

Computer Networking A Top-Down Approach

James F. Kurose • Keith W. Ross

Capítulo 1: Computer networks and the internet	6
1.1 What's the internet?	6
• Nuts and Bolts	6
• Proveedor de servicios	6
• Protocolos	6
1.2 The network edge	6
1.3 The Network Core	8
• Packet switching	8
• Circuit Switching	9
• A Network of Networks	9
1.4 Delay, Loss and Throughput in Packet-Switched Networks	10
• Throughput (tasa de transferencia)	11
1.5 Protocol Layers and Their Service Model	11
Capítulo 2 - Capa de aplicación	13
2.1 Principios de las aplicaciones de red	13
• Servicios que ofrece la capa de transporte	14
• Servicios de transporte que provee Internet	14
• Protocolos de la capa de aplicación	14
2.2 La Web y HTTP	15
• Visión general de HTTP	15
• Conexiones persistentes y no persistentes	15
• Formato de mensajes HTTP:	15
• Interacción cliente-servidor: Cookies	16
• Web Caching:	16
• HTTP/2	17
2.3 Electronic Mail	18
• Interacciones SMTP	19
• Formato de mensajes mail	19
2.4 DNS- Domain Name System	19
• Servicios que provee DNS:	19
• Como funciona DNS a grandes rasgos	19
• Problemas de un servidor DNS centralizado	20
• DNS distribuido jerárquicamente	20
• Como realmente funciona DNS:	20
• Mensajes y registros DNS	21
• Mensajes DNS:	21
2.5 Peer-To-Peer	22

● Auto Escalabilidad de P2P:	22
● BitTorrent:	23
2.6 Video Streaming and Content Distribution Networks	24
● Internet Video	24
● HTTP Streaming and DASH	24
● Content Distribution Networks	25
Capítulo 3 - Capa de transporte	26
3.1 Introduction and Transport-Layer Services	26
Relación entre la capa de transporte y la de aplicación	26
Mirada general sobre los protocolos de transporte	26
3.2 Multiplexación y demultiplexación	27
Connectionless Multiplexing and Demultiplexing (UDP)	27
Connection-Oriented Multiplexing and Demultiplexing (TCP)	27
3.3 Connectionless Transport: UDP	28
UDP Segment Structure	29
UDP Checksum	29
3.4 Principios de la transferencia de datos confiables	29
Creando un protocolo de transferencia de datos confiable con máquinas de estado finitas	30
Pipelined Reliable Data Transfer Protocols	35
Go-Back-N (GBN)	36
Selective Repeat (SR)	37
3.5 Connection-Oriented Transport: TCP (Transmission Control Protocol)	38
Conexión TCP	38
Estructura del segmento TCP	39
Números de secuencia y de ACKs	39
Round-Trip Time Estimation and Timeout	40
Transferencia de datos confiable	40
Fast Retransmit	41
Control de flujo	42
Establecimiento de conexión TCP	42
Capítulo 4: The network Layer: Data Plane	43
4.1 Overview of Network Layer	43
Forwarding and Routing	43
Control Plane: Traditional Approach	44
Control Plane: SDN Approach	44
Modelo de servicios de Red	44
4.2 What's inside a Router?	44
Input Port Processing and Destination-Based Forwarding	45
Switching	45
Output Port Processing	46
¿Dónde ocurre el encolamiento?	46
Input Queuing	47

Output Queueing	47
Packet Scheduling	47
Neutralidad de la red	48
4.3 The Internet Protocol (IP): IPv4, Addressing, IPv6 and More	48
Formato del datagrama de IPv4	48
IPv4 Addressing	49
Obtaining a Block of Addresses	49
The Dynamic Host Configuration Protocol (DHCP)	50
Network Address Translation (NAT)	50
IPv6	51
Formato del datagrama IPv6	51
Transición de IPv4 a IPv6: Tunneling	52
4.4 Generalized Forwarding and SDN	53
Match	53
Action	53
Ejemplos de Match-plus-action	54
4.5 Middleboxes	54
Capítulo 5: The network Layer: Control Plane	54
5.1 Introduction	54
5.2 Routing Algorithms	55
Link-State (LS) Routing Algorithm	56
The Distance-Vector(DV) Routing Algorithm	57
Distance-Vector (DV) Algorithm	58
Distance-Vector Algorithm: difusión de información	59
Cambios en los costos de los enlaces	59
Comparación entre LS y VS	60
5.3 Intra-AS Routing in the Internet: OSPF	61
Open Shortest Path First(OSPF)	61
5.4 Routing Among the ISPs: BGP	62
The Role of BGP	63
Advertising BGP Route Information	63
Determining the Best Routes	64
Hot Potato Routing	64
Route-Selection Algorithm	65
IP-Anycast	65
Routing Policy	66
¿Por qué existe ruteo inter-dominio e intra-dominio?	66
Capítulo 6: The Link Layer and LANs	67
6.1 Introduction to the Link Layer	67
Servicios que provee la capa de enlace	67
Donde se implementa la capa de enlace?	67

6.2 Error-Detection and Correction Techniques	68
Control de paridad	68
Checksumming Methods	69
Cyclic Redundancy Check (CRC)	69
6.3 Multiple Access Links And Protocols	70
Channel Partitioning Protocols	71
TDMA (Time Division Multiple Access)	71
FDMA (Frequency Division Multiple Access)	71
CDMA (Code Division Multiple Access)	72
Random Access Protocols	72
Slotted ALOHA	72
Aloha	73
Carrier Sense Multiple Access (CSMA)	74
Carrier Sense Multiple Access with Collision Detection (CSMA/CD)	74
Taking-Turns Protocols	75
Polling	75
Token-passing protocol	76
DOCSIS: The Link-Layer Protocol for Cable Internet Access	76
6.4 Switched Local Area Networks	77
Link-Layer Addressing and ARP	77
MAC Address	77
Address Resolution Protocol (ARP)	78
Sending a Datagram off the Subnet	78
Ethernet	79
Ethernet Frame Structure	80
Servicio Ethernet	80
Tecnologías Ethernet	81
Link-Layer Switches	81
Forwarding and Filtering	81
Self-Learning	82
Properties of Link-Layer Switching	83
Switches vs Routers	84
Virtual Local Area Networks (VLANs)	84
6.5 Link Virtualization: A network as a Link Layer	85
Multiprotocol Label Switching (MