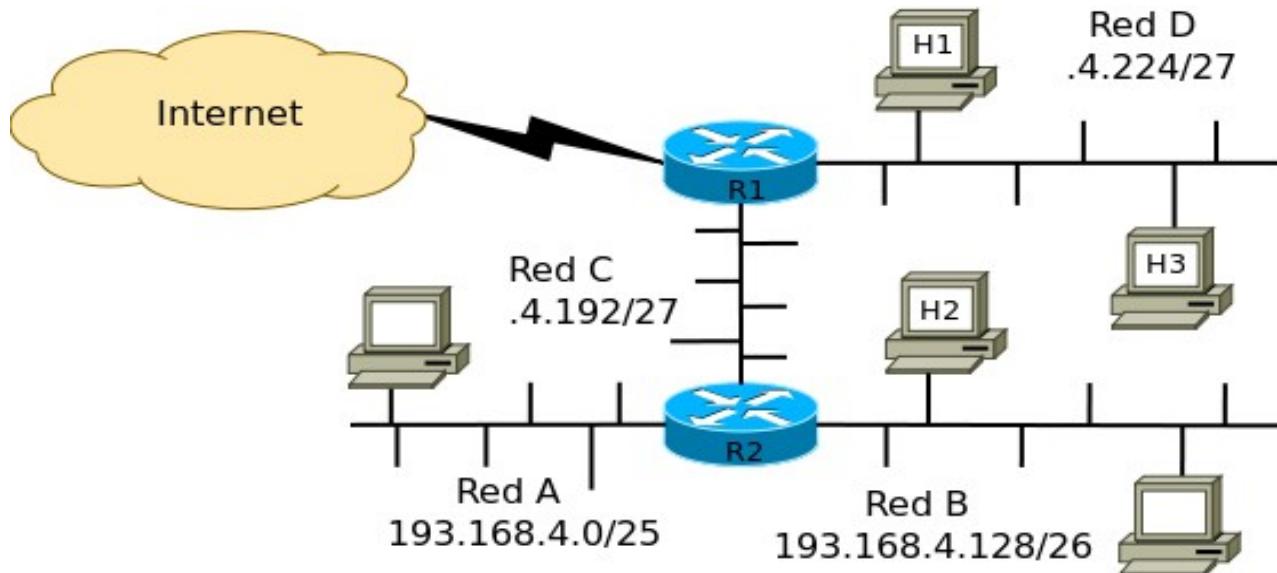
# Tabla de Ruteo (H2)



```
andres@h2:~$ netstat -nr
```

Destination	Gateway	Genmask	Metric	Iface
193.168.4.128	0.0.0.0	255.255.255.192	0	e0
0.0.0.0	193.168.4.129	0.0.0.0	-	e0

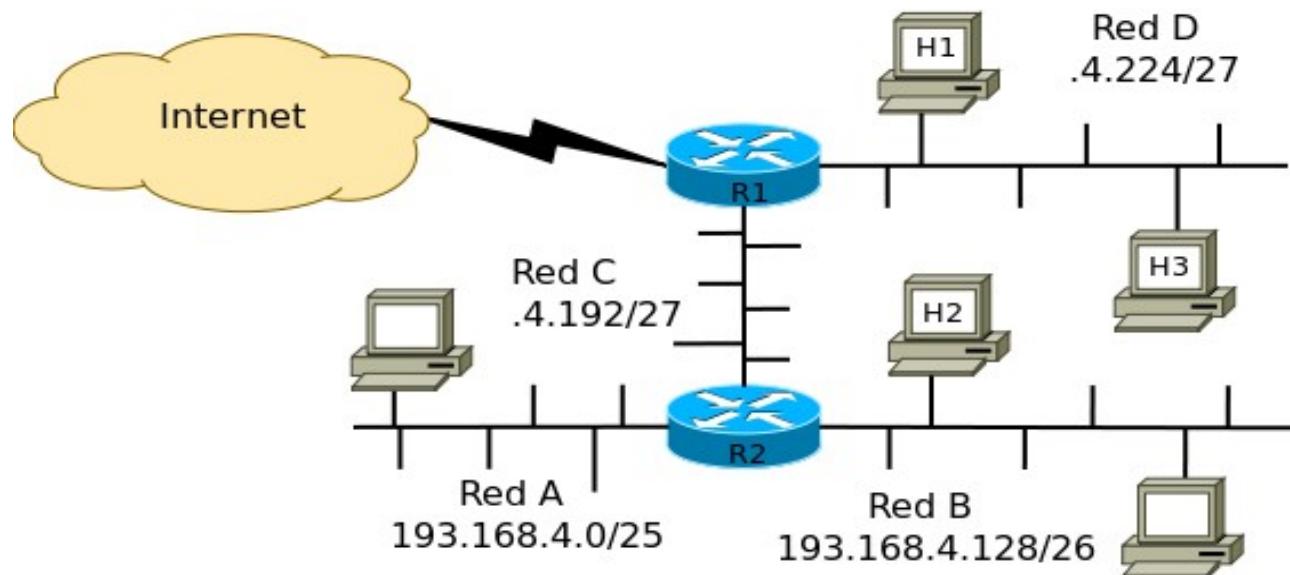
# Tabla de Ruteo (R1)



```
andres@r1:~$ netstat -nr
```

Destination	Gateway	Genmask	Metric	Iface
193.168.4.224	0.0.0.0	255.255.255.224	0	e1
193.168.4.192	0.0.0.0	255.255.255.224	0	e0
200.3.4.0	0.0.0.0	255.255.255.252	0	ppp0
193.168.4.0	193.168.4.194	255.255.255.128	0	e0
193.168.4.128	193.168.4.194	255.255.255.192	0	e0
0.0.0.0	200.3.4.1	0.0.0.0	-	ppp0

# Tabla de Ruteo (R1-a)



## ■ Alternativa

```
andres@r1:~$ netstat -nr
```

Destination	Gateway
193.168.4.224	0.0.0.0
193.168.4.192	0.0.0.0
200.3.4.0	0.0.0.0
193.168.4.0	193.168.4.194
0.0.0.0	200.3.4.1

Genmask	Metric	Iface
255.255.255.224	0	e1
255.255.255.224	0	e0
255.255.255.252	0	ppp0
255.255.255.0	0	e0
0.0.0.0	-	ppp0

# Tareas de Ruteo

- Validación de datagrama: IP header.
- Calcula checksum (solo header).
- Leer IP destino.
- Buscar en tabla de ruteo, seleccionar prefijo más largo (“best match”).
- Decrementar TTL.
- Fragmentar (alternativo).
- Transmitir o Descartar.
- Generar ICMP (alternativo).

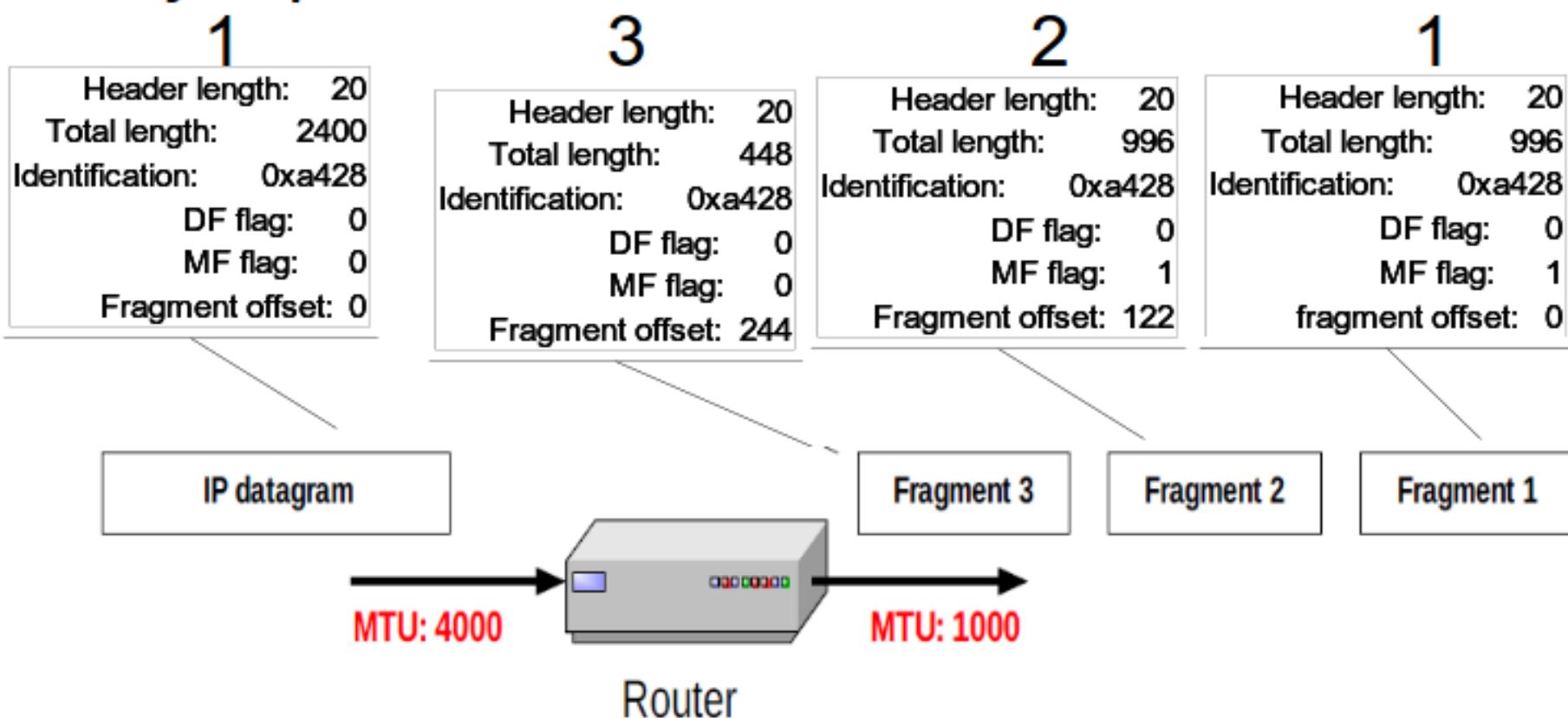
# Fragmentación

- Debido a que hay diferentes capas de enlaces con diferentes MTUs.
- Fragmentos múltiplos de 8 bytes.
  - Offset en unidades de 8 bytes.
- Se deben agregar los headers.

version	header length	DS	ECN	total length (in bytes)			
Identification				0	D	M	
time-to-live (TTL)				F	F		Fragment offset
protocol				header checksum			

# Fragmentación

## Ejemplo:



# Referencias:

- Richard Stevens. TCP/IP Illustrated. Vol 1. The Protocols.
- Douglas Comer. Internetworking with TCP/IP. Vol 1.

# Referencias:

- RFC 791 – IP.
- RFC 792 - ICMP.
- RFC 796 - Address mappings.
- ·RFC 1812 Requirements for IP Version 4 Routers.
- RFC 917 - Internet subnets.
- RFC 940 - Toward an Internet standard scheme for subnetting.
- RFC 950 - Internet Standard Subnetting Procedure.

# Referencias:

- RFC 1878 - Variable Length Subnet Table For IPv4.
- RFC 1918 - Address Allocation for Private Internets.
- RFC 1631 - The IP Network Address Translator (NAT) -Obsoleta por 3022-
- RFC 3022 – Traditional NAT.
- RFC 1517 - Applicability Statement for the Implementation of CIDR.
- RFC 1518 - An Architecture for IP Address Allocation with CIDR.
- RFC 1519 - Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy.



# ICMP (Internet Control Message Protocol)

# ICMP

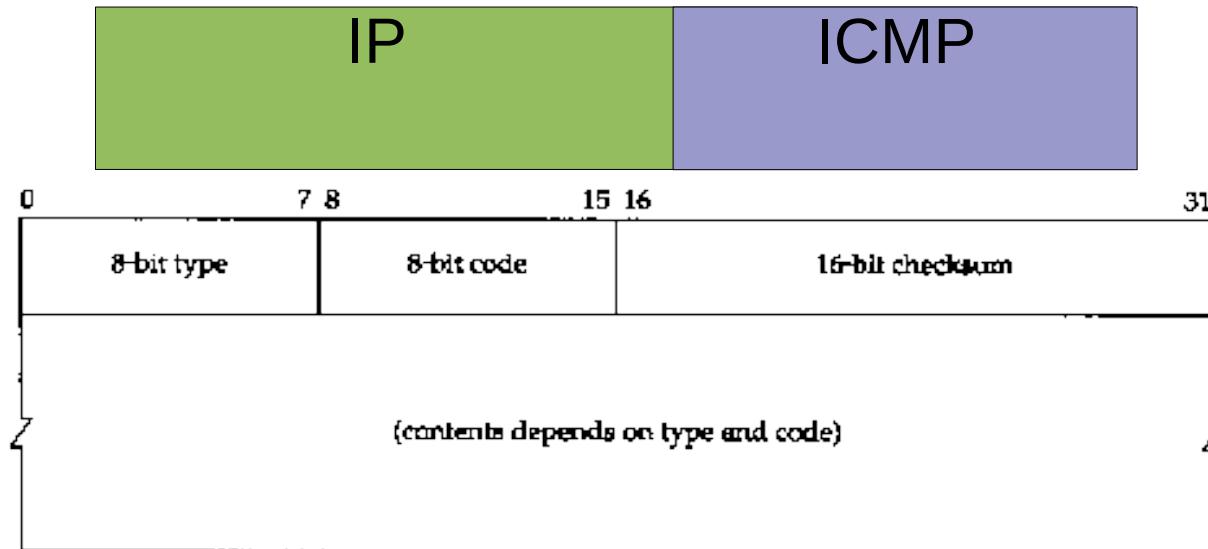
(Internet Control Message Protocol)

- Protocolo de L3.
- Protocolo “Helper” de IP.
- IP carece de control, el mismo es dado por un protocolo auxiliar.
- ICMP no le agrega confiabilidad a IP, solo brinda un “feedback” para poder resolver problemas en la red.
- ICMP podría ser prescindible en IPv4, aunque en la RFC se indica que se debe implementar en cada módulo IP.
- Definido en RFC-792.

# ICMP

(Cont.)

- ICMP se encapsula en IP.
- ICMP no es un protocolo de transporte, ya que no fue concebido para llevar datos de usuario.
- Formato de Mensaje:



# MENSAJES ICMP

- Algunos Tipos de Mensajes ICMP:
  - Echo Request/Echo Reply (PING).
  - Destino Inalcanzable.
  - TTL expirado.
  - Source Quench (Control de Congestión).
  - Redirección de Ruta.
  - Address Mask y Timestamp.

# ICMP PING (Echo)

- Pensado para probar conectividad IP entre dos hosts.
- Sirve para medir el RTT min/avg/max/dev y loss, de esta forma poder diagnosticar problemas.
- Packet INternet Gopher, el nombre basado en el sonido de un sonar de un submarino al escanear.
- Si un nodo recibe un ICMP ***Echo Request***, debe responder copiando el contendio con un ***Echo Reply (PONG)***. RFC-1122.
- Actualmente, muy desprestigiado ;-) Se filtra.

# ICMP PING (Echo) (Cont.)

No. .	Time	Source	Destination	Protocol	Info
3	0.001053	200.1.1.201	200.1.1.254	ICMP	Echo (ping) request
4	0.001917	200.1.1.254	200.1.1.201	ICMP	Echo (ping) reply
5	1.006012	200.1.1.201	200.1.1.254	ICMP	Echo (ping) request
6	1.009470	200.1.1.254	200.1.1.201	ICMP	Echo (ping) reply
7	2.006780	200.1.1.201	200.1.1.254	ICMP	Echo (ping) request

Frame 3 (98 bytes on wire, 98 bytes captured)

Ethernet II, Src: RealtekU\_12:34:56 (52:54:00:12:34:56), Dst: RealtekU\_12:34:57 (52:54:00:12:34:57)

Internet Protocol, Src: 200.1.1.201 (200.1.1.201), Dst: 200.1.1.254 (200.1.1.254)

Internet Control Message Protocol

Type: 8 (Echo (ping) request)

Code: 0 ()

Checksum: 0x3d87 [correct]

Identifier: 0x3119

Sequence number: 1 (0x0001)

Data (56 bytes)

# ICMP PING (Echo) (Cont.)

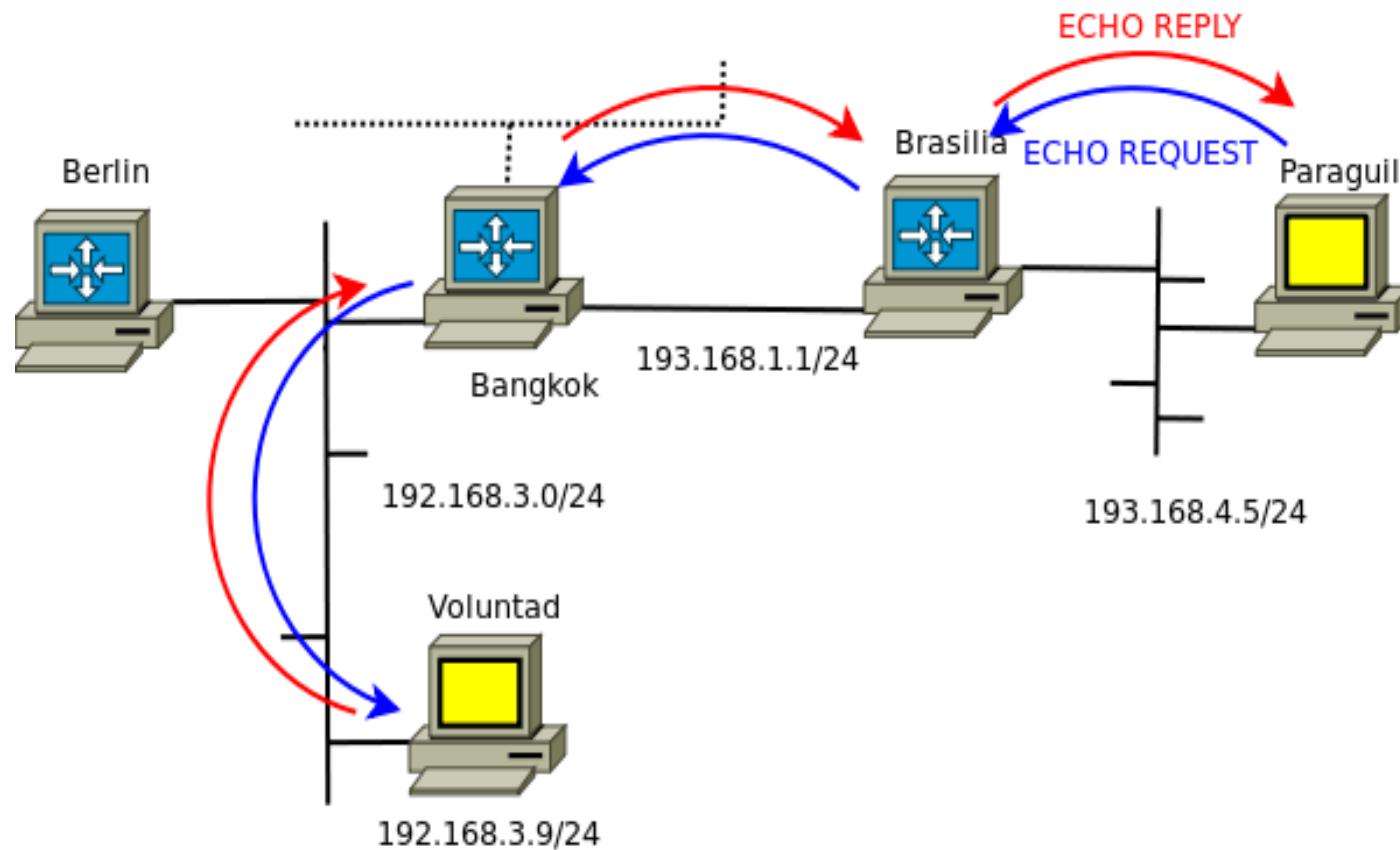
## ■ *Echo Request , Echo Reply:*

```
andres@h1(paraguil):~$ ping -c 3 193.168.3.9
PING 193.168.3.9 (193.168.3.9) 56(84) bytes of data.
64 bytes from 193.168.3.9: icmp_seq=1 ttl=53 time=149 ms
64 bytes from 193.168.3.9: icmp_seq=2 ttl=53 time=150 ms
64 bytes from 193.168.3.9: icmp_seq=3 ttl=53 time=147 ms

--- 193.168.3.9 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time
2002ms
rtt min/avg/max/mdev = 147.498/149.014/150.128/1.197 ms
```

# ICMP PING (Echo) (Cont.)

## ■ *Echo Request , Echo Reply:*



# ICMP Destino Inalcanzable

- Para indicar que una red, un host, un puerto es inalcanzable, diferentes causas.
  - Host Inalcanzable (Host Unreachable):
    - Posibles causas, no esta encendido el host, no responde ARP.
  - Red Inalcanzable (Network Unreachable):
    - No tiene el router una ruta en la tabla de ruteo a esta red.
  - Puerto Inalcanzable (Port Unreachable):
    - No hay un proceso UDP en el puerto.
  - Los mensajes requieren fragmentación.
  - El mensaje fue filtrado (admin).
  - Otros.

# ICMP Destino Inalcanzable

(Cont.)

## ■ ***Echo Request , Host Unreachable:***

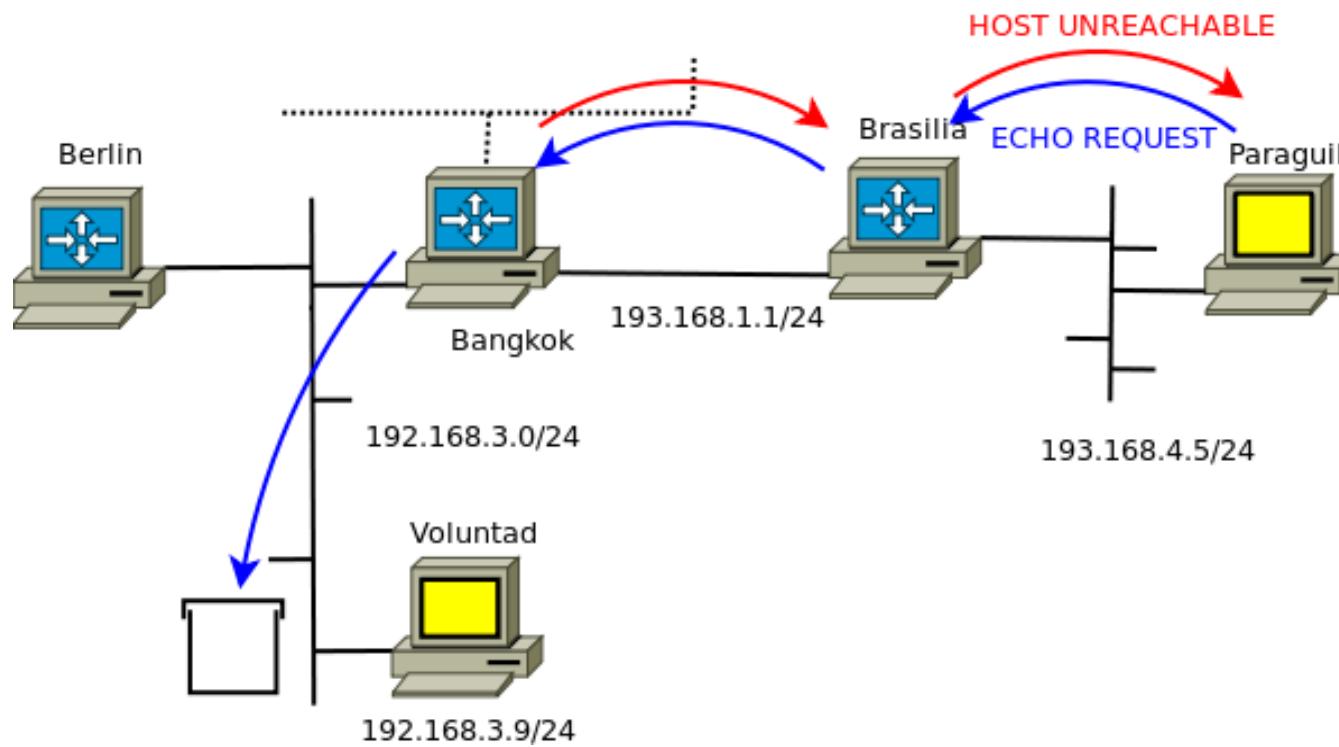
```
andres@h1(paraguil):~$ ping -c 3 193.168.3.10
PING 193.168.3.10 (193.168.3.10) 56(84) bytes of data.
From 193.168.1.2 icmp_seq=1 Destination Host Unreachable
From 193.168.1.2 icmp_seq=2 Destination Host Unreachable
From 193.168.1.2 icmp_seq=3 Destination Host Unreachable

--- 193.168.3.10 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet
loss, time 2007ms
, pipe 3
```

# ICMP Destino Inalcanzable

(Cont.)

## ■ *Echo Request , Host Unreachable:*



# ICMP TTL Expirado

- El tiempo de vida ha expirado. En realidad es el hop count con el cual salió el mensaje ha expirado.
- Time Exceeded:
  - Tiempo Excedido en viaje.
  - Tiempo Excedido en re-ensamblado.
- TTL en IP, no solo ICMP.
- Valor máximo de TTL=255.
- Puede salir con otro valor.
- Si TTL == 0, pero ya llegó a la red destino debería enviarse.
- Utilizado por traceroute(8) con UDP o ICMP.

# ICMP TTL Expirado (Cont.)

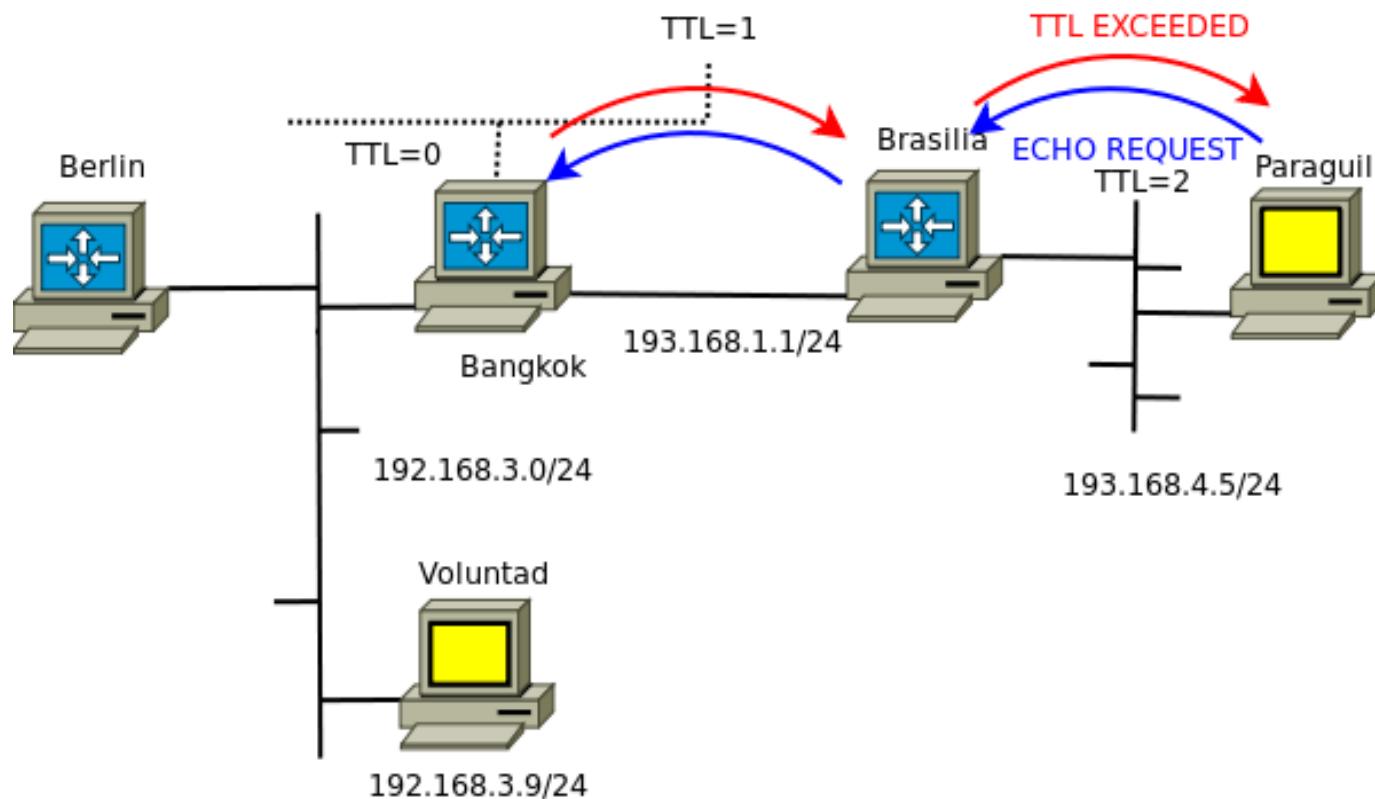
- ***Echo Request , TTL Exceeded:***

```
andres@h1(paraguil):~$ ping -c 2 -t 2 193.168.50.50
PING 193.168.50.50 (193.168.50.50) 56(84) bytes of
data.
From 193.168.1.2 icmp_seq=1 Time to live exceeded
From 193.168.1.2 icmp_seq=2 Time to live exceeded
From 193.168.1.2 icmp_seq=3 Time to live exceeded

--- 193.168.50.50 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100%
packet loss, time 2003ms
```

# ICMP TTL Expirado (Cont.)

- **Echo Request , TTL Exceeded:**



# ICMP TTL Expirado (Cont.)

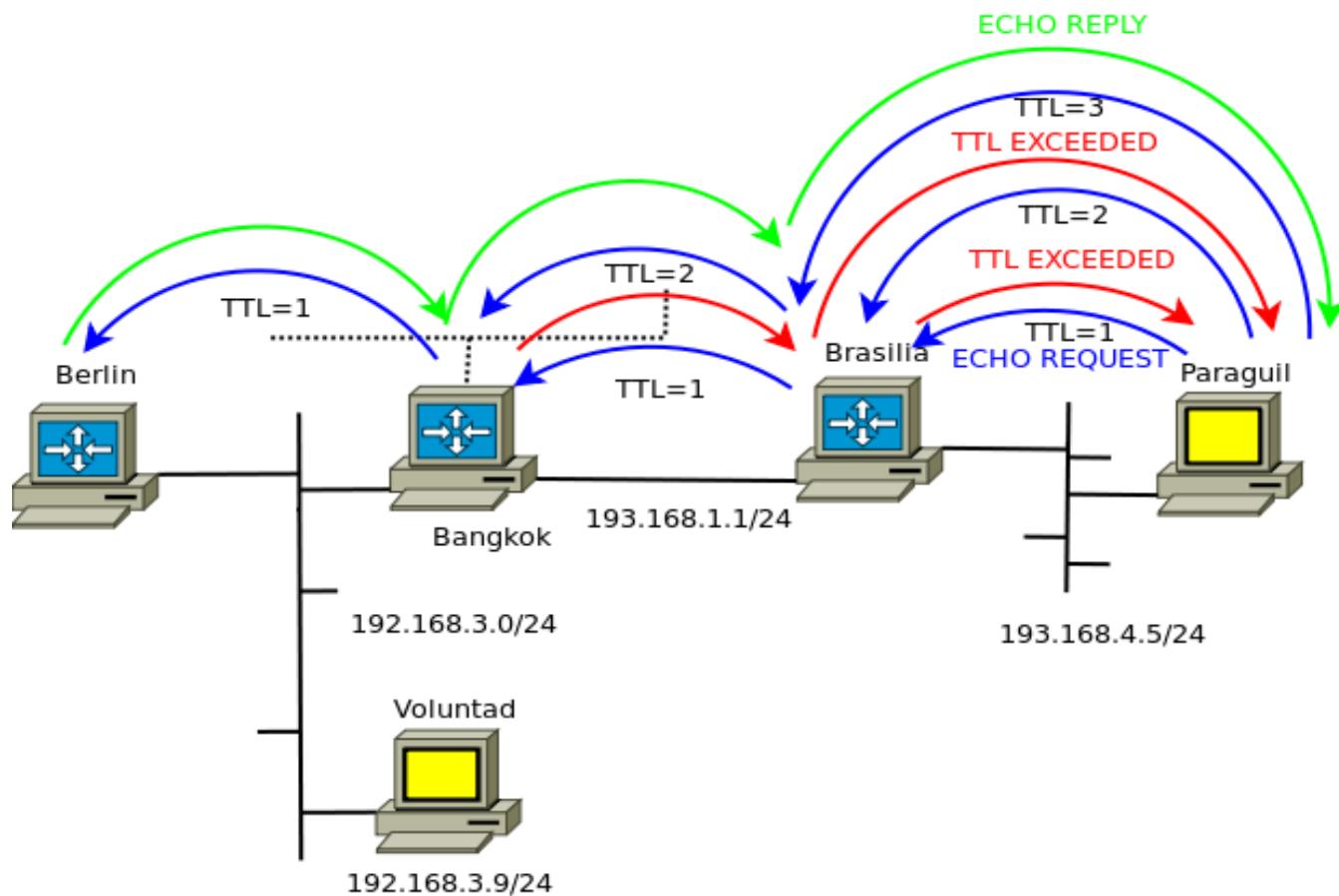
- **Echo Request , TTL Exceeded , Echo Reply:**

```
root@h1(paraguil)~# traceroute -n -I 193.168.3.254
traceroute to 193.168.3.254 (193.168.3.254), 30 hops max,
60 byte packets
```

```
1 193.168.4.65 3.698 ms 3.740 ms 4.405 ms
2 193.168.1.2 9.185 ms 9.809 ms 14.130 ms
3 193.168.3.254 14.973 ms 15.073 ms 17.312 ms
```

# ICMP TTL Expirado (Cont.)

- **Echo Request , TTL Exceeded , Echo Reply:**



# ICMP Route Redirect

- **Echo Request , Route Redirect , Echo Reply:**

```
andres@h1(paraguil):~$ ping 193.168.6.1
```

```
andres@r2(bangkok):~$ netstat -nr
```

Destination	Gateway	Genmask	Flag	Iface
193.168.6.0	193.168.3.9	255.255.255.0	G	e0
...				

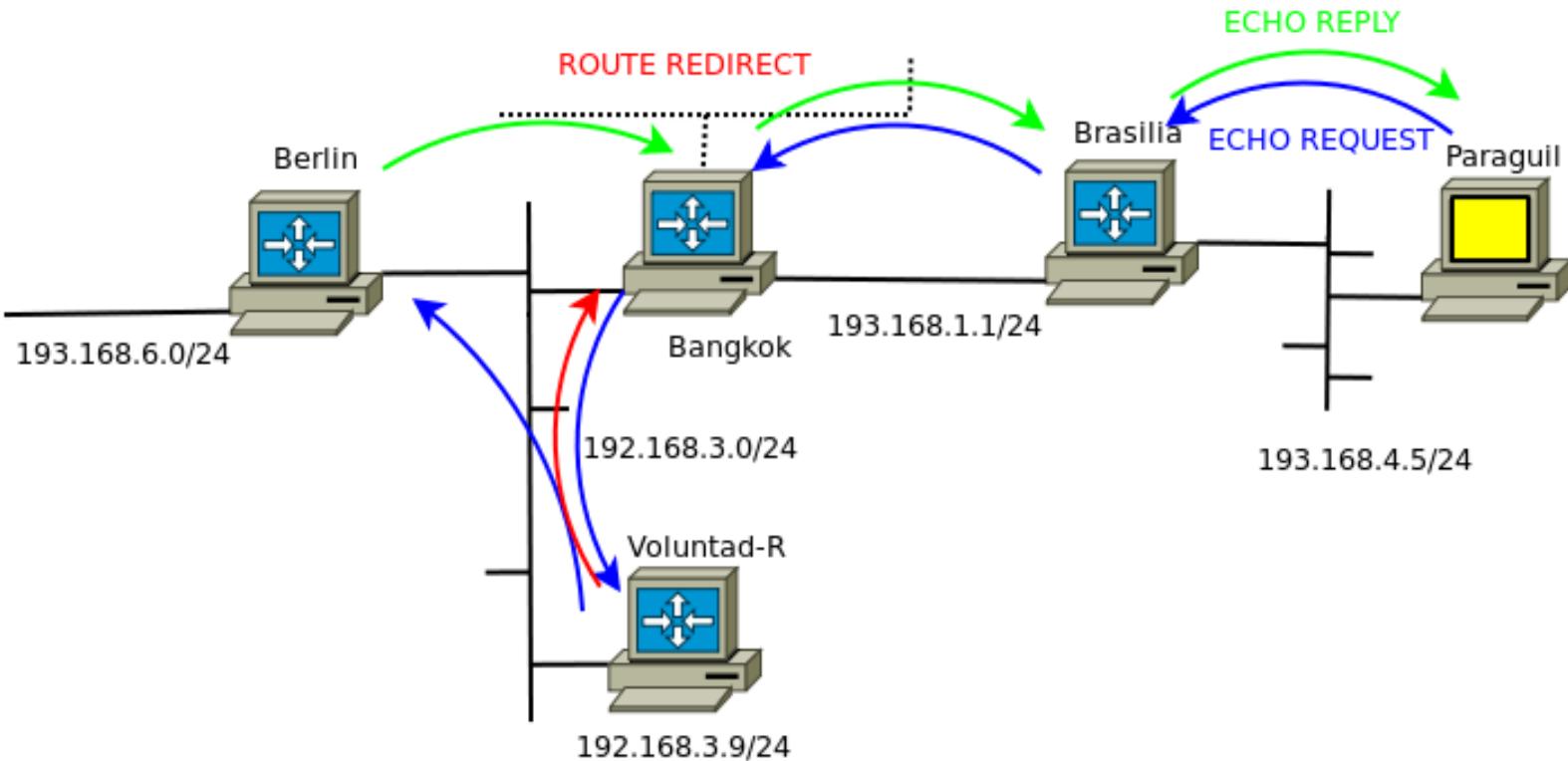
Luego del Redirect:

```
andres@r2(bangkok):~$ netstat -nr
```

Destination	Gateway	Genmask	Flag	Iface
193.168.6.0	193.168.3.9	255.255.255.0	G	e0
193.168.6.1	193.168.3.254	255.255.255.255	DH	e0
...				

# ICMP Route Redirect (Cont.)

- **Echo Request , Route Redirect , Echo Reply:**



Hoy desaconsejado y desactivado.

# Referencias:

- Richard Stevens. TCP/IP Illustrated. Vol 1. The Protocols.
- Douglas Comer. Internetworking with TCP/IP. Vol 1.
- Data & Computer Communications (6th Edition), William Stallings.

# Protocolos de Ruteo Dinámicos (Introducción)

2020

Facultad de  
Informática



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Contenidos

1

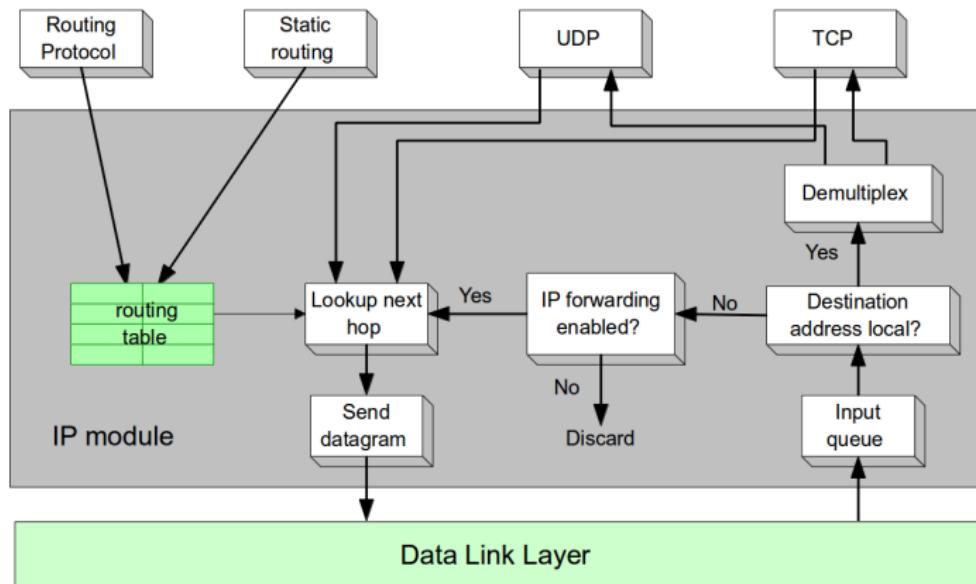
Introducción al Ruteo Dinámico

Clasificación de Protocolos de Ruteo

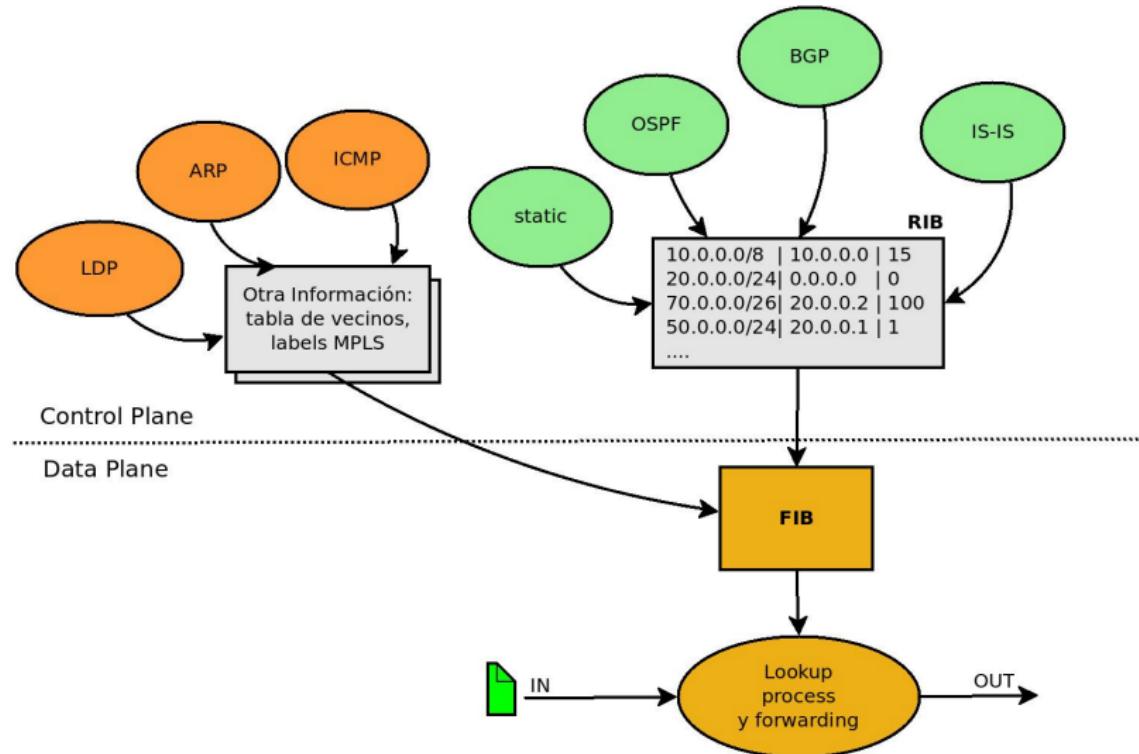
# Introducción, Conceptos

- El protocolo **ENRUTADO (Routed)** IP, requiere los servicios del protocolo de **ENRUTAMIENTO/RUTEO (Routing)** para construir las tablas de ruteo en cada router (gateway)
- **Forwarding/Switching:** selecciona un port de salida en función de la dirección de destino y tabla de ruteo. Usado por el protocolo ENRUTADO. **Plano de Datos.** Todos los router lo hacen
- **Ruteo:** proceso mediante el cual se construye la tabla de ruteo o **RIB (Routing Information Base)** Protocolo de ENRUTAMIENTO. **Plano de Control.** Algunos routers lo hacen
- **FIB:** Forwarding Information Base / Forwarding Table, el proceso de forwarding que se hacer a partir de la RIB se optimiza generando una tabla más eficiente, FIB, por ejemplo implementada como TCAM en multilayer-switching

# Routing y Forwarding



# Routing y Forwarding (Cont'd)



# Routing

- Decisiones de “forwarding” en IP se llevan a cabo localmente
- Deriva en conectividad entre los diferentes puntos de la red
- Se requieren recolectar y procesar un estado global
- Se mantiene un estado global localmente en cada router
- Los estados locales deben ser consistentes, si son inconsistentes la red no habrá convergido a un estado estable, se generan loops
- Se requiere:
  - Consistencia
  - Completitud
  - Escalabilidad
- Se desea:
  - Camino óptimo
  - Balanceo
  - Adaptabilidad

# Tipos de Routing

- Todos los equipos en la red corren el protocolo ENRUTADO, por ejemplo IP (IPv4 o IPv6)
- Los host no requieren correr protocolos de ENRUTAMIENTO/RUTEO
- Los router requieren hacer el ENRUTAMIENTO podrían trabajar de dos formas/ tipos de Routing:
  - Ruteo Estático
  - Ruteo Dinámico
- Una red compleja: muchos routers y enlaces requiere un protocolo de ruteo dinámico
- Routers pueden participar de forma activa en el routing: reciben, generan y propagan información, los hosts lo hacen de forma pasiva (no envían ni propagan información)

# Ruteo Estático

- Las rutas son establecidas por el administrador manualmente
- Propenso a errores
- Si se cambia la topología requiere cambios manuales en los routers
- Sirve cuando se tiene una red sencilla
- No tiene problemas de seguridad ni de incompatibilidad
- No implica costo de procesamiento extra
- Mayor control
- Esquema NO escalable y NO tolerante a fallos

# Ruteo Dinámico

- Requiere una configuración inicial por el administrador
- Si se cambia la topología se adapta de forma automática
- Facilita mantenimiento cuando se tiene una red compleja
- Implica costo de procesamiento extra
- Esquema escalable y tolerante a fallos
- Resolución de Problemas/Debugging, más complejo
- Caminos “óptimos” de acuerdo a la información manejada por el protocolo (métrica, costo)

# Routing Domain / Definición de AS

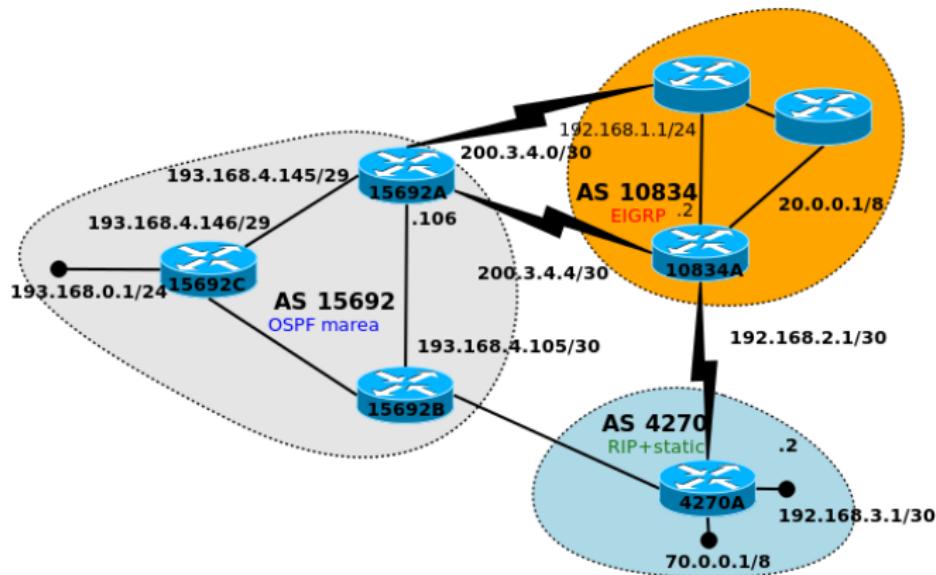
**Routing Domain:** seleccionamos el/los protocolo/s de Ruteo en un **Routing Domain**, conjunto de routers con **Routing Protocols** comunes. Uno o más de estos incluidos en un AS.

**Sistema Autónomo (Autonomous System, AS):** conjunto de redes bajo la misma administración (podría ser gestionada por más de un operador de red), y utilizando un protocolo de ruteo o combinaciones para rutear internamente, independientemente de la red de su proveedor. clara y única política de ruteo. Cada AS (Sistema Autónomo) en Internet (necesidad de intercambiar tráfico con otros Dominios) debe tener un número identificador: ASN (AS Number). Relacionado con BGP, otorgado por los RIRs, LACNIC, RIPE, ARIN, AFRINIC, ARIN.

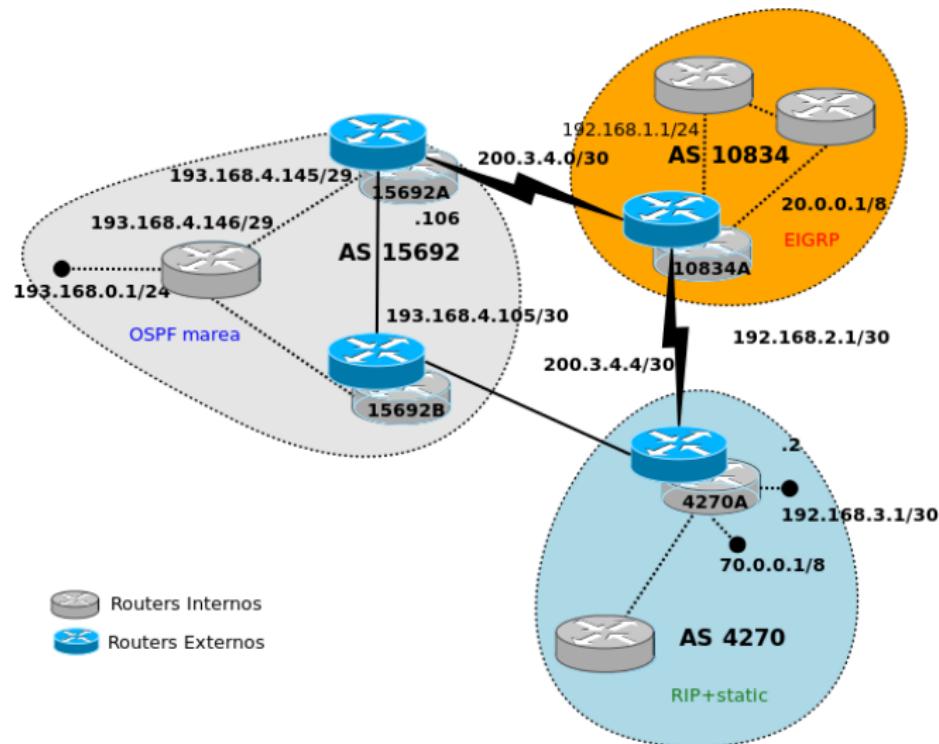
# Clasificación Protocolos de Ruteo Dinámico

- IGP (Interior Gateway Protocols), trabajan en el mismo AS
  - RIP (Routing Internet Protocol) , v1, v2 (estándar IETF)
  - IGRP e EIGRP (-Enhanced- Interior Gateway Routing Protocol) (propietarios de cisco)
  - OSPF (Open Shortest Path First) , v2 , v3 (estándar IETF)
  - IS-IS (Intermediate System to Intermediate System) (estándar de la ISO)
- EGP (Exterior Gateway Protocols), trabajan entre diferentes AS
  - GGP (Gateway to Gateway Protocol) (antecesor)
  - EGP (Exterior Gateway Protocol) (estándar IETF, en desuso)
  - BGP (Border Gateway Protocol) (estándar IETF)

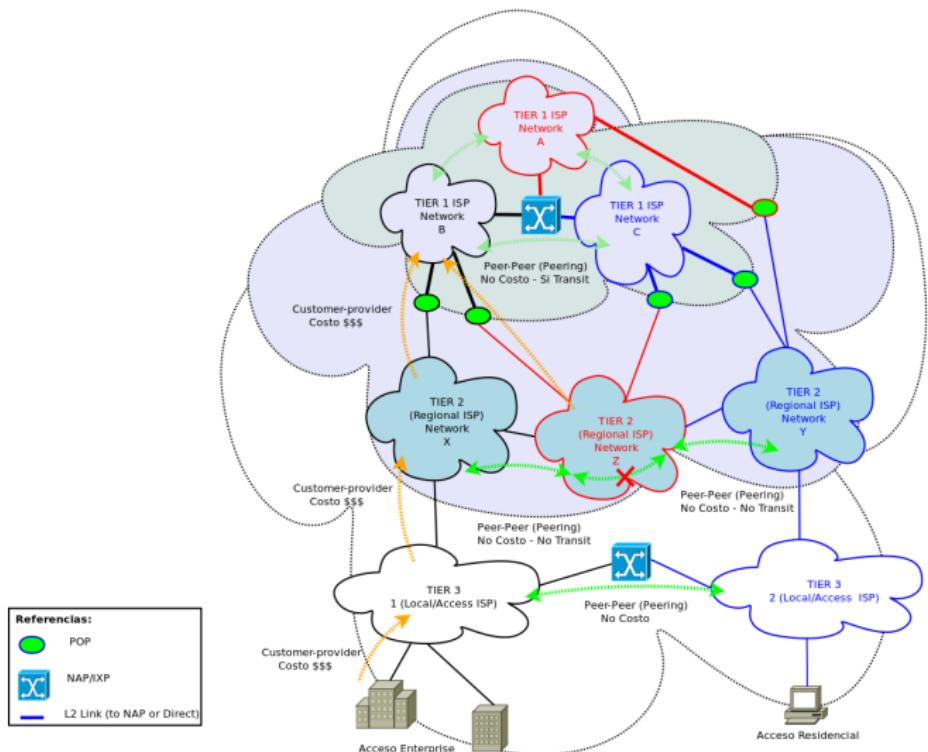
# Protocolos IGP vs EGP



# Protocolos IGP vs EGP (Cont'd)

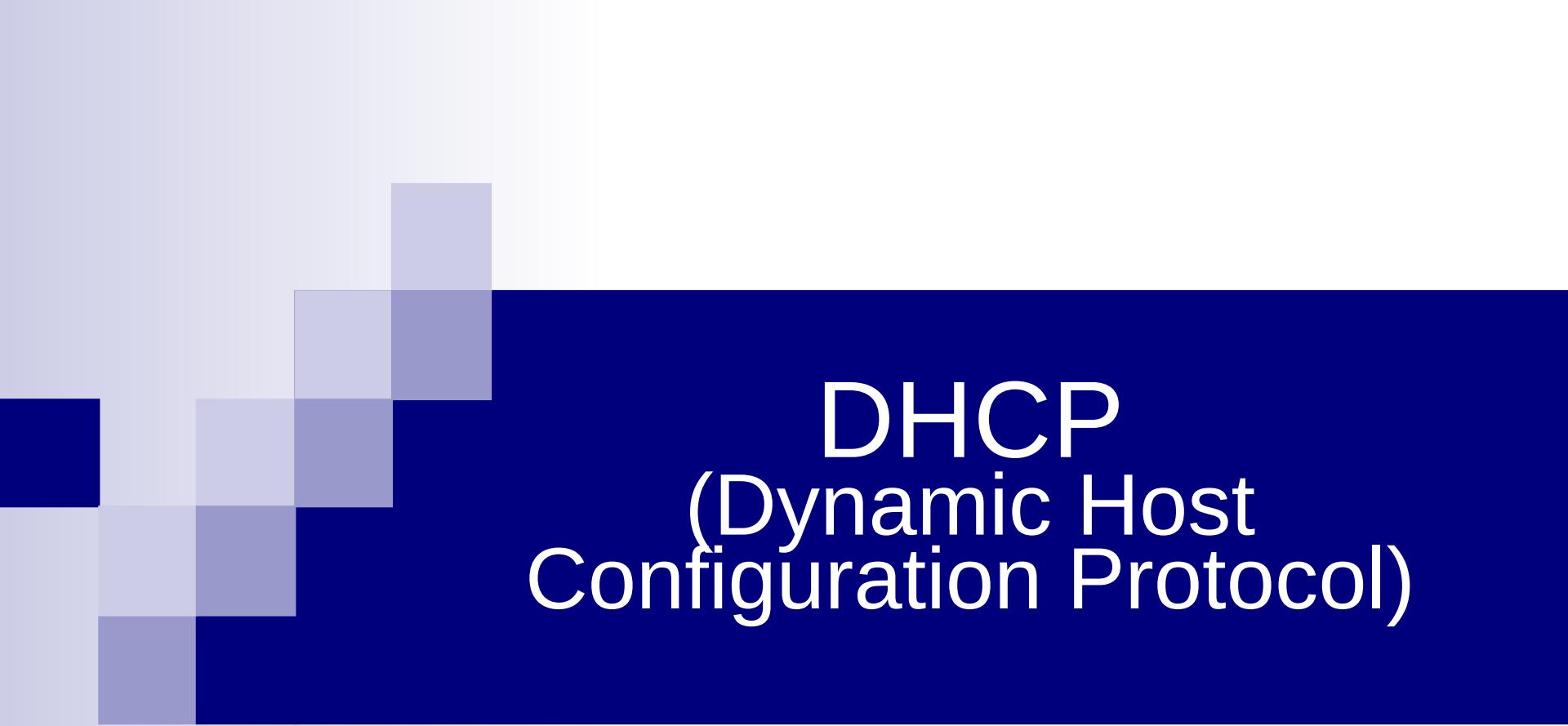


# Estructura de Internet



# Otra Clasificación de Protocolos de Ruteo

- Protocolos de DV (Vector de Distancia)
  - RIP, v1, v2
  - IGRP
  - GGP
- Protocolos de PV (Vector de Camino)
  - BGP
  - EGP
- Link State (Estado de Enlace)
  - OSPF
  - IS-IS
- Vector de Distancia Avanzado (Advanced VD) (considerado Híbrido)
  - EIGRP



# DHCP (Dynamic Host Configuration Protocol)

# DHCP

- Un host para conectarse a una red IP requiere 3 parámetros + 1.
- Para conectarse a la red local:
  - Dirección IP
  - Máscara de red.
- Para conectarse a otras redes:
  - Router por default (Default Gateway).
- Para usar servicios
  - Servidor(es) de DNS.

# DHCP (Cont.)

- Forma de Obtener los Parámetros:
  - Configuración manual/estática:
    - Difícil de mantener.
    - No escalable (recolección, re-asign.)
    - No sirve para movilidad.
  - Configuración dinámica:
    - RARP.
    - ICMP.
    - BOOTP.
    - DHCP.

# DHCP (Cont.)

## (Dynamic Host Configuration Protocol)

- Protocolo de L3.
- Protocolo “Helper” de IP.
- Al estar montado sobre UDP se lo suele considerar protocolo de nivel de aplicación.
- Tanto para IPv4 o IPv6.
- Permite la configuración dinámica de los parámetros de red de los hosts.
- Definido en RFC-2131.

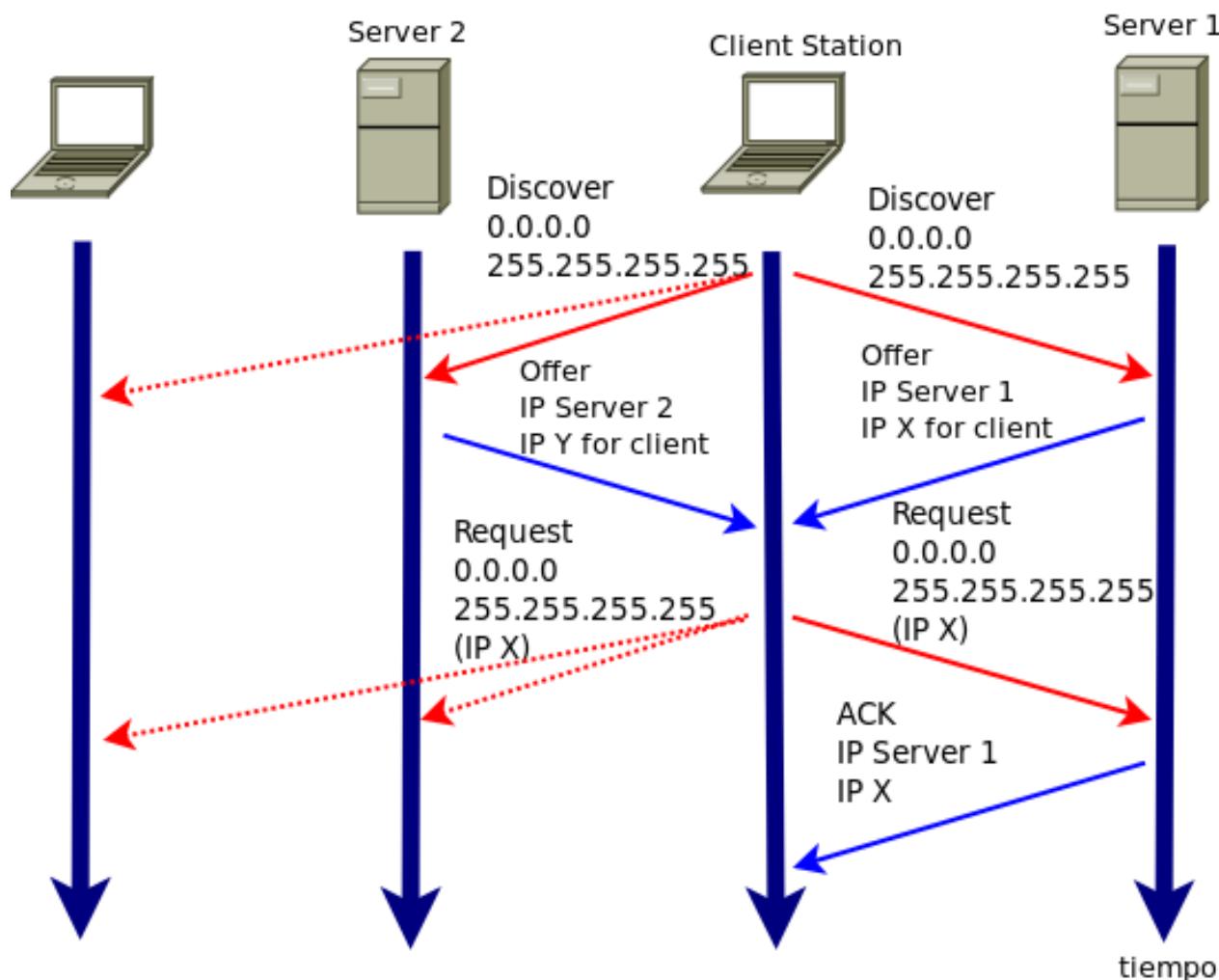
# DHCP (Cont.)

- Los host al arrancar solo tienen acceso a su red local de forma broadcast.
- En la red local existe un o más servidor de auto-configuración:
  - DHCP servers.
- Los host sin parámetros de red envían requerimiento.
- Los servidores los atienden asignando los valores que brindan conectividad.
- El parámetro se reserva por un tiempo.

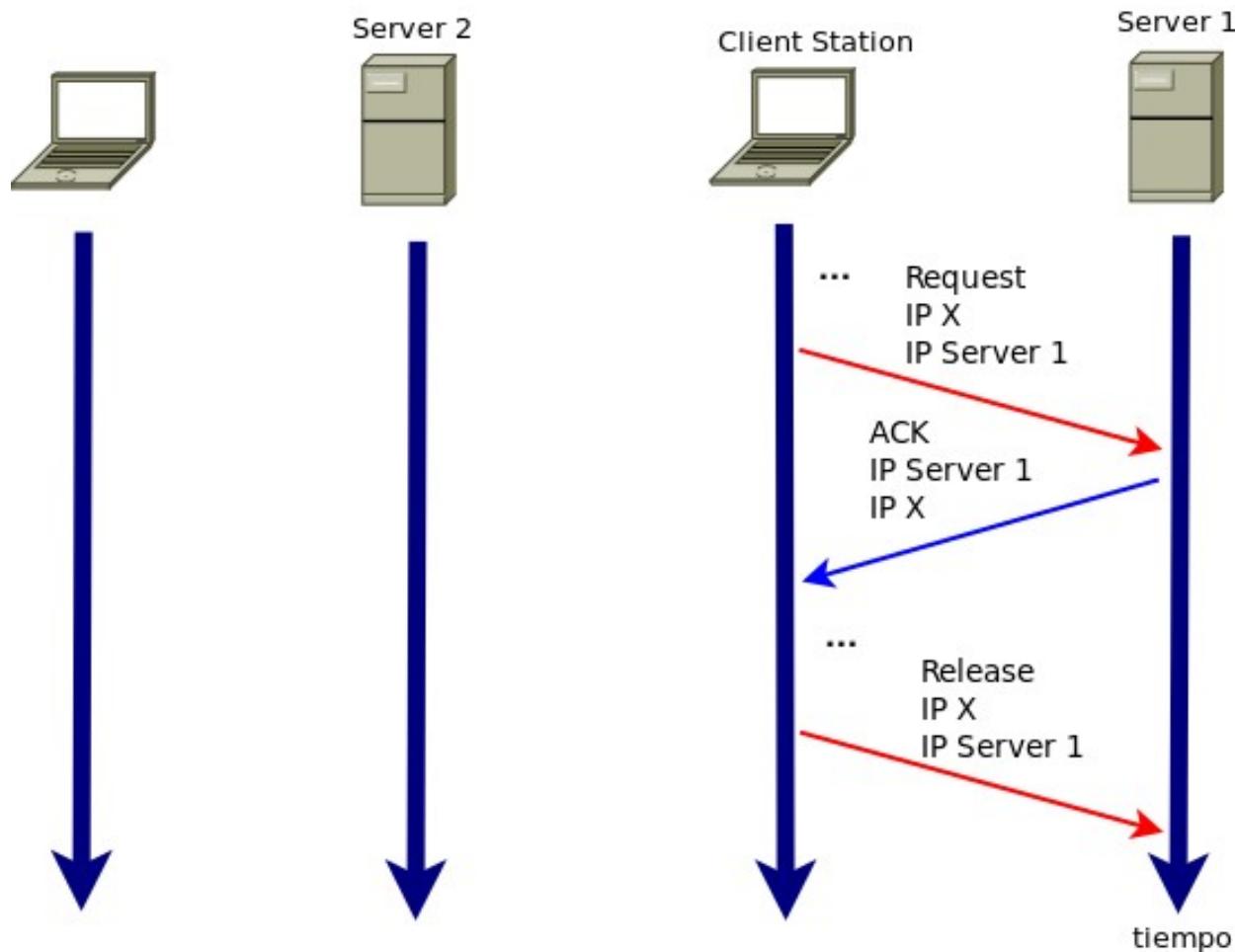
# DHCP Mensajes

- Algunos Mensajes DHCP:
  - Discover.
  - Offer.
  - Request.
  - ACK.
  - Release.
  - NAK.
- Montado sobre UDP:
  - Bootpc (client) 68
  - Bootps (server) 67

# DHCP Ejemplo



# DHCP Ejemplo



# DHCP Ejemplo

```
root@h1(berlin):~# dhclient eth0
```

```
root@h1(berlin):~#ping www.google.com
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0xb947b252
2	0.001106	10.0.2.2	10.0.2.15	DHCP	590	DHCP Offer - Transaction ID 0xb947b252
3	0.003496	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request - Transaction ID 0xb947b252
4	0.003646	10.0.2.2	10.0.2.15	DHCP	590	DHCP ACK - Transaction ID 0xb947b252

```
► User Datagram Protocol, Src Port: bootps (67), Dst Port: bootpc (68)
```

## ▼ Bootstrap Protocol

```
Message type: Boot Reply (2)
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0xb947b252
Seconds elapsed: 0
► Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: 10.0.2.15 (10.0.2.15)
Next server IP address: 10.0.2.4 (10.0.2.4)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: CadmusCo_21:2c:e0 (08:00:27:21:2c:e0)
Client hardware address padding: 000000000000000000000000
Server host name not given
Boot file name: Ubuntu-5.10(berlin).pxe
Magic cookie: DHCP
► Option: (t=53,l=1) DHCP Message Type = DHCP Offer
► Option: (t=1,l=4) Subnet Mask = 255.255.255.0
► Option: (t=3,l=4) Router = 10.0.2.2
► Option: (t=6,l=8) Domain Name Server
```

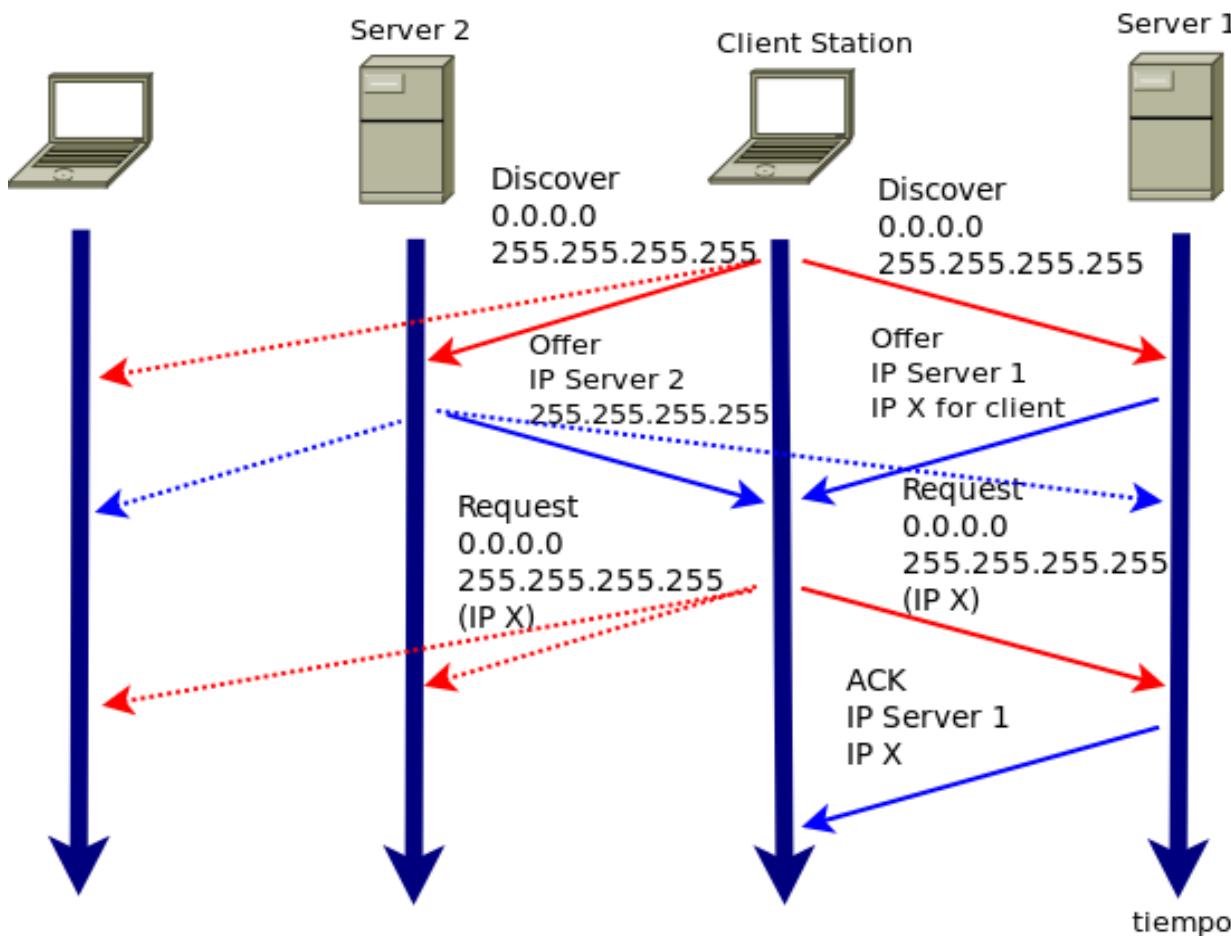
# DHCP Ejemplo

Time	0.0.0.0	255.255.255.255	10.0.2.2	10.0.2.15	CadmusCo_21:2c:e0	Broadcast	RealtekU_12:35:0	Comment
0.000								DHCP: DHCP Discover - Transaction ID 0xb947b252
0.001								DHCP: DHCP Offer - Transaction ID 0xb947b252
0.003								DHCP: DHCP Request - Transaction ID 0xb947b252
0.004								DHCP: DHCP ACK - Transaction ID 0xb947b252
33.667								ARP: Who has 10.0.2.2? Tell 10.0.2.15
33.668								ARP: 10.0.2.2 is at 52:54:00:12:35:02
33.668								DNS: Standard query A www.google.com
33.744								DNS: Standard query response A 74.125.234.116 A 74.125.234.113 A 74.125.234.111
33.744								ICMP: Echo (ping) request id=0xfe0d, seq=1/256, ttl=64
33.837								ICMP: Echo (ping) reply id=0xfe0d, seq=1/256, ttl=63
33.837								DNS: Standard query PTR 116.234.125.74.in-addr.arpa
33.908								DNS: Standard query response PTR gru03s08-in-f20.1e100.net
34.743								ICMP: Echo (ping) request id=0xfe0d, seq=2/512, ttl=64
34.841								ICMP: Echo (ping) reply id=0xfe0d, seq=2/512, ttl=63
34.841								DNS: Standard query PTR 116.234.125.74.in-addr.arpa
34.941								DNS: Standard query response PTR gru03s08-in-f20.1e100.net
35.743								ICMP: Echo (ping) request id=0xfe0d, seq=3/768, ttl=64
35.805								ICMP: Echo (ping) reply id=0xfe0d, seq=3/768, ttl=63
35.805								DNS: Standard query PTR 116.234.125.74.in-addr.arpa
35.850								DNS: Standard query response PTR gru03s08-in-f20.1e100.net
47.002								DHCP: DHCP Request - Transaction ID 0x1368c207
47.003								DHCP: DHCP ACK - Transaction ID 0x1368c207
60.561								ARP: Who has 10.0.2.2? Tell 10.0.2.15
60.561								ARP: 10.0.2.2 is at 52:54:00:12:35:02
60.561								DHCP: DHCP Release - Transaction ID 0xd089ee55

# DHCP Mensajes Broadcast

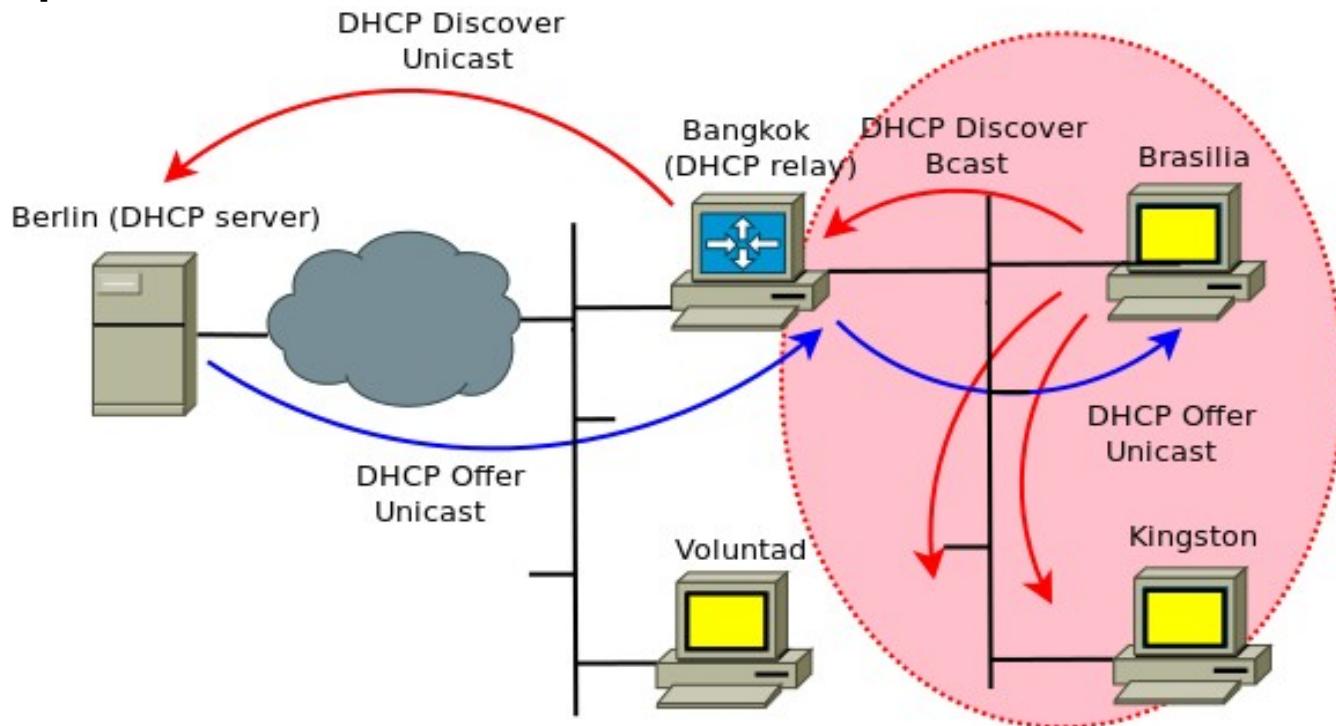
- Broadcast
  - Discover.
  - Request.
- Unicast/Broadcast
  - Offer.
- El Offer en general se envía unicast, pero debido a que pueden existir equipos que no procesan mensajes unicast antes de tener configurada la dirección IP completa, se podrían enviar en forma broadcast.

# DHCP Ejemplo (Offer bcast)



# DHCP Relay

- Los routers pueden funcionar como agentes DHCP Relay y enviar los mensajes de DHCP broadcast de forma unicast a helper (DHCP server).



# Referencias:

- Richard Stevens. TCP/IP Illustrated. Vol 1. The Protocols.
- Douglas Comer. Internetworking with TCP/IP. Vol 1.



# NAT (Network Address Translation)

Fac. Informática - UNLP

# Problemas con IPv4

- IPv4 tiene el espacio de direcciones “casi” agotado.
- Soluciones temporales:
  - CIDR: Tablas de ruteo.
  - DHCP: direcciones escasas, facilidad de administración.
  - NAT: direcciones escasas.
- Solución definitiva:
  - IPv6

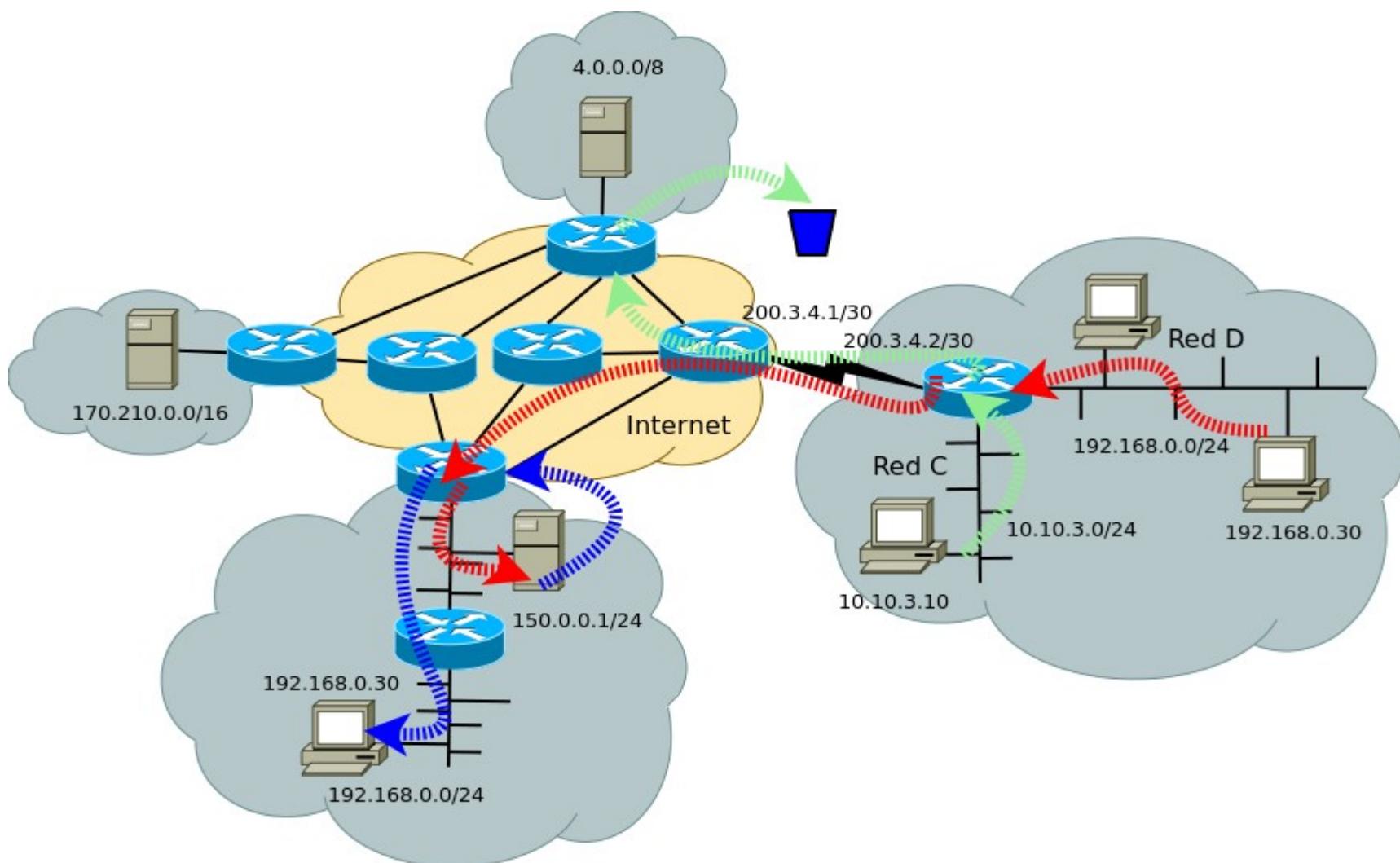
# NAT (Network Address Translation)

- Traslación de direcciones de un espacio privado (**no** “enrutable” en Internet) a un espacio público.
- Direcciones Privadas: RFC-1918:
  - 1 Clase A: 10.0.0.0/8
  - 16 Clases B: 172.16.0.0/12.
  - 256 Clases C: 192.168.0.0/16.
- Proceso definido en **RFC-3022**, hace obsoleta a **RFC-1631**.

# Problemas con IP Privadas

- No son únicas, por lo tanto:
  - Las rutas pueden ser confundidas.
  - Habitualmente son filtradas por routers de borde.
  - Algunos protocolos no funcionan adecuadamente, FTP, VoIP, etc.

# Problemas con IP Privadas



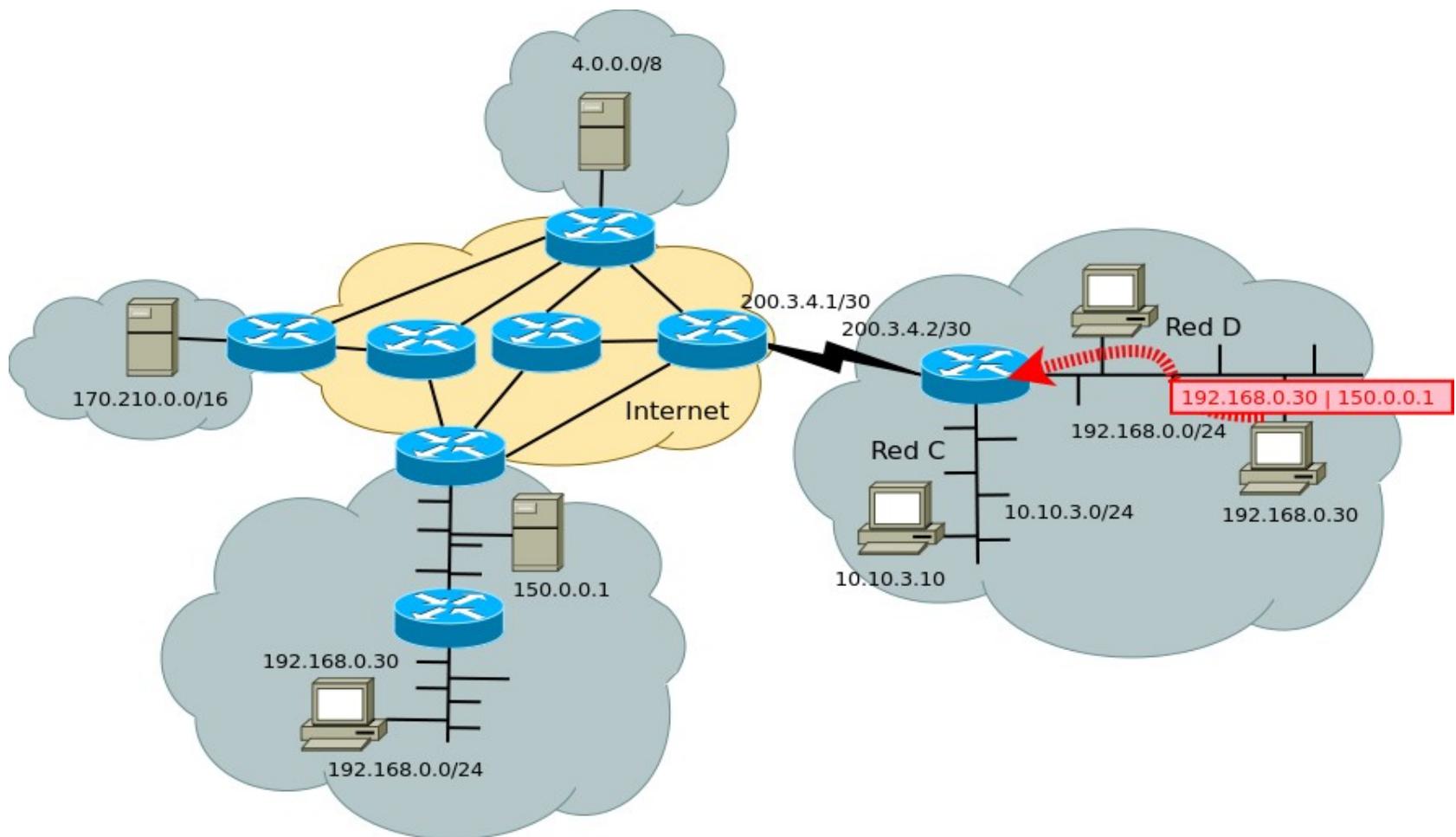
# Procesos de Traslación

- Se realizan sobre redes stubs (solo una salida).
- Se deben mantener tablas de translaciones.
- Varias formas de realizarlo:
- **NAT** (Network Address Translation):
  - Estático.
  - Dinámico.
- **NAPT** (Network Address Port Translation):
  - Dinámico sobre pool.
  - Dinámico sobre dir. overload/masquerade.
- Modificación de direcciones, ports, checksums.

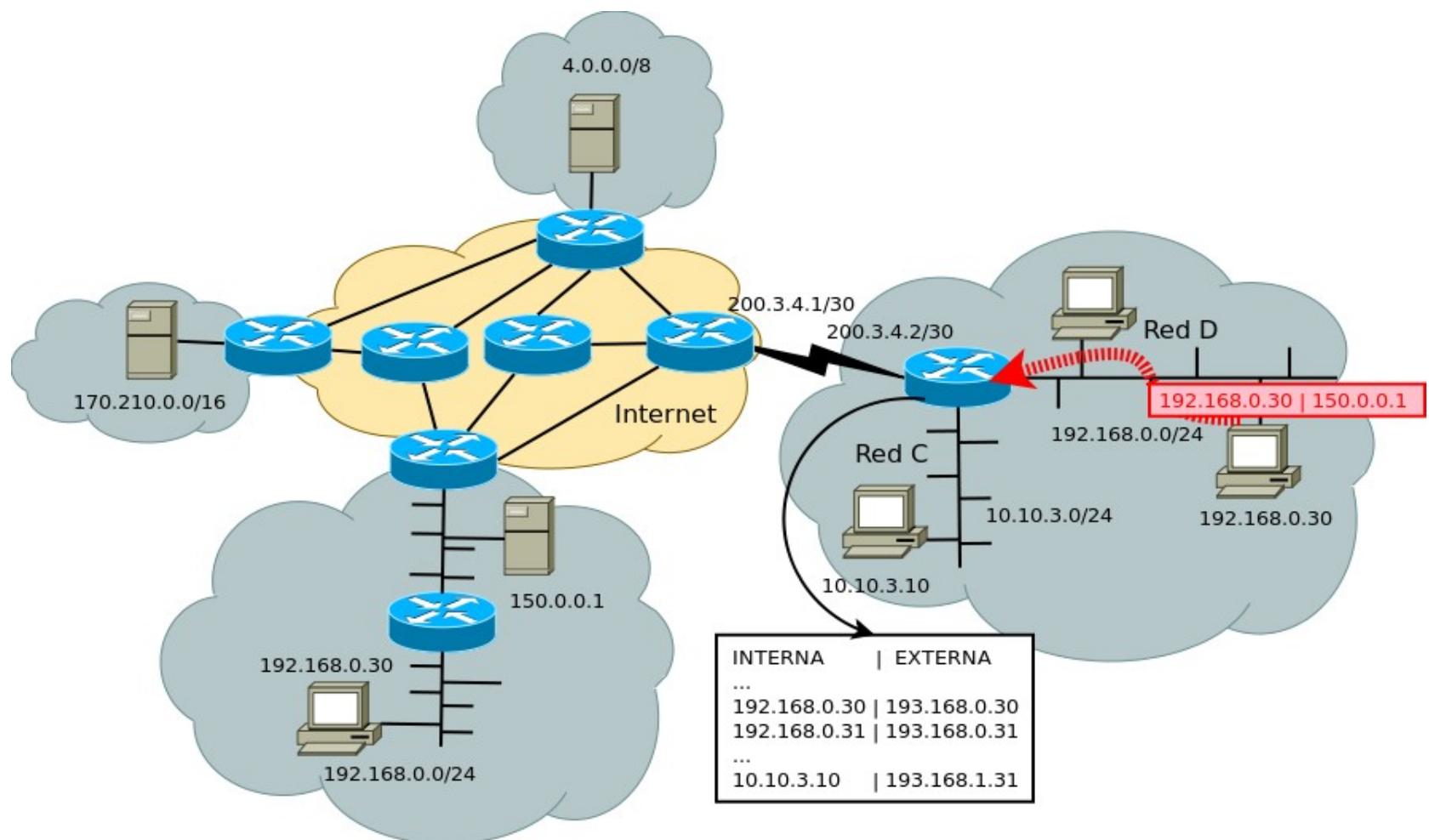
# NAT (NAT básico)

- Una forma de realizarlo es: “**one-to-one**” (uno a uno), **NAT básico**:
- Se mapea una dirección IPv4 privada a una dirección IPv4 pública.
- Si se hace de forma **estática** requiere tantas direcciones públicas como privadas.
- Permite acceso en ambas direcciones.
- Si se hace de forma **dinámica** no es necesario, pero sí se requiere un timer por cada entrada. Limita acceso simultaneo de acuerdo al pool pub.

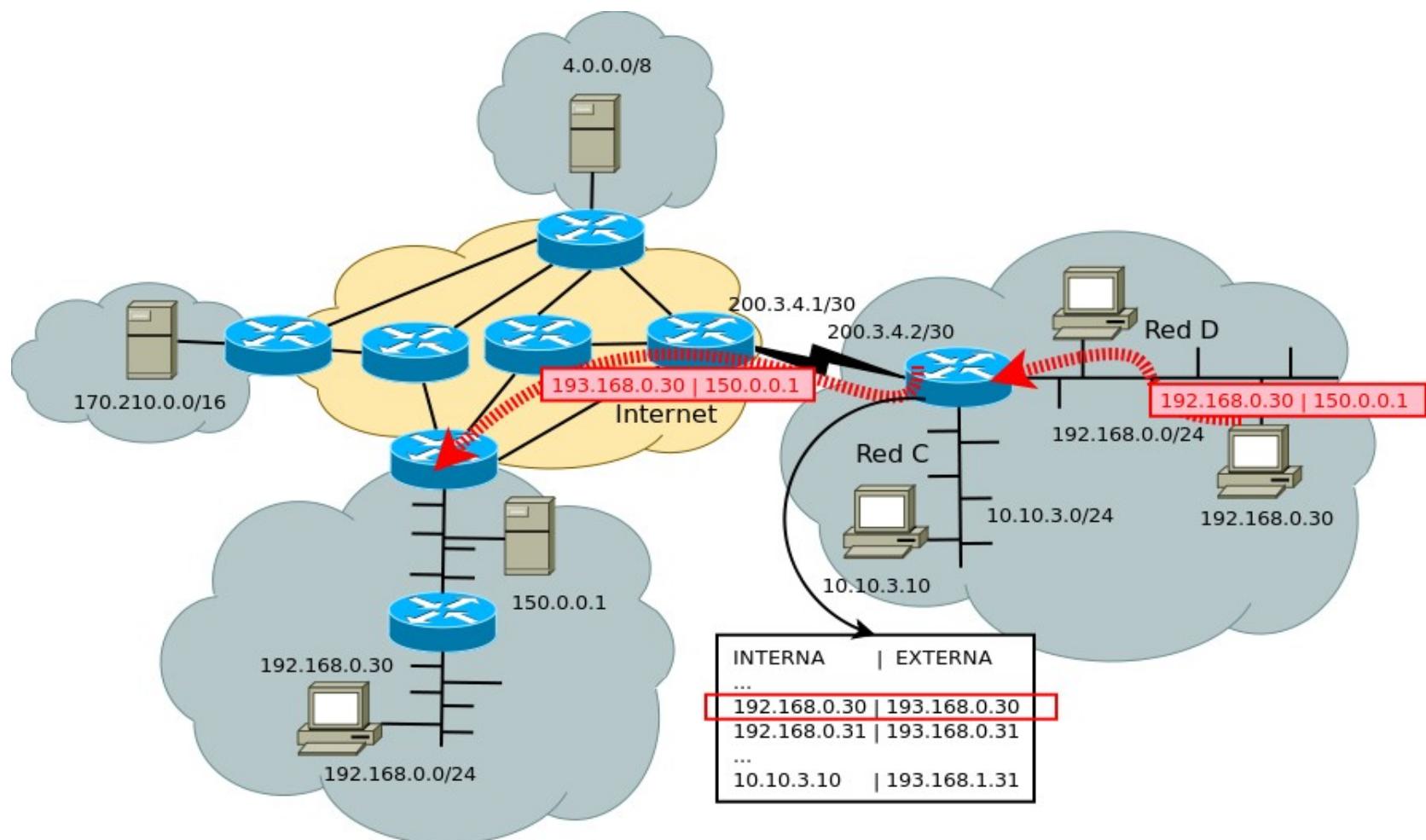
# Ejemplo NAT estático (1)



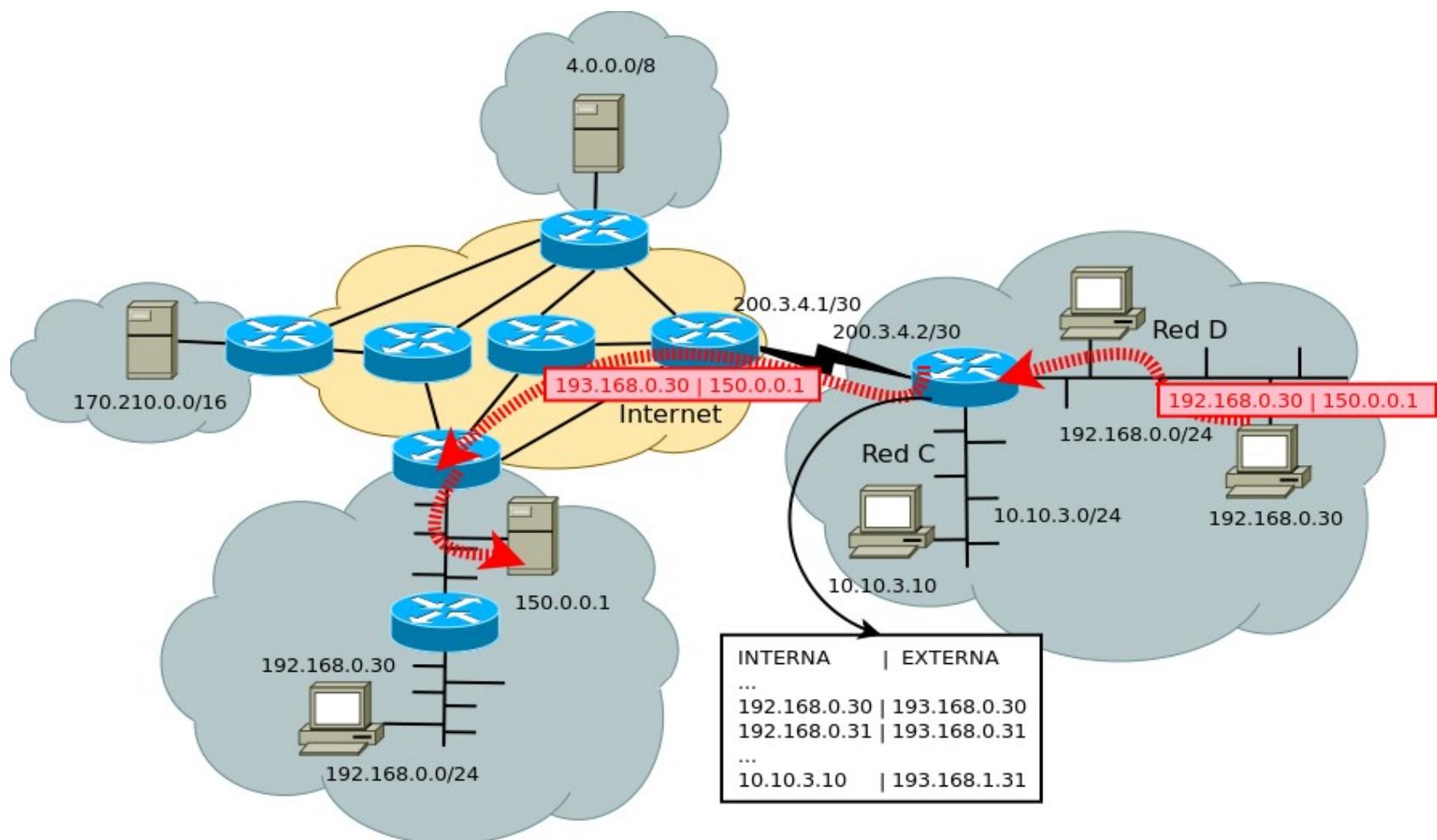
# Ejemplo NAT estático (2)



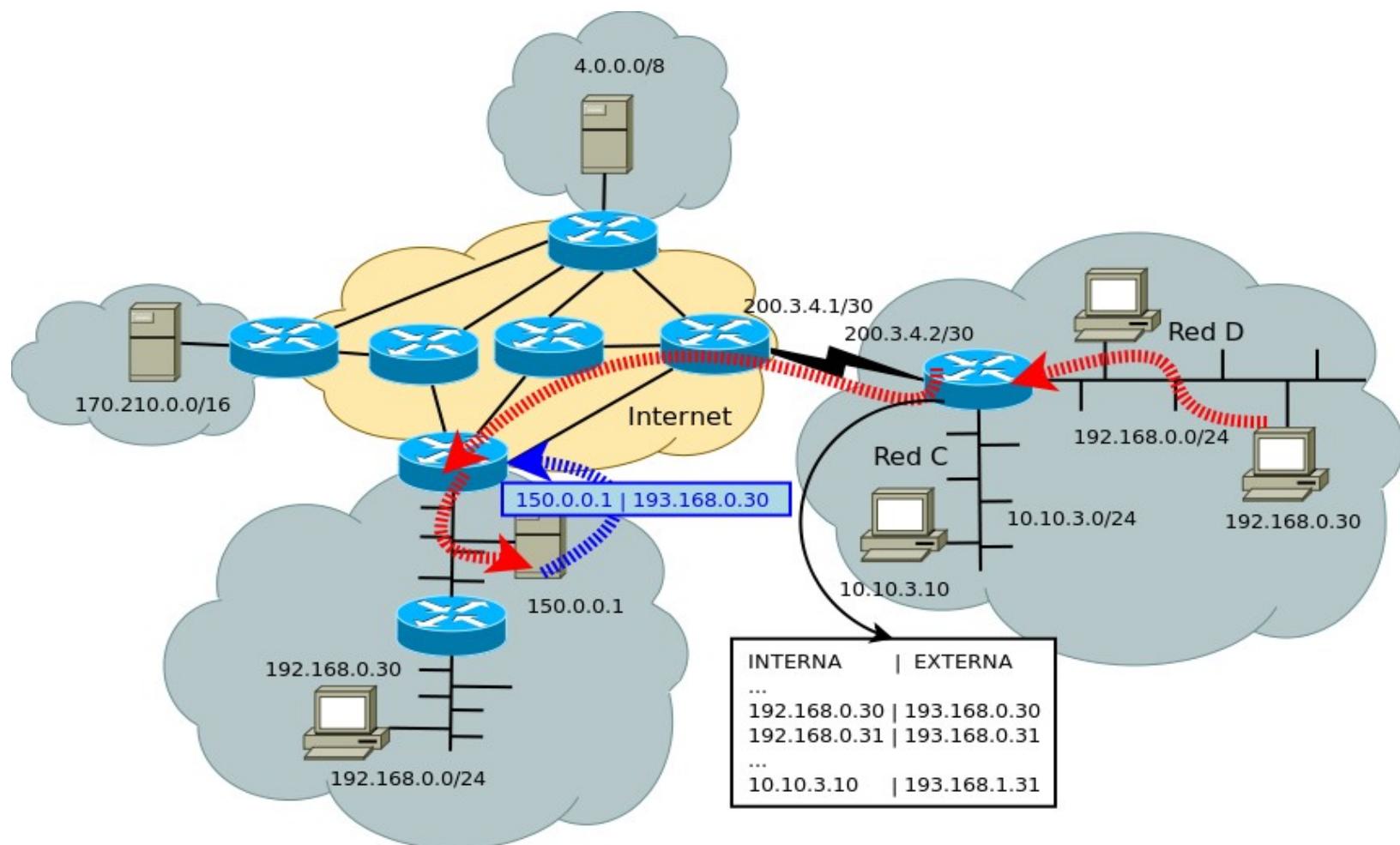
# Ejemplo NAT estático (3)



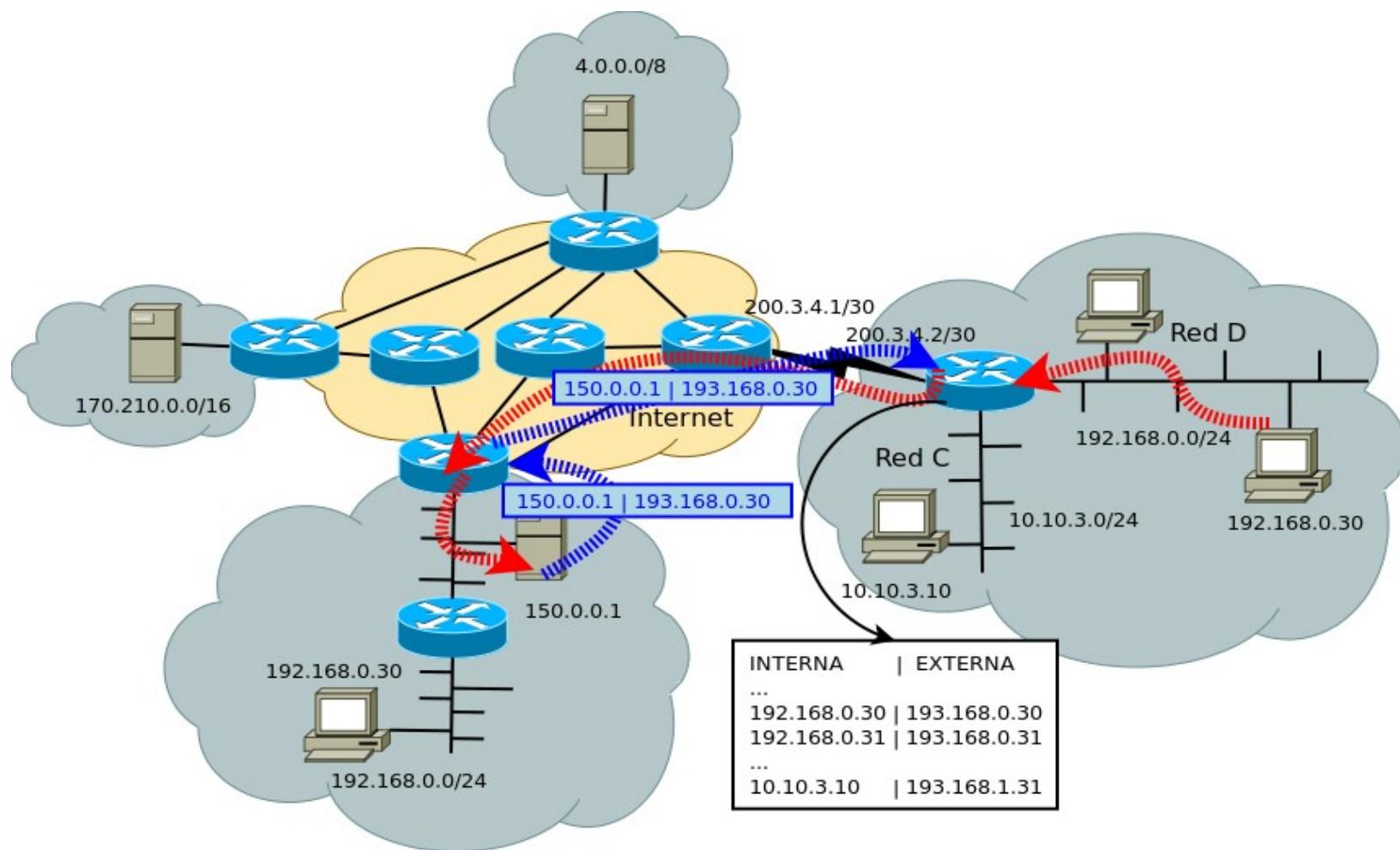
# Ejemplo NAT estático (4)



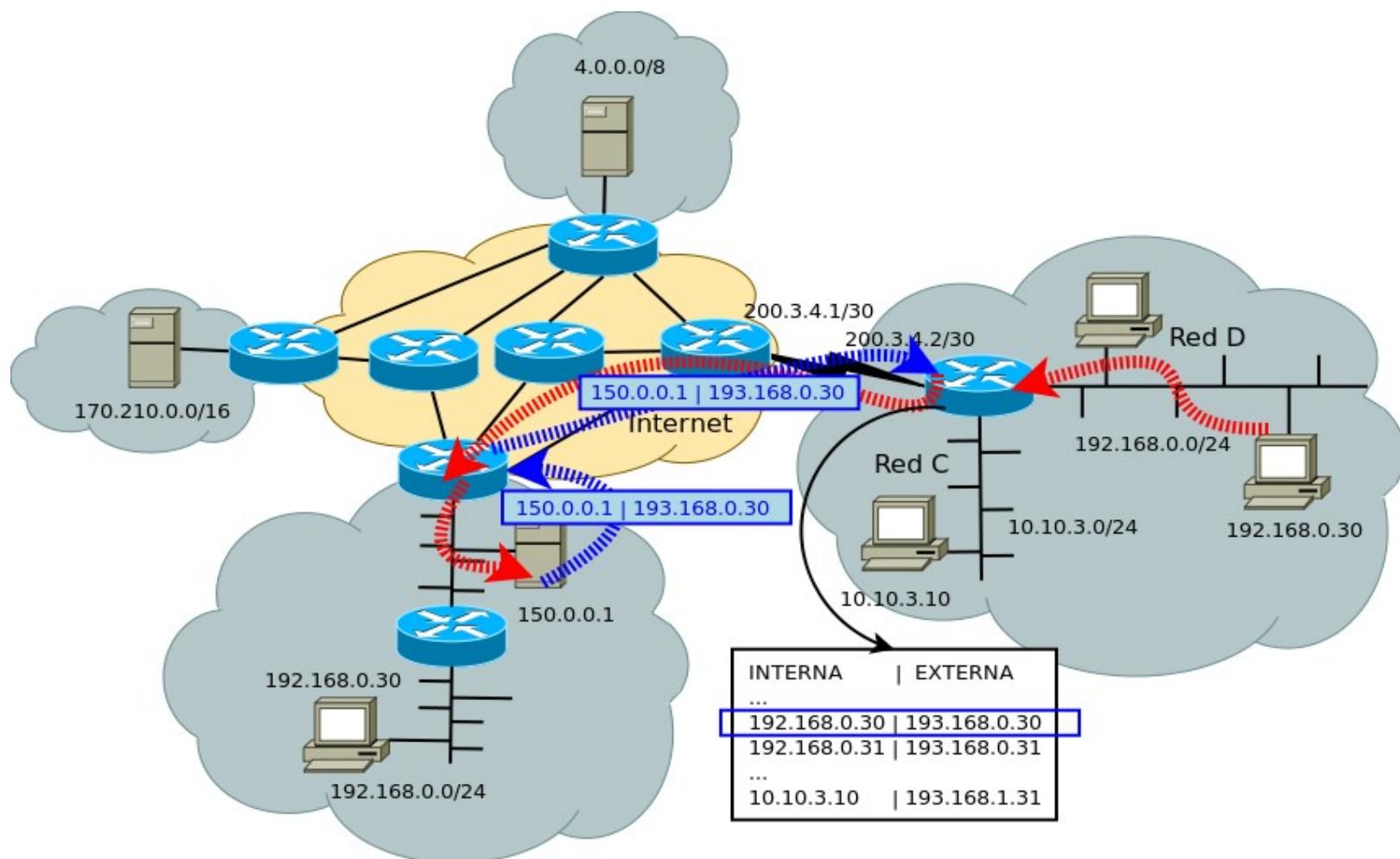
# Ejemplo NAT estático (5)



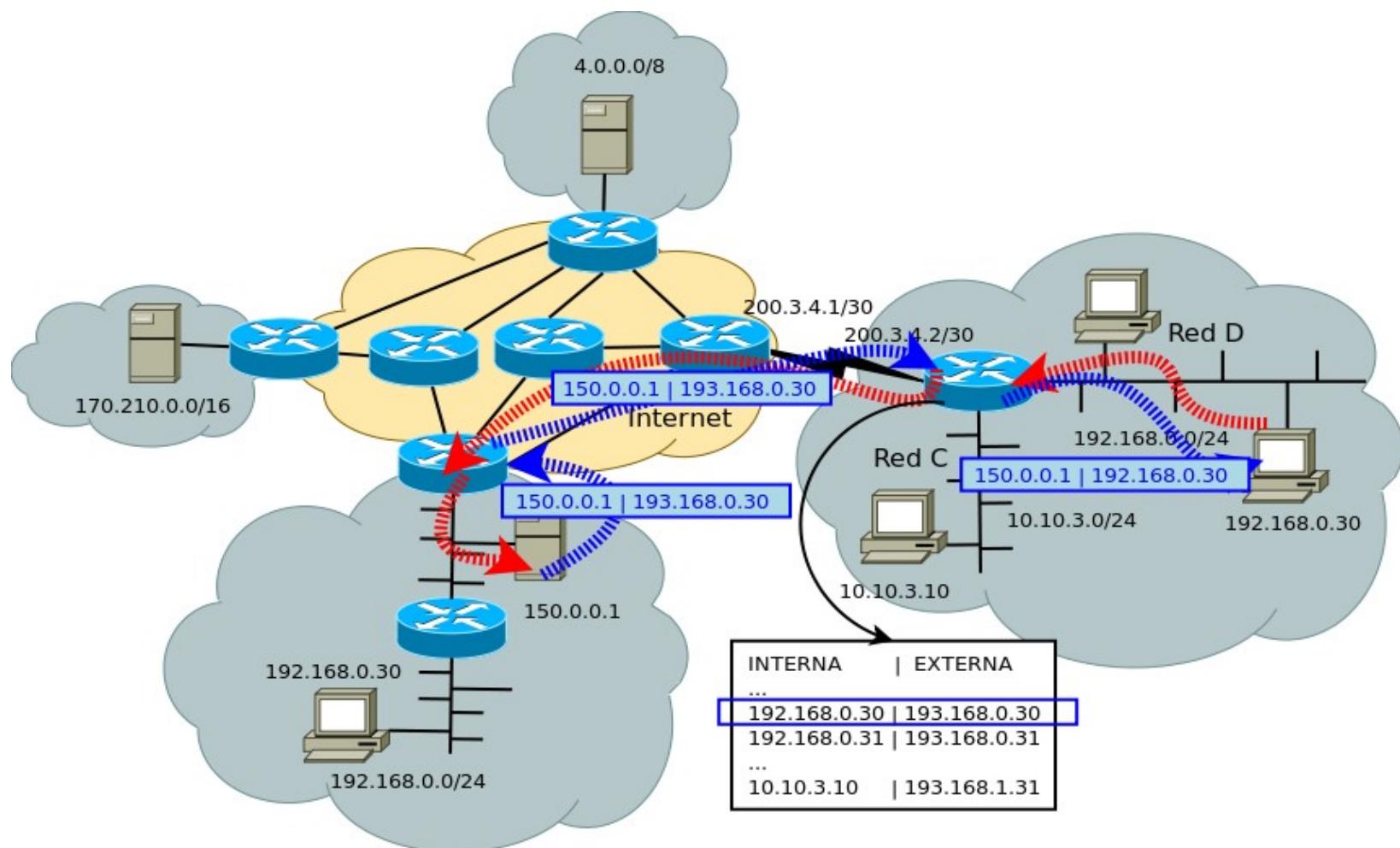
# Ejemplo NAT estático (6)



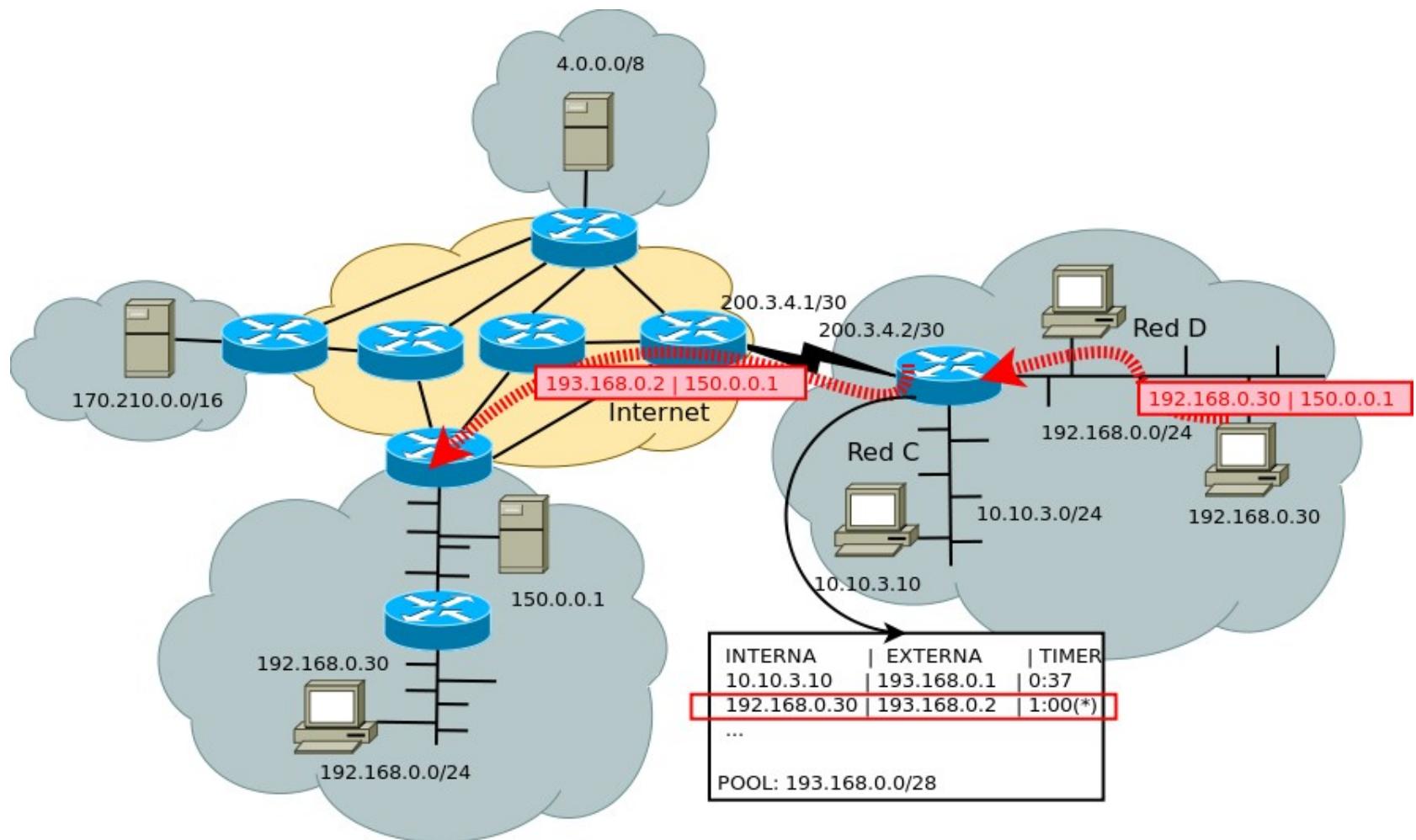
# Ejemplo NAT estático (7)



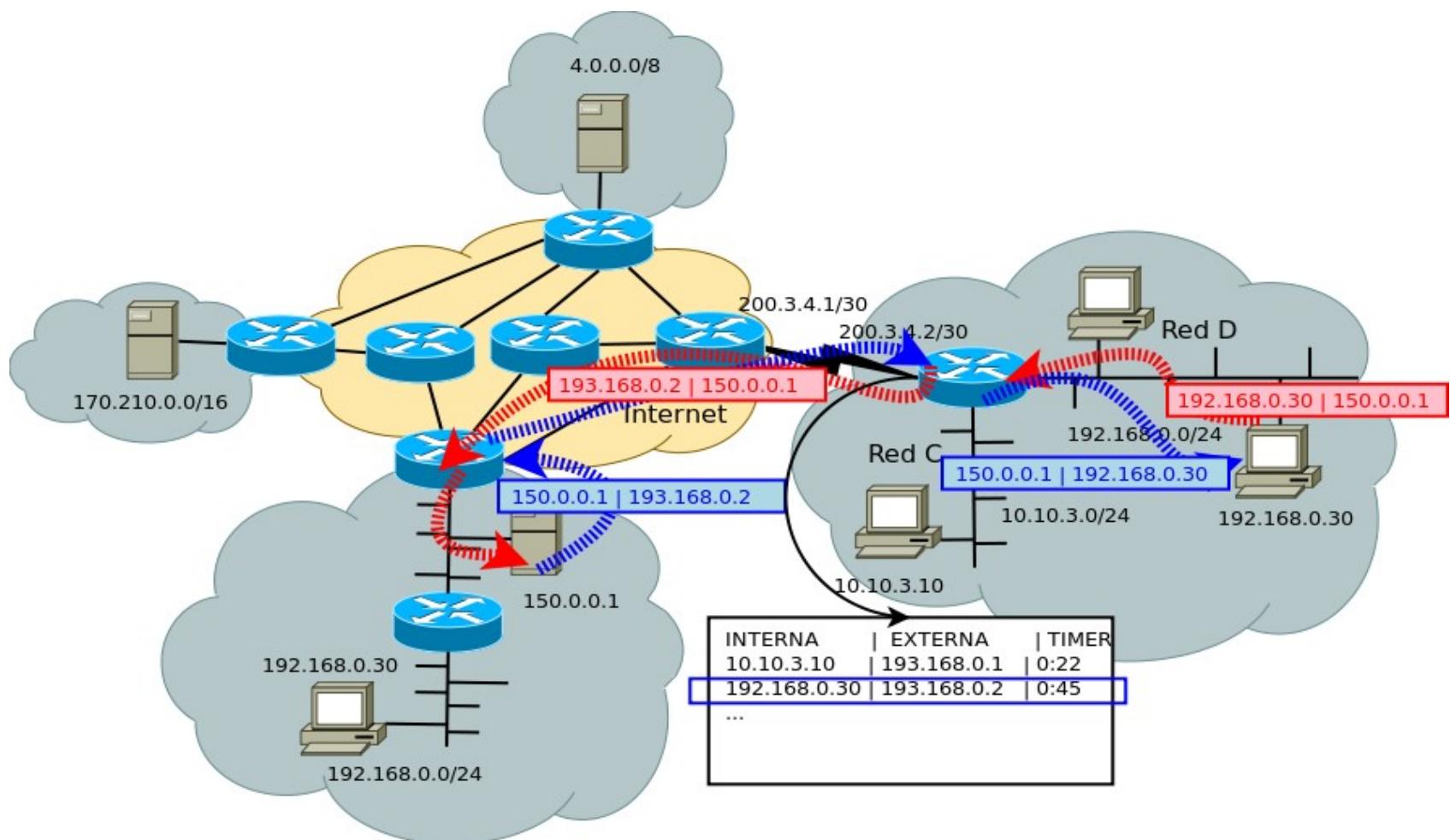
# Ejemplo NAT estático (8)



# Ejemplo NAT dinámico (1)



# Ejemplo NAT dinámico (2)



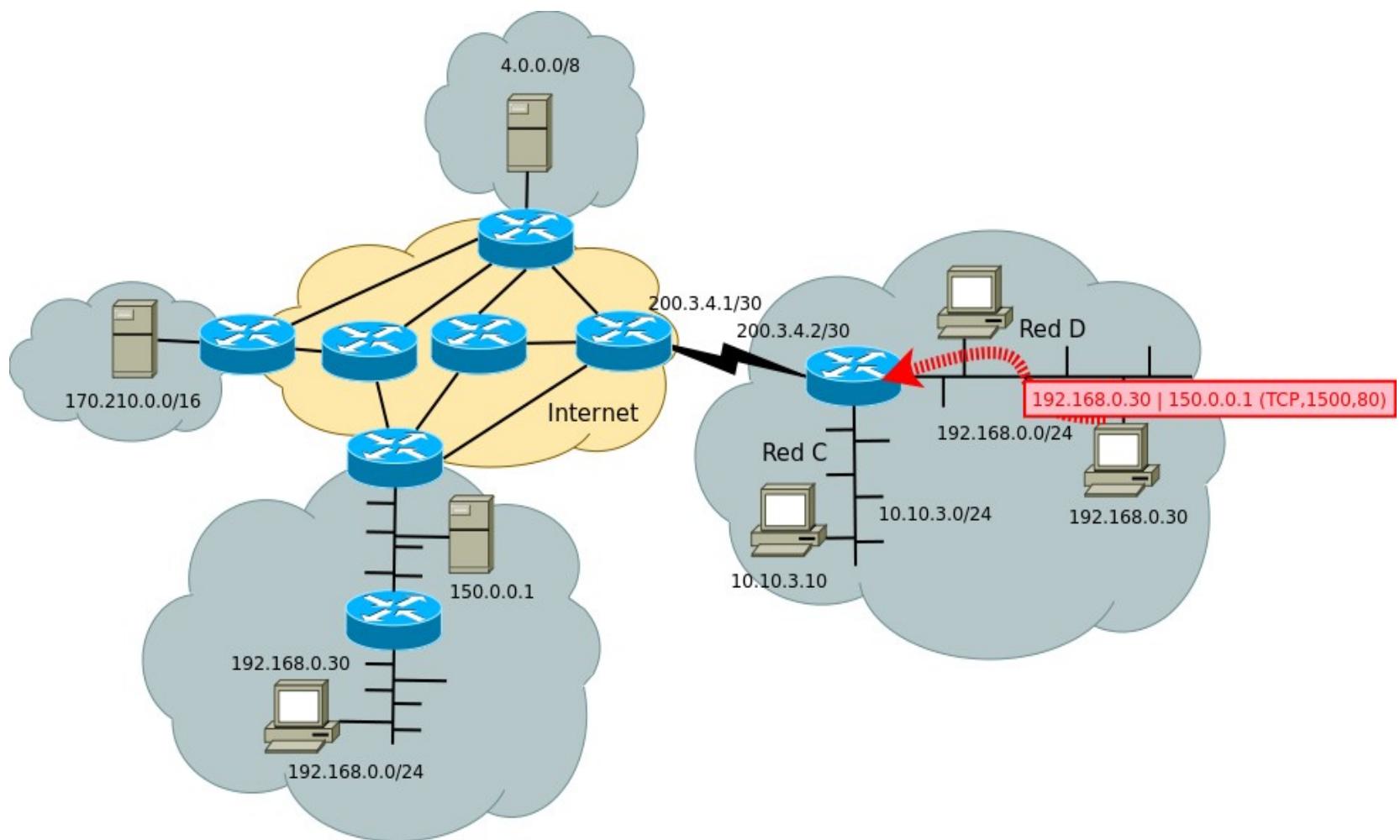
# NAPT (Network Address Port Translation)

- NAT no es implementable cuando se tiene un **pool chico** de direcciones o no se posee direcciones publicas asignadas.
- En ese caso se debe trabajar con campos de la capa de transporte o del payload.
- **NAPT** es conocido como **PAT (Port Address Translation)**: “**one-to-many**”.
- Se utilizan los **puertos** de los protocolos u otros valores como ICMP **Identifier** para resolver el mapeo.
- Se pueden usar timers y sesión del protocolo.

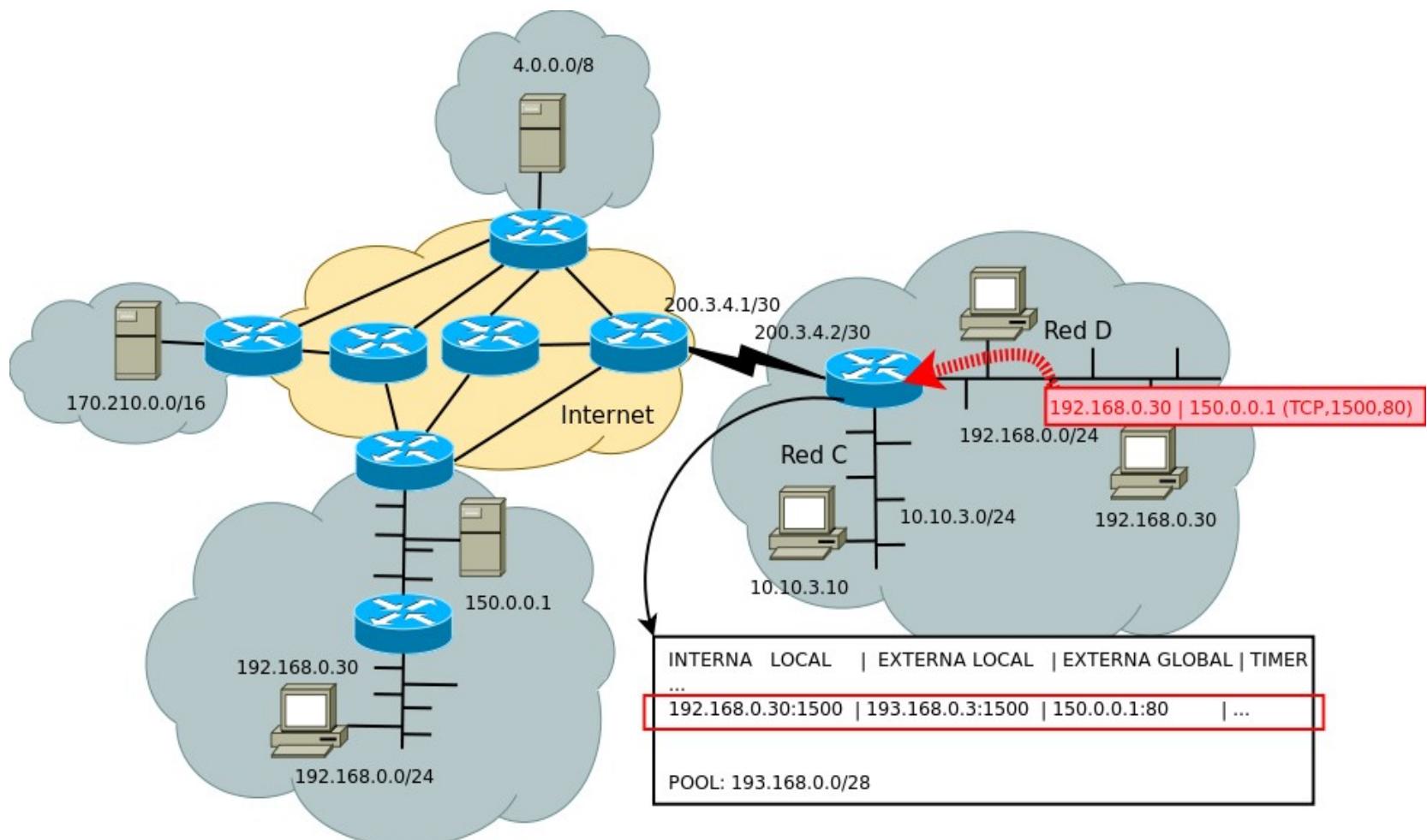
# NAPT (Network Address Port Translation)

- En la tabla de translaciones se mantienen el **protocolo y los puertos origen y destino**.
- Se intenta conservar el puerto origen, pero si esta “ocupado” se debe reemplazar por otro.
- El dispositivo debe “violar” los límites impuestos por la división en capas.
- Dos alternativas:
  - Utilizando un pool y haciendo PAT sobre este.
  - Utilizando la dir. IP externa y haciendo **overloading/masquerading** sobre esta.

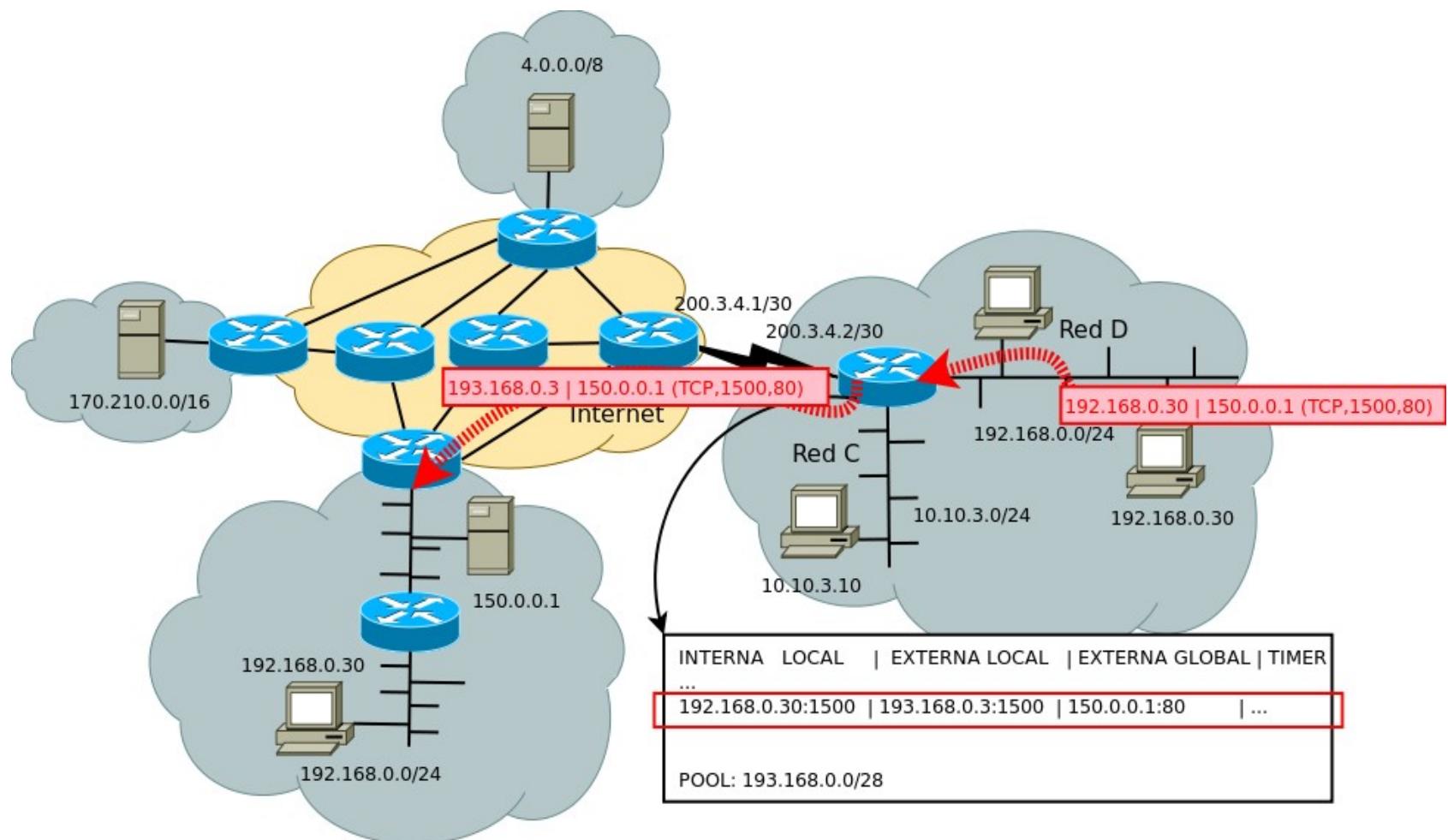
# Ejemplo de NAPT (Pool) (1)



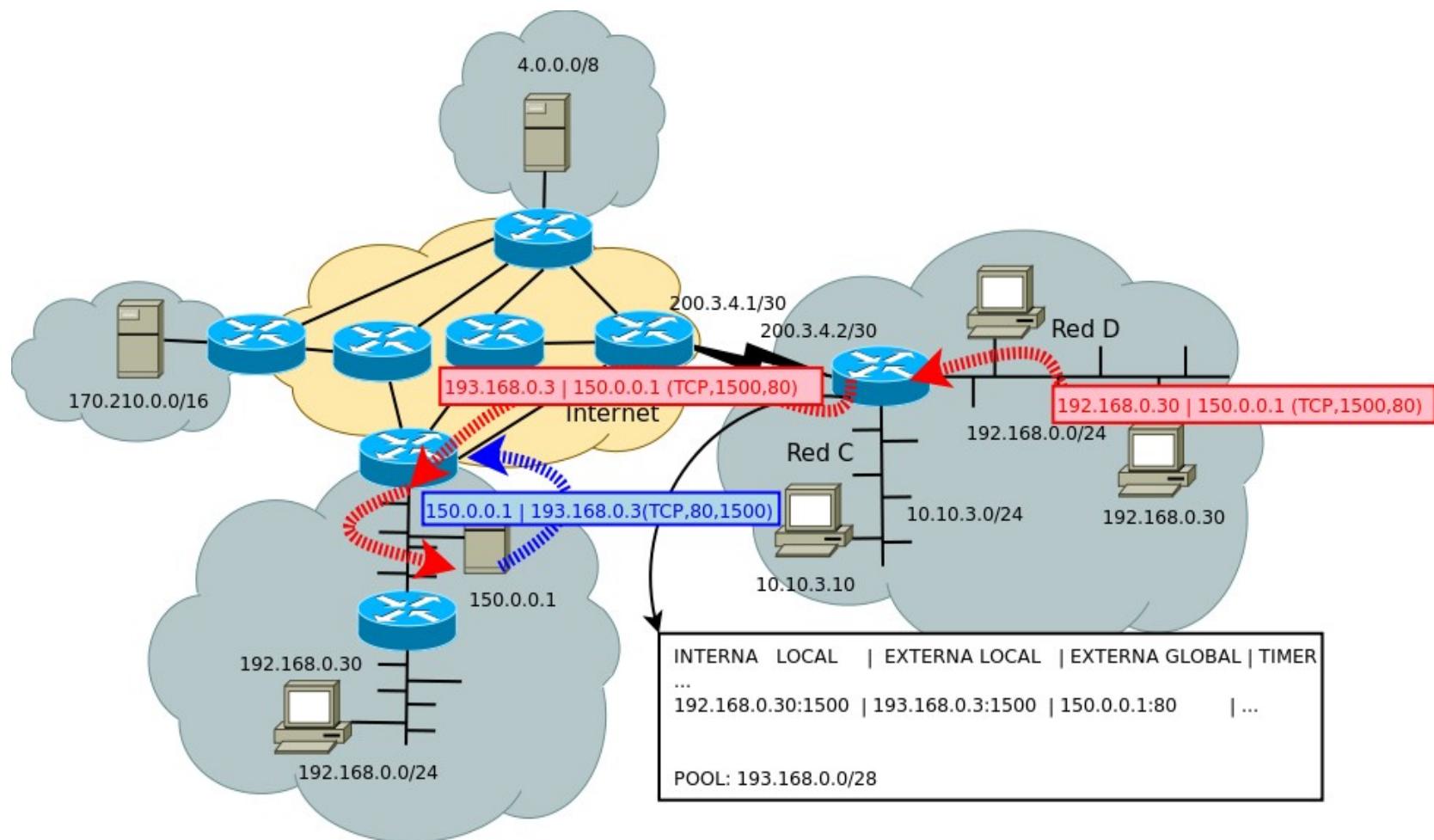
# Ejemplo de NAPT (Pool) (2)



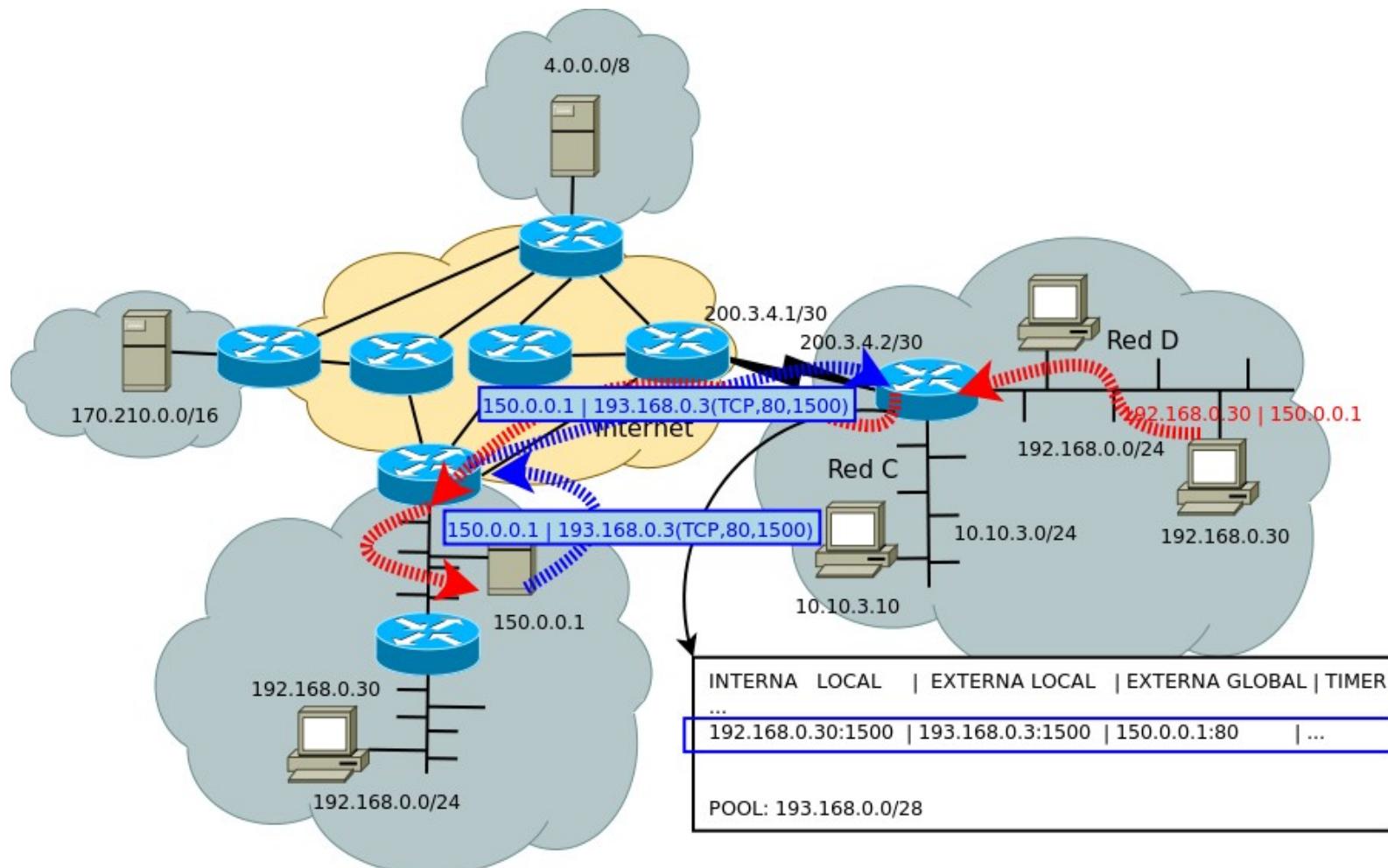
# Ejemplo de NAPT (Pool) (3)



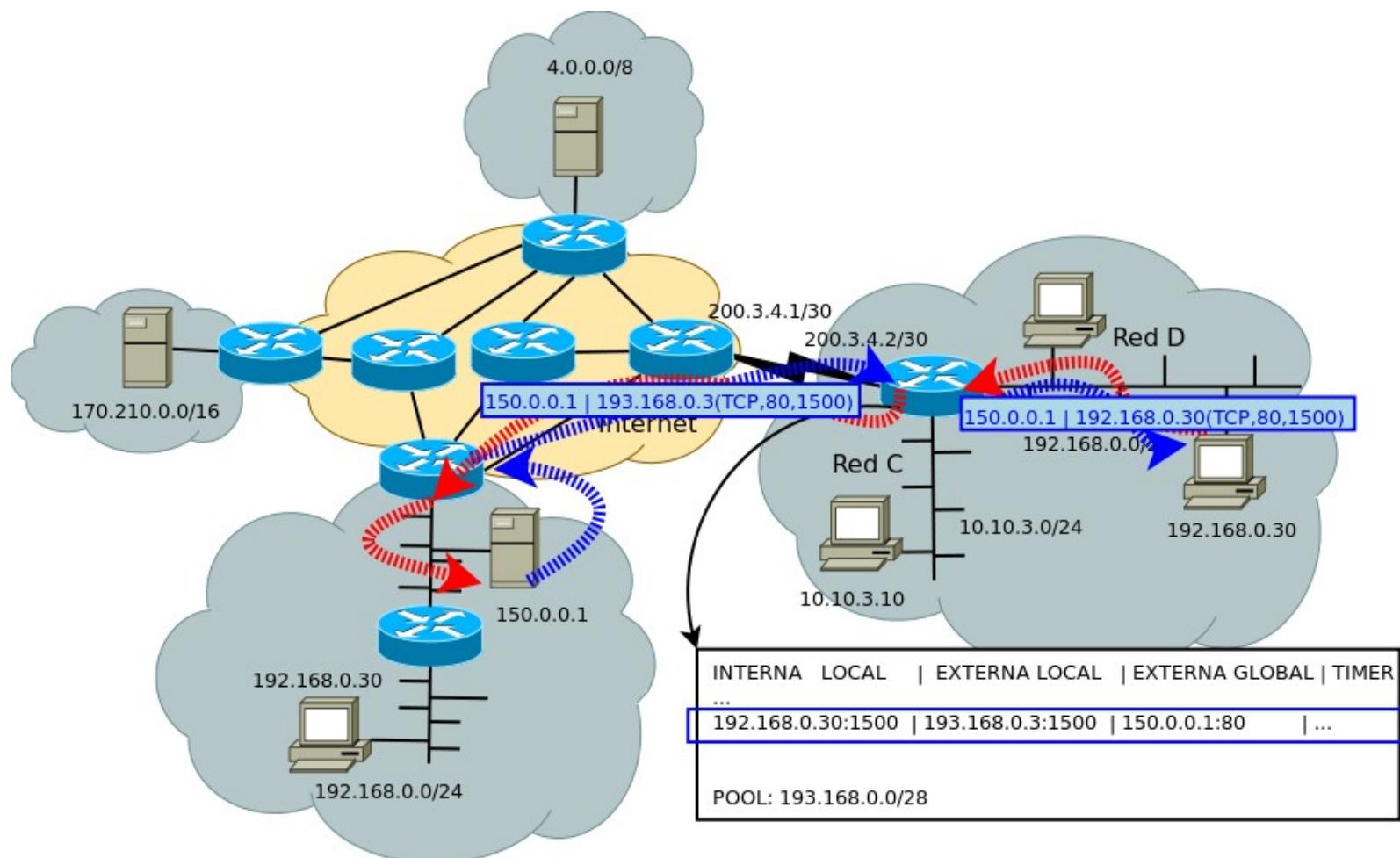
# Ejemplo de NAPT (Pool) (4)



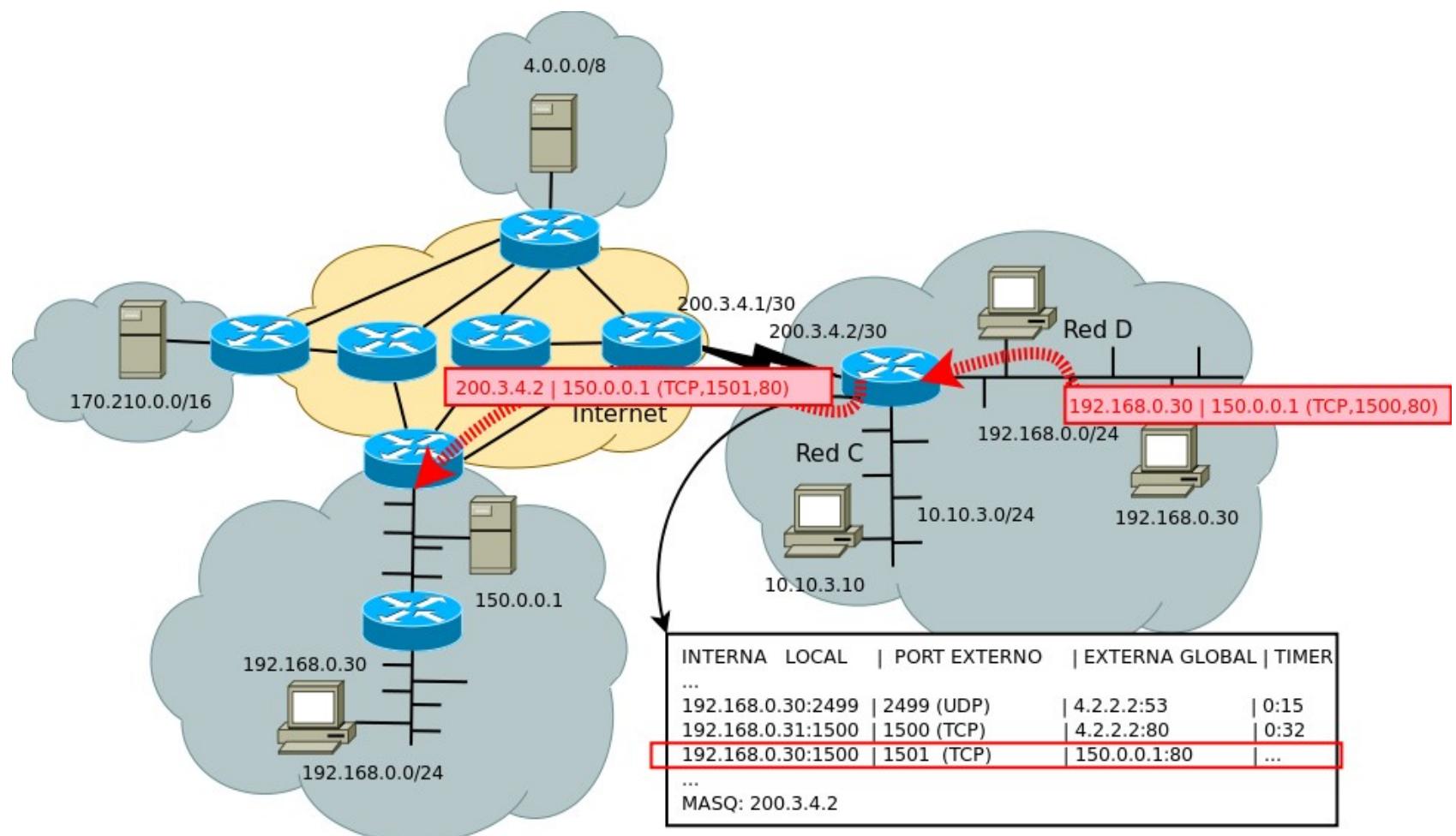
# Ejemplo de NAPT (Pool) (5)



# Ejemplo de NAPT (Pool) (6)



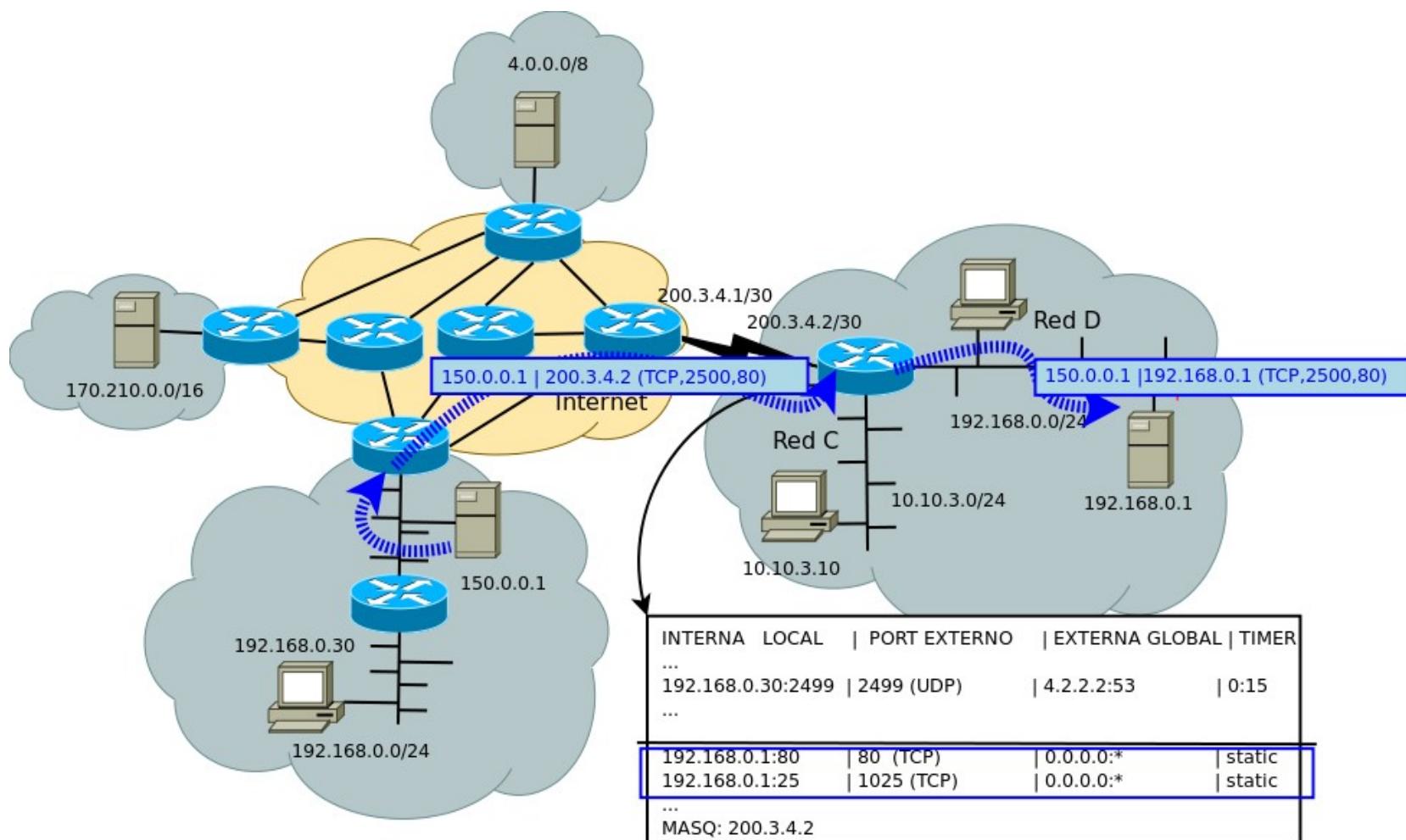
# Ejemplo de NAPT (OVERLOAD/MASQ)



# Port Forwarding

- Overloading/Masq no permiten acceso desde “afuera” hacia “adentro”.
- Solo se permiten entrar tráfico de conexiones generadas internamente.
- Mediante **Port Forwarding** (Re-envío de puerto) se permite poder tener servicios en una red privada accesibles desde “afuera”.
- No se requiere NAT estático, se implementa con NAPT y mapeo reverso estático de puertos.

# Ejemplo de Port Forwarding



# Conclusiones

- NAT/NAPT resuelve temporalmente la escasez de direcciones Ipv4.
  - Algunos servicios no funcionan.
- Da una “sensación” de seguridad, aunque no siempre es verdad.
- Se pierde la idea de IP end-to-end.
- Firewalls más complejos.

# Referencias:

- Kurose/Ross: Computer Networking (5th Edition).
- Cisco CCNAv3.1.
- Cisco CCNA v4.0 exploration.
- RFC-1918 - Address Allocation for Private Internets.
- RFC-3022 - Traditional IP Network Address Translator (Traditional NAT).
- RFC-2663 - IP Network Address Translator (NAT) Terminology and Considerations

# IPv6 (Internet Protocol version 6) (parte I)

2020

Facultad de  
Informática



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Contenidos

1

Comutación de Paquetes

2

Revisión de IPv4

3

Introducción a IPv6

- Generalidades de IPv6
- Direccionamiento IPv6
- Ruteo IPv6

4

Referencias

# Contenidos

1

Comutación de Paquetes

2

Revisión de IPv4

3

Introducción a IPv6

- Generalidades de IPv6
- Direccionamiento IPv6
- Ruteo IPv6

4

Referencias

# Contenidos

1

Comutación de Paquetes

2

Revisión de IPv4

3

Introducción a IPv6

- Generalidades de IPv6
- Direccionamiento IPv6
- Ruteo IPv6

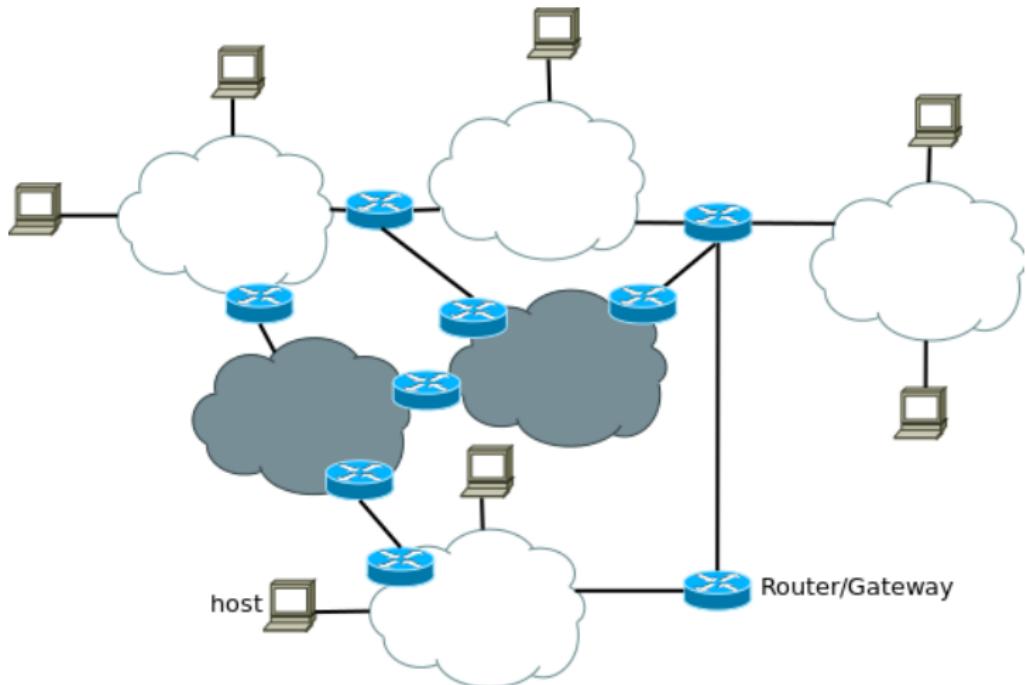
4

Referencias

# Contenidos

- 1 Comutación de Paquetes
- 2 Revisión de IPv4
- 3 Introducción a IPv6
  - Generalidades de IPv6
  - Direccionamiento IPv6
  - Ruteo IPv6
- 4 Referencias

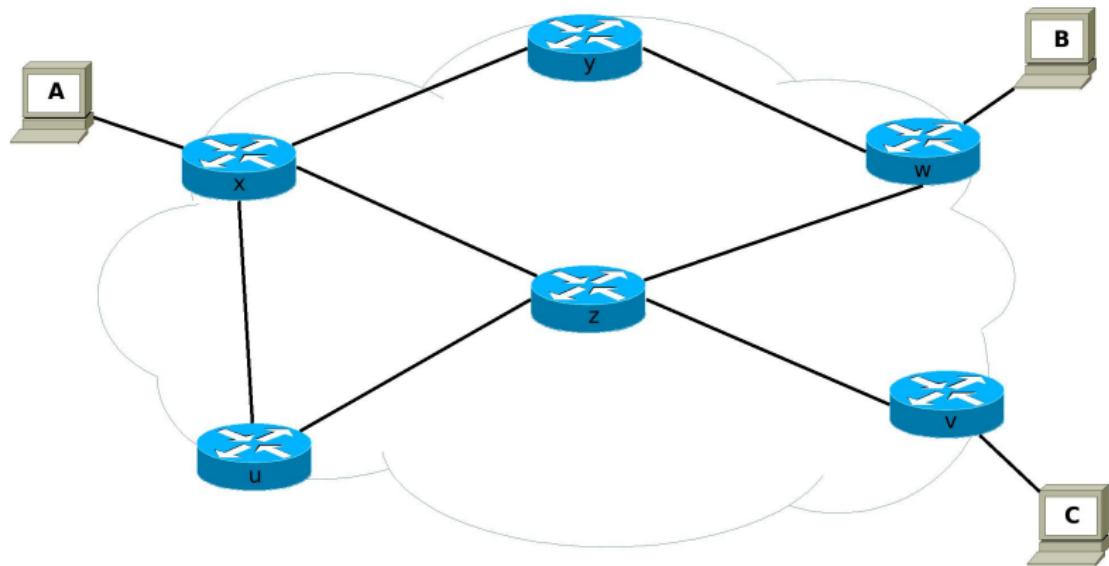
# Red de Redes



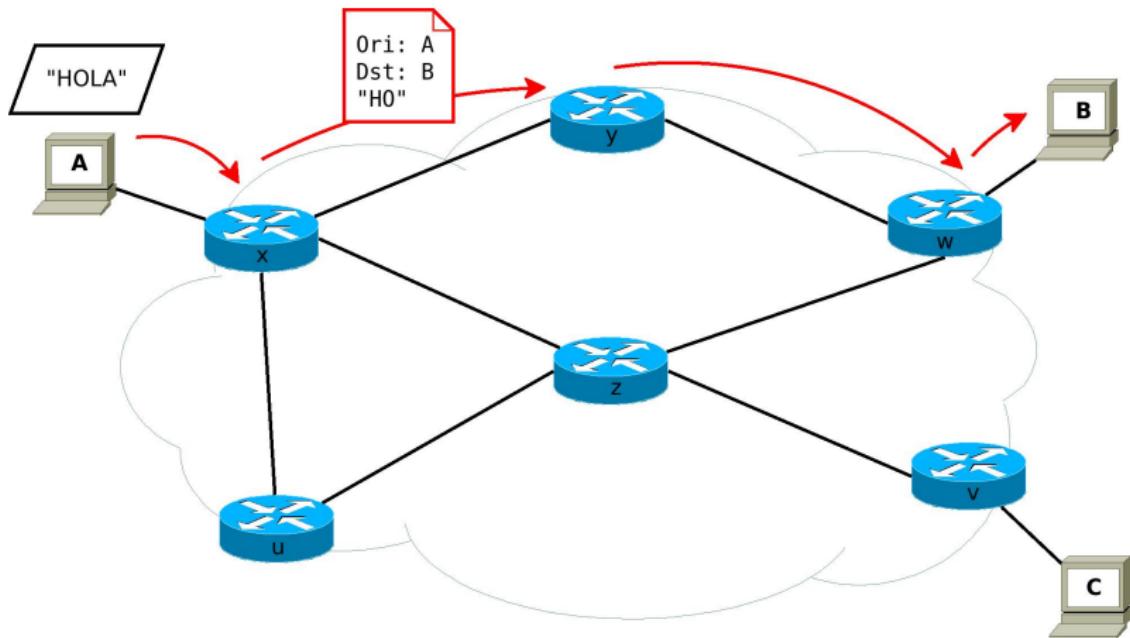
# Comutación de Paquetes

- Modelo de Red, L3: Comutación de Paquetes.
- Cada PDU: Unidad de datos, datagrama/paquete puede ser “transportado” por la red de forma independiente.
- Los datagramas tiene información en su encabezado para ser “manejados” por los dispositivos intermedios (la red).
- Componentes/dispositivos intermedios de la red: comutadores/routers/gateways.
- Routers trabajan básicamente en Store & Forward y se interconectan entre sí físicamente (a nivel de enlace).
- El ruteo se produce, hop-by-hop (salto a salto).
- Modelo best-effort, más flexible y eficiente.

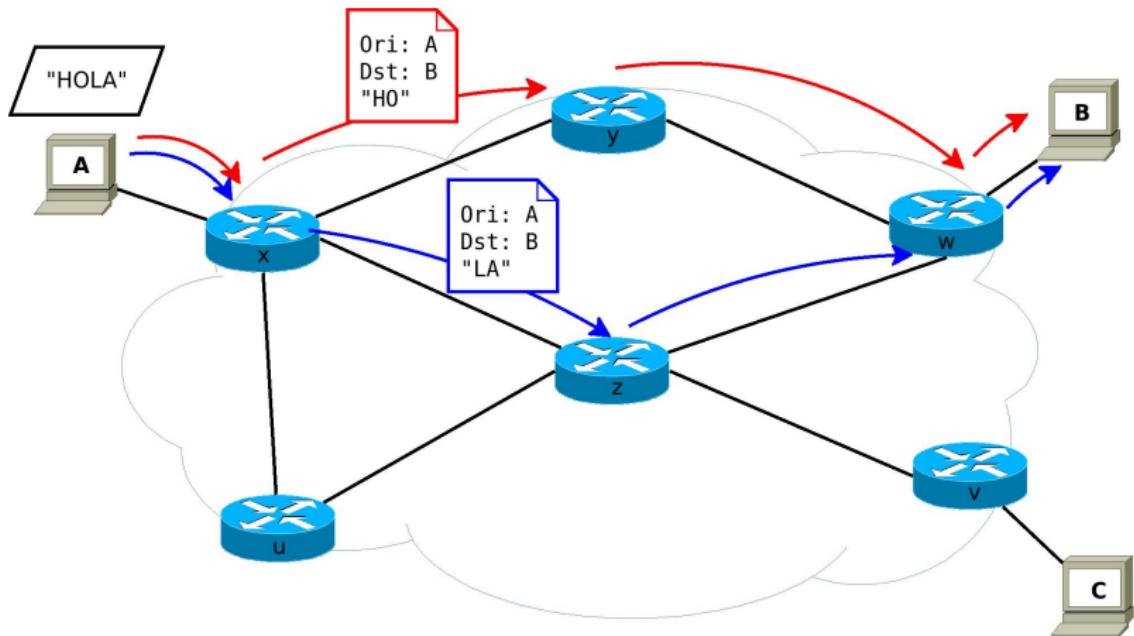
# Redes de Comunicación de Paquetes (cont.)



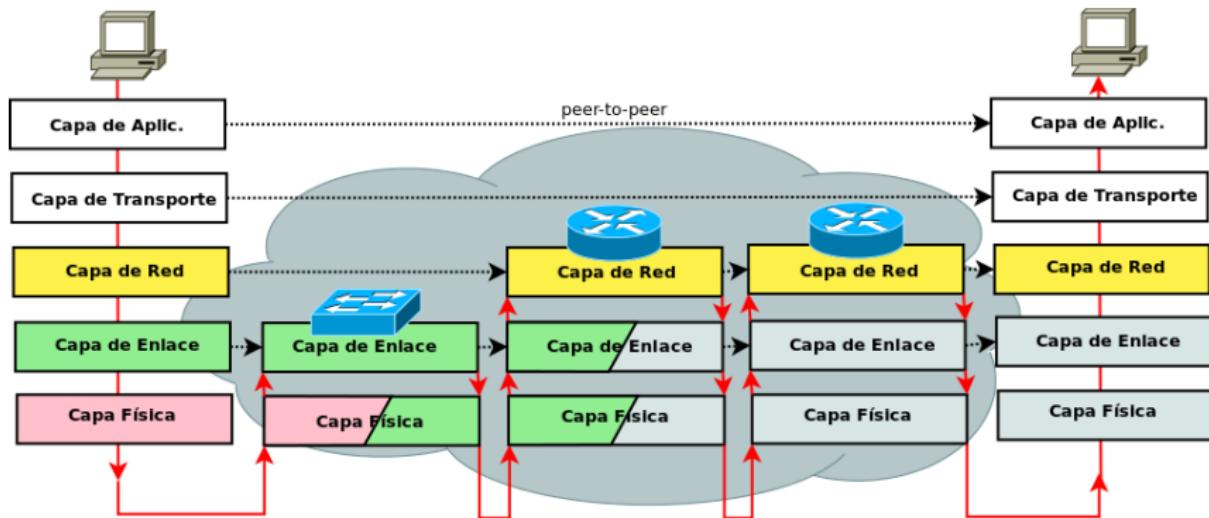
# Redes de Comutación de Paquetes (cont.)



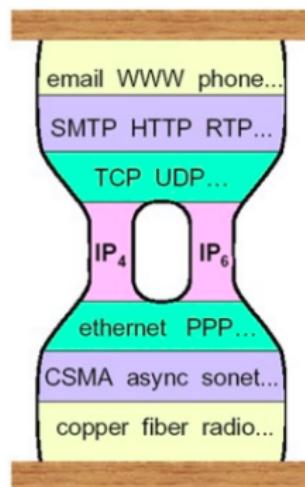
# Redes de Comutación de Paquetes (cont.)



# Comunicación entre Capas Peer-Peer



# Modelo de Internet ?

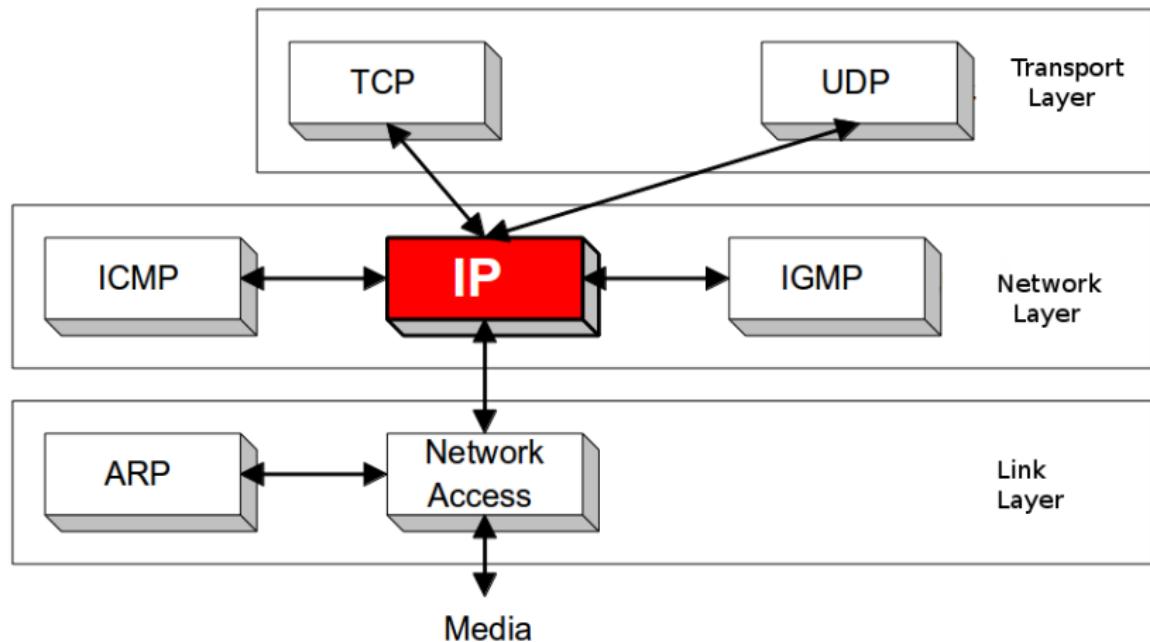


fuente: <https://www.ietfjournal.org/>

# Características de IPv4

- Protocolo de Red no orientado a conexión.
- Protocolo de Mejor Esfuerzo: best-effort, no confiable (no asegura el arribo de los mensajes).
- PDU: datagrama o paquete.
- Definido RFC 791 (STD-5).
- Funcionalidad:
  - Direccionamiento.
  - Ruteo/Forwarding.
  - Mux/Demux de protocolos superiores.
  - Fragmentación.
  - Otras: como evitar loops.
  - Detección de errores en header.

# Esquema de IPv4

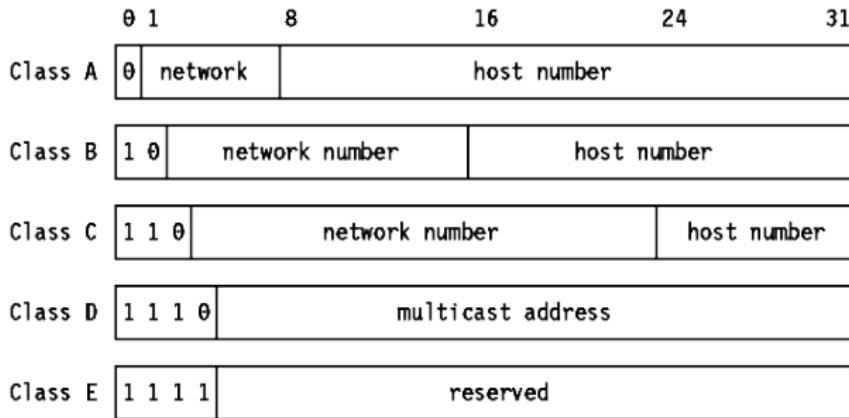


# Formato cabecera IPv4

Ver.	header	TOS	total length			
<b>identification</b>		<b>flag</b>		<b>fragment offset</b>		
<b>TTL</b>		<b>Protocol</b>	<b>Checksum</b>			
<b>32 bit Source Address</b>						
<b>32 bit Destination Address</b>						

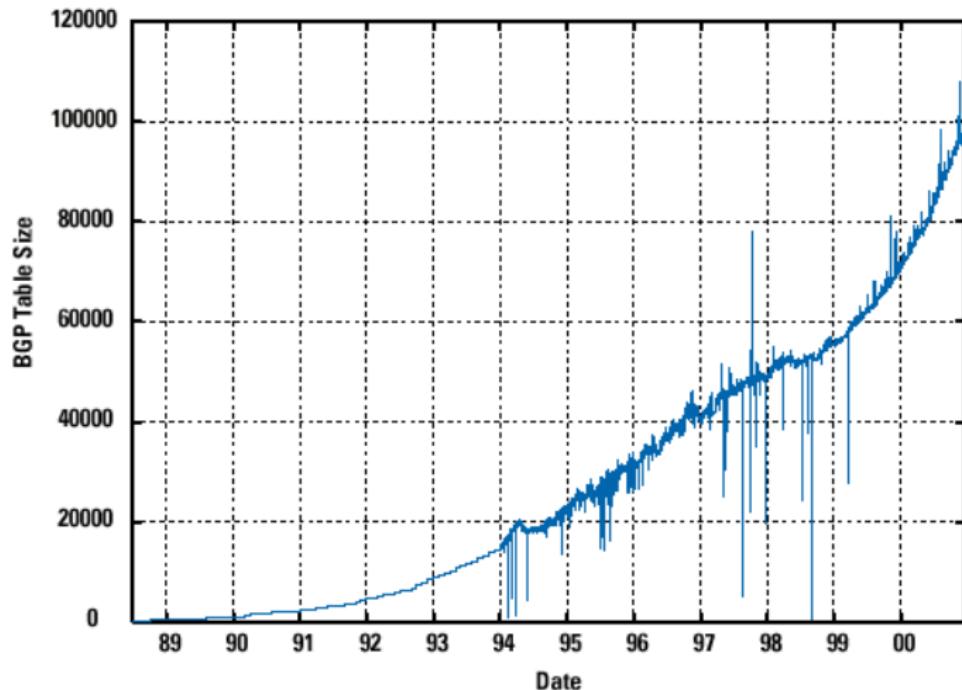
# Direcciones IPv4

- Inicialmente Classful.



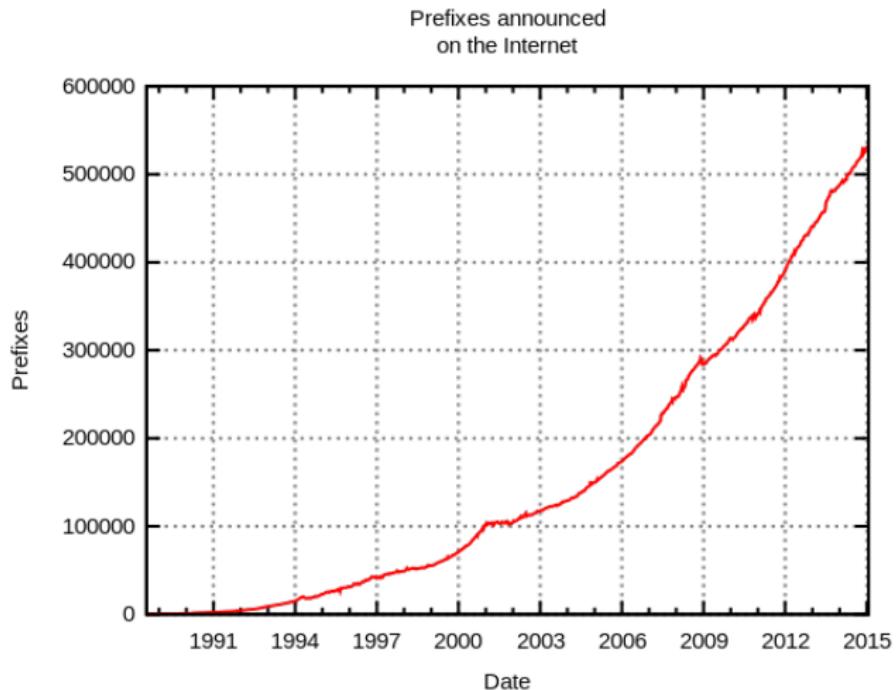
- Máscara de red, long. de prefijo: subnetting:  
192.168.30.3 255.255.255.0 == 192.168.30.3/24.
- CIDR (Classless Inter Domain Routing).

# Crecimiento Internet



fuente: [http://www.cisco.com IPJ v4n1 \(Analyzing the Internet BGP Routing Table\)](http://www.cisco.com IPJ v4n1 (Analyzing the Internet BGP Routing Table))

# Crecimiento Internet (cont.)

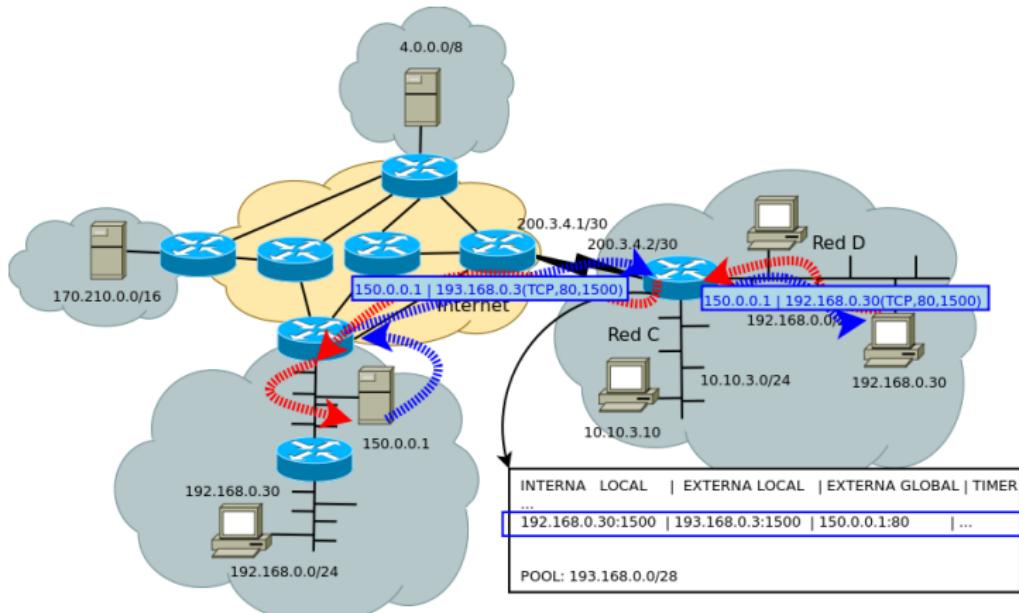


fuente: <https://commons.wikimedia.org> basado en [www.potaroo.net](http://www.potaroo.net)

# Direcciones IPv4 Privadas y NAT/NAPT

- Direcciones IPv4 Privadas: RFC-1918:
  - 1 Clase A: 10.0.0.0/8.
  - 16 Clases B: 172.16.0.0/12.
  - 256 Clases C: 192.168.0.0/16.
- Se agrega proceso de NAT (Network Address Translation), NAPT (Network Address Port Translation).
- Se pierde principio end-to-end.

# Direcciones IPv4 Privadas y NAT/NAPT (cont.)



# Problemas NAT

- Se vuelve compleja la red, dispositivos intermedios.
- Dificultades:
  - Acceso directo a red privada.
  - Protocolos Peer-to-Peer.
  - Problemas protocolos particulares: FTP, VoIP (SIP, RTP), VPN (IPSec), on-line gaming.
- Agregados (Parches):
  - Port-Forwarding, UPnP (Universal Plug & Play).
  - ISP requieren CGN (Carrier Grade NAT), NAT444, LSN: 100.64.0.0/10 (RFC-6598).
  - STUN (Session Traversal Utilities for NAT), NAT Traversal.

# Problemas en IPv4

- Direcciones IPv4 no disponibles, uso de NAT.
- Tablas de ruteo muy grandes en el backbone de Internet.
- Congestión en los routers, demasiado procesamiento.

Otras cuestiones no contempladas desde el inicio:

- Seguridad a nivel L3, IP.
- Extensiones al modelo de Calidad de Servicio (QoS).
- Fácil auto-configuración y re-numeración de direcciones.
- Movilidad a nivel de red no contemplada en el diseño del protocolo.

# Beneficios de IPv6

- No son versiones del mismo protocolo, IPv4 e IPv6.
- Mayor espacio de direcciones - 128 bits:  
340.282.366.920.938.463.463.374.607.431.768.211.456  
direcciones.
- Formato de cabecera simplificado.
- Menor overhead de procesamiento.
- Ordenar las tablas de enrutamiento.
- Conectar todo, usar auto-configuración de direcciones (plug and play).
- Arquitectura de red jerárquica para un ruteo eficiente.
- Seguridad a nivel IP (IPSec obligatorio).
- Jumbogramas, size(datagrama) > 64KB.
- Movilidad y más direcciones de multicast.

# ¿Y la versión 5?

- Los protocolos registrados en el IANA:
- La versión especifica el formato del Header IP
- La v5 ya estaba asignada
- ST: Internet Stream Protocol, orientado a conexión y con QoS, definido en 1970
- El resto otros posibles reemplazos de IP

Version	Description
0	reserved.
1	
2	
3	
4	IP, Internet Protocol.
5	<a href="#">ST, ST Datagram Mode.</a>
6	SIP, Simple Internet Protocol. SIPP, Simple Internet Protocol Plus. <a href="#">IPv6, Internet Protocol.</a>
7	<a href="#">TP/IX, The Next Internet.</a>
8	PIP, The P Internet Protocol.
9	TUBA.
10	
-	
14	
15	reserved.

# Cambios en IPv6

- Definido RFC-8200 (STD-86) 2017 (hace obsoleta RFC-2460).
- Direcciones más largas.
- Datagramas de 40 bytes (contra 20 bytes + opt, max 60B).
- Simplifica cabecera:
  - Se saca la fragmentación, se deja solo de extremo a extremo como opción.
  - Se saca checksum de cabecera.
  - Header de tamaño fijo. No existen más las Opciones.
  - Flow Label: identificador de flujo (20 bits).
  - Se renombran los campos: Traffic Class, Hop Limit, Next Header.
  - Cabeceras de extensión.

# Formato cabecera IPv6

Ver.	TrafficClass	Flow Label		
	Payload Length	Next Header	Hop Limit	
		<b>128 bit</b>		<b>Source Address</b>
		<b>128 bit</b>		<b>Destination Address</b>

# Datagrama IPv6 (Wireshark)

No.	Time	Source	Destination	Protocol	Length	Info
5	4.292394	2001:db8:20::100	2001:db8:4::1	TCP	94	58895 > irdmi [SYN] Seq=0 Win=14400 Len=0 MSS=1440 SACK_PERM
6	4.292489	2001:db8:4::1	2001:db8:20::100	TCP	94	irdmi > 58895 [SYN, ACK] Seq=0 Ack=1 Win=14280 Len=0 MSS=1440 TStamp=85877
7	4.292641	2001:db8:20::100	2001:db8:4::1	TCP	86	58895 > irdmi [ACK] Seq=1 Ack=1 Win=14400 Len=0 TStamp=85877
8	4.294168	2001:db8:20::100	2001:db8:4::1	TCP	87	58895 > irdmi [PSH, ACK] Seq=1 Ack=1 Win=14400 Len=1 TStamp=85877
9	4.294262	2001:db8:4::1	2001:db8:20::100	TCP	86	irdmi > 58895 [ACK] Seq=1 Ack=2 Win=14288 Len=0 TStamp=85877

► Frame 5: 94 bytes on wire (752 bits), 94 bytes captured (752 bits)  
 ► Ethernet II, Src: 00:00:00\_aa:00:02 (00:00:00\_aa:00:02), Dst: 00:00:00\_aa:00:03 (00:00:00\_aa:00:03)  
 ▼ Internet Protocol Version 6, Src: 2001:db8:20::100 (2001:db8:20::100), Dst: 2001:db8:4::1 (2001:db8:4::1)  
 ► 0110 .... = Version: 6  
 ► ..... 0000 0000 ..... .... .... = Traffic class: 0x00000000  
 .... .... .... 0000 0000 0000 0011 1100 = Flowlabel: 0x0000003c  
 Payload length: 40  
 Next header: TCP (0x06)  
 Hop limit: 63  
 Source: 2001:db8:20::100 (2001:db8:20::100)  
 Destination: 2001:db8:4::1 (2001:db8:4::1)  
 ► Transmission Control Protocol, Src Port: 58895 (58895), Dst Port: irdmi (8000), Seq: 0, Len: 0

0000 00 00 00 aa 00 03 00 00 00 aa 00 02 86 dd 50 00	..... . . . . .
0010 00 3c 00 28 06 3f 20 01 0d b8 00 20 00 00 00 00	.. (.?. . . . .
0020 00 00 00 00 01 20 01 0d b8 00 04 00 00 00 00	..... . . . .
0030 00 00 00 00 00 01 e6 0f 1f 40 e6 f2 d7 39 00 00	..... .@...9..
0040 00 00 a0 02 38 40 a0 4d 00 00 02 04 05 a0 04 02	....8@.M .....
0050 08 oa 00 01 4f 75 00 00 00 00 01 03 03 04	....ou.. . . .

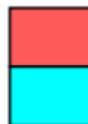
Flowlabel (ipv6.flow), 4 bytes      Packets: 30 Displayed: 30 Marked: 0 Load time: 0:00.127      Profile: Default

captures/fi-tcp-60.pcap, rows: 5

# Formato cabecera IPv4 - Cambios

- Cambios en IPv4:

Ver.	header	TOS	total length						
identification		flag	fragment offset						
TTL	Protocol	Checksum							
32 bit Source Address									
32 bit Destination Address									



removed  
changed

# Cabecera IPv4 vs IPv6

- Comparación de Cabeceras:

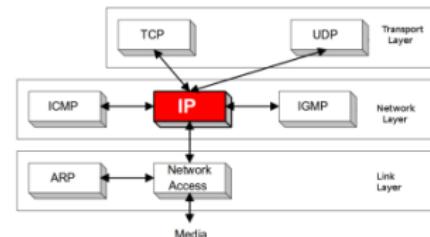
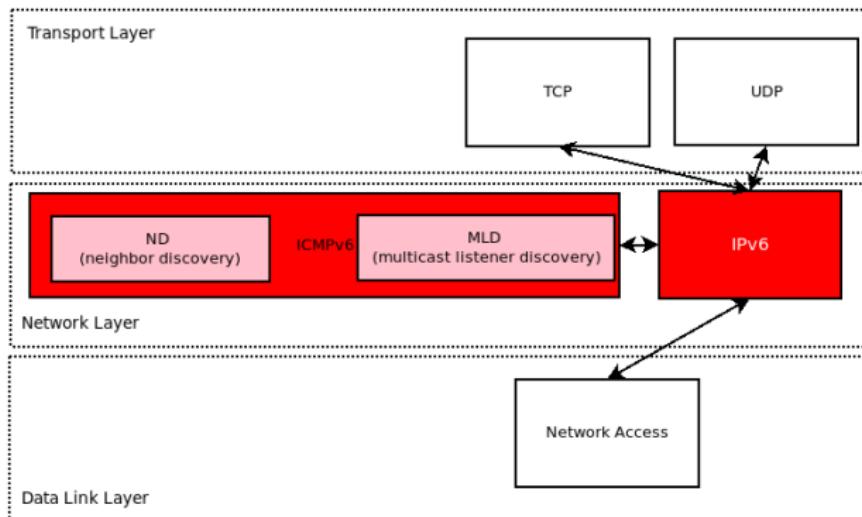


# Servicios Básicos

- Funcionalidad:
  - Direccionamiento.
  - Ruteo/Forwarding.
  - Mux/Demux de protocolos superiores.
  - Fragmentación.
  - Otras: como evitar loops.
  - ~~Detección de errores en header.~~

# IPv6 Stack

- Cambia el plano de control según IPv4



# Servicios Nuevos

- Funcionalidad:

- Descubrimiento de Vecinos (NDP):
  - ND propiamente.
  - Router discovery y auto-configuración.
- Manejo de Grupos de Multicast.

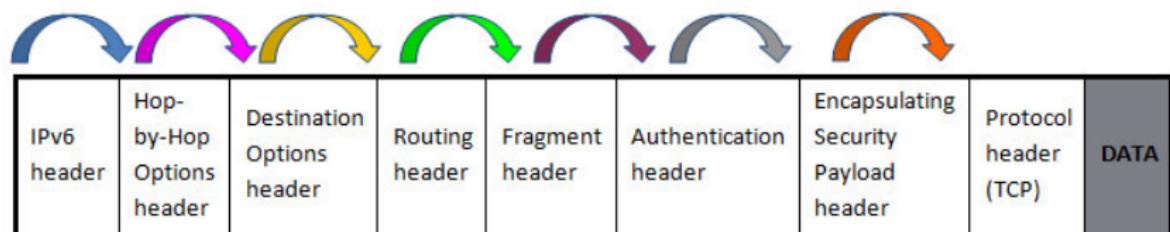
# Cabeceras de Extensión (Más servicios)

- Permite la extensibilidad del protocolo.
- Se encuentran a continuación del header.
- En general, son procesadas por los extremos.

IPv6 Header Next Header = 6 (TCP)	Segmento TCP		
IPv6 Header Next Header = 43 (Routing Header)	Routing Header Next Header = 6 (TCP)	Segmento TCP	
IPv6 Header Next Header = 43 (Routing Header)	Routing Header Next Header = 44 (Fragment Header)	Fragment Header Next Header = 6 (TCP)	Segmento TCP

# Orden Cabeceras de Extensión

- Hop-by-hop: procesado por cada router.
- Dest Opt: procesado por routers incluidos.
- Routing: procesado por routers, RH0 desaconsejado.
- Frag, Auth, Sec, Dest. procesado por extremos.



fuente: <http://www.cisco.com/web/about/security/intelligence/FNIPv6.html>

# Ejemplo NH IPv6 (Wireshark)

No.	Time	Source	Destination	Length	Info
6	4.002568	2001:db8:1234:2::10	2001:db8:1234:3::10	1294	IPv6 fragment (nxt=ICMPv6 (58) off=0 id=0xbff2f2f2)
7	4.002912	2001:db8:1234:2::10	2001:db8:1234:3::10	238	Echo (ping) request id=0x0032, seq=3, hop limit

► Frame 7: 238 bytes on wire (1904 bits), 238 bytes captured (1904 bits)

► Ethernet II, Src: 00:00:00:aa:00:06 (00:00:00:aa:00:06), Dst: 00:00:00:aa:00:04 (00:00:00:aa:00:04)

▼ Internet Protocol Version 6, Src: 2001:db8:1234:2::10 (2001:db8:1234:2::10), Dst: 2001:db8:1234:3::10 (2001:db8:1234:3::10)

- 0110 .... = Version: 6
- .... 0000 0000 .... .... .... = Traffic class: 0x00000000
- .... .... .... 0000 0000 0000 0000 0000 = Flowlable: 0x00000000
- Payload length: 184
- Next header: IPv6 fragment (44)
- Hop limit: 64
- Source: 2001:db8:1234:2::10 (2001:db8:1234:2::10)
- Destination: 2001:db8:1234:3::10 (2001:db8:1234:3::10)

▼ Fragmentation Header

- Next header: ICMPv6 (58)
- Reserved octet: 0x0000
- 0000 0100 1101 0... = Offset: 154 (0x009a)
- .... .... .... .00. = Reserved bits: 0 (0x0000)
- .... .... .... .0 = More Fragment: No
- Identification: 0xbff2f2769

► [2 IPv6 Fragments (1408 bytes): #6(1232), #7(176)]

▼ Internet Control Message Protocol v6

- Type: Echo (ping) request (128)
- Code: 0

captures/ipv6-frag-doble-nocont-i.pcap, rows: 7

# Más direcciones disponibles

- Más direcciones para más gente, más dispositivos, nuevas tecnologías, IoT.
- En un principio, discusión entre (anecdótico):
  - Direcciones de longitud fija de 64 bits.
  - Direcciones de longitud variable de 160 bits.
- Acuerdo:
  - Direcciones de longitud fija de 128 bits.
  - Varias direcciones por interfaz.
  - Direcciones con diferentes alcances y tiempos de vida.

# Direcciones IPv6

- Tipos de direcciones:
  - Unicast.
  - Anycast (tomadas del rango Unicast).
  - Multicast (no hay direcciones broadcast): FF00::/8.
- Alcance (Scope) de las direcciones Unicast:
  - Locales (Link-local): FE80::/64.
  - De sitio site-local (desaconsejadas), unique-local.
  - Compatibilidad ipv4-compat (desaconsejadas), ipv4-mapped.
  - Globales: 2000::/3.

# Direcciones IPv6 (cont.)

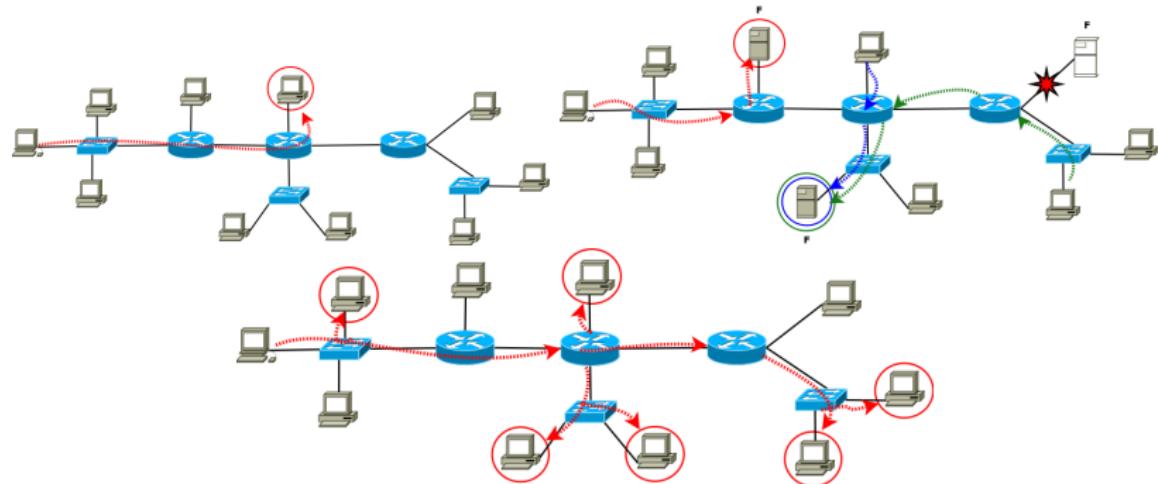
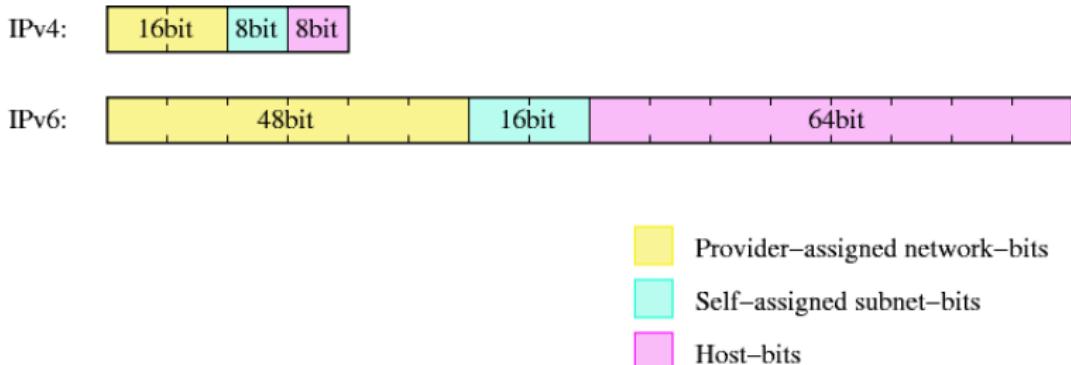


Figura: unicast, anycast y multicast

# Direcciones IPv6 (cont.)



fuente: <http://www.netbsd.org/docs/guide>

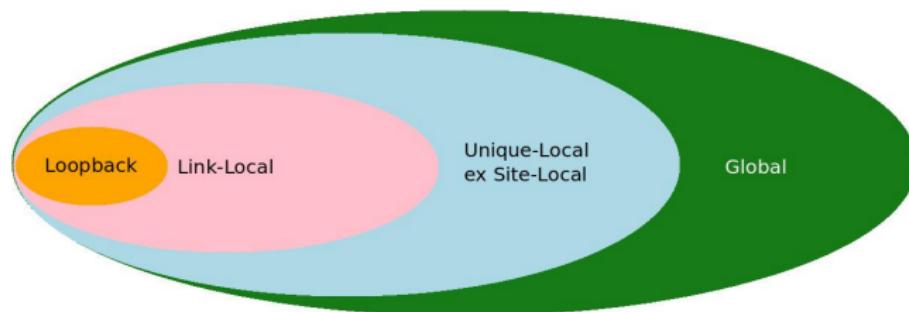
- 128 bits.
- Unicast: separadas en red, [subred] y host.
- No hay clases de direcciones.

# Notación Direcciones IPv6

- Hexadecimal en grupos de 16 bits, separadas por ":".
- `nnnn:nnnn: nnnn:ssss: hhhh:hhhh: hhhh:hhhh /pfxlen`
- Ceros al inicio de cada grupo se pueden obviar:
  - `2001:0db8:1011:0001:36ed:04ff:fe32:0076 ==`  
`2001:db8:1011:1:36ed:4ff:fe32:76`
- Ceros contiguos se puede eliminar con "::". Sólo se puede utilizar una vez:
  - `2001:db8:1011:1:0:0:0:1 == 2001:db8:1011:1::1`
  - `2001:db8:0:0:1011:0:0:1 == 2001:db8:0:0:1011::1`
  - `2001:db8:0:1011:0:0:0:1 == 2001:db8:0:1011::1`
- Se utilizan “[,]” para indicar port en URL:
  - `http://[2001:db8:1011:1:0:0:0:1]:8080`
- No se usa máscara, solo prefix length.

# Direcciones IPv6 Unicast

- Link-local.
- Site-local.
- Unique-Local.
- IPv4-Compatible.
- IPv4-Mapped.
- Global.



# Direcciones IPv6 locales

- Link-local Address



- Prefijo Asignado:** FE80::/10.
- Prefijo Utilizado:** FE80::/64 (len. en LAN /64).
- Alcance:** sólo la red directamente conectada.
- Mayormente auto-generadas stateless a partir de IID.
- IID:** Interface Identifier, 64 LSb.

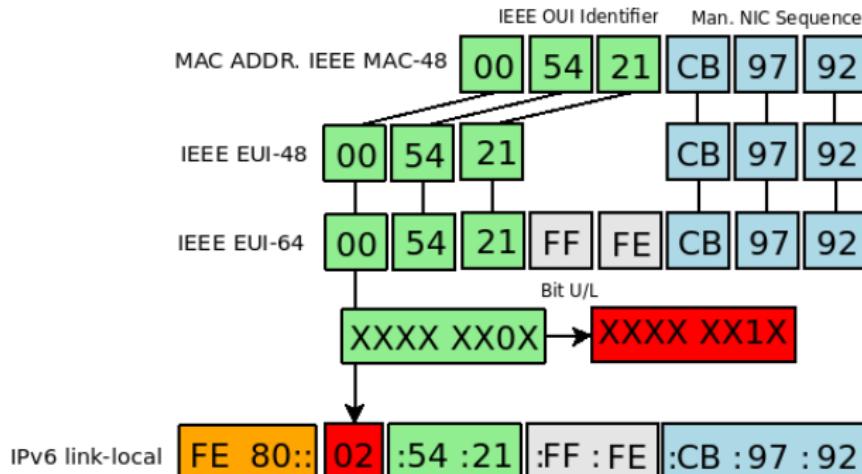
Obligatoria en todas las interfaces multiacceso

# Direcciones IPv6 locales (cont.)

- Generación de IID:
  - Usando IEEE MAC-64.
  - Extended Unique Identifier 64, EUI-64 derivado de IEEE MAC-48, EUI-48.
  - De forma manual.
  - RFC-4941 Privacy Extensions for Stateless Address Autoconfiguration (dir. temporales).
  - RFC-7217 A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC).
- Finalmente se generan con el prefijo link-local y realiza DAD (Duplicate Address Detection).

# IPv6 sobre LAN: EUI-64 (desaconsejado su uso para dir. estables)

- Redes de multi-acceso con ID de interfaces, direcciones MAC-48: Ethernet, 802.11, Bluetooth, FC y antiguas tecnologías como: FDDI, TokenRing, TokenBus, ATM.



# Direcciones IPv6 locales (ejemplo)

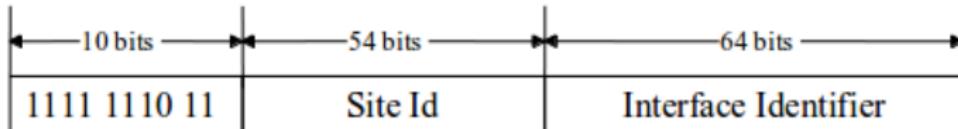
```
root@n7:/# ifconfig eth0
eth0      Link encap:Ethernet      HWaddr 00:00:00:aa:00:07
          inet addr:10.0.3.10    Bcast:0.0.0.0    Mask:255.255.255.0
          inet6 addr: 2001:db8:1234:3::10/64 Scope:Global
          inet6 addr: 2001:db8:1234:3:200:ff:feaa:7/64 Scope:Global
              inet6 addr: fe80::200:ff:feaa:7/64 Scope:Link
          inet6 addr: 2001:db8:1234:3:f8b0:a208:ca54:4ef5/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
```

```
root@n7:/# ip addr add fe80::2222:2222:2222:2222/64 dev eth0
```

```
root@n7:/# ifconfig eth0  #ip addr show dev eth0
eth0      Link encap:Ethernet      HWaddr 00:00:00:aa:00:07
          inet addr:10.0.3.10    Bcast:0.0.0.0    Mask:255.255.255.0
          inet6 addr: 2001:db8:1234:3::10/64 Scope:Global
              inet6 addr: fe80::2222:2222:2222:2222/64 Scope:Link
          inet6 addr: 2001:db8:1234:3:200:ff:feaa:7/64 Scope:Global
              inet6 addr: fe80::200:ff:feaa:7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
```

# Direcciones IPv6 de Site-Local

- **Site-local Address**

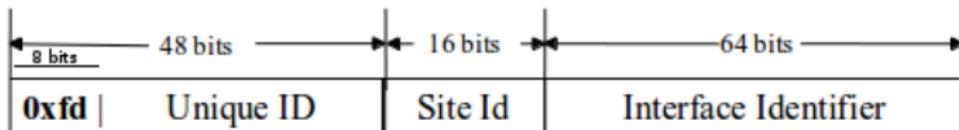


- **Prefijo:** FEC0::/10.
- **Alcance:** sitio u organización. Similar a las redes privadas de IPv4.
- Dificultad de establecer los límites.

Desaconsejado su uso en la RFC 3879, Deprecating Site Local Addresses, de 2004

# Direcciones IPv6 de Sitio Únicas

- **Unique Local Address (ULA)**



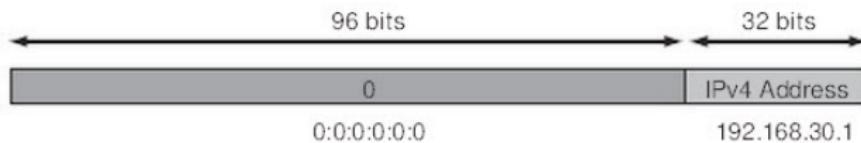
- **Prefijo:** FC00::/7, dividido en FC00::/8 y FD00::/8.
- **Prefijo Utilizado:** FD00::/8, [xxxxxxxxL] L bit = 1 (def. local).
- **Alcance:** sitio u organización.
- Definidas en RFC-4193. Reemplazan las direcciones de Site Local.

**Unique ID** debe ser generado de forma (pseudo)-aleatoria

# Direcciones IPv4-compat IPv6

- **IPv4-compat IPv6 (desaconsejadas)**

- Usadas para la transición. Definidas en RFC-4291 y desaconsejadas.
- Asigna a un IPv4 global única una IPv6.



IPv4-Compatible Address = 0:0:0:0:0:192.168.30.1  
= ::192.168.30.1  
= ::C0A8:1E01

**IPv4-Compat IPv6 son desaconsejadas en RFC-4291, 2006.**

# Direcciones IPv4-mapped IPv6

- **IPv4-mapped IPv6**

- Uso definido en RFC-4038.



IPv4-Mapped Address = 0:0:0:0:0:FFFF:192.168.30.1

fuente gráficos:

<http://blog.alansoon.com/others/ipv6-addressing-vs-ipv4-on-ipv6-day-internet-technology-review-online-infrastructure>

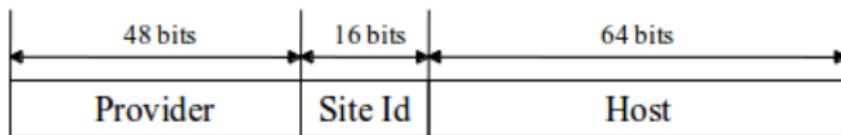
# Direcciones IPv4-mapped IPv6 (ejemplo)

```
root@n7:/# ip -6 addr add ::10.0.3.10/64 dev eth0
root@n7:/# ip -6 addr add ::ffff:10.0.3.10/64 dev eth0

root@n7:/# ifconfig eth0
eth0      Link encap:Ethernet HWaddr 00:00:00:aa:00:07
          inet addr:10.0.3.10 Bcast:0.0.0.0 Mask:255.255.255.0
            inet6 addr: 10.0.3.10/64 Scope:Global
                     inet6 addr: 2001:db8:1234:3::10/64 Scope:Global
                     inet6 addr: 2001:db8:1234:3:200:ff:feaa:7/64 Scope:Global
                     inet6 addr: fe80::200:ff:feaa:7/64 Scope:Link
                     inet6 addr: 2001:db8:1234:3:f8b0:a208:ca54:4ef5/64 Scope:Global
                      inet6 addr: ::10.0.3.10/64 Scope:Compat
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          ...
...
```

# Direcciones IPv6 Globales

- **Aggregatable Global Unicast Address**



- **Prefijo:** cedidos por un provider.
- **Alcance:** Internet. Similar a las direcciones públicas de IPv4.

# Direcciones Globales (cont.)

- Generación de IID:
  - Usando IEEE MAC-64 (**desaconsejado en RFC-8064**)
  - Extended Unique Identifier 64, EUI-64 derivado de IEEE MAC-48, EUI-48 (**desaconsejado en RFC-8064 para dir. estables**)
  - De forma manual.
  - RFC-4941 Privacy Extensions for Stateless Address Autoconfiguration (dir. temporales).
  - RFC-7217 A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC).
- Finalmente se generan con el prefijo link-local y realiza DAD (Duplicate Address Detection).

# RFC 7217, IID estables y opacos para SLAAC

- Cálculo del IID sin exponer el ID del dispositivo.
- $RID = F(Prefix, Net_{Interface}, NetworkID, DADCounter, secretkey)$
- $F()$  : función pseudorandom, como SHA1, SHA-256, MD5(no aceptable).
- $Prefix$  : valor recibido por RA o link-local.
- $Net_{Interface}$  : forma de identificar la interfaz, un índice, la MAC, el nombre u otro valor.
- $NetworkID$  : valor que identifica a la red conectada, por ejemplo SSID para el caso de wifi (opcional).
- $DADCounter$  : contador usado para resolver los conflictos con dir. Se debe almacenar en memoria persistente (HD).
- $SecretKey$  : clave secreta de al menos 128 bits.
- El IID finalmente se obtiene tomando los bits del  $RID$  necesarios desde el LSb.

# Direcciones IPv6 Globales (ejemplo)

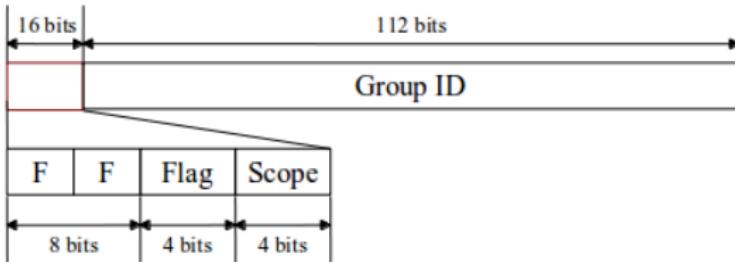
```
root@n7:/# ip addr show dev eth0
26: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:00:aa:00:07 brd ff:ff:ff:ff:ff:ff
        inet 10.0.3.10/24 scope global eth0
            inet6 2001:db8:1234:3:3cb2:67a8:b0d6:2214/64 scope global temporary dynamic
                valid_lft 86392sec preferred_lft 14392sec
            inet6 2001:db8:1234:3:200:ff:fea:7/64 scope global dynamic
                valid_lft 86392sec preferred_lft 14392sec
            inet6 2001:db8:1234:3::10/64 scope global
                valid_lft forever preferred_lft forever
            inet6 fe80::2222:2222:2222:2222/64 scope link
                valid_lft forever preferred_lft forever
            inet6 fe80::200:ff:fea:7/64 scope link
                valid_lft forever preferred_lft forever
```

# Direcciones IPv6 Globales (ejemplo 2)

```
root@n7:/# sysctl net.ipv6.conf.eth0.addr_gen_mode=2|3
net.ipv6.conf.eth0.addr_gen_mode = 3
root@n7:/# ifconfig eth0 down ; ifconfig eth0 up
root@n7:/# ip addr show dev eth0
12: eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 00:00:00:aa:00:01 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet6 2001::b7ab:71a3:b605:f45f/64 scope global dynamic mngtmpaddr stable-privacy
            valid_lft 86396sec preferred_lft 14396sec
        inet6 fe80::988e:62bb:c7f2:5617/64 scope link stable-privacy
            valid_lft forever preferred_lft forever
root@n7:/# sysctl net.ipv6.conf.eth0.stable_secret
net.ipv6.conf.eth0.stable_secret = df59:989e:cdba:94fd:b1bb:6113:d4de:56c4
```

# Direcciones IPv6 Multicast

- **Multicast Address**



- **Prefijo:** FF00::/8
- **Flags:** permanente, temporaria. Otros reservados.
- **Alcance:** 1: nodo local, 2: link local, 5: site local, 8: org. local, E: global.
- **GID:** grupo de multicast.

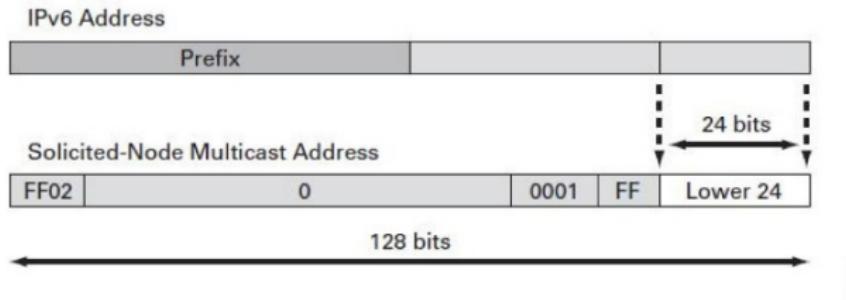
# Direcciones IPv6 Multicast (cont.)

- Node-Local/Interface-Local: FF01::1, FF01::2 (no salen a la red).
- Link-Local: (quedan en la LAN).
  - FF02::1 (todos los nodos en la LAN) equivale a 224.0.0.1. Posible reemplazo lim. broadcast: 255.255.255.255.
  - FF02::2 (todos los routers en la LAN) 224.0.0.2.
  - FF02::5 OSPFv3 All SPF routers (224.0.0.5)
  - FF02::6 OSPFv3 All DR routers (224.0.0.6)
  - FF02::8 IS-IS for IPv6 routers.
  - FF02::9 RIP routers (224.0.0.9).
  - FF02::1:2 All DHCP-Agents (255.255.255.255).
- Site-Local: (quedan en el site).
  - FF05::2 All routers.
  - FF05::1:3 All-dhcp-servers RFC-3315.
- Generales:
  - FF0X::FB mDNSv6 (Mcast DNS).
  - FF0X::102 NTP.

# Direcciones IPv6 Multicast SD

## ● **Solicited Node Multicast Address (SD)**

- Usada para ND (Neighbor Discovery) en lugar de flooding en la LAN.
- Generada a partir de unicast/anycast.
- Por cada unicast/anycast debe hacer join de la multicast.

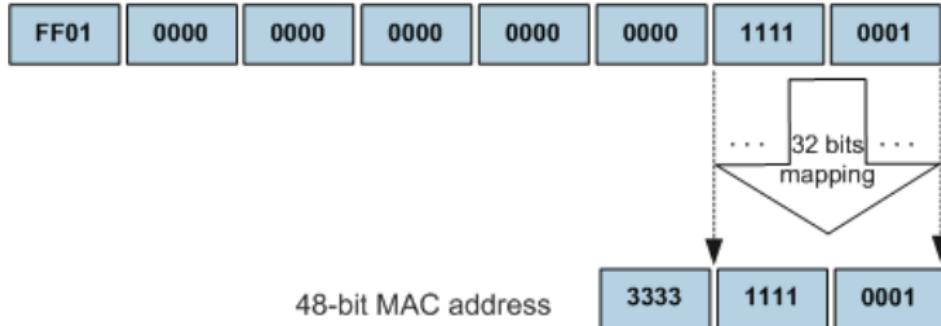


fuente: <http://www.cisco.com/>

# IPv6 Multicast mapeada en IEEE EUI-48

- IPv4 mcast mapped to Ethernet:
  - 01:00:5E:00:00:00 - 01:00:5E:7F:FF:FF.
  - 23 LS bits de mcast IPv4 en MAC.
- IPv6 mcast mapped to Ethernet:
  - 33:33:00:00:00:00 - 33:33:FF:FF:FF:FF.
  - 32 LS bits de mcast IPv6 en MAC.

128-bit IPv6 address



# Uso de Direcciones IPv6 Multicast

## ● Multicast

```
root@n7:/# netstat -g -A inet6 -n
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          ...
IPv6/IPv4 Group Memberships
Interface      RefCnt Group
-----
lo            1      ff02::1
eth0           1      ff02::1:ff54:4ef5
eth0           2      ff02::1:ffaa:7
eth0           1      ff02::1:ff00:10
eth0           1      ff02::1
```

```
root@n7:/# ip -6 maddr show
9: lo
inet6 ff02::1
26: eth0
inet6 ff02::1:ff54:4ef5
inet6 ff02::1:ffaa:7 users 2
inet6 ff02::1:ff00:10
inet6 ff02::1
```

## Uso de Direcciones IPv6 Multicast (cont.)

- Multicast

```
root@n7:/# cat /proc/net/dev_mccast
26 eth0          1   0 333300000001
26 eth0          1   0 3333ff0000010
26 eth0          1   0 01005e0000001
26 eth0          1   0 3333ffaa00007
26 eth0          1   0 3333ff544ef5
```

# Direcciones IPv6 - Casos Especiales

- Any (sin especificar):
  - ::0/0
- Loopback/Localhost:
  - ::1/128
- Documentación:
  - 2001:db8::/32
- 6Bone:
  - 3FFE::/16, devueltas al IANA en 2006.

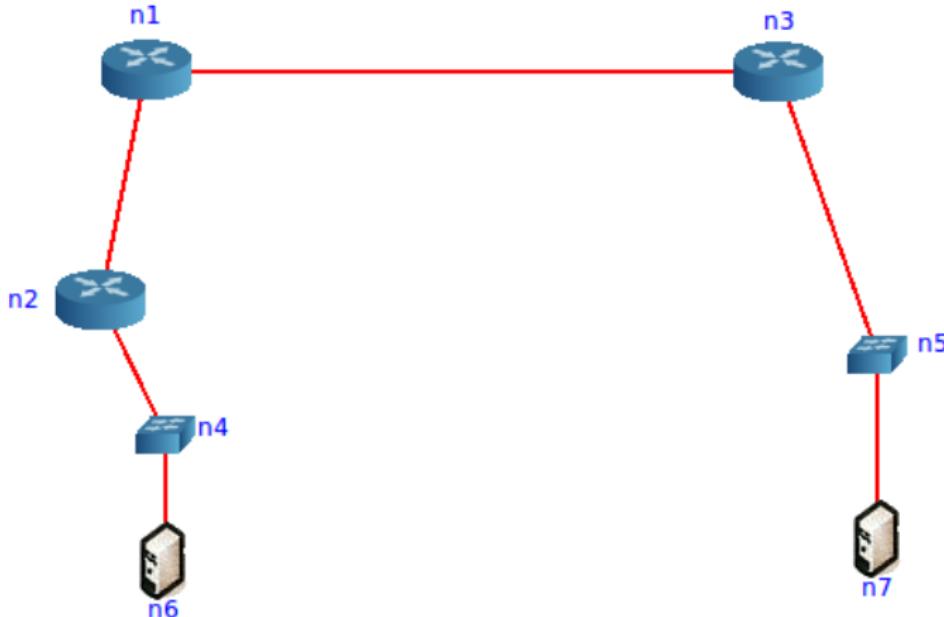
# Direcciones IPv6 loopback (ejemplo)

```
root@n7:/# ip -6 addr show dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state UNKNOWN qlen 1000
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever

root@n8:/# ifconfig lo
lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
                UP LOOPBACK RUNNING MTU:16436 Metric:1
```

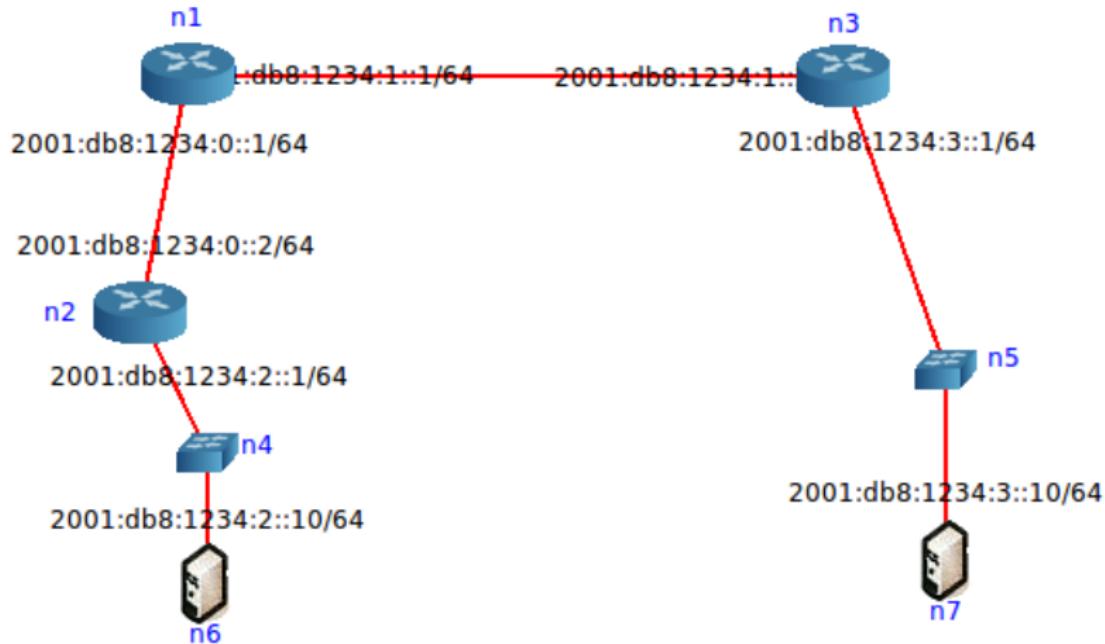
# Red de Ejemplo

- Caso 1: 2001:db8:1234::/48.
- Caso 2: 2001:db8:1234:1234::/56.
- Se requieren 4 sub-redes.



# Red de Ejemplo (cont.)

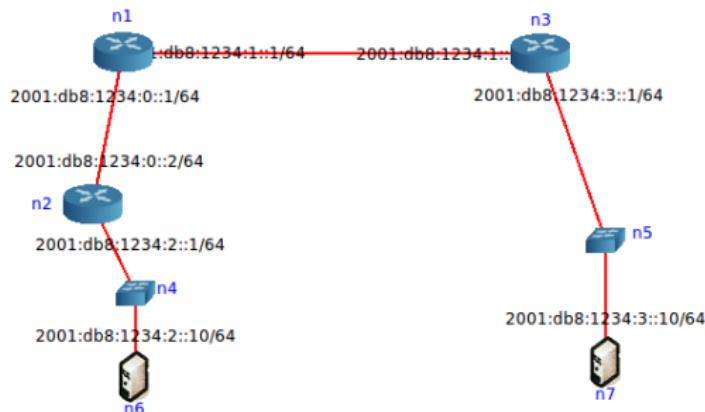
- Caso 1: 2001:db8:1234::/48.
- Tomo 16 bits para sub-redes 2001:db8:1234:\_\_\_\_\_::/64.



# Red de Ejemplo (cont.)

- Subredes:

- Red 0: 2001:db8:1234:**0000**::/64.
- Red 1: 2001:db8:1234:**0001**::/64.
- Red 2: 2001:db8:1234:**0002**::/64.
- Red 3: 2001:db8:1234:**0003**::/64.

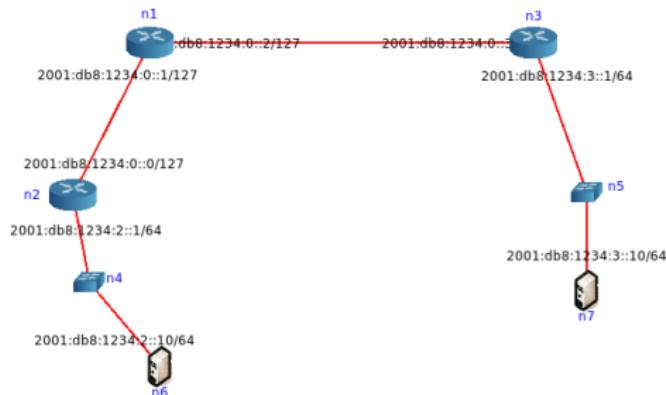


# Red de Ejemplo (cont.)

- Qué sucede si lo resuelvo con solo 2 bits ?
- Subredes:
  - Red 0: 2001:db8:1234:**0000 0000 0000 0000<sub>2</sub>**::/50,  
2001:db8:1234:00::/50.
  - Red 1: 2001:db8:1234:**0100 0000 0000 0000<sub>2</sub>**::/50,  
2001:db8:1234:40::/50.
  - Red 2: 2001:db8:1234:**1000 0000 0000 0000<sub>2</sub>**::/50,  
2001:db8:1234:80::/50.
  - Red 3: 2001:db8:1234:**1100 0000 0000 0000<sub>2</sub>**::/50,  
2001:db8:1234:C0::/50.
- No escala en caso de necesitar agregar redes.
- SLAAC no funcionaba con prefijo  $\neq /64$ .
- Sirve en caso de necesitar hacer sub-redes dentro de las subredes.

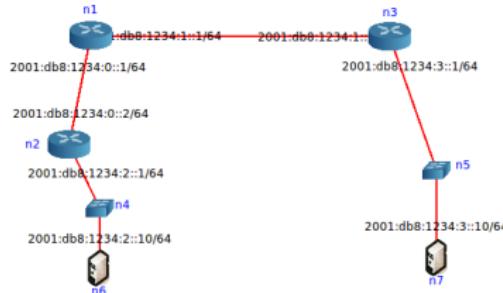
# Red de Ejemplo (cont.)

- Qué sucede si las punto a punto utilizo un /127.
- Subredes:
  - Red 0: 2001:db8:1234:0000::0/127.
  - Red 1: 2001:db8:1234:0000::2/127.
  - Red 2: 2001:db8:1234:0002::/64.
  - Red 3: 2001:db8:1234:0003::/64.
- Tengo menos desperdicio.
- No voy a tener más de 2 equipos en las p2p.



# Red de Ejemplo (cont.)

- Caso 2: 2001:db8:1234::/56.
- Tomo 8 bits para sub-redes 2001:db8:1234:00  ::/64.
- Subredes:
  - Red 0: 2001:db8:1234:0000::/64.
  - Red 1: 2001:db8:1234:0001::/64.
  - Red 2: 2001:db8:1234:0002::/64.
  - Red 3: 2001:db8:1234:0003::/64.



# Ver Tabla de Ruteo

- RIB (Tabla de Ruteo)

```
root@n7:/# ip -6 route show
2001:db8:1234:3::/64          dev eth0  proto kernel  metric 256
fe80::/64                      dev eth0  proto kernel  metric 256
default via 2001:db8:1234:3::1 dev eth0  metric 1024
default via fe80::200:ff:fea:5 dev eth0  proto kernel ... expires 24sec
...
```

```
root@n7:/# netstat -nr -A inet6
Kernel IPv6 routing table
Destination      Next Hop          Flag   Met Ref   Use If
2001:db8:1234:3::/64    ::            U      256 0     1   eth0
fe80::/64          ::            U      256 0     0   eth0
::/0                2001:db8:1234:3::1  UG      1024 0    0   eth0
::/0                fe80::200:ff:fea:5  UGDAe  1024 0    0   eth0
::1/128           ::            Un      0  1     1   lo
...
```

# Tabla de Ruteo en Routers

```
root@n1:/# ip -6 route show
2001:db8:1234::/127 dev eth0 proto kernel metric 256
2001:db8:1234::2/127 dev eth1 proto kernel metric 256
2001:db8:1234:2::/64 via 2001:db8:1234:: dev eth0 metric 1024
2001:db8:1234:3::/64 via 2001:db8:1234::3 dev eth1 metric 1024
fe80::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth1 proto kernel metric 256
```

```
root@n3:/# ip -6 route show
2001:db8:1234::2/127 dev eth0 proto kernel metric 256
2001:db8:1234:3::/64 dev eth1 proto kernel metric 256
fe80::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth1 proto kernel metric 256
default via 2001:db8:1234::2 dev eth0 metric 1024
```

# Ruteo en IPv6

- Ruteo Estático.
- RIP-ng.
- OSPFv3.
- IS-IS.
- MP-BGP.
- ...

# Referencias

- IPv6 Essentials (Integrating IPv6 into your IPv4 Network) Silvia Haggen. O'Reilly. 2nd Ed, 2006.
- Migrating to IPv6. Marc Blanchet. John Wiley and Sons, 2006.
- Tutorial de Jordi Palet Martínez:  
[http://www.consulintel.es/html/ForoIPv6/Documentos/Tutorial de IPv6.pdf](http://www.consulintel.es/html/ForoIPv6/Documentos/Tutorial%20de%20IPv6.pdf).
- TCP/IP Illustrated, Volume 1: The Protocols, 2nd Ed. W. Richard Stevens, Kevin R. Fall. Addison-Wesley Professional Computing Series, 2011.
- IPv6 for All: <http://www.ipv6tf.org/pdf/ipv6forall.pdf>.

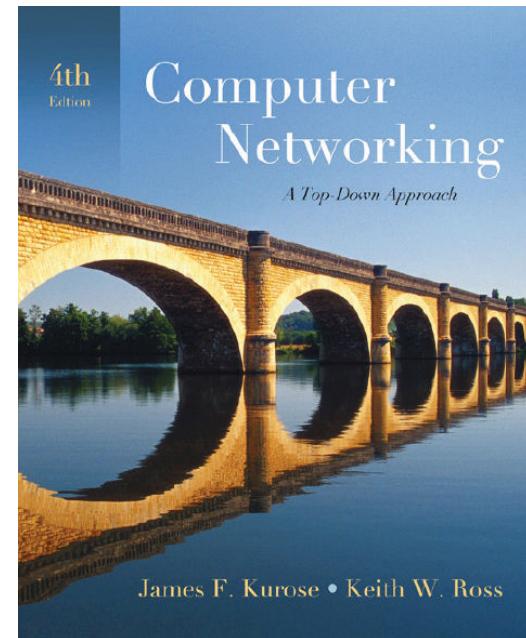
# Introducción a las Redes de Computadores

## Capítulo 5 Capa de Enlace y LANs

Nota acerca de las transparencias del curso:

Estas transparencias están basadas en el sitio web que acompaña el libro y han sido modificadas por los docentes del curso.

All material copyright 1996-2007  
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking:  
A Top Down Approach  
4<sup>th</sup> edition.  
Jim Kurose, Keith Ross  
Addison-Wesley, July  
2007.*

# Capítulo 5: La Capa de Enlace de Datos

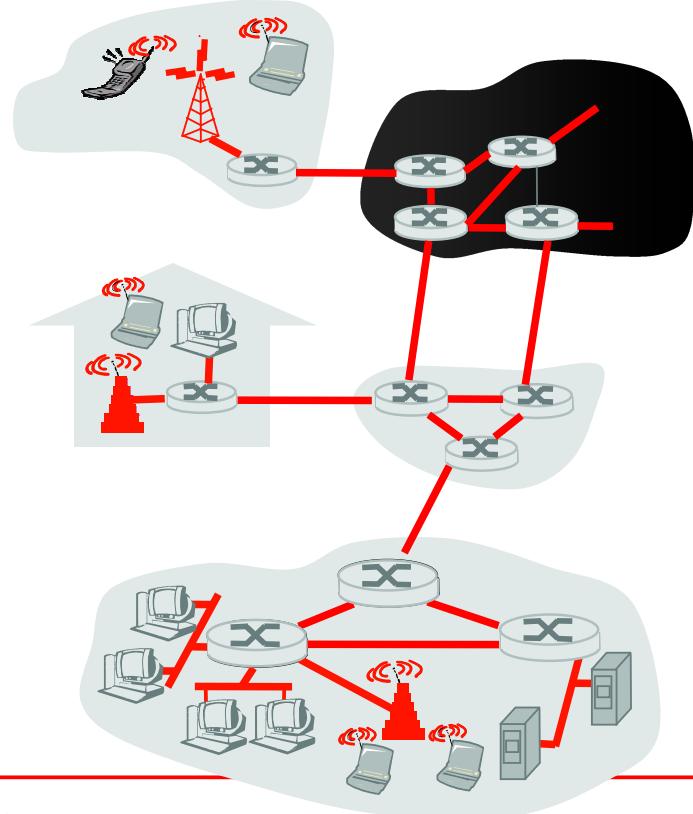
## Objetivos:

- Entender los principios detrás de los servicios de la capa de enlace de datos:
  - detección de errores; corrección
  - compartir un canal de *broadcast*: acceso múltiple
  - direccionamiento de capa de enlace
  - transferencia de datos confiable, control de flujo
- Algunas tecnologías de Capa de Enlace

# Capa de Enlace: Introducción

## Algo de terminología:

- hosts y routers son **nodes**
- los canales de comunicación que conectan nodos adyacentes a través de caminos de comunicación son **links**
  - enlaces cableados
  - enlaces inalámbricos
  - LANs
- la PDU de capa 2 es el **frame**, que encapsula un datagrama



**la capa de enlace de datos tiene la responsabilidad de transferir datagramas desde un nodo a otro nodo adyacente, a través de un link**

# Capa de enlace: contexto

- los datagramas son transferidos por diferentes protocolos de enlace sobre diferentes enlaces:
  - p.e., Ethernet en el primer enlace, Frame Relay en los enlaces intermedios, 802.11 en el último enlace
- cada protocolo de enlace brinda diferentes tipos de servicios
  - p.e., puede o no proveer rdt (*reliable data transfer*) sobre el enlace

## Analogía transporte

- Viaje desde Montevideo a Mar del Plata
  - remise: Montevideo a Carrasco
  - avión: Carrasco a Aeroparque
  - ómnibus: Aeroparque a Mar del Plata
- turista = **datagrama**
- segmento de transporte = **enlace de comunicación**
- modo de transporte = **protocolo de capa de enlace**
- agencia de viaje = **algoritmo de enrutamiento**

# Servicios de Capa de Enlace

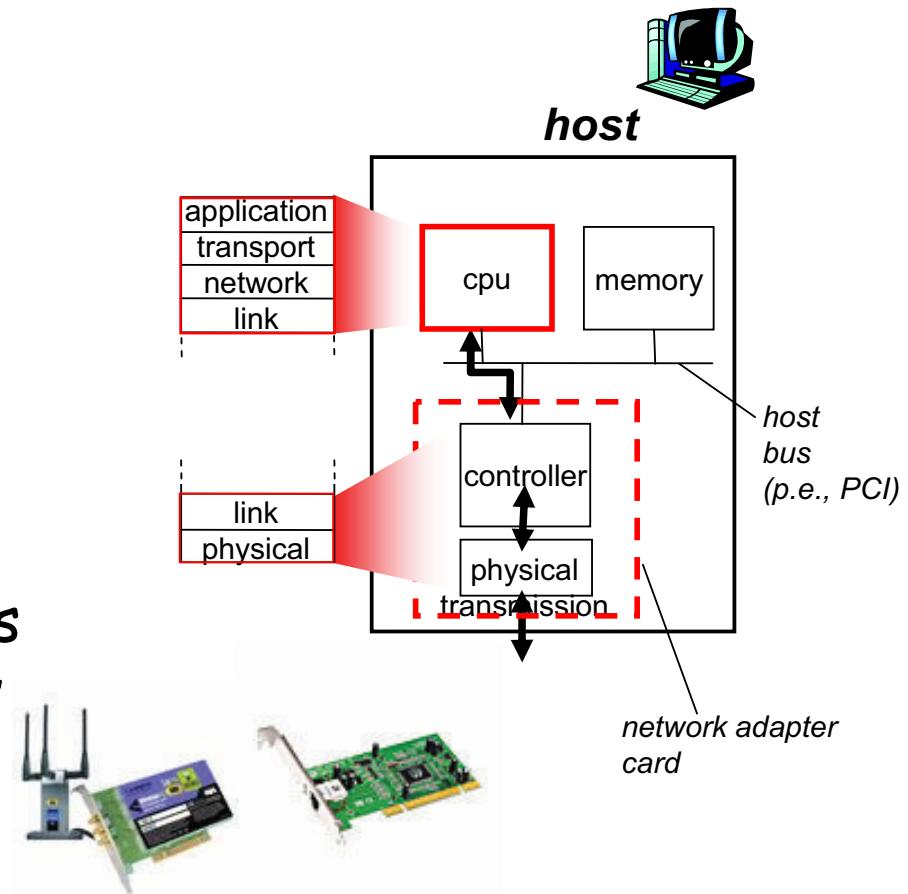
- **entramado (framing):**
  - encapsulado del datagrama en la trama, agregando encabezado (*header*) y cola (*trailer*)
- **acceso al enlace:**
  - acceso al canal si es un medio compartido (*Medium Access Control*)
  - direcciones "MAC" *addresses* utilizadas en los encabezados de las tramas para identificar el origen y el destino
    - distintas de las direcciones IP
- **entrega confiable:**
  - entre nodos adyacentes
  - ¡ya aprendimos cómo hacer ésto (teo Capa de Transp.)!
  - rara vez utilizados en enlaces de pocos errores (fibra óptica, algunos pares trenzados)
  - enlaces inalámbricos: alta tasa de error
    - P: ¿Por qué confiabilidad a nivel de enlace y *end-end*?

# Servicios de Capa de Enlace (más)

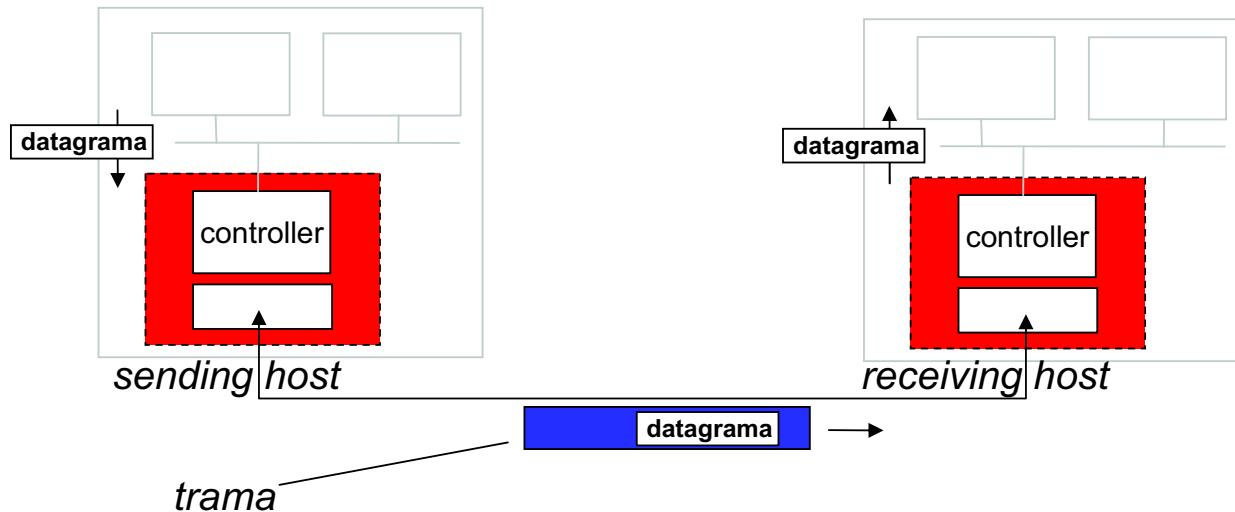
- *control de flujo:*
  - acuerdo entre los nodos emisor y receptor (aquí, adyacentes)
  - Recordar: *buffers* y capacidad de procesamiento
- *detección de errores:*
  - errores causados por atenuación de la señal, por ruido.
  - el receptor detecta presencia de errores:
    - señala al emisor para una retransmisión o descarta la trama
- *corrección de errores (FEC: Forward Error Correction):*
  - el receptor identifica *y corrige* el/los error/es en bit/s sin necesidad de retransmisión
- *half-duplex and full-duplex:*
  - con *half-duplex*, los nodos en los extremos del enlace pueden transmitir, pero no al mismo tiempo

# ¿Dónde está implementada la Capa de Enlace?

- En todos los *hosts*
- En el adaptador de red  
*(Network Interface Card: NIC)*
  - Tarjetas Ethernet, PCMCIA, 802.11
  - Implementa las capas de Enlace y Física (como mínimo)
- Incorporadas a los buses del sistema de los *hosts*
- combinación de *hardware, software, firmware*



# Comunicación de adaptadores



## □ lado emisor:

- encapsula el datagrama en una trama
- agrega bits de chequeo de error, rdt, control de flujo, etc.

## □ lado receptor:

- busca por errores, rdt, control de flujo, etc
- extrae el datagrama y lo pasa a las capas superiores

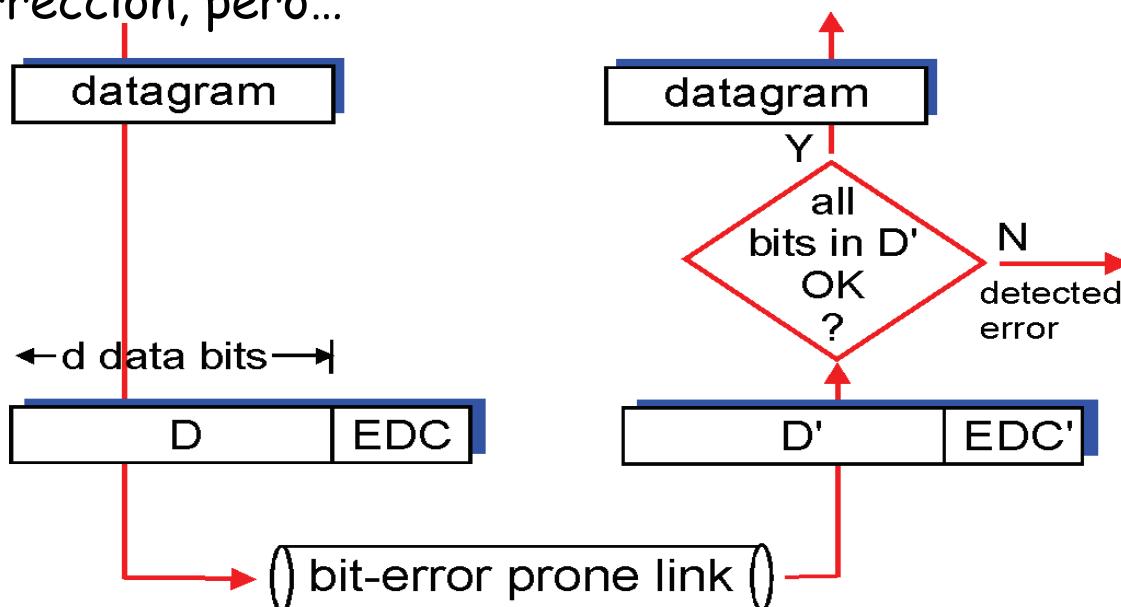
# Detección de errores

EDC= Error Detection and Correction bits (**redundancia**)

D = Datos protegidos por chequeo de errores; puede incluir campos del encabezado

!La detección de errores no es 100% confiable!

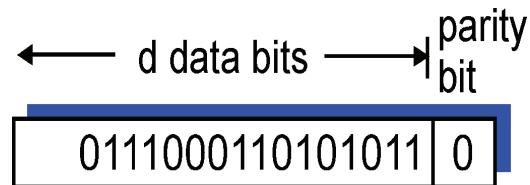
- el protocolo puede perder algunos errores
- un campo de EDC mayor proporciona mejor detección y corrección, pero...



# Chequeo de paridad

## Paridad de un bit:

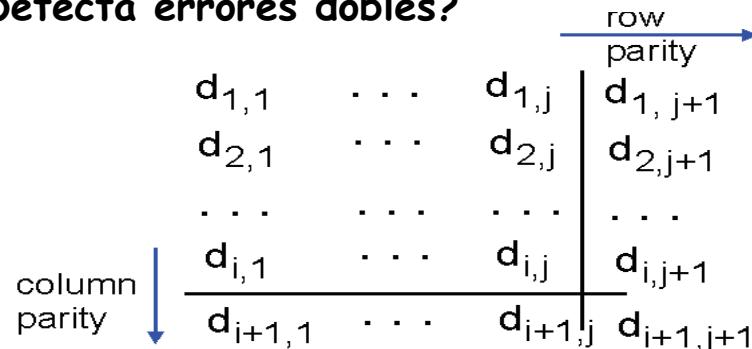
Detecta errores en 1 bit



## Paridad en dos dimensiones:

Detecta y corrige errores en 1 bit

¿Detecta errores dobles?



101011  
111100  
011101  
001010  
*no errors*

101011  
101100  
011101  
001010  
*parity error*

*correctable  
single bit error*

## Internet checksum (suma de comprobación)

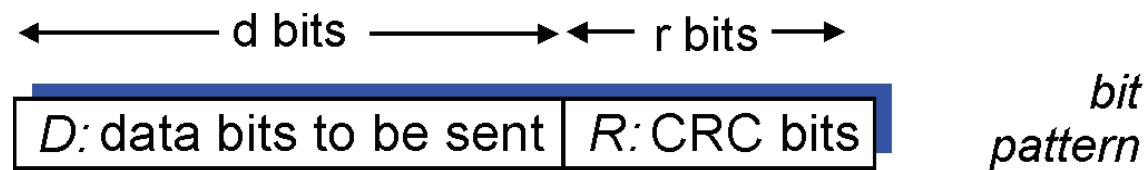
- Objetivo: detectar “errores” (bits cambiados) en el paquete transmitido (nota: generalmente utilizado en la capa de transporte)
- Recordar lo visto en Capa de Transporte
- En general es un método menos potente que el próximo que veremos

## Cyclic Redundancy Check

- códigos CRC o códigos polinómicos
- ampliamente utilizado en la práctica (Ethernet, 802.11 WiFi, ATM)
- ver a los bits de datos, **D**, como los coeficientes de un polinomio
  - por ejemplo: 110001 es  $x^5+x^4+1$
- Toda la aritmética que se utiliza es módulo 2 sin *carry* en las operaciones (sumas y restas equivalentes a XOR)
- elegimos un patrón de  $r+1$  bits (polinomio generador), **G**, de grado  $r$ , que conocen el transmisor y el receptor

## Cyclic Redundancy Check

- objetivo: determinar  $r$  CRC bits,  $R$ , tal que
  - $\langle D, R \rangle$  (concatenado) es divisible exactamente por  $G$ 
    - $D \times 2^r$  es desplazar hacia la izquierda  $r$  bits y agregando 0s
    - $D \times 2^r + R$  es concatenarlos
  - el receptor divide  $\langle D, R \rangle$  entre  $G$ . Si el resto es distinto de cero: **error detectado!**



$$D * 2^r \text{ XOR } R \qquad \textit{mathematical formula}$$

## Ejemplo CRC

- El emisor busca  $R$ , tal que exista  $Q$  que cumpla:

$$D \cdot 2^r \text{ XOR } R = Q \cdot G$$

Que  $G$  divida a  $D \cdot 2^r - R$   
sin resto

$$D \cdot 2^r \text{ XOR } R = Q \cdot G$$

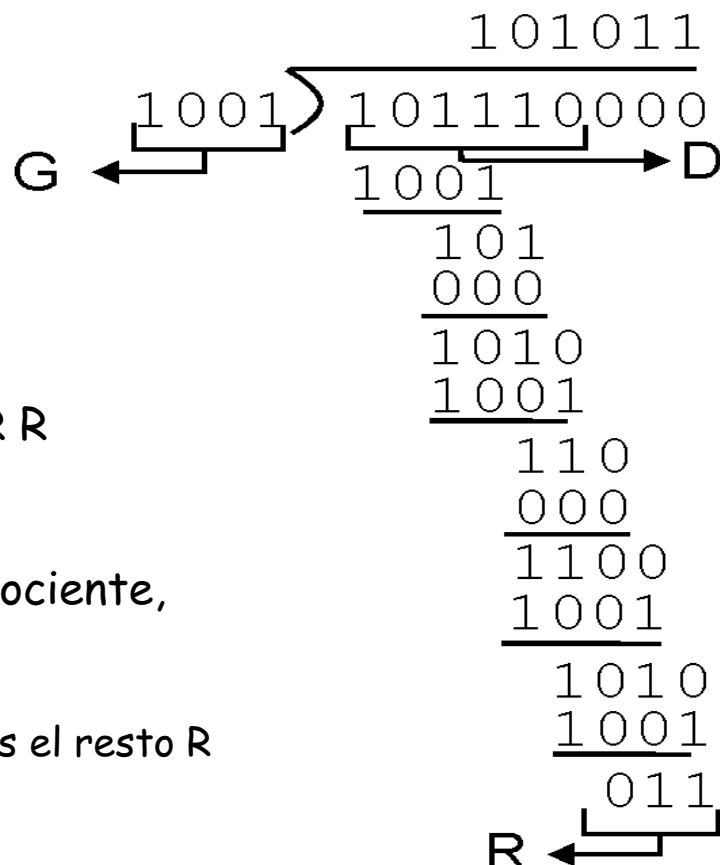
$$D \cdot 2^r \text{ XOR } R \text{ XOR } R = Q \cdot G \text{ XOR } R$$

$$D \cdot 2^r = nG + R$$

$D \cdot 2^r$ : dividendo,  $G$ : divisor,  $Q$ : cociente,  
 $R$ : resto

- si dividimos  $D \cdot 2^r$  por  $G$ , buscamos el resto  $R$

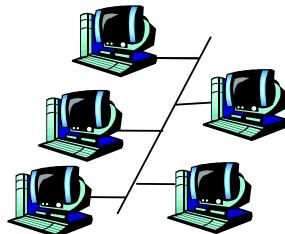
$$R = \text{resto } [ \frac{D \cdot 2^r}{G} ]$$



# Protocolos y enlaces de acceso múltiple

Dos tipos de enlaces:

- punto a punto
  - PPP para acceso discado
  - Enlace punto a punto entre switch Ethernet y *host*
- broadcast** (cable o medio compartido)
  - Ethernet "legacy"
  - HFC: *Hybrid Fiber Cable*
  - 802.11: LAN inalámbrica



cable compartido (p.e.,  
cable Ethernet)



RF compartida  
(p.e., 802.11 WiFi)



RF compartido  
(satélite)



personas en una fiesta  
(aire compartido)

# Protocolos de acceso múltiple

- Único canal *broadcast* compartido
- Dos o más transmisiones simultáneas: interferencia
  - Colisión
    - si un nodo recibe dos o más señales al mismo tiempo
    - simultaneidad en el tiempo y en la frecuencia de dos o más tramas en el mismo medio físico

## Protocolo de Acceso Múltiple

- Algoritmo distribuido que determina cómo los nodos comparten el canal, y determina cuándo el nodo puede transmitir
- La comunicación acerca de compartir el canal debe utilizar el mismo canal
  - no canal *out-of-band* para coordinación

# Protocolo de acceso múltiple ideal

## Canal Broadcast con velocidad R bps

1. cuando un nodo quiere transmitir, lo hará a una velocidad R.
2. cuando M nodos quieren transmitir, cada uno enviará a una velocidad promedio de  $R/M$
3. completamente descentralizado:
  - no hay un nodo especial para coordinar las transmisiones
  - no hay sincronización de relojes, *slots*
4. simple

# Protocolos MAC: taxonomía

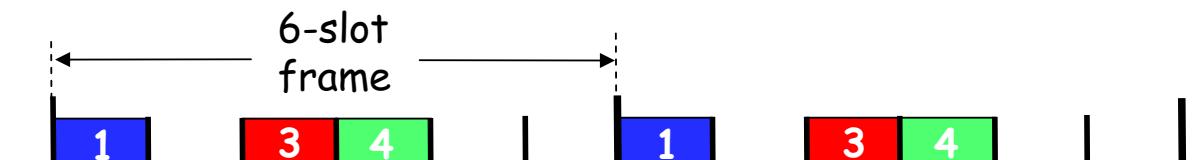
Tres grandes clases:

- Particionado del canal**
  - Protocolos de arbitraje
  - divide el canal en pequeñas "piezas" (ranuras de tiempo, frecuencia, código)
  - asigna una pieza a un nodo para su uso exclusivo
  - estrategia estática
  - equitativo
- Acceso Randómico**
  - el canal no se divide, permite colisiones
  - "recuperación" de colisiones
  - estrategia dinámica
- "Toma de turnos"**
  - Los nodos toman turnos, pero los nodos con más tramas para enviar podrían tomar turnos más largos
  - estrategia dinámica
  - estrategias de reserva o centralizada

## Protocolos MAC de particionado del canal: TDMA

### *TDMA: Time Division Multiple Access*

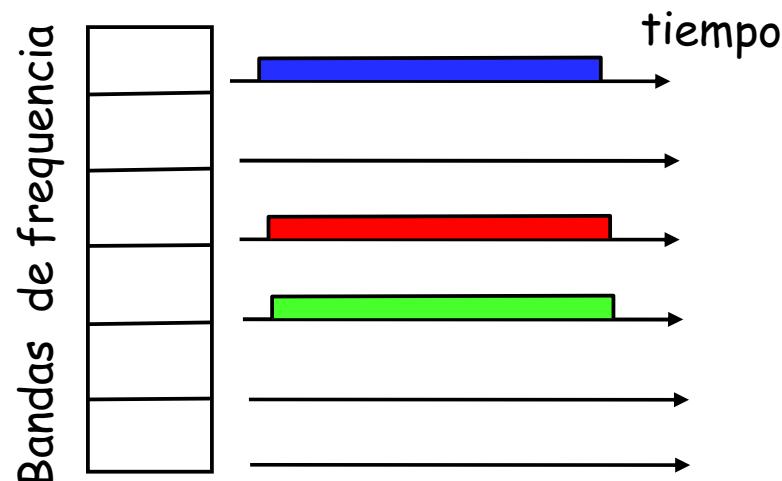
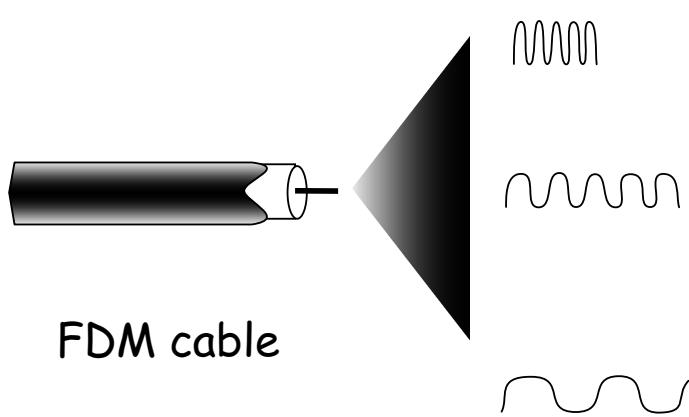
- acceso al canal rotativo
- cada estación tiene un *slot* de longitud fija (longitud = tiempo de transm. de la trama) en cada vuelta
- los *slots* sin usar quedan libres
- ejemplo: LAN con 6 estaciones, 1,3 y 4 tiene trama; los *slots* 2,5 y 6 quedan libres



# Protocolos MAC de particionado del canal: FDMA

## **FDMA: Frequency Division Multiple Access**

- el espectro del canal se divide en bandas de frecuencia
- a cada estación se le asigna una banda de frecuencia fija
- el tiempo de transmisión no utilizado en las bandas de frecuencia queda libre
- ejemplo: LAN con 6 estaciones, 1,3 y 4 tienen trama; las bandas de frecuencia 2,5 y 6 están libres



# Protocolos de acceso randómico

- cuando un nodo tiene un paquete para enviar
  - transmite a la velocidad total del canal,  $R$
  - no existe *a priori* coordinación entre nodos
- dos o más nodos transmitiendo □ “colisión”
- **protocolos MAC de acceso randómico** especifican:
  - cómo detectar colisiones (directa o indirecta)
  - cómo recuperarse de las colisiones (p.e., a través de re-transmisiones retrasadas)
- ejemplos de protocolos MAC de acceso randómico:
  - ALOHA ranurado, ALOHA
  - CSMA, CSMA/CD, CSMA/CA
  - También se les conoce como sistemas de contención o sistemas de contienda

# CSMA (*Carrier Sense Multiple Access*)

**CSMA**: escuchar antes de transmitir

- Si el canal está libre: transmitir la trama entera
- Si el canal está ocupado: diferir la transmisión
  - volver a escuchar después de un tiempo
  - seguir escuchando hasta que quede libre y transmitir
  - seguir escuchando hasta que quede libre y transmitir con probabilidad  $p$
- Analogía humana: ¡no interrumpir a los otros!

## CSMA/CD (*Collision Detection*)

- **CSMA/CD:** si hay presencia de portadora, se difiere la transmisión, como en CSMA
  - las transmisiones que colisionan son abortadas, reduciendo el desperdicio de canal
  - colisión = desperdicio del canal
- detección de colisión:
  - relativamente fácil en LANs cableadas
  - difícil en LANs inalámbricas

# Protocolos MAC “Toma de turnos”

protocolos MAC de particionado del canal:

- compartir el canal *justo y eficiente* a alta carga
- ineficiente a baja carga: retardo en el acceso al canal, ancho de banda  $1/N$  asignado aún si hay un sólo nodo activo

protocolos MAC de acceso randómico

- eficiente a baja carga: un único nodo puede utilizar completamente el canal
- alta carga: *overhead* por colisión

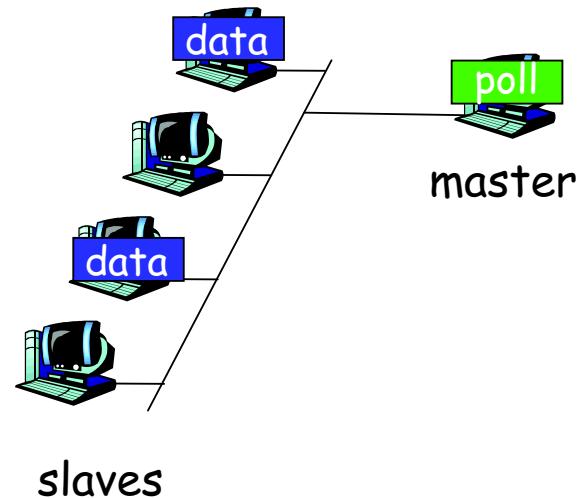
protocolos de “toma de turnos”

busca lo mejor de los dos mundos

# Protocolos MAC "Tomando turnos"

## *Polling:*

- el nodo *master* "invita" a los nodos *slaves* a transmitir en turnos
- típicamente utilizado con dispositivos *slaves* "tontos"
- sin colisiones
- determinístico
- involucra:
  - *overhead* por *polling*
  - latencia
  - único punto de falla (*master*)
- ejemplo
  - Bluetooth
    - IEEE 802.15
  - Un modo de operación de 802.11 (Wi Fi)



# Resumen de protocolos MAC

- *particionado de canal*, en tiempo, frecuencia
  - división en el tiempo, división en la frecuencia
- *acceso randómico* (dinámico),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - Escucha de portadora: fácil en algunas tecnologías (cableadas), difícil en otras (inalámbricas)
  - CSMA/CD utilizado en Ethernet
  - CSMA/CA (*Collision Avoidance*) utilizado en 802.11
- *toma de turnos*
  - *polling* desde un sitio central, pasaje de *token*
  - Bluetooth, Token Ring

# LAN

- Recordar que LAN (*Local Area Network*) es una red concentrada en un área geográfica concreta que podemos asimilarla a una oficina, un piso, un edificio, un campus.
- Recordar además:
  - PAN
  - MAN, WAN
- Velocidades típicas actuales: 10 Mbps, 100 Mbps, 1 Gbps.
- Ya es una realidad: 10 Gbps en cobre.

# Direcciones MAC

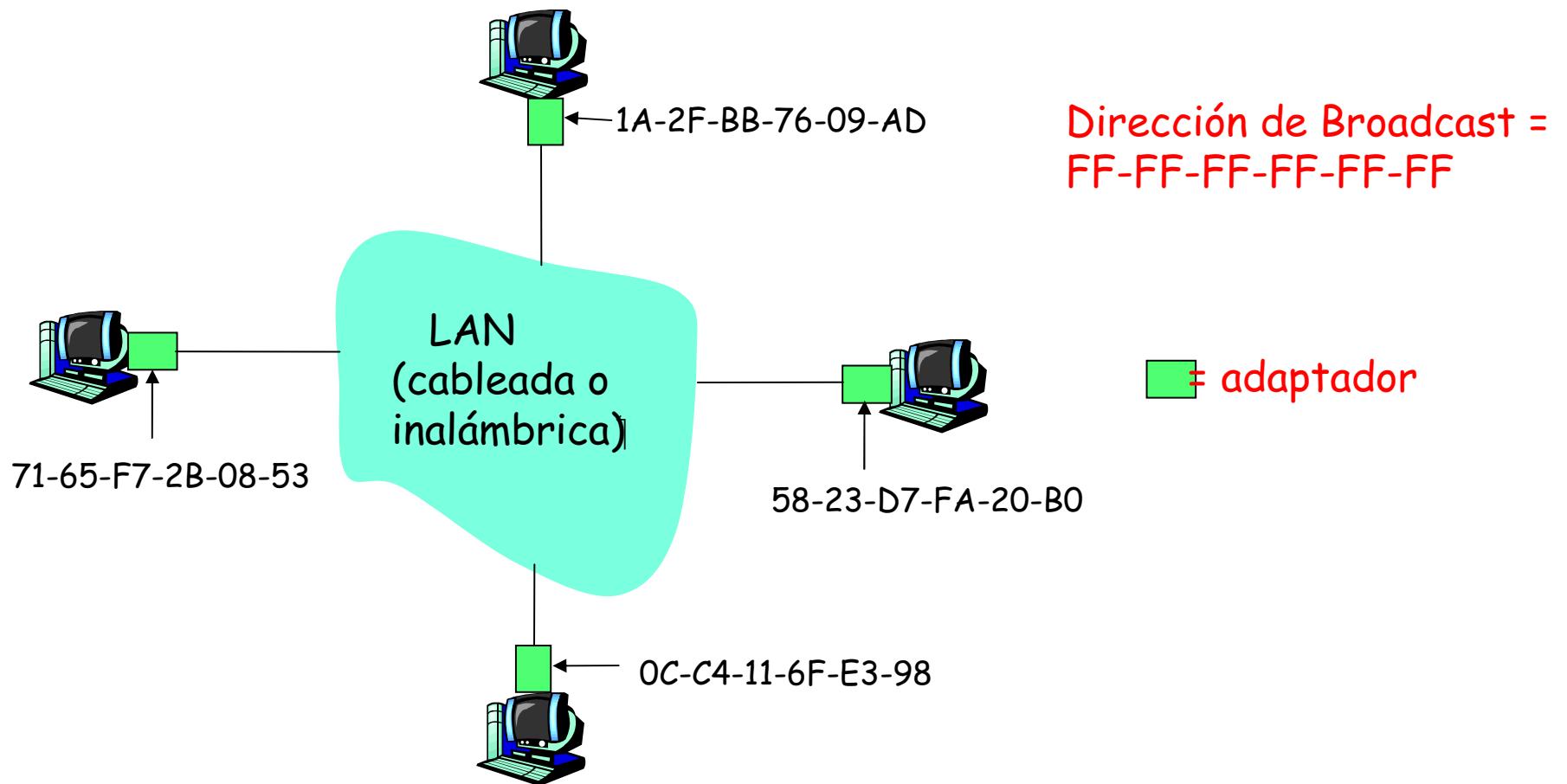
- Direcciones IP de 32 bits:
  - direcciones de la *capa de red*
  - utilizada para llevar el datagrama a la subred IP destino
  
- Dirección MAC (o LAN o física o hardware o del adaptador o "Ethernet"):
  - función: *llevar la trama de una interfaz a otra interfaz físicamente conectada (misma red)*
  - Direcciones MAC de 48 bits (en la mayoría de las redes LAN)
    - grabada en la ROM de la NIC; en algunos casos (cada vez más) configurable por software

# Direcciones MAC

- asignación de direcciones MAC administrada por IEEE
- los fabricantes compran porciones del espacio de direcciones MAC (para asegurar unicidad)
  - OUI (*Organizationally Unique Identifier*): 3 primeros octetos, asignados a las compañías (*company\_id*)
    - <http://standards.ieee.org/regauth/oui/index.shtml>
  - Restantes 3 octetos (*NIC Specific*): administrados por cada compañía
- Dirección MAC plana → portable
  - puedo mover la tarjeta de una LAN a otra
- Dirección IP jerárquica → no portable
  - la dirección depende de la subred IP a la que el nodo está conectado

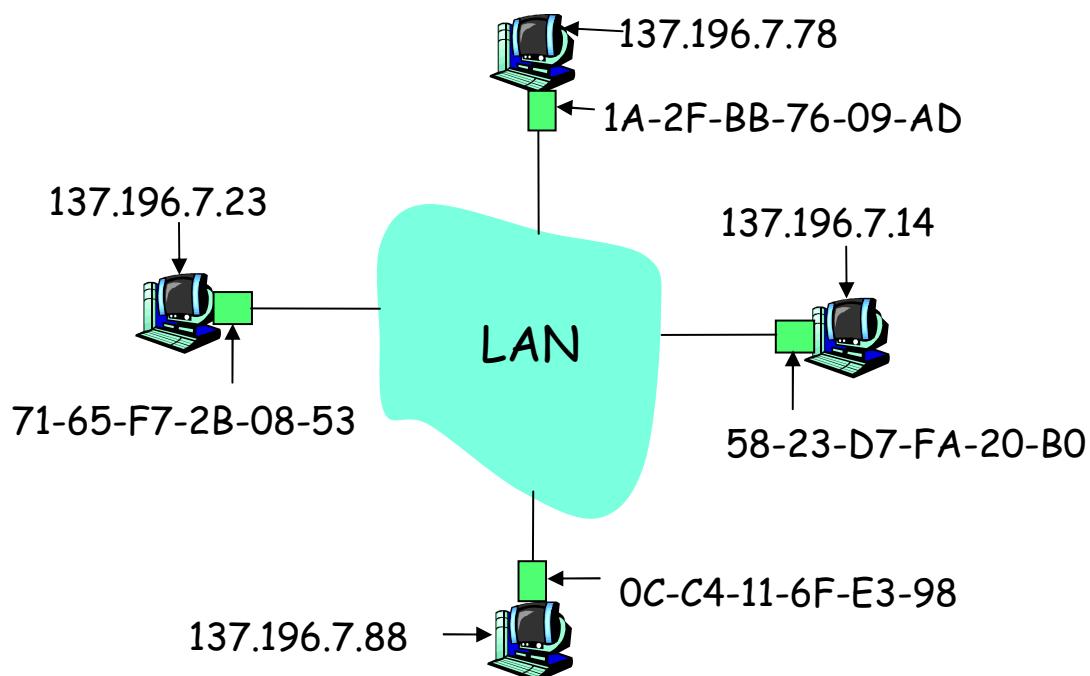
# Direcciones MAC

Cada adaptador en la LAN tiene una dirección LAN única



# ARP: Address Resolution Protocol

Pregunta: ¿Cómo determinamos la dirección MAC de B, conociendo la dirección IP de B?

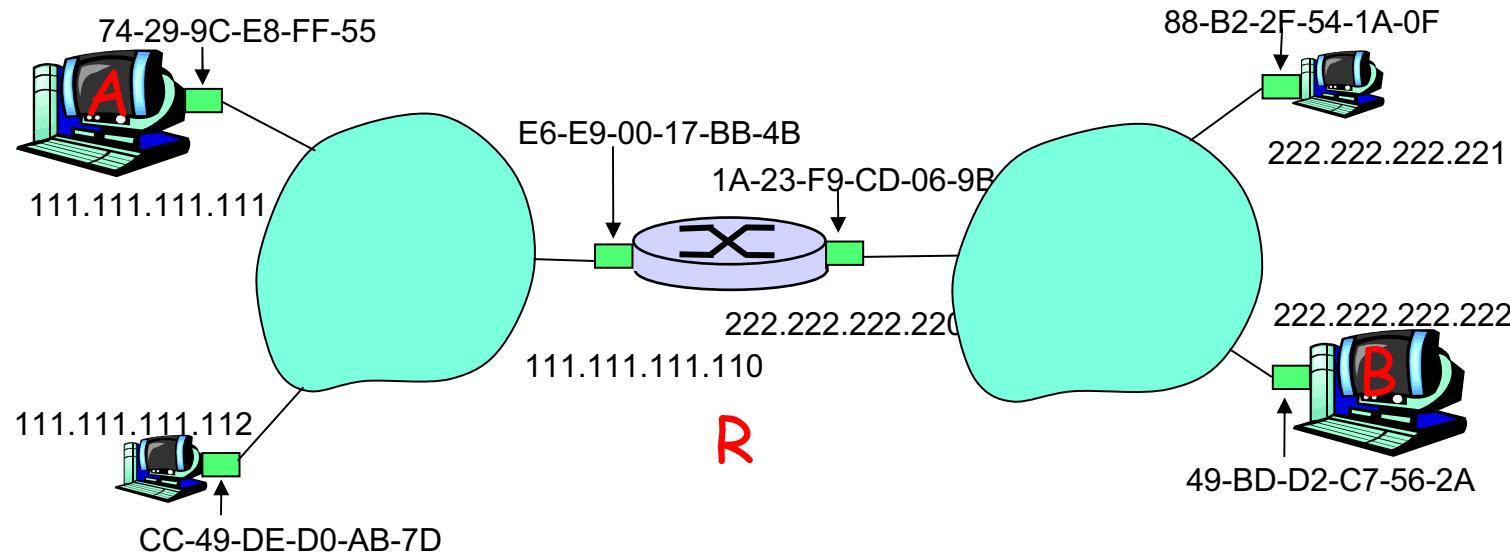


- Cada nodo IP (host, router) en la LAN tiene una tabla **ARP**
- Tabla ARP: mapeo de direcciones IP/MAC para algunos nodos de la LAN
  - < dirección IP; dirección MAC; TTL>
  - TTL (*Time To Live*): tiempo después del cual el mapeo de direcciones debe ser olvidado (por ejemplo, 20 min)

# Direccionamiento: *routing* hacia otra LAN

datagrama desde A hasta B, vía R

asumimos que A conoce la dirección IP de B



- dos tablas ARP en el router R, una para cada red IP (LAN)

# Ethernet

Tecnología LAN cableada dominante:

- Creada "en los 70" (Metcalfe & Boggs)
- NICs baratas (USD 5) y switches baratos
- Primera tecnología LAN ampliamente utilizada
- Más simple y barata que *token LANs* y ATM
- Velocidades: 10 Mbps - 10 Gbps

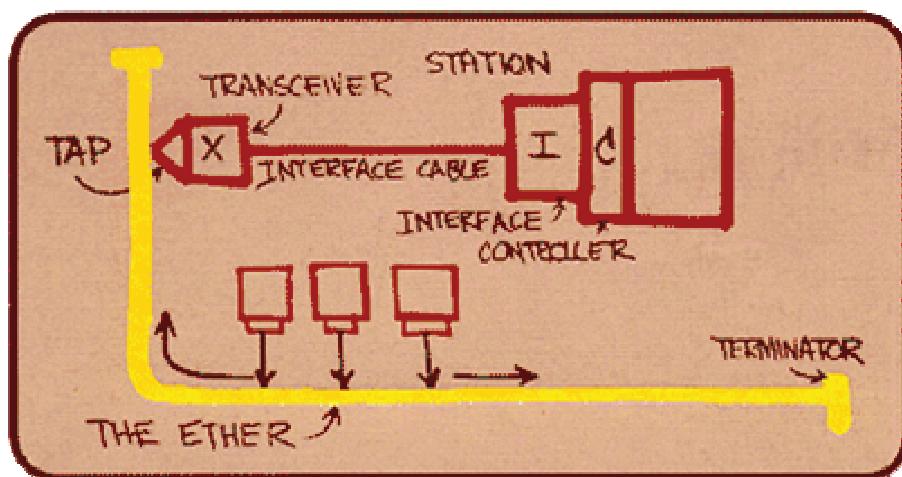
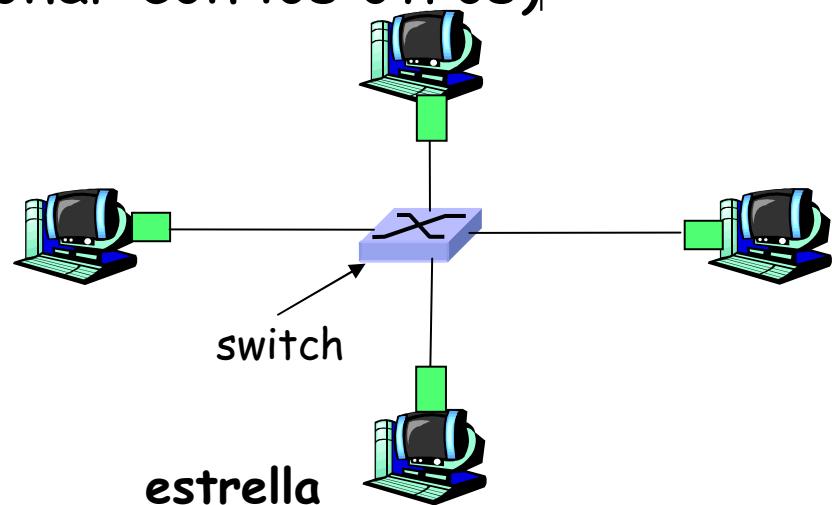
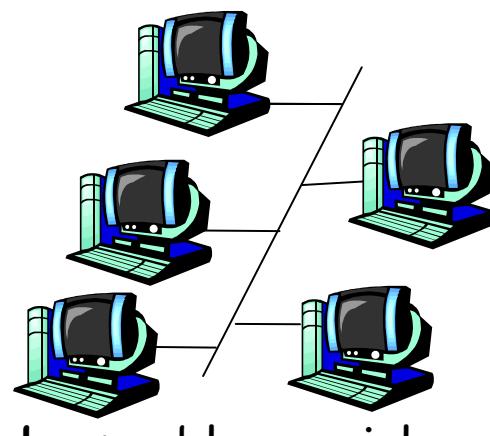


Diagrama de Ethernet  
de Robert Metcalfe

# Topología en estrella

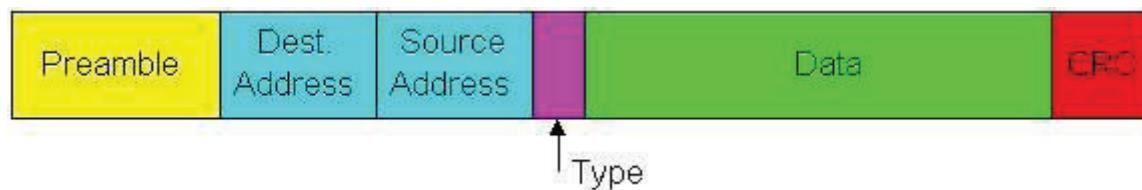
- la topología en bus fue popular hasta mediados de los 90
  - todos los nodos en el mismo dominio de colisión (pueden colisionar con cualquiera de los otros)
- hoy: prevalece la topología *estrella*
  - **switch** activo en el centro (desde "fines de los 90")
  - cada "spoke" corre el protocolo Ethernet (los nodos no pueden colisionar con los otros)



# Estructura de la trama

## Ethernet

- El adaptador del emisor encapsula el datagrama IP (u otro paquete de protocolo de capa de red) en una **trama Ethernet**

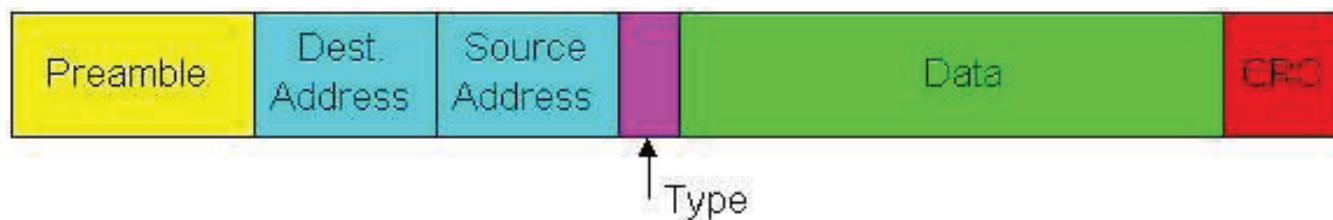


### *Preamble:*

- siete bytes con el patrón 10101010 seguido por un byte con el patrón 10101011
- utilizado para despertar al receptor y sincronizar los relojes de emisor y receptor

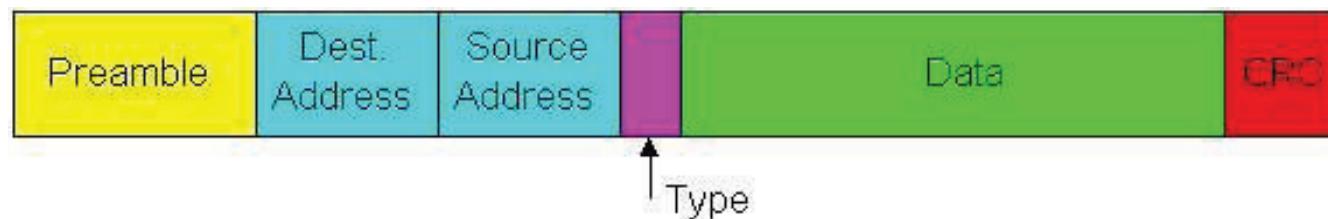
# Estructura de la trama Ethernet (más)

- **Direcciones:** 6 bytes cada una
  - si el adaptador recibe una trama con dirección destino la suya o la dirección de *broadcast*, (ej. paquete ARP), pasa los datos en la trama al protocolo de capa de red
  - en otro caso, el adaptador descarta la trama
- **Type:** 2 bytes
  - multiplexación
  - indica el protocolo de la capa superior (casi siempre IP pero otros es posible, p.e., IPX, AppleTalk)



# Estructura de la trama Ethernet (más)

- **Data:** de 46 a 1500 bytes
- **CRC:** 4 bytes
  - CRC-32
  - chequeado en el receptor, si un error es detectado, la trama es descartada
  - Para calcularlo se utiliza todo menos el "Preamble"

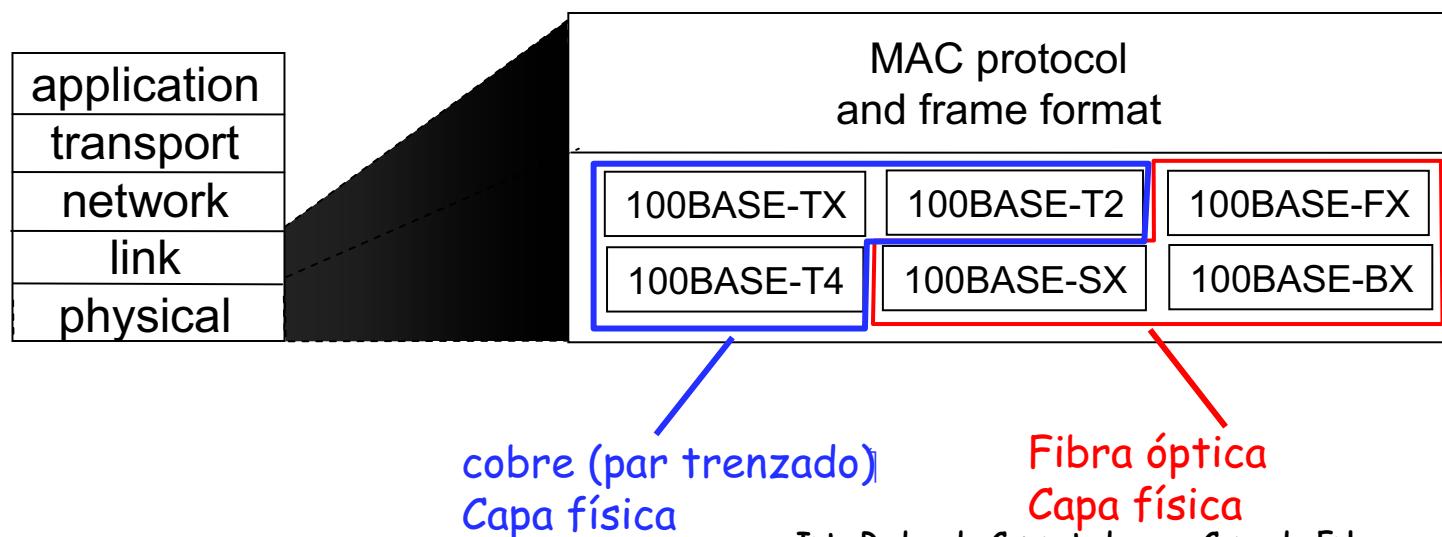


# Ethernet: servicio no confiable, no orientado a conexión

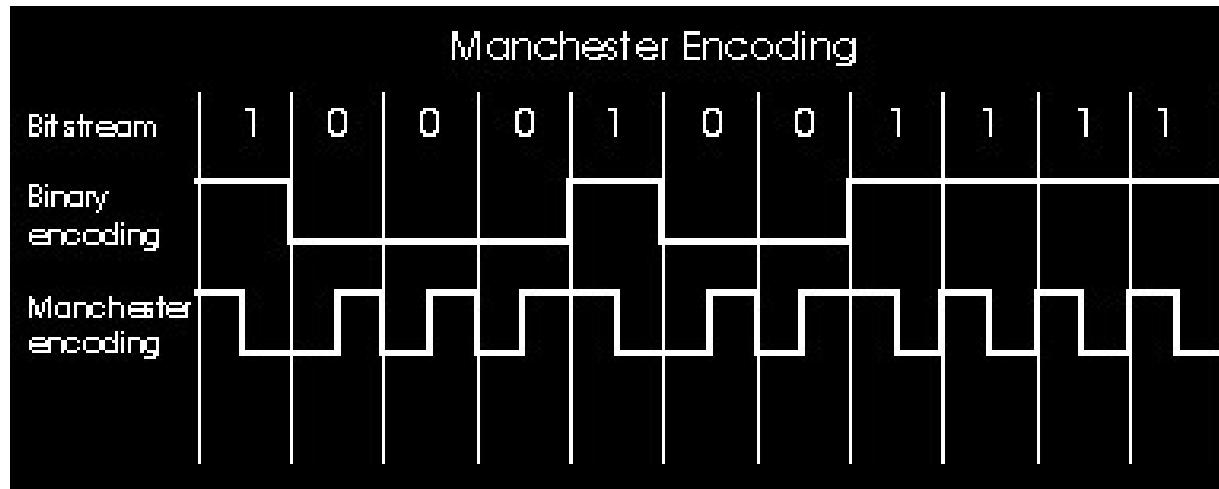
- **No orientado a conexión:** No hay *handshaking* entre las NICs de emisor y receptor
- **No confiable:** la NIC que recibe no envía ACKs o NAKs a la NIC emisora
  - el flujo de datagramas pasados a la capa de red puede tener huecos (datagramas perdidos)
  - los huecos serán llenados si la aplicación utiliza TCP
  - en otro caso, la aplicación verá los huecos
- Protocolo MAC de Ethernet: **CSMA/CD**
- La detección de colisiones es un servicio de Capa Física

## 802.3 Ethernet Standards: Capas de Enlace y Física

- **varios** diferentes estándares Ethernet
  - protocolo MAC y formato de trama único
  - diferentes velocidades: 2 Mbps, 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps
  - diferentes medios físicos: fibra óptica, cable



# Codificación Manchester

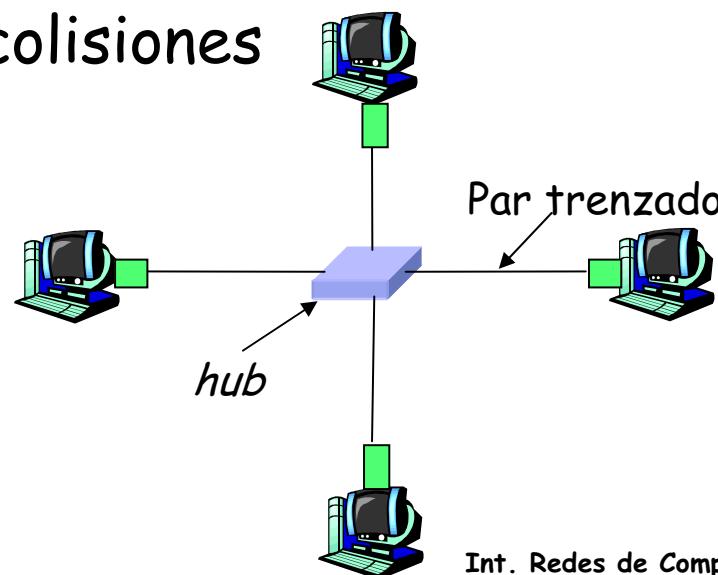


- Utilizado en 10BaseT
- Cada bit tiene una transición
- Permite que los relojes de los nodos emisores y receptores siempre estén sincronizados entre sí
  - No se requiere un reloj centralizado, global

# Hubs

... repetidores de Capa Física ("tonto"):

- los bits que llegan en un *link* salen por *todos* los otros *links* a la misma velocidad
- todos los nodos conectados al *hub* pueden colisionar con los otros
- no existe *buffering* de tramas
- no hay *CSMA/CD* en el hub: la *NIC* del *host* detecta las colisiones

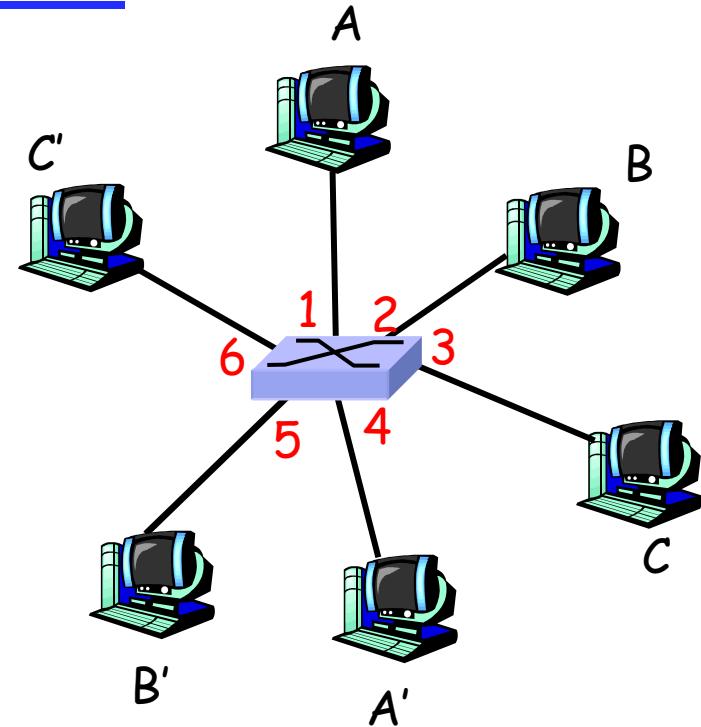


# Switch

- dispositivo de Capa de Enlace: más "inteligente" que los *hubs*, tienen un rol *activo*
  - almacenamiento, envío de tramas Ethernet
  - examina la dirección MAC destino de la trama entrante, realiza un envío **selectivo** de la trama a uno o más *links* de salida; cuando la trama será enviada en un segmento, utiliza CSMA/CD para acceder al segmento
- *transparente*
  - los *hosts* no se "enteran" de la presencia de los *switches*
- *plug-and-play, self-learning*
  - los *switches* no necesitan ser configurados (para su operación básica)

## Switch: permite múltiples transmisiones simultáneas

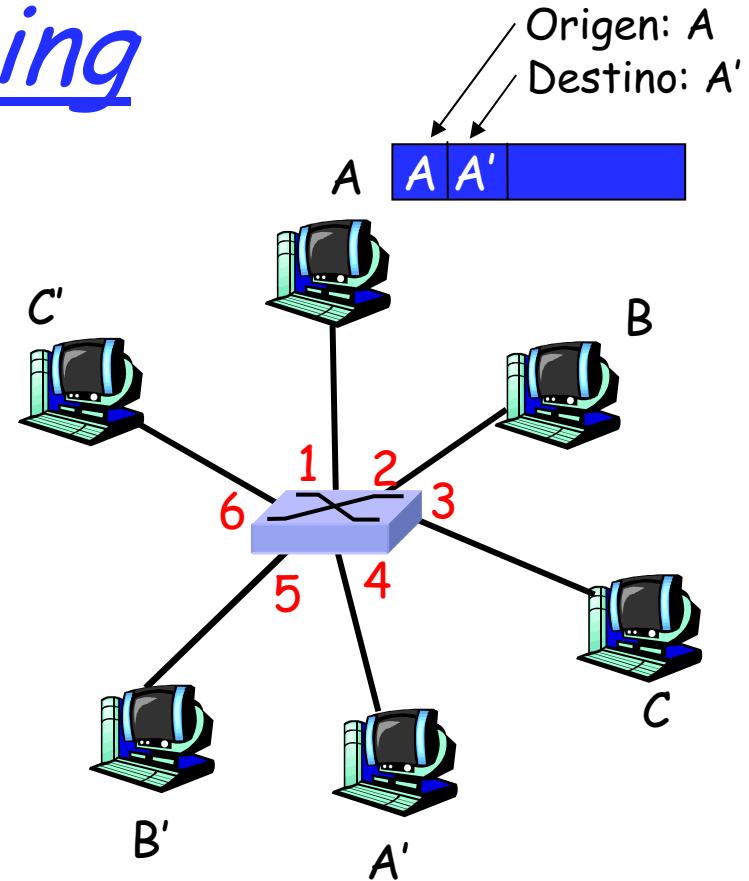
- Los hosts tienen conexiones dedicadas, directas al switch
- Los switches hacen buffer de las tramas
- El protocolo Ethernet es utilizado en *cada link entrante*, pero no hay colisiones; *full duplex*
  - cada link es su propio dominio de colisión
- switching:** A-to-A' and B-to-B' simultáneamente, sin colisiones
  - no posible con hub



switch con seis interfaces  
(1,2,3,4,5,6)

# Switch: self-learning

- el switch *aprende* qué hosts puede ser alcanzado a través de qué interfaces
  - cuando una trama es recibida, el switch "aprende" la ubicación del emisor: el segmento LAN de entrada
  - registra el par emisor/ubicación en la tabla del switch



Dir. MAC	interfaz	TTL
A	1	60

*Tabla del switch  
(inicialmente vacía)*

# Switch: filtering/forwarding de tramas

Cuando una trama es recibida:

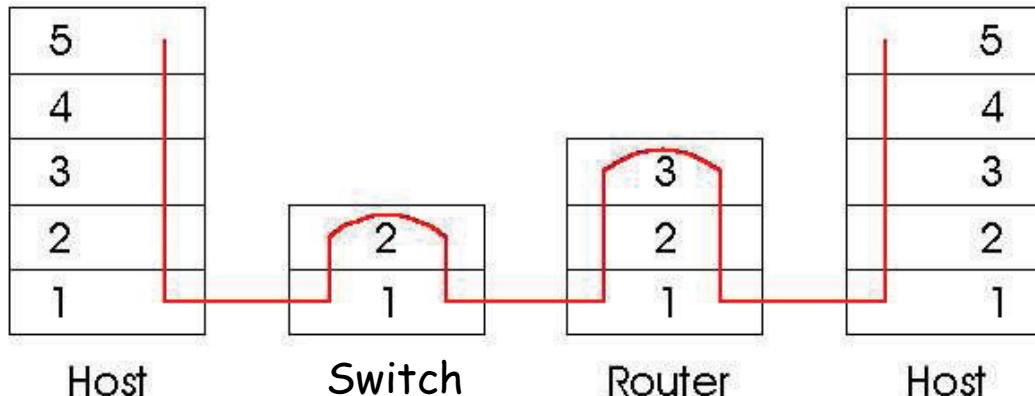
1. registra el link asociado con el host que envía
  2. busca en la *switch table* utilizando la dirección MAC destino
  3. if encuentra una entrada para el destino  
    then {  
        if destino en segmento de donde arribó la trama  
            then descartar la trama  
        else forward de la trama en la interfaz indicada  
    }  
    else flood
- forward en todas las interfaces  
menos en la que arribó*

# Técnicas de commutación de tramas

- Técnicas utilizadas por los *switches* para pasar la trama desde el puerto de entrada hasta el puerto de salida
- Se decide en función de la DA
- Dos grandes familias
  - *Cut-through*
    - Sólo espera la *Destination Address*
    - No realiza FCS (Frame-Check-Sequence)
  - *Store & Forward*
    - Espera toda la trama
    - Realiza FCS

# Switches vs. Routers

- ambos son dispositivos *store-and-forward*
  - routers: dispositivos de capa de red (examina encabezados de capa de red)
  - switches: dispositivos de capa de enlace
- los routers mantienen tablas de *routing*; implementan algoritmos de *routing*
- los switches mantienen tablas de switch, implementan filtrado, algoritmos de aprendizaje



# Segmentando redes LAN...

- "Teoría de Darwin de las redes LAN" ☺ :
  - la evolución del **hub** al **switch**
  - existió un dispositivo intermedio que vivió poco: el **bridge**
- Hub
  - Capa Física
  - 1 dominio de colisión y 1 dominio de broadcast
- Bridge
  - Capa de Enlace de Datos
  - 1 dominio de colisión en cada puerta y 1 dominio de broadcast
- Switch
  - Capa de Enlace de Datos
  - 1 dominio de colisión en cada puerta y 1 dominio de broadcast
  - Pero además, mayor
    - cantidad de puertas que un bridge
    - capacidad de conmutación de tramas que un bridge

# Red "switcheadas"

- Redundancia
  - Confiabilidad, disponibilidad
  - Costos
  - Pero quizás también, inestabilidad
    - Por ejemplo, un simple *ARP request* puede generar una tormenta de broadcast y afectar la *performance* de los switches de toda la red
    - Algo similar puede ocasionar un *unicast*
    - Precisamos una solución que evite los *loops* pero sin perder las bondades de la redundancia
  - En capa de enlace no existe el concepto de TTL
- *Spanning-Tree Protocol (STP)*: Protocolo de gestión de capa de enlace que pone a disposición la redundancia de caminos pero previene de posibles *loops* en la red de *switches* ( posible origen de duplicación de mensajes)

# Protocolo Spanning-Tree (STP)

- El objetivo es que en cada instante exista un solo camino activo entre dos *switches*
  - Que existan *loops* físicos pero no lógicos
- Se define un árbol a través del cual se alcanza a todos los *switches* pero el árbol se “poda” de tal forma que algunos puertos quedan bloqueados a la espera de algún cambio topológico y los restantes puertos están en estado forwarding
- Algunos comentarios
  - Protocolo transparente a los usuarios
  - Radia Perlman -> IEEE 802.1D
  - “Protocolo de árbol de expansión”
  - Referencias en la bibliografía
    - Secciones 4.4 o 4.7 “del Tanenbaum”
    - Sección 5.6 “del Kurose & Ross”

# VLAN: Virtual LAN

- Empresa con  $k$  departamentos
  - 1 red LAN por departamento
    - Agrupar lógicamente usuarios de la red y recursos conectados a puertos definidos administrativamente
    - Broadcast
    - Seguridad
    - Carga
- En los 90's:  $k$  redes LAN independientes significaba instalar  $k$  hubs (como mínimo)
- Luego, se incorporaron los *switches*
- Ahora:  $k$  redes LAN, técnicamente puede significar simplemente instalar 1 *switch*

## VLAN: Virtual LAN (más)

- IEEE 802.1Q
- Permite crear “*switches* virtuales” en uno o más *switches* y de esa forma separar dominios de broadcast (más pequeños)
- Se debe definir:
  - Cantidad
  - Nombre de cada una (“color”)
  - Miembros de cada una
- En cada puerto del *switch*, una sola VLAN posible, salvo en los *trunks*

# Enlace de Datos Punto a Punto

- un emisor, un receptor, un enlace: más fácil que un enlace *broadcast*:
  - no se requiere *Medium Access Control*
  - no se necesita direccionamiento MAC explícito
  - p.e., enlace discado
- protocolos *point-to-point* más populares:
  - PPP: *Point-to-Point Protocol*
  - HDLC: *High level Data Link Control*

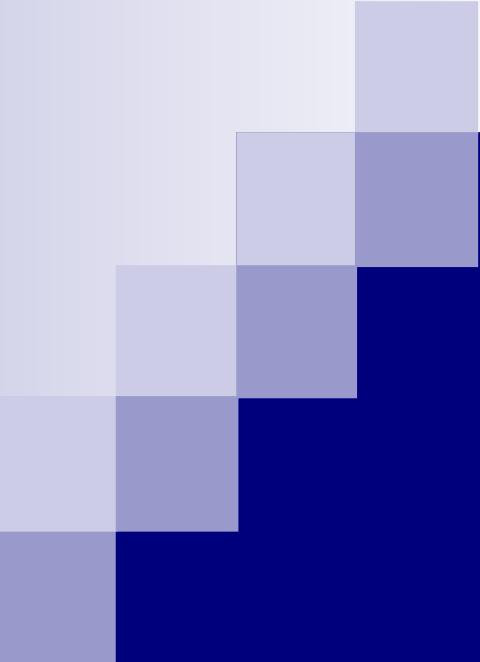
# PPP (RFC 1547, 1661, 1962, 2153)

- Requerimientos de diseño de PPP: RFC 1547
  - simple
  - **entramado de paquete:** encapsulado del datagrama de capa de red en una trama de capa de enlace
  - **transparencia:** debe poder llevar cualquier patrón de bit en el campo de datos (incluso los vinculados al *framing*)
  - **multiplexación:** porta datos de capa de red de cualquier protocolo (no solamente IP) al mismo tiempo
    - posibilidad de demultiplexar
  - **detección de error** (no corrección)
  - **estado de la conexión:** detectar y señalizar a la capa de red sobre falla en el *link*
  - **negociación de la dirección de la capa de red:** un *endpoint* puede configurar la dirección de red del otro
  - **posibilidad de negociación de opciones**
  - **posibilidad de compresión de datos**

## No requerimientos de PPP

- corrección/recuperación de errores
- control de flujo
- entrega de tramas en orden (secuenciamiento)
- no hay necesidad de soporte de enlaces multipunto (p.e., *polling*)

Recuperación de errores, control de flujo, re-ordenamiento de datos  
son relegados a las capas superiores



# Bridging y Switching (Ethernet)

UNLP – Fac. De Informática  
2020

# Contenido

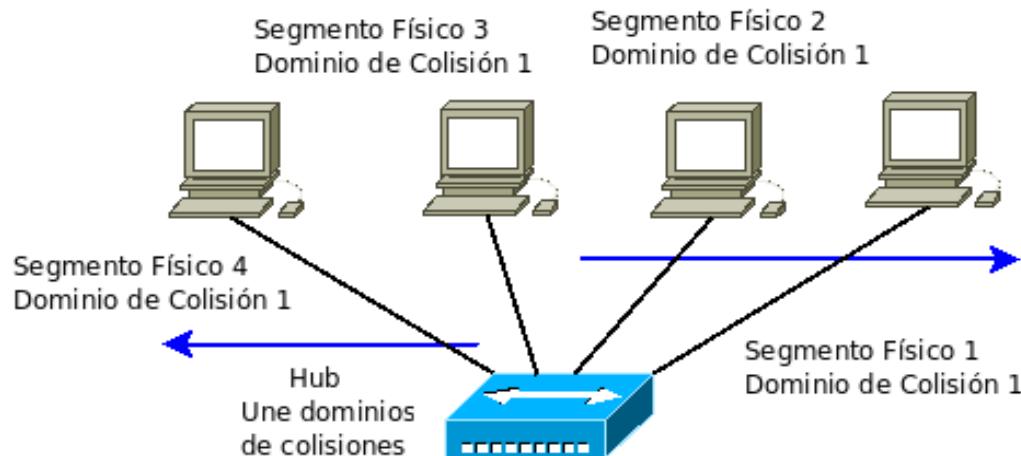
- Dispositivos de LAN.
- Tablas de MAC.
- Métodos de Switching.
- Spanning Tree Protocol.
- Administración.
- VLANs.

# Dispositivos de LAN

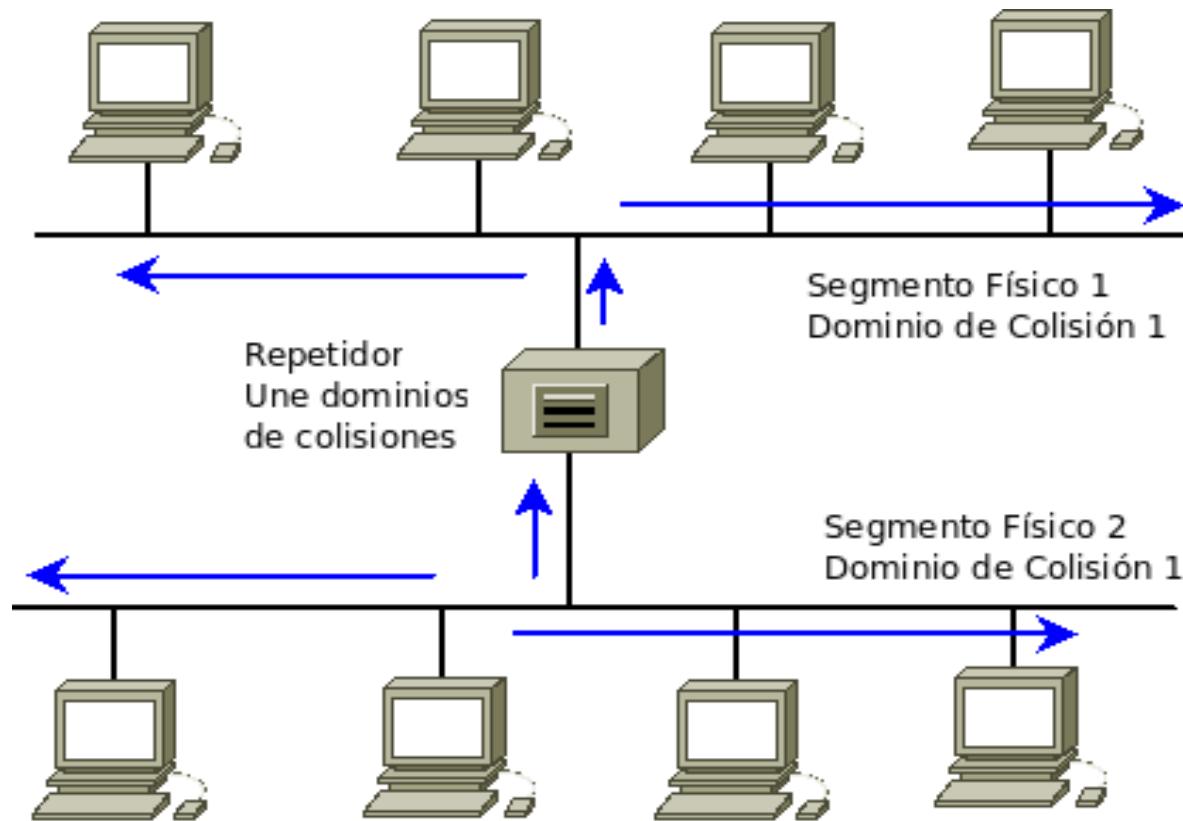
- Dominio de Colisión vs. Dominio de Broadcast.
- Micro-segmentación.
- Dispositivos:
  - Repetidor.
  - Bridge.
  - Switch.
  - Router.

# Repetidor/Hub

- **Repetidor:** amplificador digital, dos puertos. Regenera la señal une dominios de colisión generando un único, permite extensión.
- **Hub:** repetidor multipuerto. Usado en 10BaseT y 100BaseT.

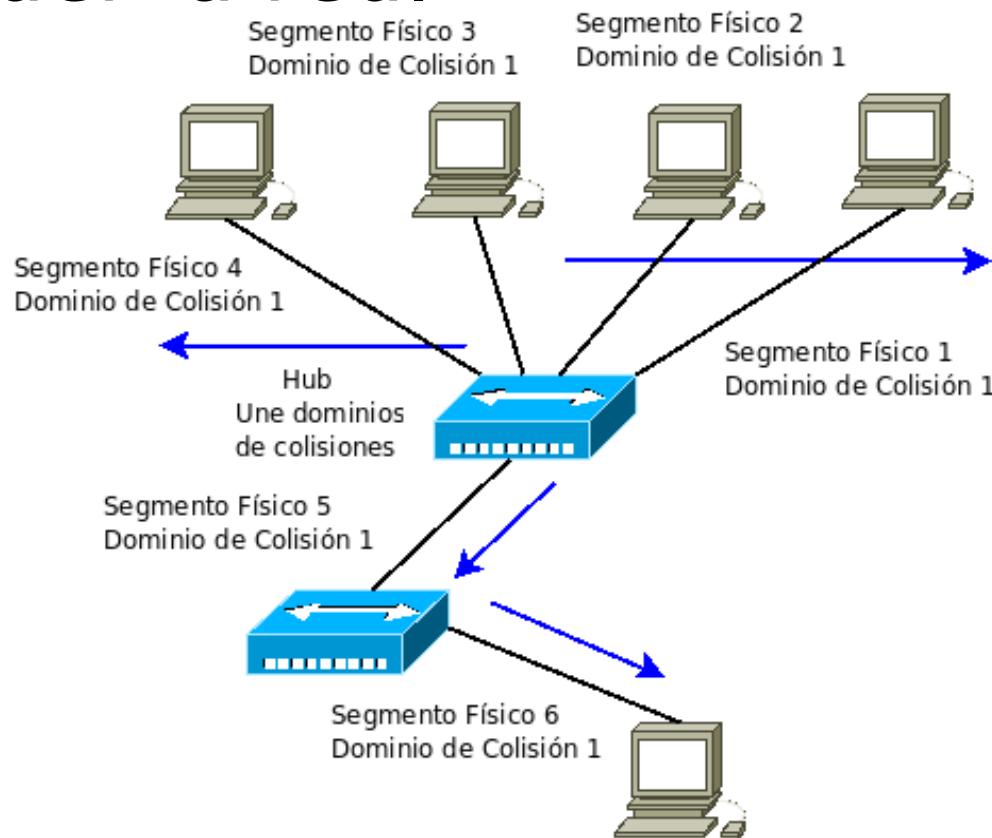


# Repetidor



# Cascadas/Uplinks (N2N)

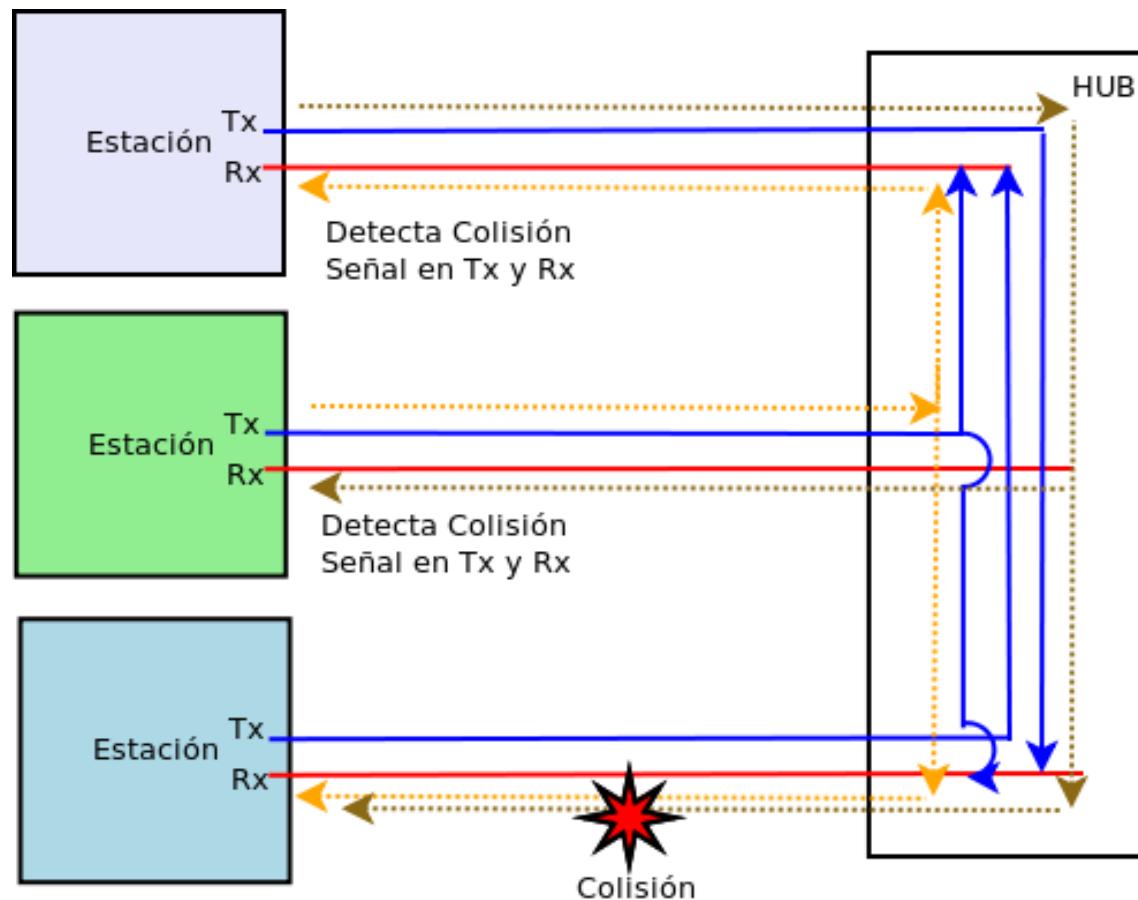
- Permitir interconectar concentradores y repetidores para extender la red.



# Dominio de Colisión

- Hasta donde pueden extenderse las colisiones.
- Hasta donde llega la señal de una trama unicast.
- Todas las estaciones en el mismo dominio de colisiones ven los datos transmitidos de cada una.
- Un repetidor o un hub extienden un dominio de colisión.

# Hubs y Colisiones



# Tipos de Hubs

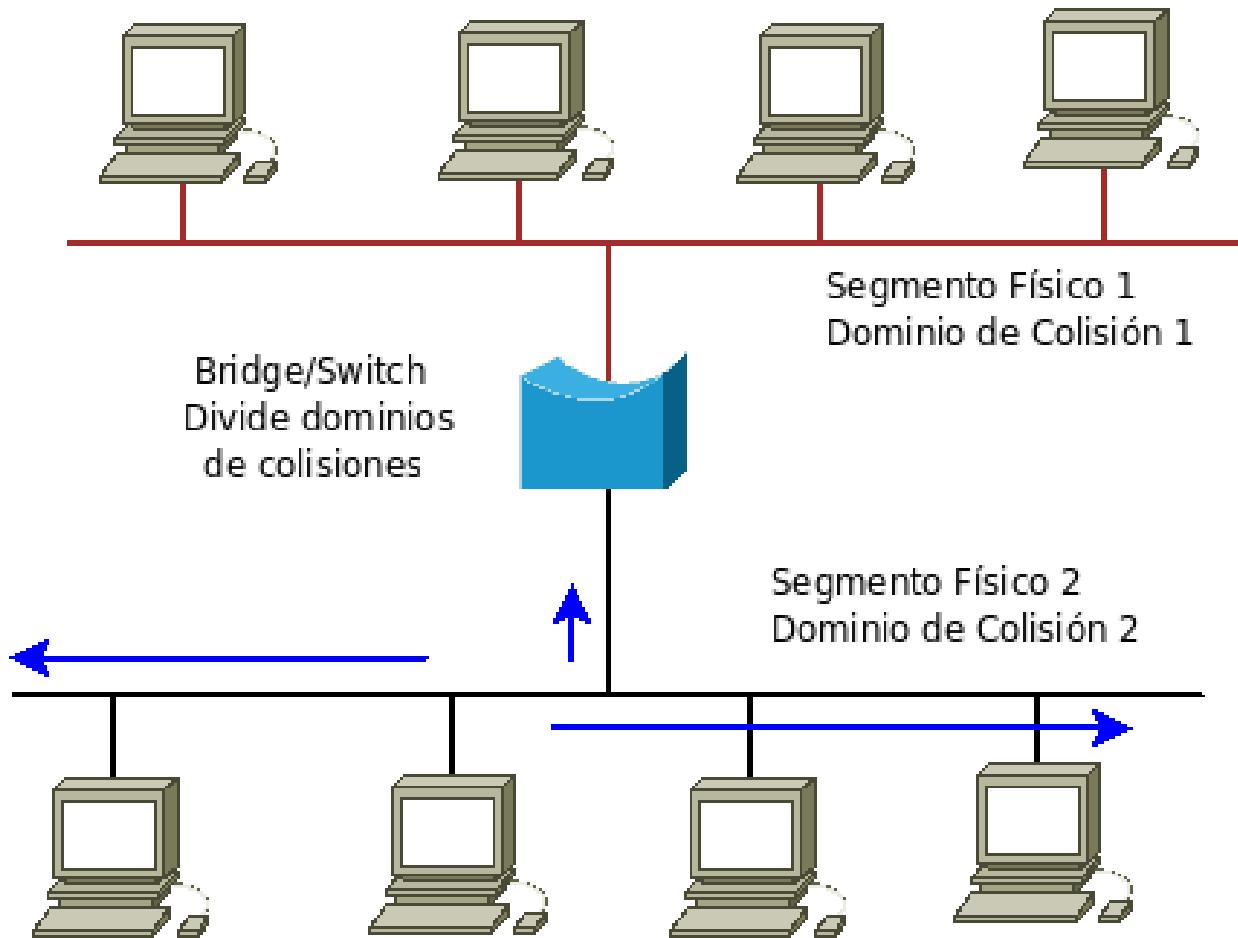
- **Hubs pasivos:** solo envían la señal por todos los puertos restantes.
- **Hubs activos:** regeneran la señal, mayor alcance.
- **Hubs inteligentes:** pueden poseer administración, permiten detectar problemas.
- Los hubs pueden detectar colisiones y generar JAMs.

# Bridge

**Bridge:** Poder adaptar entre dos protocolos de nivel de enlace o físico, pueden ser diferentes. Dividir dominio de colisión.

- Dividir la red en partes más pequeñas: dominios de colisiones.
- Permitir escalabilidad.
- Implementado por software.
- Dos puertos en general.
- Bridge Ethernet podría adaptar dos tecnologías de nivel físicas, e.g.: 10Base2 y 10BaseT.

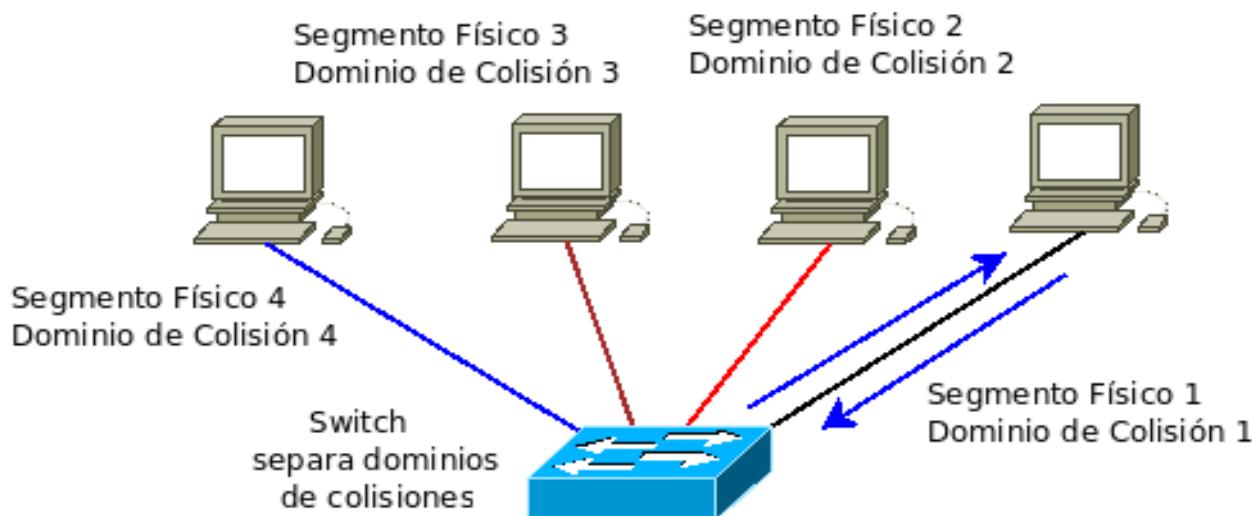
# Bridge



# Switch

**Switch:** un bridge multipuerto que trabaja con la misma tecnología de enlace y física en c/u.

- Trabaja en hardware, ASIC, múltiples puertos.
- Puertos trabajan en FDX, micro-segmentación.



# Switching

## Razones para usar switches en una red:

- Dividir la red en partes más pequeñas (dominios de colisiones, micro-segmentación).
- Seguridad: VLANs, admin.
- Mejorar el rendimiento de la red. FDX vs. HDX.
- No hay colisiones.
- Los switches tienen menor delay.
- Actualidad Switches multilayers o L3/capa3.



# Bridging/Switching

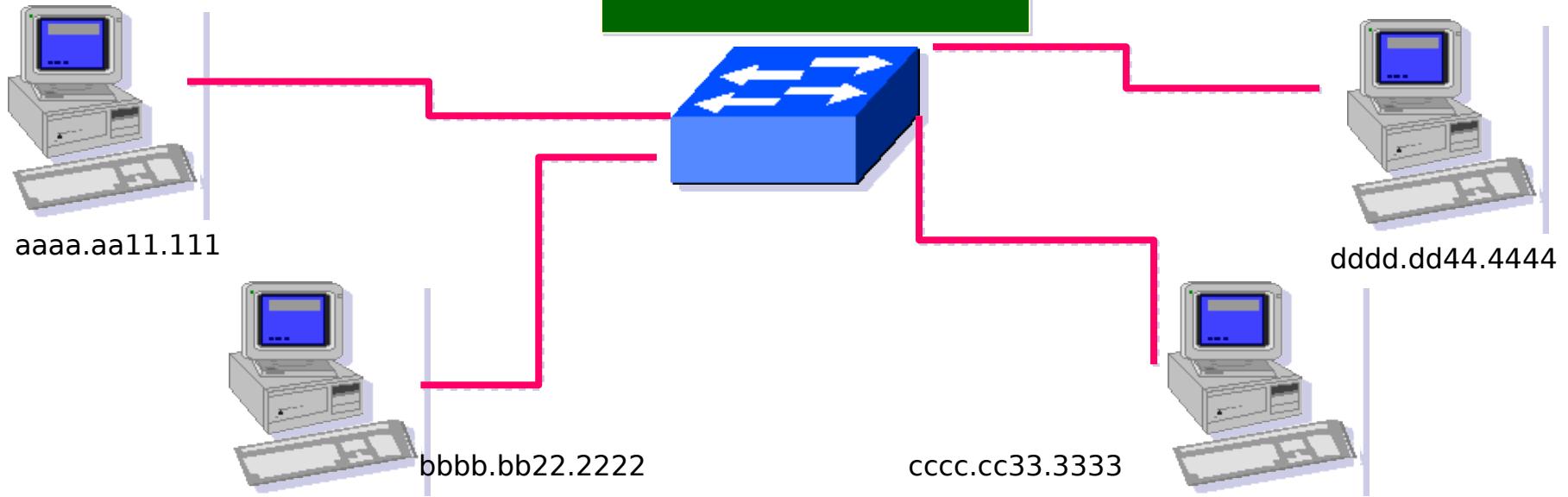
## Funciones del switch

- **Aprender direcciones MAC:**  
El dispositivo guarda las direcciones MAC asociadas a cada puerto en una base de datos.
- **Reenviar / filtrar paquetes:**  
Al recibir una trama, el switch revisa su base de datos MAC para determinar a través de que puerto puede alcanzar la dirección de destino.
- **Evitar bucles de capa 2:**  
Los switches administran los bucles de redundancia con STP. Bridges solo una instancia de STP, switches podrían correr varias,

# Switching

## Aprendizaje de direcciones

Tabla direcciones MAC, mem CAM

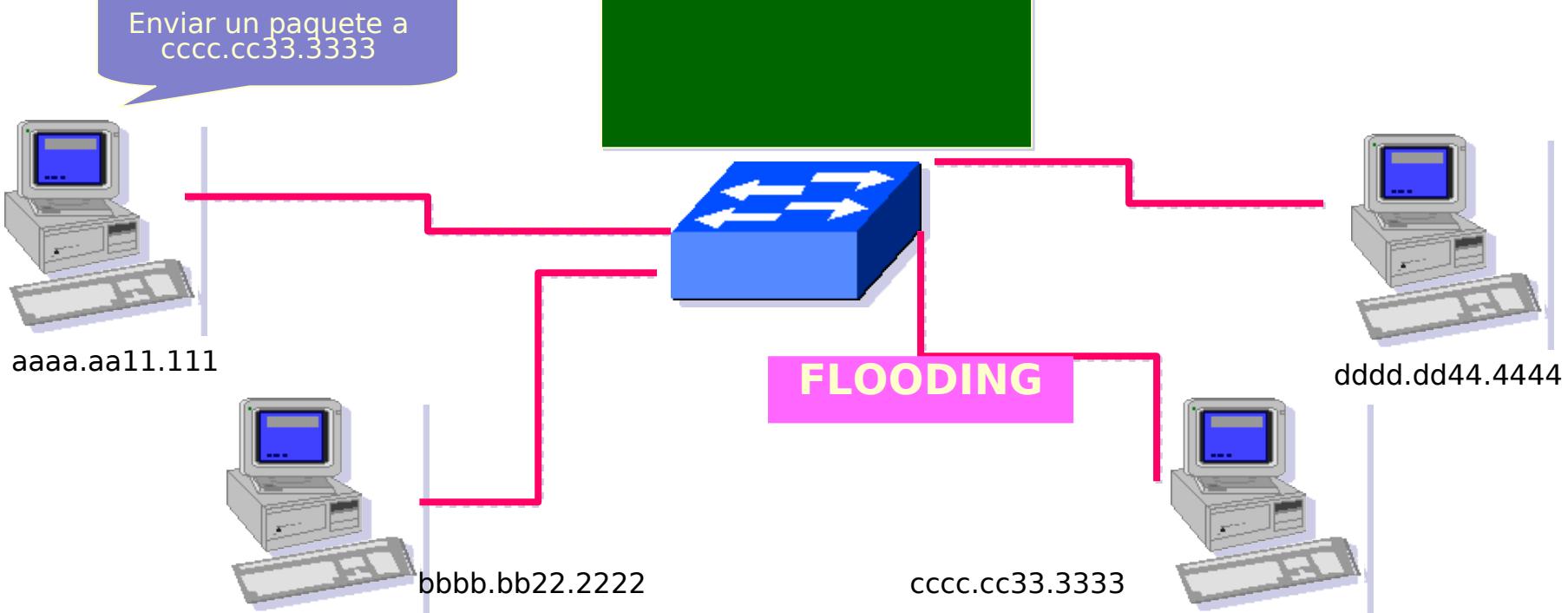


# Switching

## Aprendizaje de direcciones

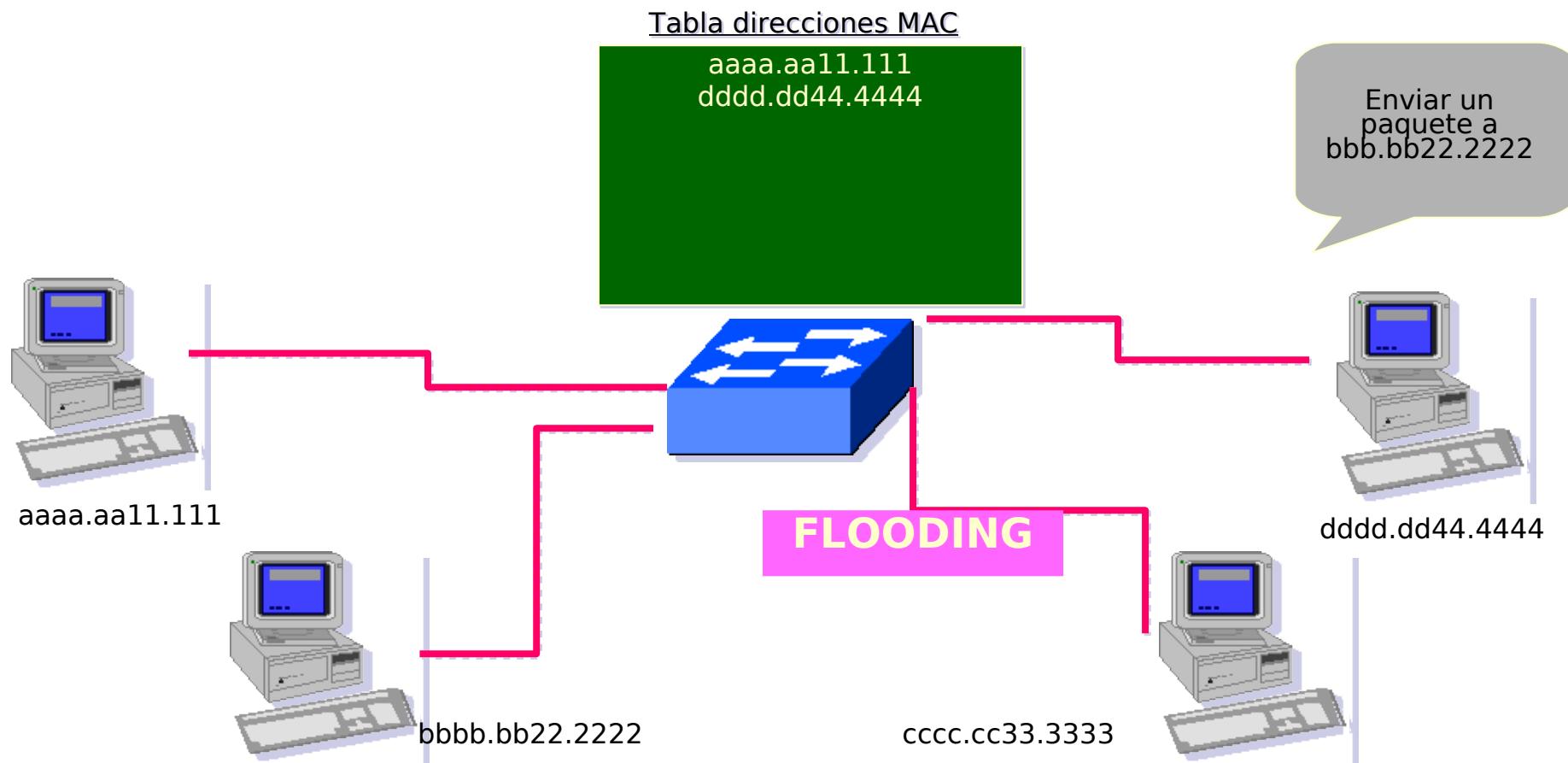
Tabla direcciones MAC, mem CAM

aaaa.aa11.111



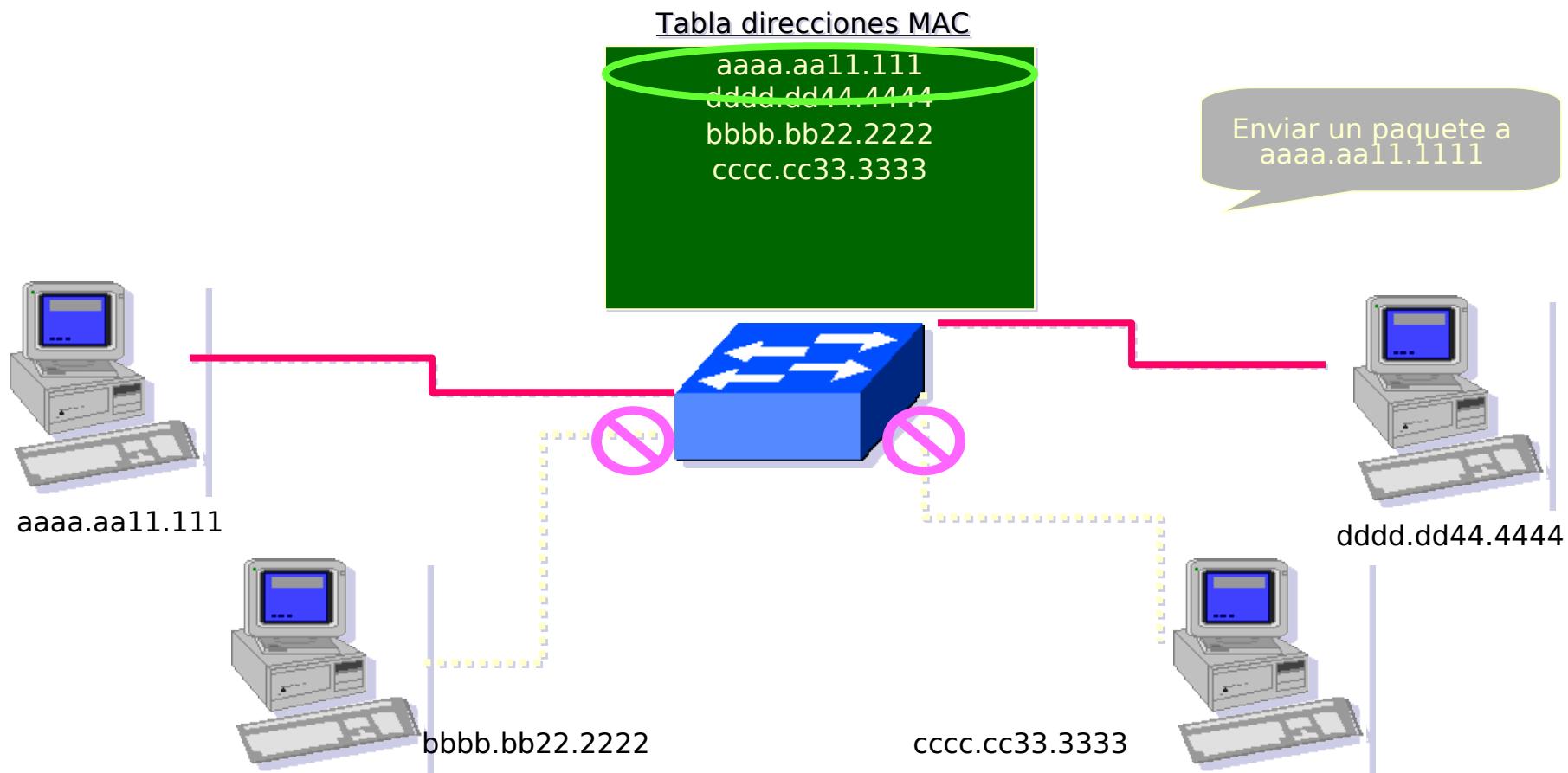
# Switching

## Aprendizaje de direcciones



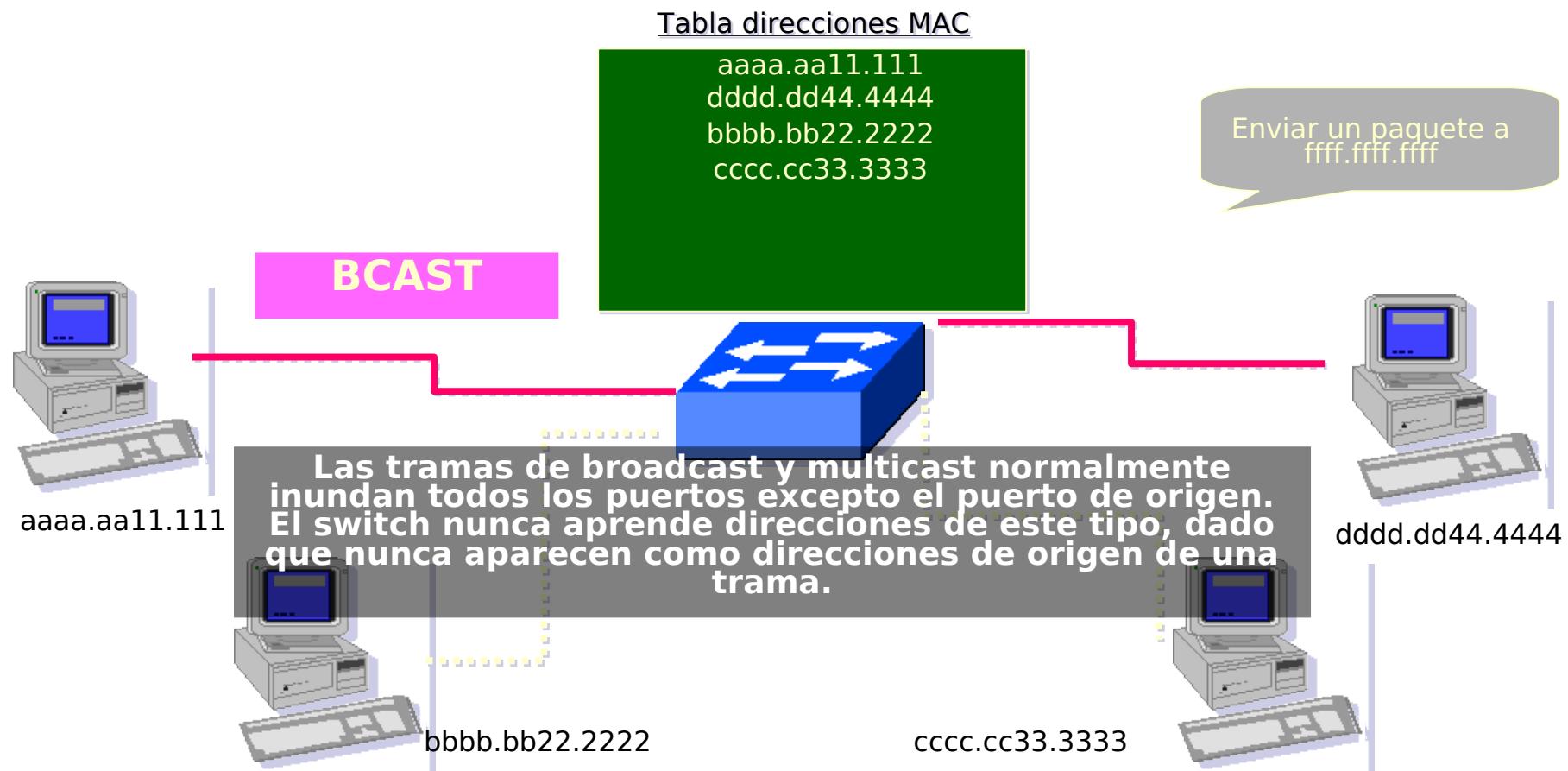
# Switching

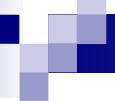
## Reenvío / filtrado de paquetes



# Switching

## Tramas broadcast/multicast





# **Switching**

## Métodos de Conmutación

- **Store and Forward (Almacena y Envía):**

- Lee toda la trama y chequea CRC.
- Mas seguro.

- **Fragment Free (Libre de Fragmetos):**

- Lee los primeros 64 bytes.

- **Cut-through (de corte):**

- Lee hasta la dirección destino.
- Más rápido.



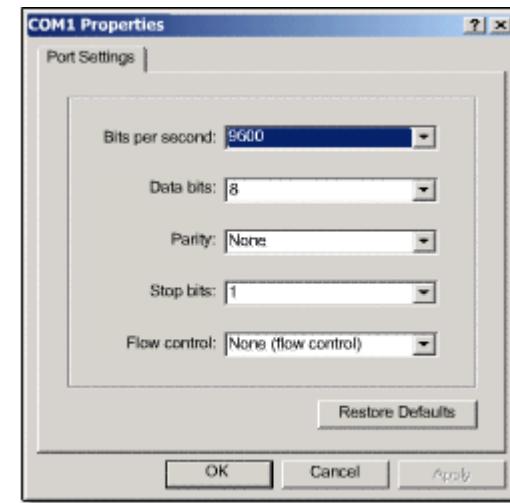
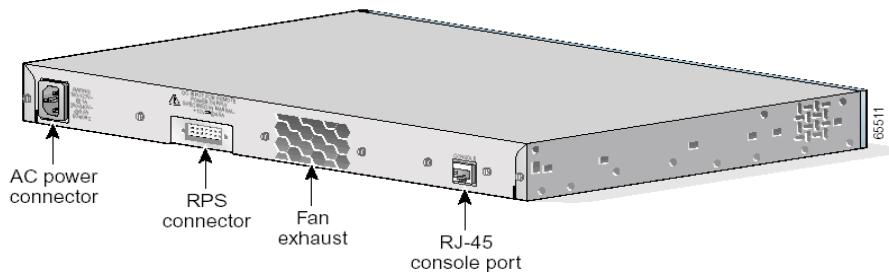
# Características Extras

# Switches Administrables

- **Administracion remota:**
  - **IN-Band.**
  - **OUT-Band.**
- **SNMP.**
- **VLANs.**
- **QoS.**
- **Ruteo L3 (MLS).**
- **Agregación de Enlaces (Ether-channel) 802.3ae.**
- **Flow-Control.**

# Mantenimiento del Switch

- Conectarse al switch por medio de una conexión de consola



- Brindan la interfaz Web , ssh o telnet para administración.
- Además brindan monitoreo por SNMP.

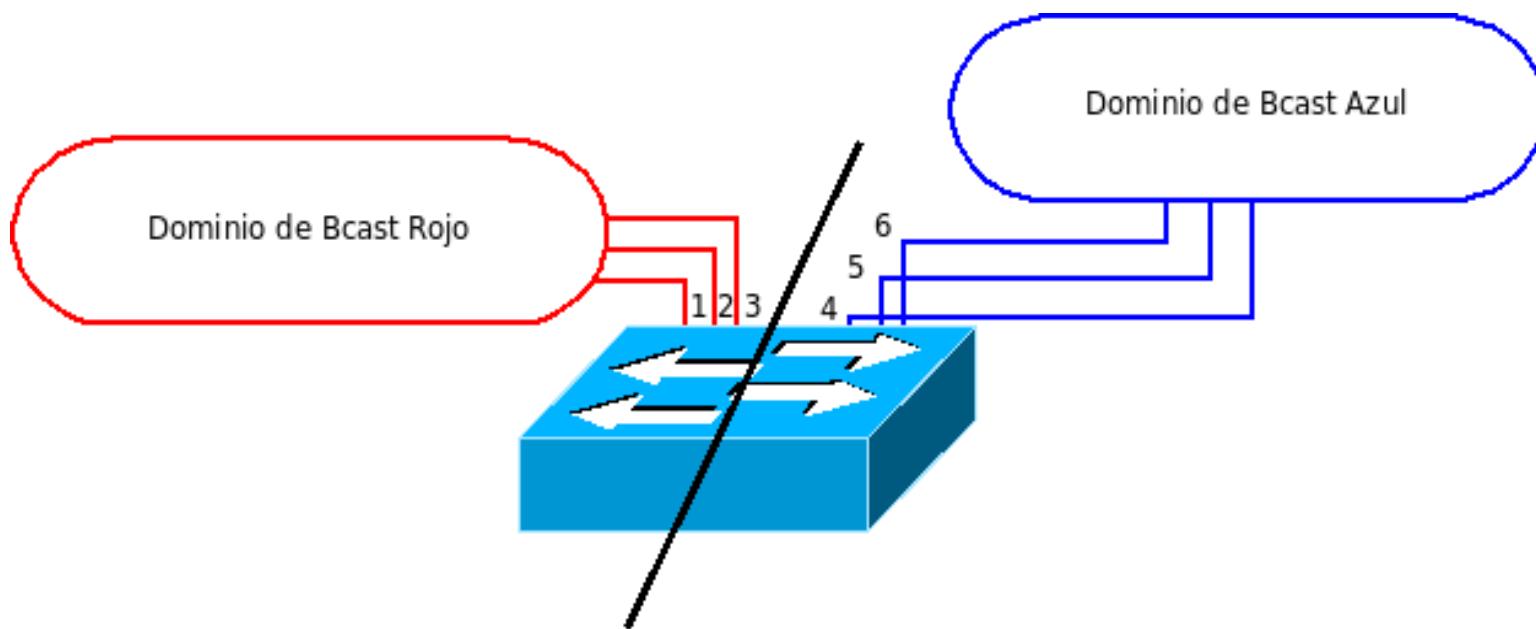
# Mantenimiento del Switch

- Los administradores de red deben documentar y mantener los archivos de configuración operacional de los dispositivos de red.
- Debe realizarse una copia de seguridad del archivo de configuración actual en un servidor o en un disco.
- También debe realizarse una copia de seguridad del firmware/OS en un servidor local. Entonces se puede recargarlo en la memoria flash si es necesario.

# VLANs

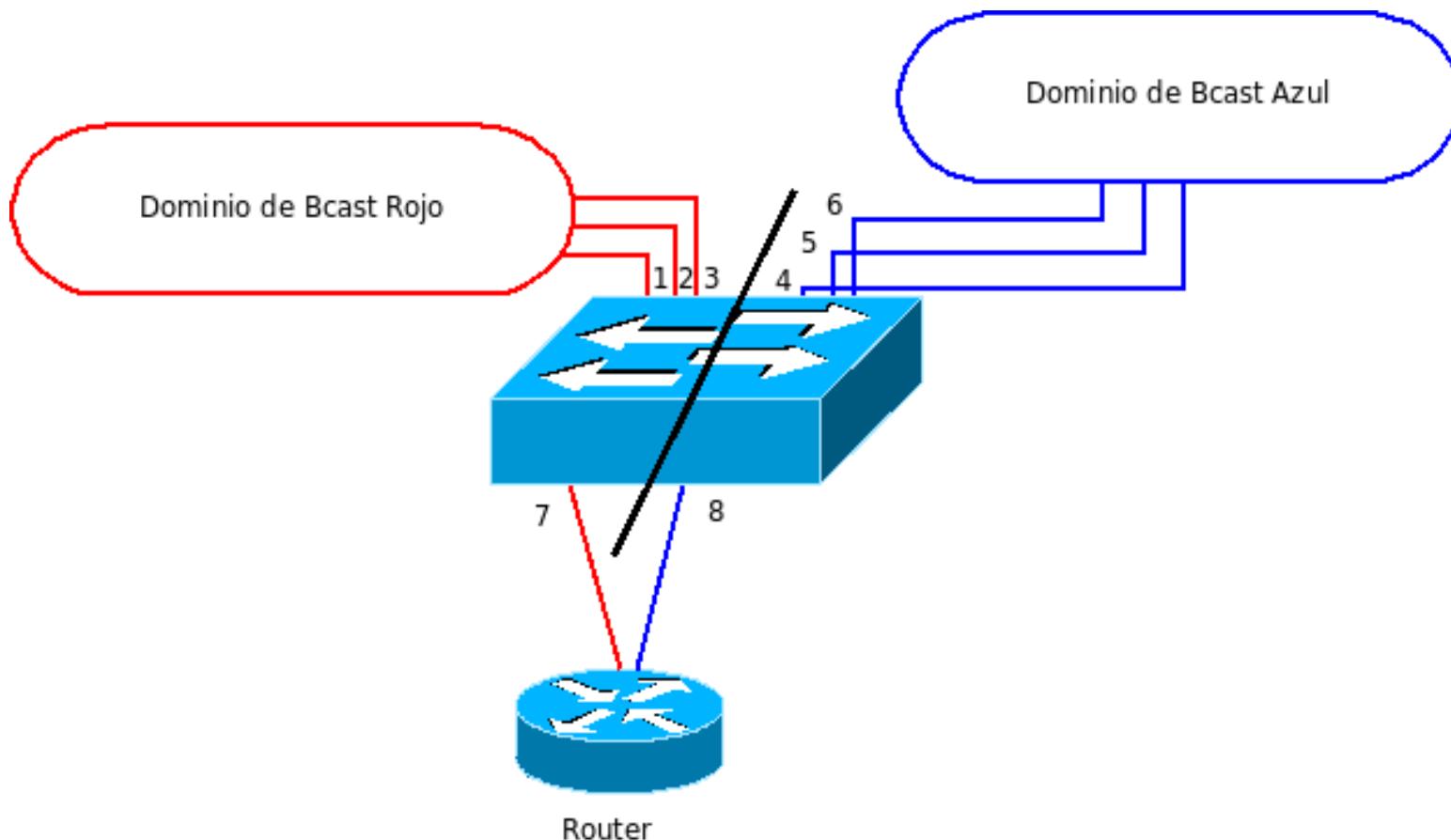
- Dividir un switch en switches virtuales cada uno sobre una VLAN (Virtual LAN).
- Cada VLAN es un dominio de broadcast independiente.
- Para lograr conectividad se deben conectar mediante uplinks o routers.
- Los uplinks compartidos entre VLANs marcan el tráfico con TAGs: 802.1Q tagged ports/trunks:

# VLANs (separadas)



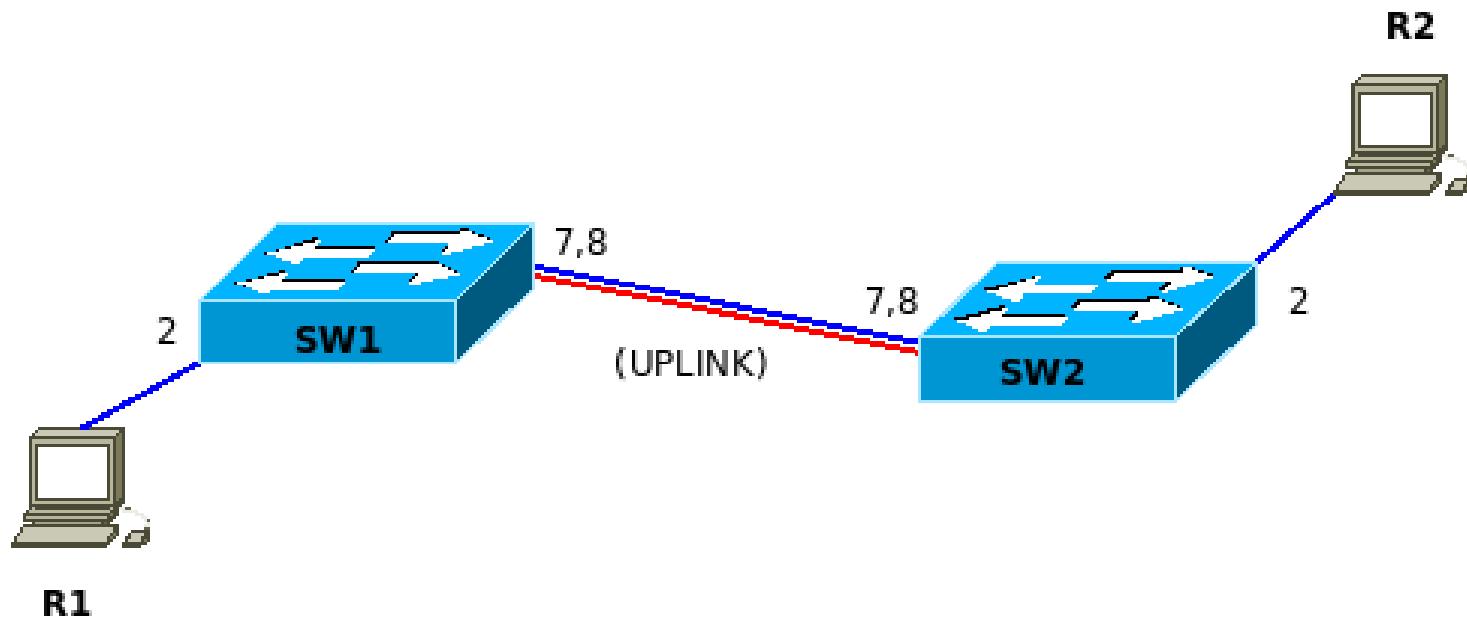
- Divide dominios de broadcast independientes.
- Cada puerto en una VLAN (dominio).

# VLANs (conectadas)



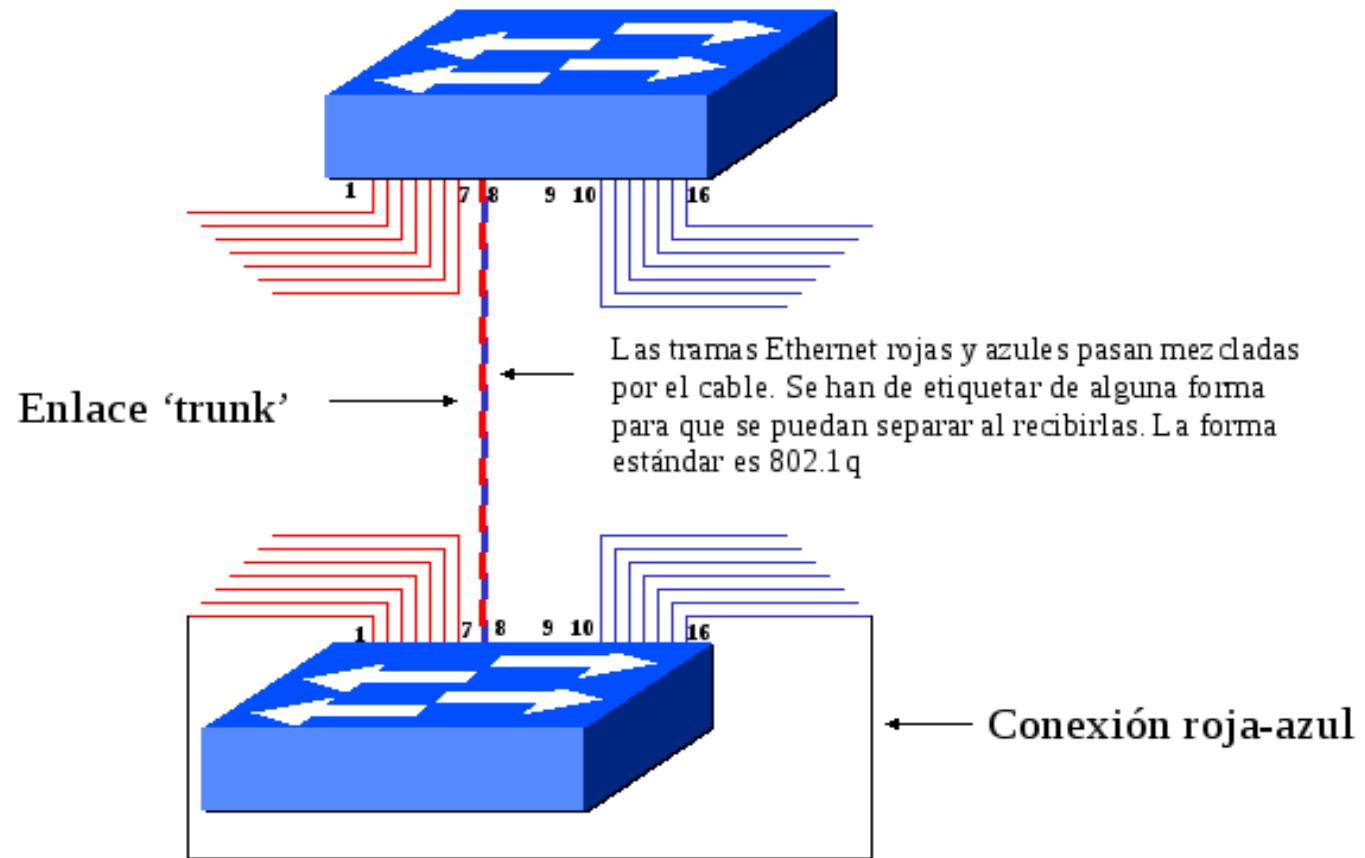
- Interconectar VLANs mediante dispositivo L3 (router).

# VLANs (uplinks)



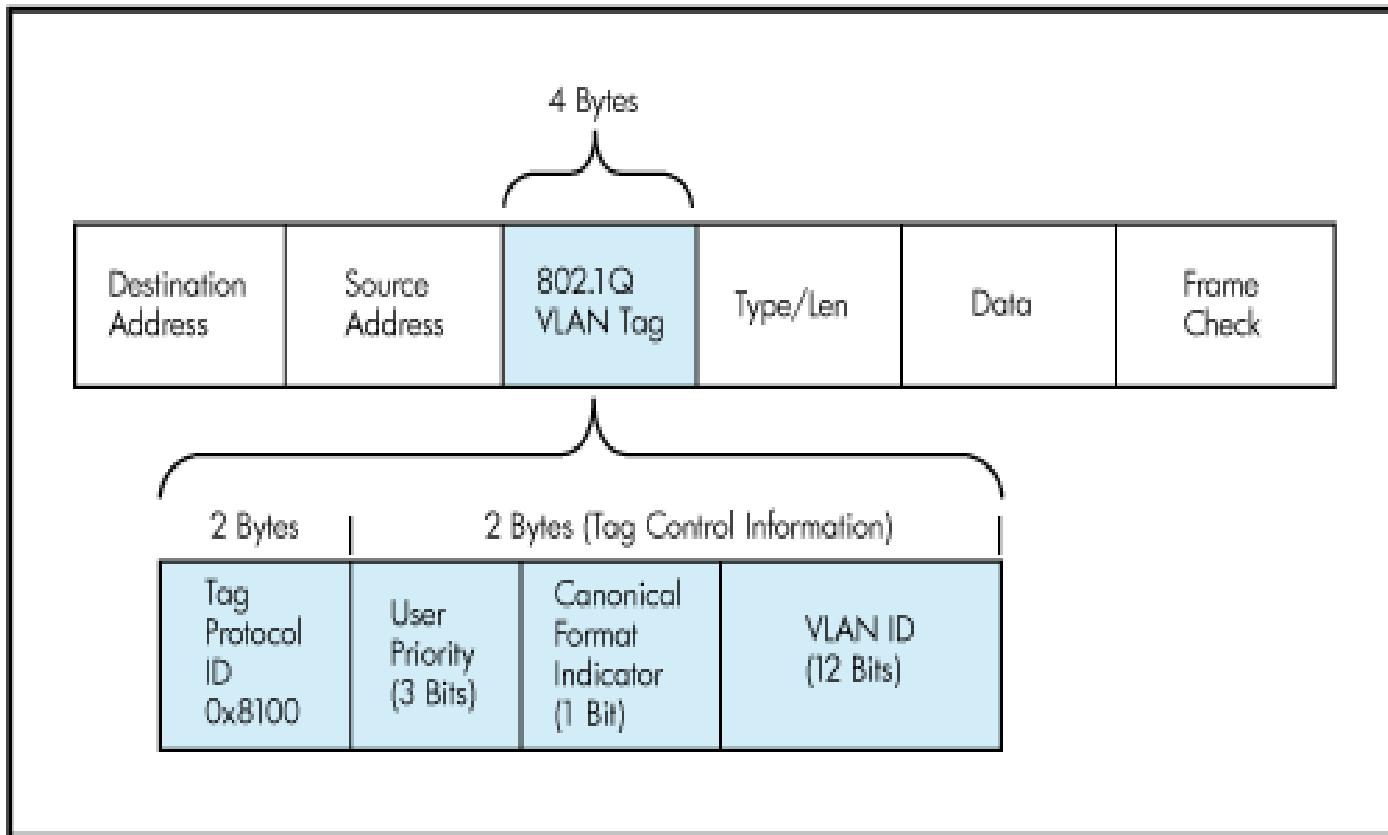
- Hacer uplinks entre VLANs requeriría un enlace por VLAN si cada puerto solo en una VLAN.

# VLANs (tagging)

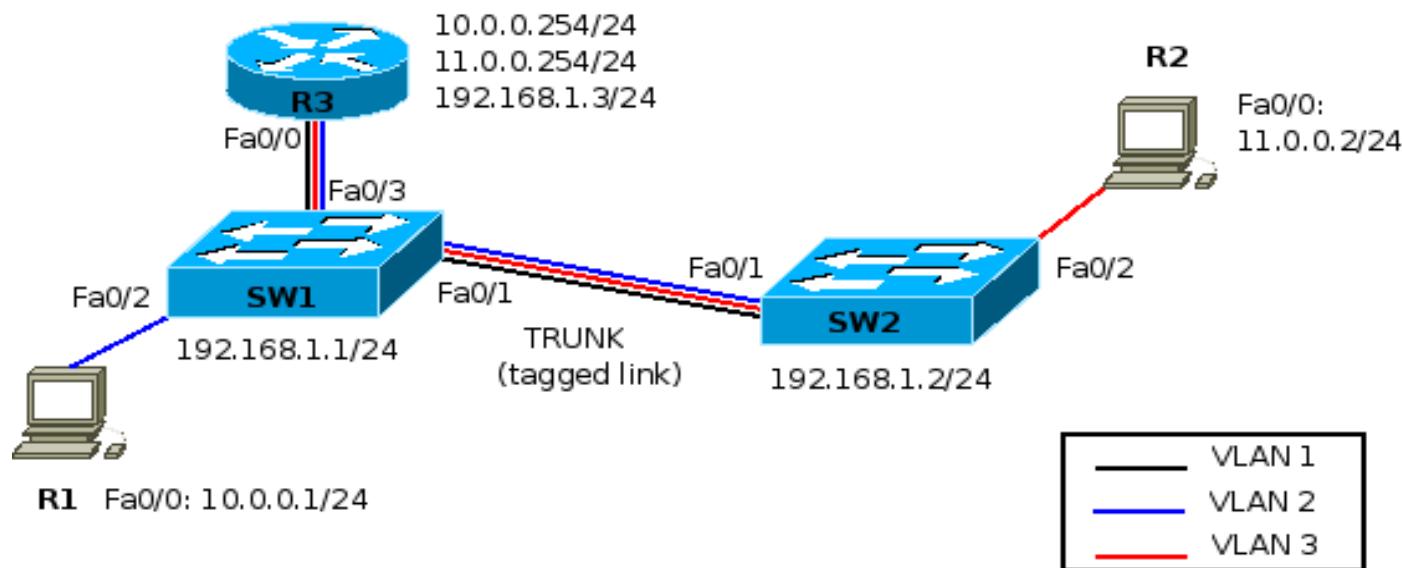
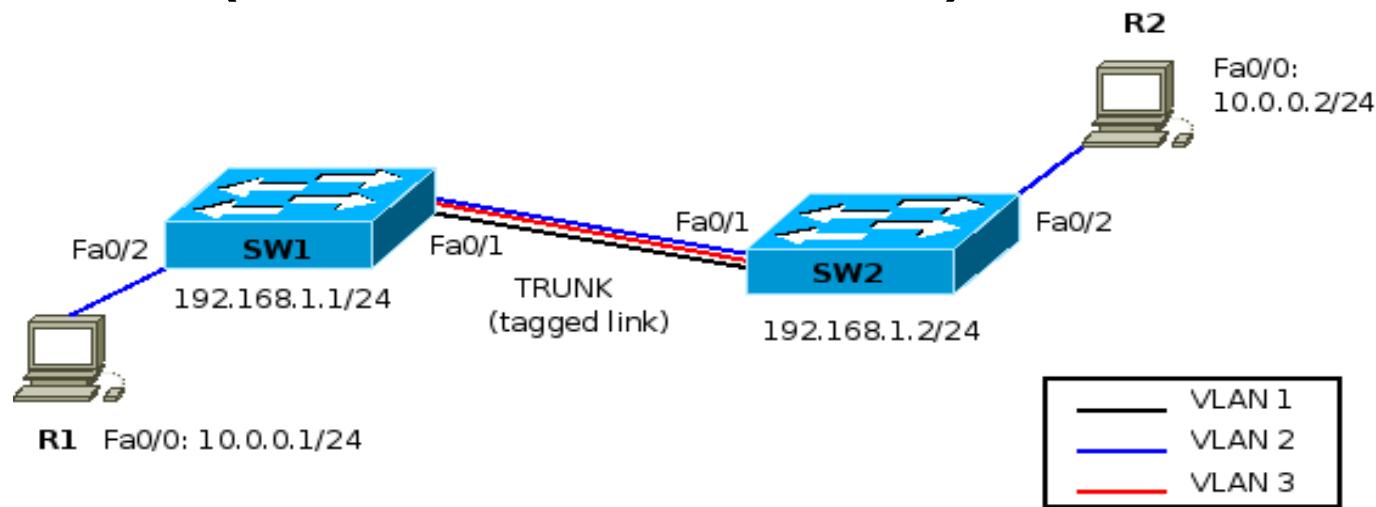


- Puertos especiales de trunking, taggeados.

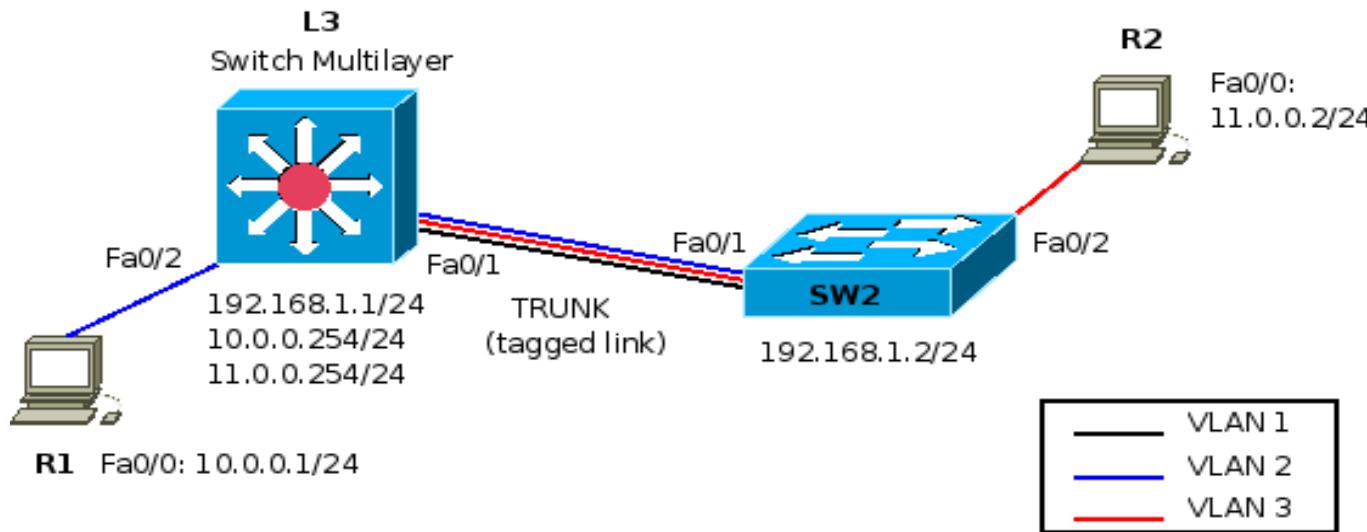
# VLANs (tagging)



# VLANs (route on stick)



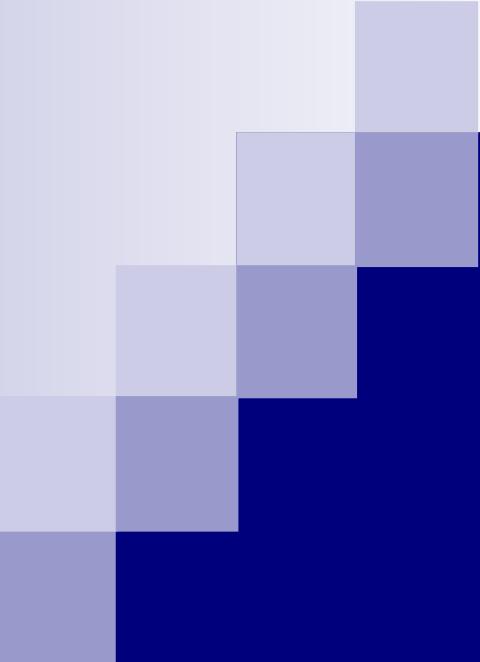
# VLANs (multilayer switch)



- Dispositivos L2/L3, switches-routers integrados.
- Mayor eficiencia.

# Referencias:

- Cisco CCNAv3.1.
- Data & Computer Communications (6th Edition), William Stallings.
- <http://www.ieee802.org/>
- Ethernet: The Definitive Guide. Charles E. Spurgeon. O'Reilly and Associate. Feb 2000. 1st. Edition.



# ARP

(Address Resolution Protocol)

UNLP – Fac. De Informática  
2020

# Contenido

- ARP (Address Resolution Protocol).

Extra:

- InARP (Inverse ARP).
- RARP (Reverse ARP).

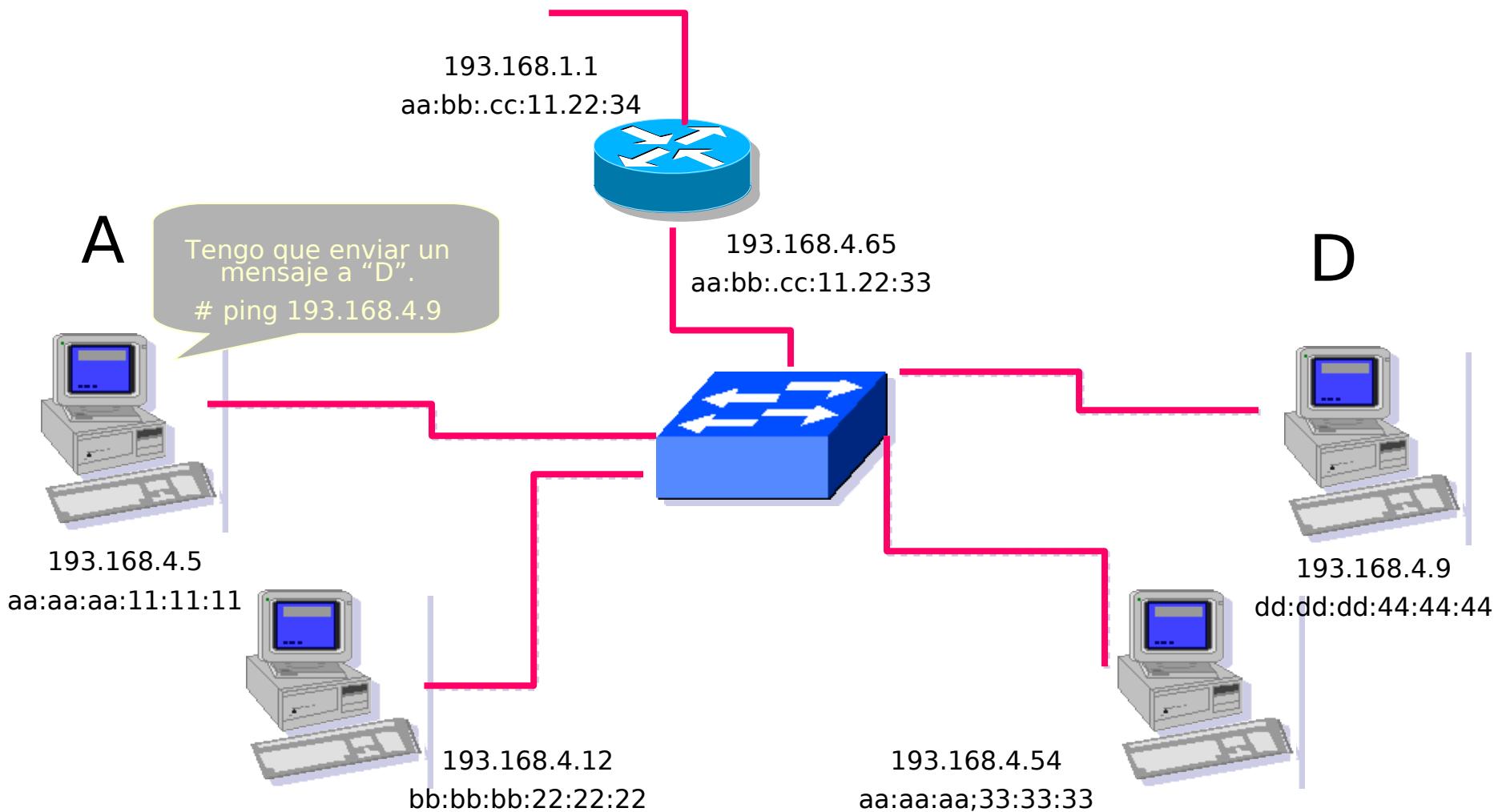
# ARP

(Address Resolution Protocol)

- Protocolo de L2, a veces considerado L3.
- Protocolo “Helper” de IP.
- Mapea Dir. Lógicas (IP) a Dir. Hardware (MAC).
- Trabaja conjuntamente con Ethernet (u otros protocolos de L2 multiacceso con broadcast: Token Ring, FDDI, 802.11).
- Trabaja de forma dinámica, auto-aprendizaje, sin configuración.
- Puede configurarse de forma estática.
- Definido en RFC-826.

# ARP

## Aprendizaje de direcciones

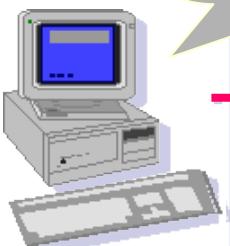


# ARP

## Aprendizaje de direcciones

A

Tengo que enviar un  
mensaje a "D", pero  
NO lo tengo en la  
tabla ARP.  
# ping 193.168.4.9



193.168.4.5

aa:aa:aa:11:11:11

D



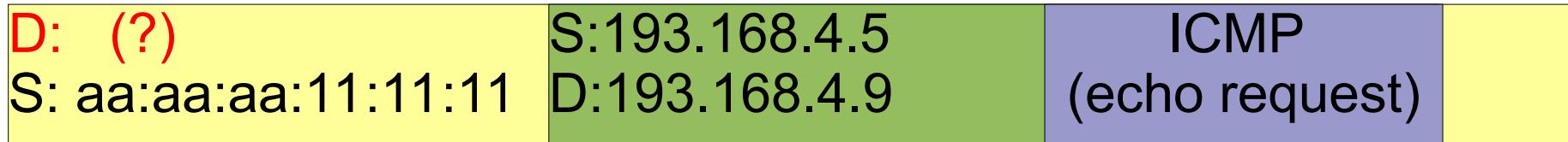
193.168.4.9

dd:dd:dd:44:44:44

andres@h1(paraguil):~\$ arp -a -n  
 (?) 193.168.4.65 at aa:bb:cc:11.22:33 on eth0  
 (?) 193.168.4.62 at <incomplete> on eth0

# ARP

- “A” (h1-paraguil) construye un paquete IP y lo debe encapsular en una trama Ethernet.



- “A” no sabe la Dir. MAC de “D”, debe resolverla para completar la trama.

# ARP

- “A” debe recurrir a un ARP Request:
  - Quién es 192.168.4.9 ? (ARP Request)
  - Como no sabe la MAC debe ser broadcast L2.

D: ff:ff:ff:ff:ff:ff	YO: 193.168.4.5, aa:aa:aa:11:11:11	
S: aa:aa:aa:11:11:11	RQ:193.168.4.9, 00:00:00:00:00:00	

- “D” procesa el requerimiento y responde con ARP Reply:
  - Yo soy dd:dd:dd:44:44:44 de forma unicast.

D: aa:aa:aa:11:11:11	YO: 193.168.4.9, dd:dd:dd:44:44:44	
S: dd:dd:dd:44:44:44	RP: 193.168.4.5, aa:aa:aa:11:11:11	

# Mensaje ARP Request

No..	Time	Source	Destination	Protocol	Info
1	0.000000	RealtekU_12:34:56	Broadcast	ARP	Who has 200.1.1.254? Tell 200.1.1.201
2	0.000943	RealtekU_12:34:57	RealtekU_12:34:56	ARP	200.1.1.254 is at 52:54:00:12:34:57

▶ Frame 1 (42 bytes on wire, 42 bytes captured)  
▶ Ethernet II, Src: RealtekU\_12:34:56 (52:54:00:12:34:56), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
▼ Address Resolution Protocol (request)  
    Hardware type: Ethernet (0x0001)  
    Protocol type: IP (0x0800)  
    Hardware size: 6  
    Protocol size: 4  
    Opcode: request (0x0001)  
    [Is gratuitous: False]  
    Sender MAC address: RealtekU\_12:34:56 (52:54:00:12:34:56)  
    Sender IP address: 200.1.1.201 (200.1.1.201)  
    Target MAC address: 00:00:00\_00:00:00 (00:00:00:00:00:00)  
    Target IP address: 200.1.1.254 (200.1.1.254)

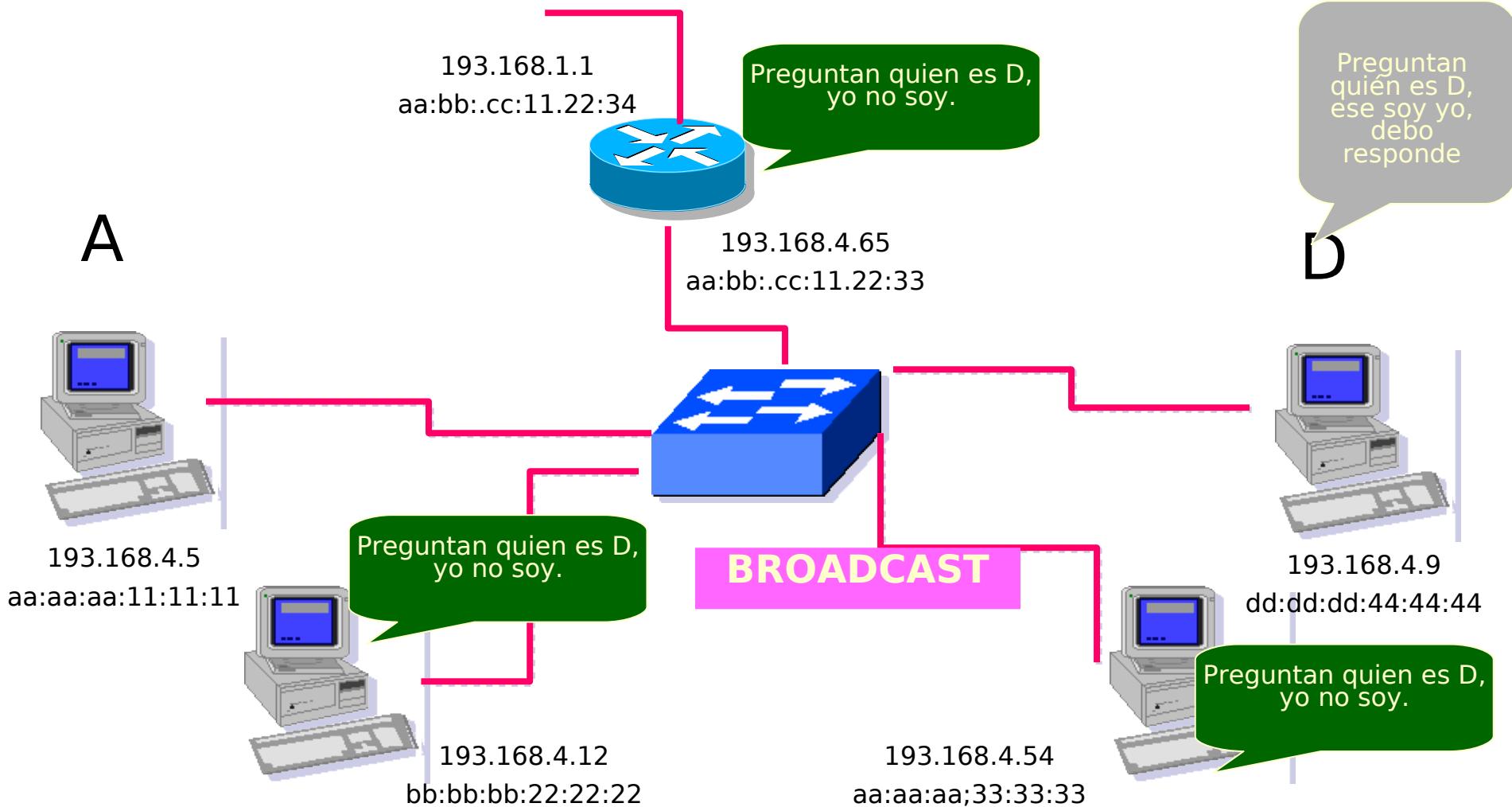
# Mensaje ARP Reply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	RealtekU_12:34:56	Broadcast	ARP	Who has 200.1.1.254? Tell 200.1.1.201
2	0.000943	RealtekU_12:34:57	RealtekU_12:34:56	ARP	200.1.1.254 is at 52:54:00:12:34:57

▶ Frame 2 (60 bytes on wire, 60 bytes captured)  
▶ Ethernet II, Src: RealtekU\_12:34:57 (52:54:00:12:34:57), Dst: RealtekU\_12:34:56 (52:54:00:12:34:56)  
▼ Address Resolution Protocol (reply)  
    Hardware type: Ethernet (0x0001)  
    Protocol type: IP (0x0800)  
    Hardware size: 6  
    Protocol size: 4  
    Opcode: reply (0x0002)  
    [Is gratuitous: False]  
    Sender MAC address: RealtekU\_12:34:57 (52:54:00:12:34:57)  
    Sender IP address: 200.1.1.254 (200.1.1.254)  
    Target MAC address: RealtekU\_12:34:56 (52:54:00:12:34:56)  
    Target IP address: 200.1.1.201 (200.1.1.201)

# ARP

## Aprendizaje de direcciones

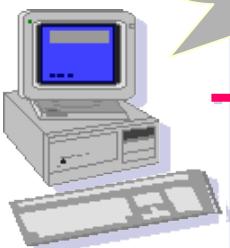


# ARP

## Aprendizaje de direcciones

A

Ahora lo tengo en la  
tabla ARP, lo puedo  
usar.  
# ping 193.168.4.9



193.168.4.5

aa:aa:aa:11:11:11

D



193.168.4.9

dd:dd:dd:44:44:44

```
andres@h1(paraguil):~$ arp -a -n
(?) 193.168.4.65 at aa:bb:cc:11.22:33 on eth0
(?) 193.168.4.62 at <incomplete> on eth0
(?) 193.168.4.9 at dd:dd:dd:44:44:44 on eth0
```

# ARP

- “A” terminar de construir la trama Ethernet que enviará de forma unicast.



- “D” luego la recibirá y lo responderá.

# ARP

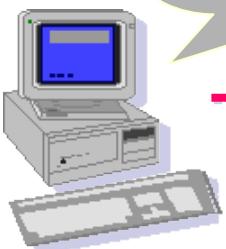
## Agregar direcciones de forma estática permanente

A

Puedo agregar a D  
de forma estática.

#arp -s ...

# ping 193.168.4.9



193.168.4.5  
aa:aa:aa:11:11:11



D



193.168.4.9  
dd:dd:dd:44:44:44

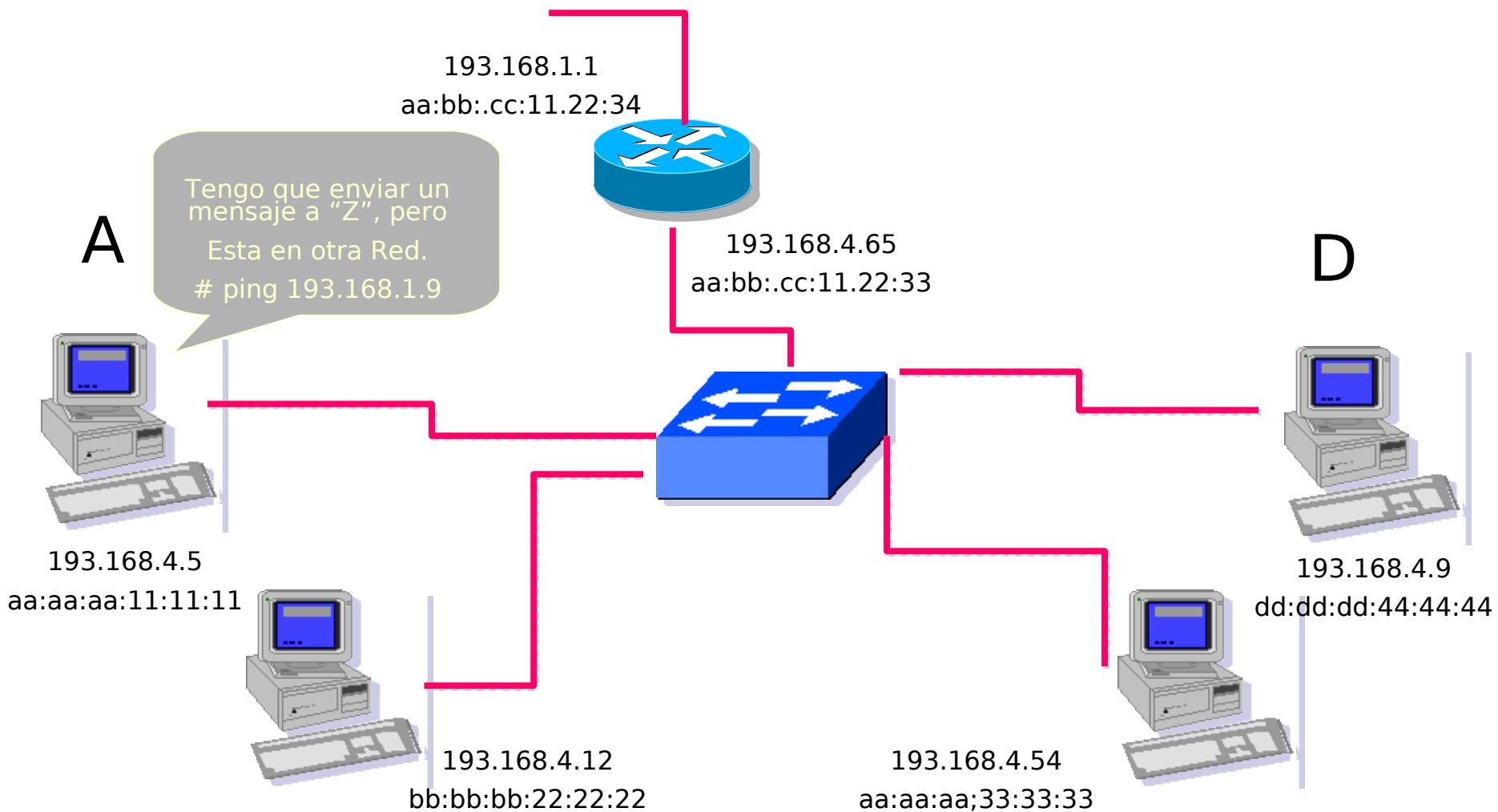
```
andres@h1 (paraguil):~$ arp -a -n  
 (?) 193.168.4.65 at aa:bb:cc:11.22:33 on eth0
```

```
andres@h1 (paraguil):~$ arp -s 193.168.4.9 dd:dd:dd:44:44:44
```

```
andres@h1 (paraguil):~$ arp -a -n  
 (?) 193.168.4.65 at aa:bb:cc:11.22:33 on eth0  
 (?) 193.168.4.9 at dd:dd:dd:44:44:44 PERM on eth0
```

# ARP

## Aprendizaje de direcciones

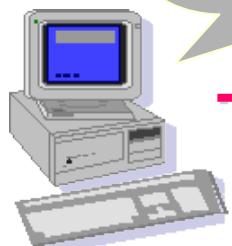


# ARP

Para otras redes utiliza la MAC del default gw

A

Debo consultar el  
default gateway.  
# ping 193.168.1.9



193.168.4.5

aa:aa:aa:11:11:11



R



193.168.4.65

aa:bb:cc:11:22:33

```
andres@h1(paraguil):~$ netstat -nr
```

Destination	Gateway	Genmask	Metric	Iface
193.168.1.5	0.0.0.0	255.255.255.224	0	e0
0.0.0.0	193.168.4.65	0.0.0.0	-	e0

```
andres@h1(paraguil):~$ arp -a -n
```

(?) 193.168.4.65 at aa:bb:cc:11:22:33 on eth0

# RARP

(Reverse Address Resolution Protocol)

- Protocolo de L2, utilizado para mapear direcciones físicas (MAC) a direcciones Lógicas (IP).
- Utilizados en redes multiacceso como Ethernet.
- Utilizado por estaciones sin disco para obtener su dirección IP.
- Hoy es un protocolo en desuso, superado por BOOTP/DHCP.
- RFC-903.

# Referencias:

- Richard Stevens. TCP/IP Illustrated. Vol 1. The Protocols.
- Douglas Comer. Internetworking with TCP/IP. Vol 1.
- Data & Computer Communications (6th Edition), William Stallings.

# IPv6 (Internet Protocol version 6) (parte II)

2020

Facultad de  
Informática



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Contenidos

1

- Protocolos dentro de IPv6
  - ICMPv6
  - Auto-configuración
  - Neighbor Discovery (NDP)
  - PMTU Discovery

2

- Estado de IPv6
  - Transición/Coexistencia
  - Evolución de IPv6

3

- Referencias

# Contenidos

1

- Protocolos dentro de IPv6
  - ICMPv6
  - Auto-configuración
  - Neighbor Discovery (NDP)
  - PMTU Discovery

2

- Estado de IPv6
  - Transición/Coexistencia
  - Evolución de IPv6

3

- Referencias

# Contenidos

1

- Protocolos dentro de IPv6
  - ICMPv6
  - Auto-configuración
  - Neighbor Discovery (NDP)
  - PMTU Discovery

2

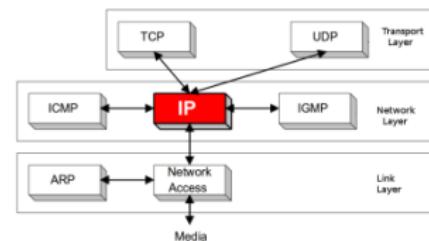
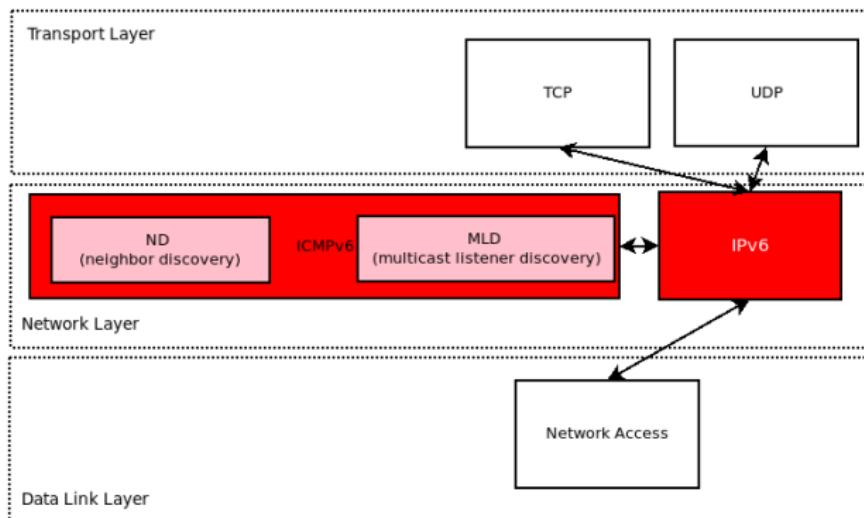
- Estado de IPv6
  - Transición/Coexistencia
  - Evolución de IPv6

3

- Referencias

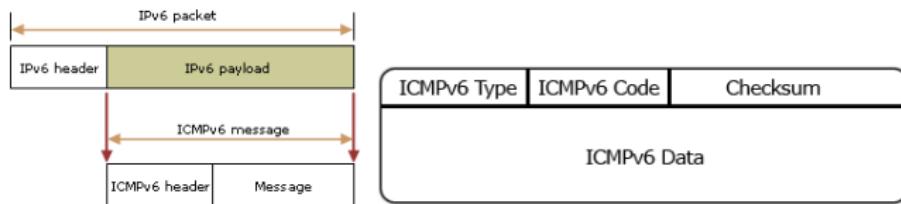
# IPv6 Stack

- Cambia el plano de control según IPv4



# Internet Control Message Protocol v6 (ICMPv6)

- ICMPv6 definido en RFC-4443.
- Parte fundamental del stack IPv6.
- Resuelve:
  - Multicast Listener Discovery (MLD), reemplazo de IGMP.
  - Neighbor Discovery Protocol (NDP), reemplazo de ARP y mensajes Router Discovery(Auto-configuración), Redirect.
  - Mensajes de control de ICMP: informativos (ping), errores.



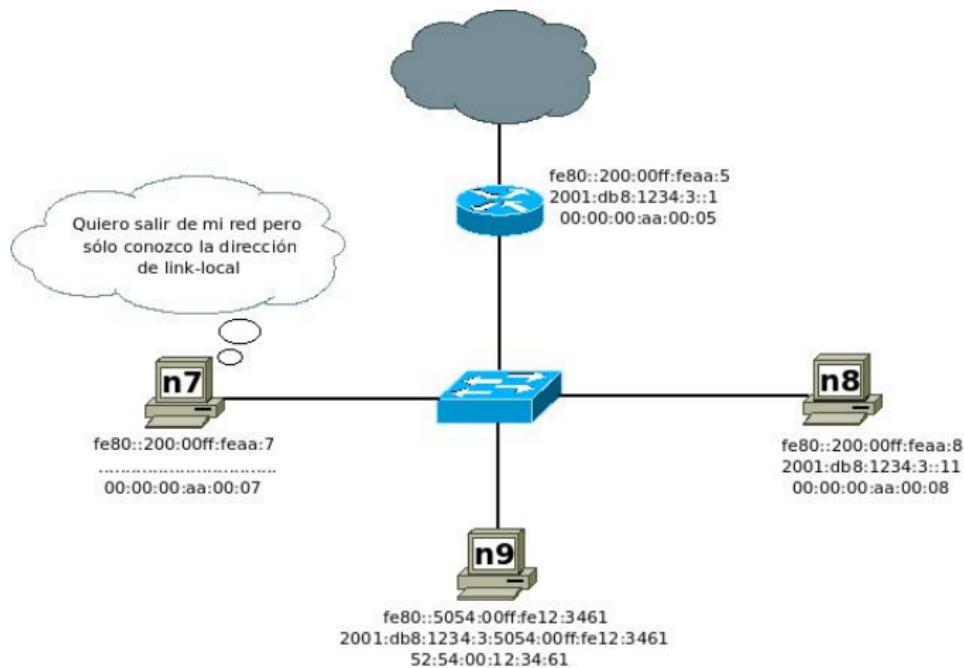
fuente: <http://www.microsoft.com>, <http://www.ipv6.com>

# IPv6 Stateless Autoconfiguration

- Parte del NDP.
- Reemplaza la configuración manual de direcciones IPv6 en IPv4.
- Alternativa básica a DHCPv6, pero sin estados, **SLAAC**.
- El router anuncia uno ó más prefijos de red mediante mensajes Router Advertisement RA (134).
- Se pueden solicitar bajo demanda Router Solicitation RS (133).
- Los hosts auto-configuran su dirección de link-local y solicitan el prefijo a algún router de la red.
- Una vez obtenido se auto-configuran generando su propias direcciones, previo realizar DAD (Duplicate Address Detection).
- Determinan y configuran el default gateway a partir de los Router Advertisement recibidos.
- Router Advertisement puede llevar opciones de configuración del DNS, RFC-6106.

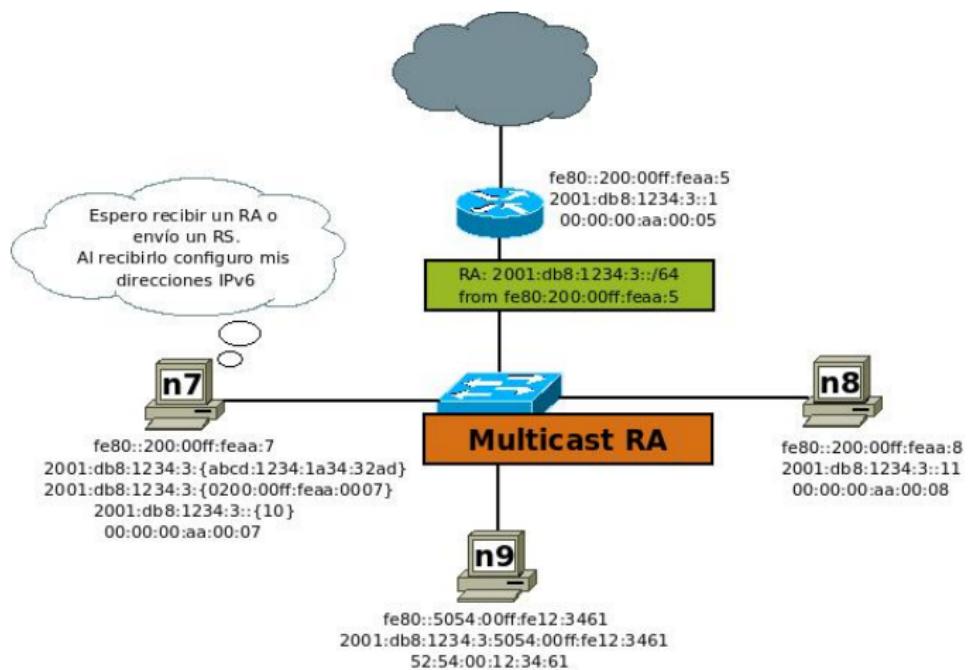
# Router Discovery

- Aprendizaje de su propia configuración



# Router Discovery

- Aprendizaje de su propia configuración



# Mensajes RS y RA

```
root@n7:/# ifconfig eth0 down;sleep 1;ifconfig eth0 up
...
root@n7:/# sleep 4;ping6 2001:db8:1234:1::1
PING 2001:db8:1234:1::1(2001:db8:1234:1::1) 56 data bytes
64 bytes from 2001:db8:1234:1::1: icmp_seq=1 ttl=63 time=0.251 ms
64 bytes from 2001:db8:1234:1::1: icmp_seq=2 ttl=63 time=0.265 ms
...
...
```

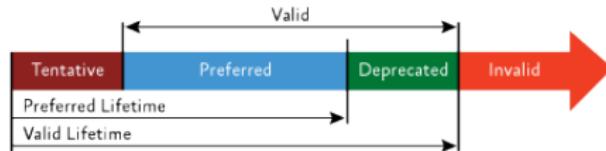
No.	Time	Source	Destination	Length	Info
1	0.000000	::	ff02::1:ffaa:7	78	Neighbor Solicitation for fe80::200:ff:fea:7
2	1.000020	fe80::200:ff:fea:7	ff02::2	70	Router Solicitation from 00:00:00:aa:00:07
3	1.001236	fe80::200:ff:fea:5	ff02::1	110	Router Advertisement from 00:00:00:aa:00:05
4	1.010120	fe80::200:ff:fea:7	ff02::16	110	Multicast Listener Report Message v2
5	1.750071	::	ff02::1:ffaa:7	78	Neighbor Solicitation for 2001:db8:1234:3:200:ff:fea:7
6	1.860154	::	ff02::1:ff88:4bb4	78	Neighbor Solicitation for 2001:db8:1234:3:6846:670b:7588:4bb4
7	3.745713	2001:db8:1234:3:6846:670b:7588	2001:db8:1234:1::1	118	Echo (ping) request id=0x0036, seq=1, hop limit=64 (reply in 8)

► Frame 3: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)  
 ► Ethernet II, Src: 00:00:00:aa:00:05 (00:00:00:aa:00:05), Dst: 33:33:00:00:00:01 (33:33:00:00:00:01)  
 ► Internet Protocol Version 6, Src: fe80::200:ff:fea:5 (fe80::200:ff:fea:5), Dst: ff02::1 (ff02::1)  
 ▾ Internet Control Message Protocol, v6  
 Type: Router Advertisement (134)  
 Code: 0  
 Checksum: 0x2add [correct]  
 Cur hop limit: 64  
 Flags: 0x18  
 Router lifetime (s): 30  
 Reachable time (ms): 0  
 Retrans timer (ms): 0  
 ► ICMPv6 Option (Prefix information : 2001:db8:1234:3::/64)  
 ► ICMPv6 Option (Source link-layer address : 00:00:00:aa:00:05)

captures/ipv6-rs-ra-dad-icmp-i.pcap, rows: 3 captures/ipv6-rs-ra-dns.pcap, row: 3

# Detalles mensaje RA - RFC 4861

- Flags: L(on-link), A(Autonomous), R(Router).
  - L, indica que esta asignado a una interfaz (si no esta no se asume off-link).
  - A, sirve para autoconf. global.
  - R, indica que es router, sirve para NUD (Neighbor Unreach. Detection).
- Otros parámetros: Tiempo de vida válido y preferido (valid and preferred lifetime).



fuente: <http://www.microsoft.com>

# IPv6 Autoconfiguration DHCP6

- Configuración manual es posible (routers, servers).
- Configuración automática, hay variantes: SLAAC, DHCP6.
- Combinaciones:
  - SLAAC solo, hoy puede obtener conf. básica para Internet, Prefijo, Router y DNS (otros MTU).
  - SLAAC solo, algunos equipos no soportan la opción de DNS, requieren DHCP6.
  - SLAAC + DHCP6, conf. básica más parámetros extras por DHCP6.
  - DHCP6 solo, requiere RA para Router/Gateway de la red.

# IPv6 Autoconfiguration DHCP6 (cont.)

- Flags en RA:
  - O bit - Other Configuration Flag, RFC4861, indica que puede usar DHCP6 para obtener otros parámetros, por ejemplo DNS info.
  - M bit - Managed Address Configuration Flag, RFC4861, indica que usa DHCP6.
- Combinaciones:
  - $O = 0, M = 0$ , conf. vía SLAAC, stateless; si hay DHCP6 no loaría.
  - $O = 1, M = 0$ , conf. vía SLAAC, stateless; por DHCP6 obtiene parámetros adicionales. *AdvOtherConfigFlag*.
  - $O = *, M = 1$ , conf. vía DHCP6 stateful; salvo router. *AdvManagedFlag*.

# Datagrama DHCP6 (Wireshark)

No.	Time	Source	Destination	Length	Info
24	47.468477	fe80::200:ff:fea:5	ff02::1	110	Router Advertisement from 00:00:00:aa:00:05
25	52.675902	fe80::200:ff:fea:5	ff02::5	90	Hello Packet
26	52.720484	fe80::200:ff:fea:7	ff02::1:2	118	Solicit XID: 0x28f60b CID: 00010001le44f6e6000000aa0007
27	52.722313	fe80::200:ff:fea:5	ff02::1:ffaa:7	86	Neighbor Solicitation for fe80::200:ff:fea:7 from 00:00:00:00:00:00
28	52.722390	fe80::200:ff:fea:7	fe80::200:ff:fea:5	86	Neighbor Advertisement fe80::200:ff:fea:7 (sol, ovr) is
29	52.722461	fe80::200:ff:fea:5	fe80::200:ff:fea:7	204	Advertise XID: 0x28f60b IAA: 2001:db8:1234:3::1ed6 CID:
30	52.724592	fe80::200:ff:fea:7	ff02::1:2	164	Request XID: 0x1afc22 CID: 00010001le44f6e6000000aa0007
31	52.725194	fe80::200:ff:fea:5	fe80::200:ff:fea:7	204	Reply XID: 0x1afc22 IAA: 2001:db8:1234:3::1ed6 CID: 0001

► Frame 24: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)  
 ► Ethernet II, Src: 00:00:00:aa:00:05 (00:00:00:aa:00:05), Dst: 33:33:00:00:00:01 (33:33:00:00:00:01)  
 ► Internet Protocol Version 6, Src: fe80::200:ff:fea:5 (fe80::200:ff:fea:5), Dst: ff02::1 (ff02::1)  
 ▼ Internet Control Message Protocol v6  
   Type: Router Advertisement (134)  
   Code: 0  
   Checksum: 0x2a9d [correct]  
   Cur hop limit: 64  
 ▼ Flags: 0x58  
   0.... .... = Managed address configuration: Not set  
   .1.... .... = Other configuration: Set  
   ..0.... .... = Home Agent: Not set  
   ...1 1... = Prf (Default Router Preference): Low (3)  
   .... .0.. = Proxy: Not set  
   .....0.. = Reserved: 0  
   Router lifetime (s): 30  
   Reachable time (ms): 0

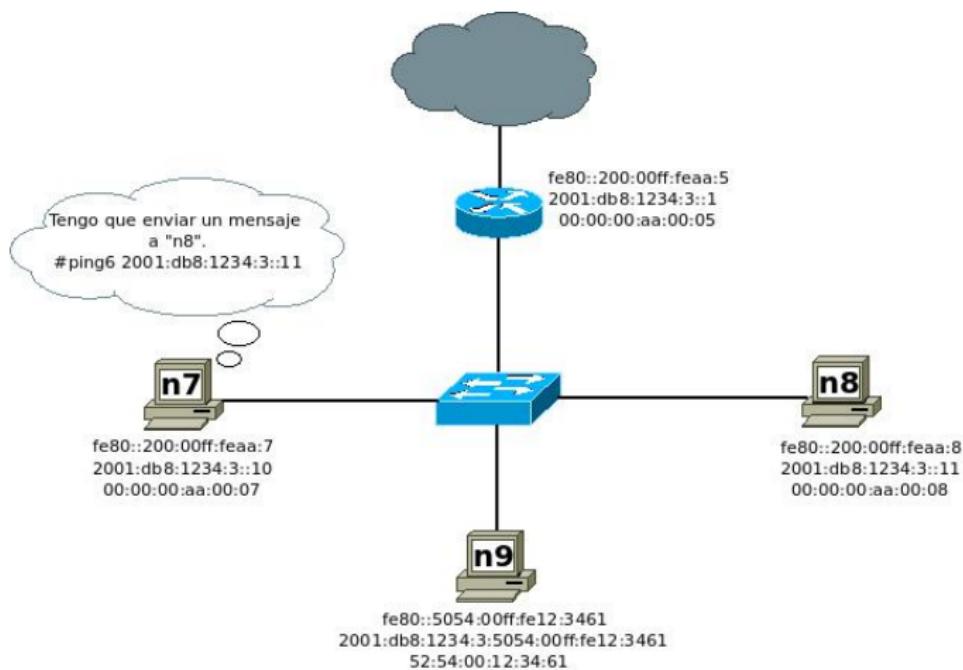
captures/ipv6-radvd-dhcp6.pcap, rows: 24,26,29,...

# Neighbor Discovery

- Reemplaza básicamente al protocolo ARP de IPv4.
- Mapea direcciones lógicas (IPv6) a direcciones de Hardware (MAC, EUI-48, EUI-64).
- Trabaja conjuntamente con Ethernet (u otros protocolos de L2 multiacceso con broadcast: Bluetooth, 802.11, (Token-Ring, FDDI)).
- Trabaja de forma dinámica, auto-aprendizaje, sin configuración.
- Puede configurarse de forma estática.
- Definido en RFC-4861.
- 2 tipos de mensajes: Neighbor Solicitation NS(135) y Neighbor Adv. NA(136).

# Neighbor Discovery (cont.)

- Aprendizaje de direcciones:



# Neighbor Discovery - NS y NA

- “n7” debe recurrir a un Neighbor Solicitation (NS).
- Como no sabe la MAC debe enviar un multicast L2 y L3.

```
root@n7:/# ip -6 addr show dev eth0
27: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc 1000
...
    inet6 2001:db8:1234:3::10/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:7/64 scope link
        valid_lft forever preferred_lft forever

root@n7:/# ip -6 neigh show
fe80::200:ff:feaa:5 dev eth0 lladdr 00:00:00:aa:00:05 router STALE

root@n7:/# ping6 -c 5 2001:db8:1234:3::11 -I 2001:db8:1234:3::10
```

# Neighbor Discovery NS

- Mensajes NS (Neighbor Solicitation):

No.	Time	Source	Destination	Length	Info
1	0.000000	2001:db8:1234:3::10	ff02::1:ff00:11	86	Neighbor Solicitation for 2001:db8:1234:3::11 from 00:00:00:aa:00:07
2	0.000151	2001:db8:1234:3::11	2001:db8:1234:3::10	86	Neighbor Advertisement 2001:db8:1234:3::11 (sol, ovr) is at 00:00:00:aa:00:07
3	0.000161	2001:db8:1234:3::10	2001:db8:1234:3::11	118	Echo (ping) request id=0x004e, seq=1, hop limit=64 (reply in 4)
4	0.000186	2001:db8:1234:3::11	2001:db8:1234:3::10	118	Echo (ping) reply id=0x004e, seq=1, hop limit=64 (request in 3)

► Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)  
 ▷Ethernet II, Src: 00:00:00:aa:00:07 (00:00:00:aa:00:07), Dst: 33:33:ff:00:00:11 (33:33:ff:00:00:11)  
 ▷Internet Protocol Version 6, Src: 2001:db8:1234:3::10 (2001:db8:1234:3::10), Dst: ff02::1:ff00:11 (ff02::1:ff00:11)  
 ► 0110 .... = Version: 6  
 ►.... 0000 0000 .... .... .... = Traffic class: 0x00000000  
 .... .... 0000 0000 0000 0000 = Flowlabel: 0x00000000  
 Payload length: 32  
 Next header: ICMPv6 (58)  
 Hop limit: 255  
 Source: 2001:db8:1234:3::10 (2001:db8:1234:3::10)  
 Destination: ff02::1:ff00:11 (ff02::1:ff00:11)  
 ▷ Internet Control Message Protocol V6  
 Type: Neighbor Solicitation (135)  
 Code: 0  
 Checksum: 0xf8db [correct]  
 Reserved: 00000000  
 Target Address: 2001:db8:1234:3::11 (2001:db8:1234:3::11)  
 ▷ICMPv6 Option (Source link-layer address : 00:00:00:aa:00:07)  
 Type: Source link-layer address (1)  
 Length: 1 (8 bytes)  
 Link-layer address: 00:00:00:aa:00:07 (00:00:00:aa:00:07)

captures/ipv6-ns-na-icmp.pcap, rows: 1

## Neighbor Discovery NS, NA Addresses

```
** Request: Neighbor Solicitation (NS)

From: MAC:<my-MAC-address> , IPv6:<my-Link-Local-Address> / IPv6: <my-global-Address>
To:   MAC:<MAC-multicast-address> , IPv6:<solicited-node_multicast_address>

<solicited-node-multicast-address> (SNMA) : ff02:0000:0000:0000:0000:0001:ff00:0000/104
                                         + last 24bits IPv6
<MAC-multicast-address> (MMA)          : 33:33: + last 32bits IPv6 mcast address

n7: mac address:           00:00:00:aa:00:07
    ipv6 address:          2001:db8:1234:3::10/64
    ipv6 link-local address: fe80::0200:00ff:fe00:0007
"n7" Try to discover:      2001:db8:1234:3::11/64 MAC address ("n8")

SNMA: ff02:0000:0000:0000:0000:0001:ff00:0000/104 + 00:00:11
      = ff02:0000:0000:0000:0000:0001:ff00:0011
      https://tools.ietf.org/html/rfc4291
MMA:  33:33:               + ff:00:00:01 = 33:33:ff:00:00:11
                                         https://tools.ietf.org/html/rfc2464#section-7
                                         https://tools.ietf.org/html/rfc6085
                                         https://tools.ietf.org/html/rfc7042
From: MAC: 00:00:00:aa:00:07 , IPv6: fe80::0200:00ff:fe00:0007 / 2001:db8:1234:3::10
To:   MAC: 33:33:ff:00:00:11   IPv6: ff02:0000:0000:0000:0001:ff00:0011

** Reply: Neighbor Advertisement (NA)

From: MAC:<my-MAC-address> , IPv6:<my-IPv6-Requested-Address>
To:   MAC:<MAC-who-request-address> , IPv6:<IPv6-who-request-address>

From: MAC: 00:00:00:aa:00:08 , IPv6: fe80::0200:00ff:fe00:0008 / 2001:db8:1234:3::11
To:   MAC: 00:00:00:aa:00:07   IPv6: fe80::0200:00ff:fe00:0007 / 2001:db8:1234:3::10
```

# Neighbor Discovery NA

- “n8” procesa el requerimiento y responde con un Neighbor Advertisement (NA):

No.	Time	Source	Destination	Length	Info
1	0.000000	2001:db8:1234:3::10	ff02::1:ff00:11	86	Neighbor Solicitation for 2001:db8:1234:3::11 from 00:00:00:aa:00:07
2	0.000151	2001:db8:1234:3::11	2001:db8:1234:3::10	86	Neighbor Advertisement 2001:db8:1234:3::11 (sol, ovr) is at 00:00:00:aa:00:07
3	0.000161	2001:db8:1234:3::10	2001:db8:1234:3::11	118	Echo (ping) request id=0x004e, seq=1, hop limit=64 (reply in 4)
4	0.000186	2001:db8:1234:3::11	2001:db8:1234:3::10	118	Echo (ping) reply id=0x004e, seq=1, hop limit=64 (request in 3)

► Frame 2: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)  
 ► Ethernet II, Src: 00:00:00:aa:00:08 (00:00:00:aa:00:08), Dst: 00:00:00:aa:00:07 (00:00:00:aa:00:07)  
 ▾ Internet Protocol Version 6, Src: 2001:db8:1234:3::11 (2001:db8:1234:3::11), Dst: 2001:db8:1234:3::10 (2001:db8:1234:3::10)  
 ► 0110 .... = Version: 6  
 ► .... 0000 0000 .... .... .... = Traffic class: 0x00000000  
 .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000  
 Payload length: 32  
 Next header: ICMPv6 (58)  
 Hop limit: 255  
 Source: 2001:db8:1234:3::11 (2001:db8:1234:3::11)  
 Destination: 2001:db8:1234:3::10 (2001:db8:1234:3::10)  
 ▾ Internet Control Message Protocol v6  
 Type: Neighbor Advertisement (136)  
 Code: 0  
 Checksum: 0x54ef [correct]  
 ► Flags: 0x00000000  
 Target Address: 2001:db8:1234:3::11 (2001:db8:1234:3::11)  
 ▾ ICMPv6 Option (Target link-layer address : 00:00:00:aa:00:08)  
 Type: Target link-layer address (2)  
 Length: 1 (8 bytes)  
 Link-layer address: 00:00:00:aa:00:08 (00:00:00:aa:00:08)

captures/ipv6-ns-na-icmp.pcap, rows: 2

# Neighbor Discovery NS, NA y PING-PONG

- “n7” debe recurrir a un Neighbor Solicitation:

```
root@n7:/# ping6 -c 5 2001:db8:1234:3::11 -I 2001:db8:1234:3::10
PING 2001:db8:1234:3::11(2001:db8:1234:3::11) from 2001:db8:1234:3::10 : 56 data bytes
64 bytes from 2001:db8:1234:3::11: icmp_seq=1 ttl=64 time=0.222 ms
64 bytes from 2001:db8:1234:3::11: icmp_seq=2 ttl=64 time=0.273 ms
64 bytes from 2001:db8:1234:3::11: icmp_seq=3 ttl=64 time=0.344 ms
64 bytes from 2001:db8:1234:3::11: icmp_seq=4 ttl=64 time=0.219 ms
64 bytes from 2001:db8:1234:3::11: icmp_seq=5 ttl=64 time=0.181 ms
--- 2001:db8:1234:3::11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.181/0.247/0.344/0.059 ms
```

```
root@n7:/# ip -6 neigh show
2001:db8:1234:3::11 dev eth0 lladdr 00:00:00:aa:00:08 REACHABLE
fe80::200:ff:fea:8 dev eth0 lladdr 00:00:00:aa:00:08 REACHABLE
fe80::200:ff:fea:5 dev eth0 lladdr 00:00:00:aa:00:05 router STALE
```

# Neighbor Discovery Flow

Time	Source	Destination	Comment
0.000000	01:db8:1234:3::10	2001:db8:1234:3::11	ICMPv6: Neighbor Solicitation for 2001:db8:1234:3::11 from 00:00:00:aa:00:07
0.000151	ff02::1:ff00:11	ff80::200:ff:fea:8	ICMPv6: Neighbor Advertisement 2001:db8:1234:3::11 (sol, ovr) is at 00:00:00:aa:00:08
0.000161			ICMPv6: Echo (ping) request id=0x004e, seq=1, hop limit=64 (request in 4)
0.000186			ICMPv6: Echo (ping) reply id=0x004e, seq=1, hop limit=64 (request in 3)
1.001056			ICMPv6: Echo (ping) request id=0x004e, seq=2, hop limit=64 (reply in 6)
1.001276			ICMPv6: Echo (ping) reply id=0x004e, seq=2, hop limit=64 (request in 5)
2.000626			ICMPv6: Echo (ping) request id=0x004e, seq=3, hop limit=64 (reply in 8)
2.000883			ICMPv6: Echo (ping) reply id=0x004e, seq=3, hop limit=64 (request in 7)
3.001138			ICMPv6: Echo (ping) request id=0x004e, seq=4, hop limit=64 (reply in 10)
3.001305			ICMPv6: Echo (ping) reply id=0x004e, seq=4, hop limit=64 (request in 9)
4.000855			ICMPv6: Echo (ping) request id=0x004e, seq=5, hop limit=64 (reply in 12)
4.000982			ICMPv6: Echo (ping) reply id=0x004e, seq=5, hop limit=64 (request in 11)
5.004705			ICMPv6: Neighbor Solicitation for 2001:db8:1234:3::10 from 00:00:00:aa:00:08
5.004917			ICMPv6: Neighbor Advertisement 2001:db8:1234:3::10 (sol)

captures/ipv6-ns-na-icmp.pcap

# Neighbor Discovery NA (cont.)

- Mensajes NA (Neighbor Advertisement):
- Solicitado o no solicitado.
- Solicitado, respuesta a RS Flag: (S)Solicited=1.
- NO-solicitado, para actualizar caches Flag: (O)Override=1.
- Flag (R)Router=1, lo envía el router.

# Neighbor Discovery NS (Router)

- “n7” debe recurrir a un Neighbor Solicitation, si no tiene router MAC/stale:

```
root@n7:/# ip -6 neigh show
fe80::200:ff:fea:5 dev eth0 lladdr 00:00:00:aa:00:05 router STALE

root@n7:/# ping6 -c 5 2001:db8:1234:3::1 -I 2001:db8:1234:3::10
PING 2001:db8:1234:3::1(2001:db8:1234:3::1) from 2001:db8:1234:3::10 : 56 data bytes
64 bytes from 2001:db8:1234:3::1: icmp_seq=1 ttl=64 time=0.906 ms
64 bytes from 2001:db8:1234:3::1: icmp_seq=2 ttl=64 time=0.303 ms

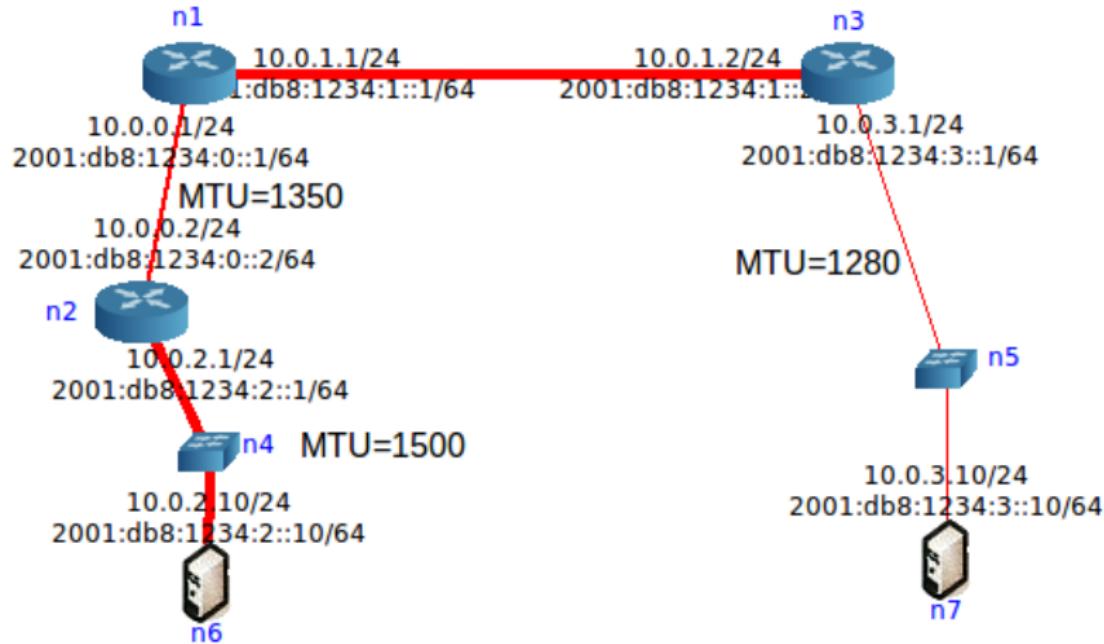
root@n7:/# ip -6 neigh show
2001:db8:1234:3::1 dev eth0 lladdr 00:00:00:aa:00:05 router REACHABLE
fe80::200:ff:fea:5 dev eth0 lladdr 00:00:00:aa:00:05 router STALE

captures/ipv6-ns-na-router-icmp.pcap, row:4
```

# PMTU Discovery

- MTU (Maximum Transmission Unit) depende de L2.
- Cada Link L2 puede tener su MTU Link-MTU.
- A lo largo de un camino se puede establecer Path MTU (PMTU).
- Para IPv6 min MTU=1280B (RFC-2460, RFC-8200), Para IPv4 68B(RFC-791).
- Recomendado, (estandarizado) 1500B.
- Las implementaciones hacen el PMTU discovery (RFC-1981).
- Se utiliza ICMPv6 Error: “Message Too Big”.
- Una vez determinado el PMTU se fragmenta de extremo a extremo.
- Se puede saltar y usar directamente 1280, no es óptimo.
- TCP trata de usar MSS (Maximum Segment Size).

# PMTU Discovery (cont.)



# PMTU Discovery (cont.)

Time	Source IP	Destination IP	Comment
0.000000	01:db8:1234:2::10	2001:db8:1234:2::1	ICMPv6: Echo (ping) request id=0x0032, seq=1, hop limit=64 (no response found!)
0.000539	(0)	(0)	ICMPv6: Packet Too Big
1.999869	(0)	(0)	IPv6 fragment (nxt=ICMPv6 (58) off=0 id=0xbff2f2768)
2.000198	(0)	(0)	ICMPv6: Packet Too Big
2.000231	(0)	(0)	ICMPv6: Echo (ping) request id=0x0032, seq=2, hop limit=64 (no response found!)
4.002568	(0)	(0)	IPv6: IPv6 Fragment (nxt=ICMPv6 (58) off=0 id=0xbff2f2769)
4.002912	(0)	(0)	ICMPv6: Echo (ping) request id=0x0032, seq=3, hop limit=64 (reply in 9)
4.003248	(0)	(0)	IPv6: IPv6 Fragment (nxt=ICMPv6 (58) off=0 id=0x55c4a2e5)
4.003259	(0)	(0)	ICMPv6: Echo (ping) reply id=0x0032, seq=3, hop limit=61 (request in 7)
6.005221	(0)	(0)	IPv6: IPv6 fragment (nxt=ICMPv6 (58) off=0 id=0xbff2f276a)
6.005583	(0)	(0)	ICMPv6: Echo (ping) request id=0x0032, seq=4, hop limit=64 (reply in 13)
6.005893	(0)	(0)	IPv6: IPv6 Fragment (nxt=ICMPv6 (58) off=0 id=0x55c4a2e6)
6.005904	(0)	(0)	ICMPv6: Echo (ping) reply id=0x0032, seq=4, hop limit=61 (request in 11)
8.007685	(0)	(0)	IPv6: IPv6 fragment (nxt=ICMPv6 (58) off=0 id=0xbff2f276b)
8.007895	(0)	(0)	ICMPv6: Echo (ping) request id=0x0032, seq=5, hop limit=64 (reply in 17)
8.008197	(0)	(0)	IPv6: IPv6 fragment (nxt=ICMPv6 (58) off=0 id=0x55c4a2e7)
8.008208	(0)	(0)	ICMPv6: Echo (ping) reply id=0x0032, seq=5, hop limit=61 (request in 15)

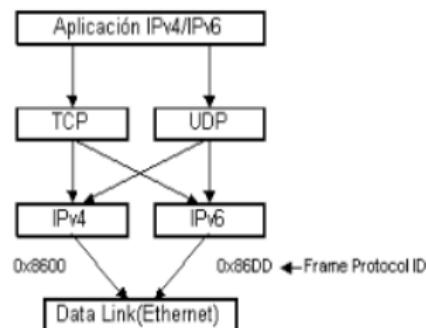
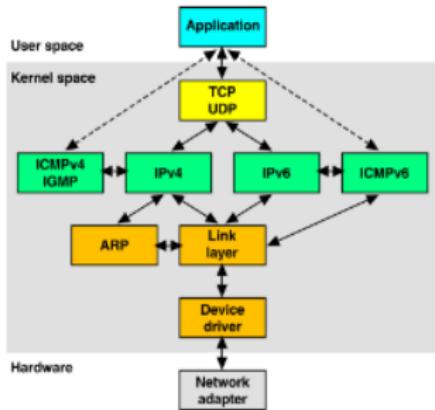
captures/ipv6-frag-doble-nocont-i.pcap

# Transición/Coexistencia de IPv4 a IPv6

- No va a existir un día “D” para cambiar de IPv4 a IPv6.
- Amplio abanico de técnicas disponibles:
  - Doble pila (Dual-Stack).
  - Técnicas de tunneling.
  - Técnicas de traducción IPv6  $\longleftrightarrow$  IPv4.
  - Técnicas combinadas.

# Dual Stack IPv4/IPv6

- No se elimina la pila IPv4.
- Técnica multi-protocolo como con: NetBIOS, Appletalk, IPX.
- Actualmente, la mayoría de los OSs soportan IPv6.
- La aplicación, biblioteca de código elige cual usar (registros de DNS AAAA y A).
- RFC-6555, “Happy Eyeballs: Success with Dual-Stack Hosts”.
- Van a co-existir por “mucho” tiempo.

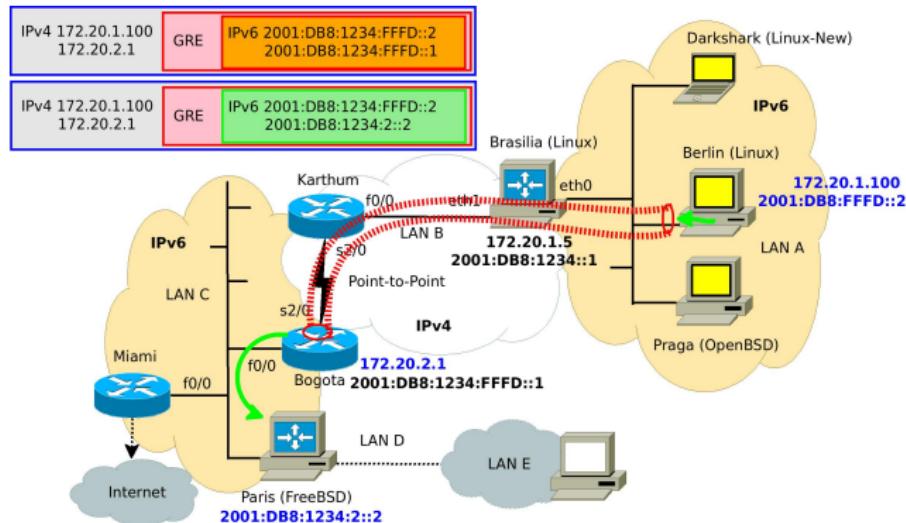


# Túneles IPv4/IPv6

- Encapsular IPv6 en IPv4 donde no hay cobertura
- Se requiere doble pila IPv4/IPv6 en los routers
- Pueden ser:
- **Manuales:**
  - GRE (point-to-point).
  - SIT (Simple Internet Tunnel) IP-IP - IPv6-IPv4 (point-to-point).
  - 6in4.
- **Tunnels Brokers, interfaces web de ayuda:**
  - 6in4+TB (<https://tunnelbroker.net/>)
- **Automáticos:**
  - 6to4 (No funciona sobre NAT)
  - ISATAP (no funciona sobre NAT)
  - Teredo (sobre NAT)

# Túneles IPv4/IPv6

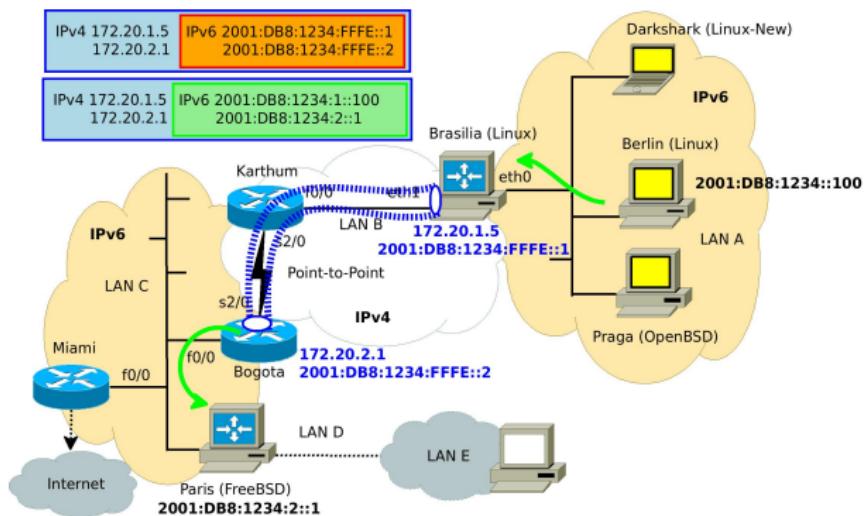
- Manuales (en caso intranet con GRE):
  - Encapsula en GRE (47) General Routing Encapsulation.
  - Punto a punto entre host y router o entre routers.



captures/113-ipv6-tun-gre-Berlin-e0.pcap. row:1

# Túneles IPv4/IPv6 (cont.)

- Manuales (en caso intranet con SIT):
  - Encapsula directamente en IPv4, proto 41(IPv6).
  - Punto a punto entre host y router o entre routers.



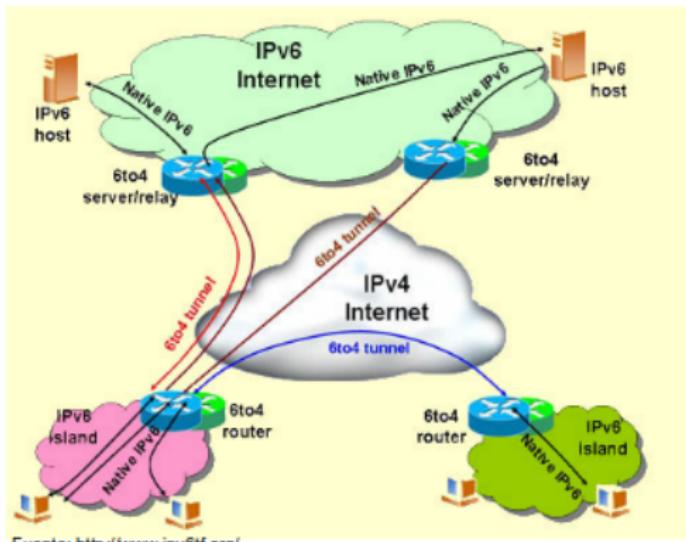
captures/110-ipv6-tun-sit.pcap, row:4 captures/111-ipv6-tun-sit-Bogota-S00.pcap, row:8

# Túneles IPv4/IPv6 (cont.)

- sit/6in4 con TB (Tunnel Broker):
  - Encapsula en IPv4 como SIT.
  - Se consideran Punto a Punto.
  - Direcciones de ambos extremos en el mismo bloque, por ejemplo /126.
  - Para funcionar con NAT requiere que no se filtre proto 41 (IPv6 en IPv4).
  - Agregan un site web que determina la conf. (TB).
  - El usuario entra al site y obtiene la conf.

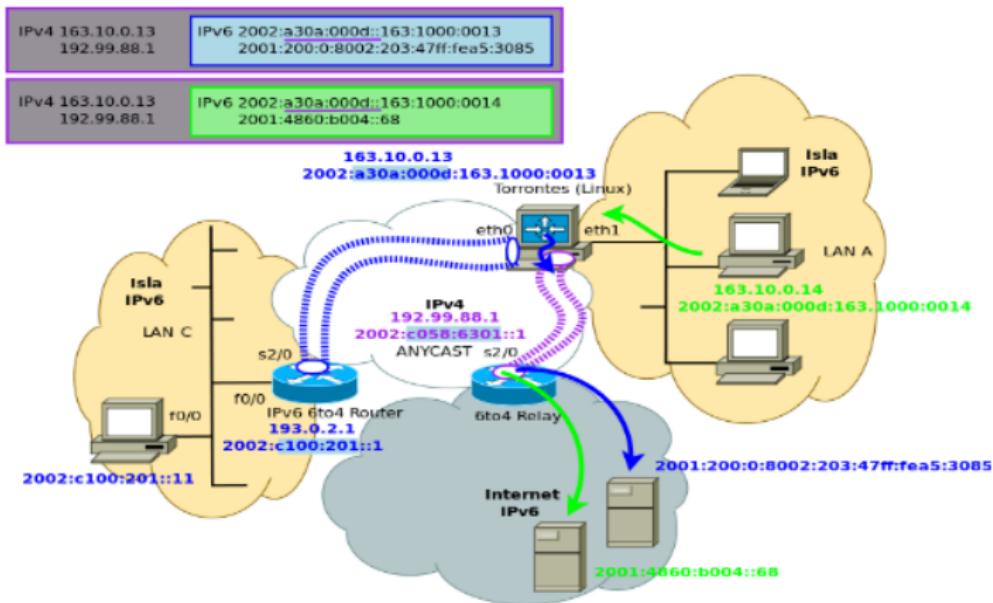
# Túneles IPv4/IPv6 (cont.)

- 6to4: sobre direcciones IPv4 globales:
  - Se usa el bloque: 2002::/16
  - Paquetes de salida siempre al 6to4 relay.
  - Paquetes de vuelta, pueden usar otro origen.
  - Prefijo 6to4 relays anycast: 192.88.99.1/24



# Túneles IPv4/IPv6 (cont.)

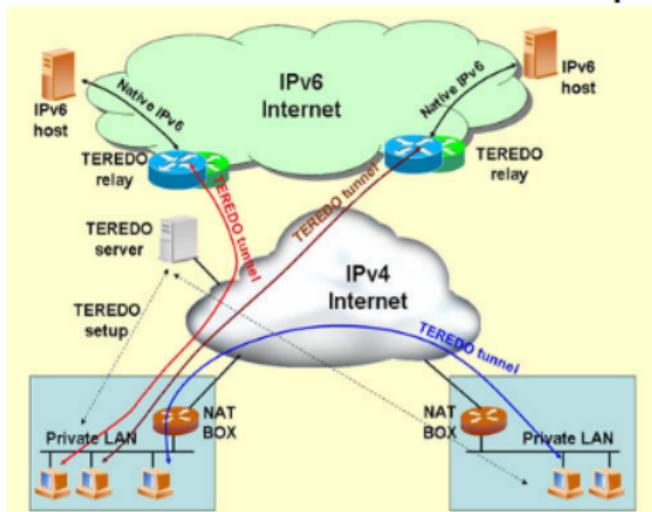
- 6to4: en Intranet



# Túneles IPv4/IPv6 (cont.)

- Teredo:

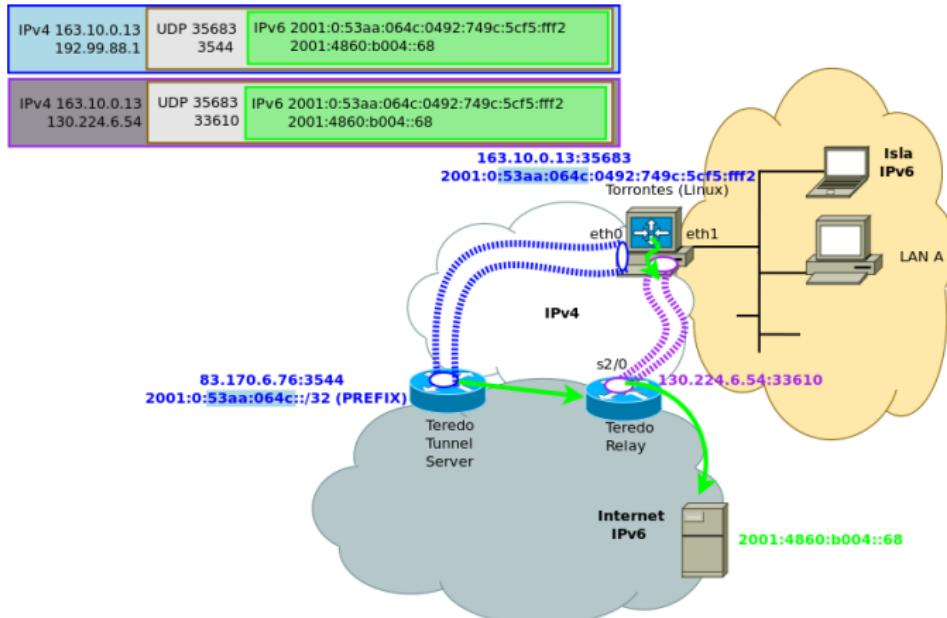
- funciona sobre direcciones IPv4 privadas, con NAT y sin Proto 41.
- Encapsula el datagramas UDP.
- Se configura el cliente contra un Server Teredo.
- El Server Teredo proporciona acceso a los Teredo Relay.



Fuente: <http://www.ipv6tf.org/>

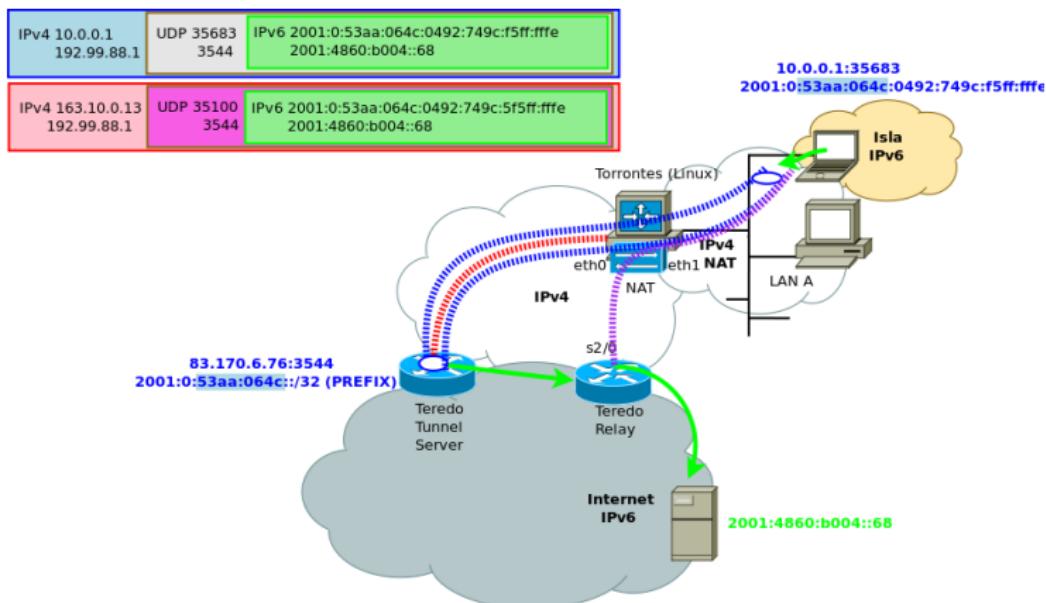
# Túneles IPv4/IPv6 (cont.)

- Teredo: sobre direcciones IPv4 públicas.



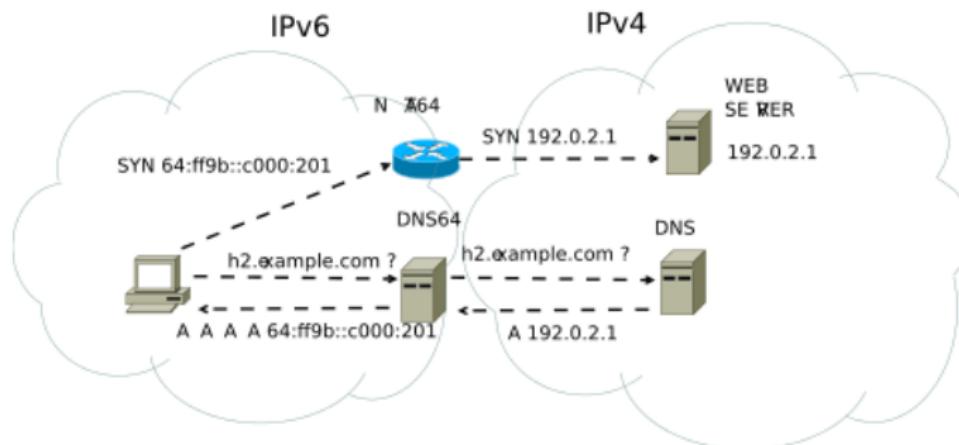
# Túneles IPv4/IPv6 (cont.)

- Teredo: sobre direcciones IPv4 privadas.



# Técnicas de Traducción

- NAT64(NATPT) + DNS64.
- No es una técnica recomendada, rompe principio end-2-end.

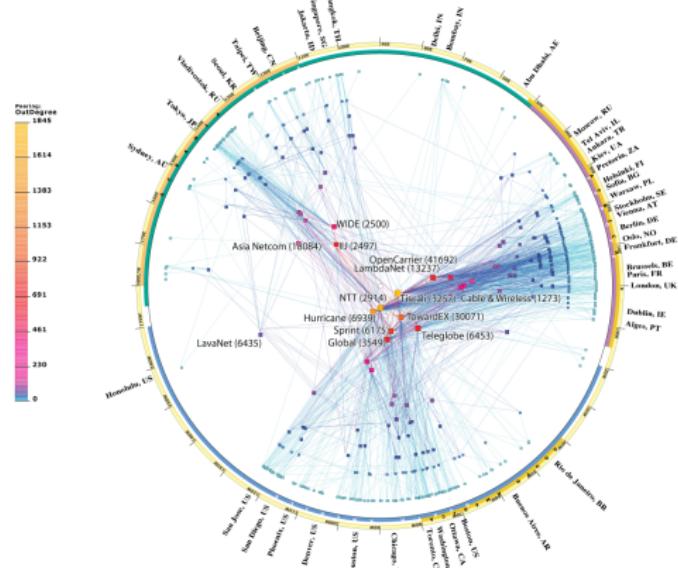


fuente: <http://www.wikipedia.com>

# IPv6 en 2008

## CAIDA's IPv6 AS Core AS-level INTERNET GRAPH

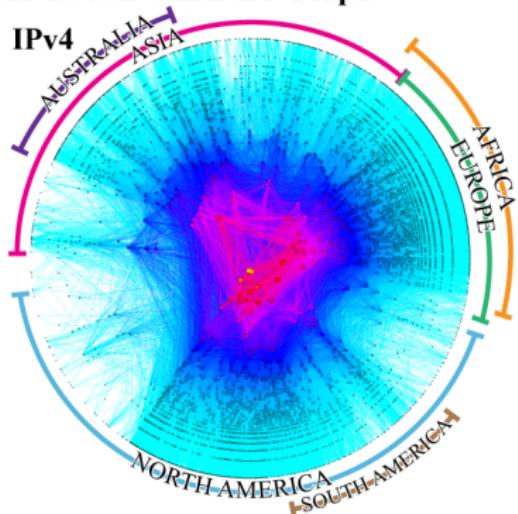
Community Collected January 2008



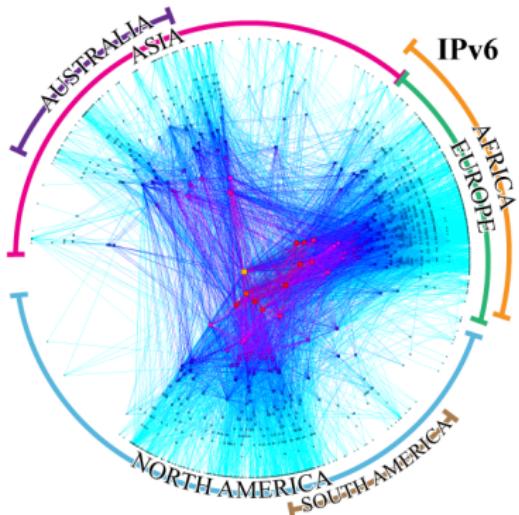
copyright © 2008 UC Regents. all rights reserved.

# IPv4 vs IPv6 en 2013

CAIDA's IPv4 & IPv6 AS Core  
AS-level INTERNET Graph



Archipelago  
Jan 2013

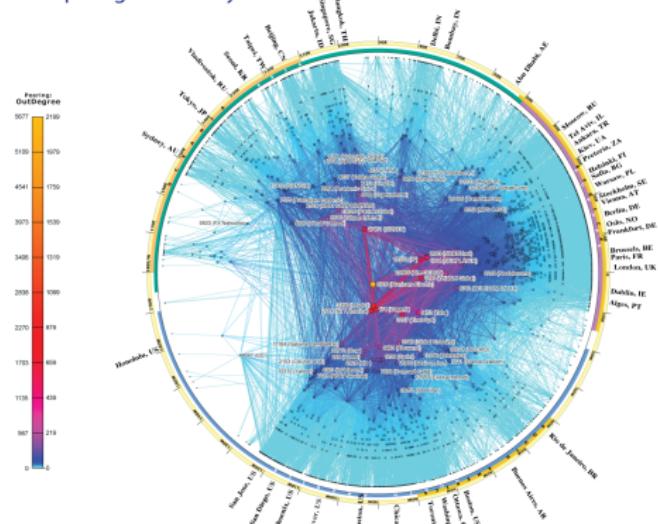


Copyright 2013 UC Regents. All rights reserved.

# La Internet Actual

## CAIDA's IPv6 AS Core AS-level INTERNET GRAPH

Archipelago January 2015



copyright © 2015 UC Regents. All rights reserved.

# Referencias

- IPv6 Essentials (Integrating IPv6 into your IPv4 Network) Silvia Haggen. O'Reilly. 2nd Ed, 2006.
- Migrating to IPv6. Marc Blanchet. John Wiley and Sons, 2006.
- Tutorial de Jordi Palet Martínez:  
[http://www.consulintel.es/html/ForoIPv6/Documentos/Tutorial de IPv6.pdf](http://www.consulintel.es/html/ForoIPv6/Documentos/Tutorial%20de%20IPv6.pdf).
- TCP/IP Illustrated, Volume 1: The Protocols, 2nd Ed. W. Richard Stevens, Kevin R. Fall. Addison-Wesley Professional Computing Series, 2011.
- IPv6 for All: <http://www.ipv6tf.org/pdf/ipv6forall.pdf>.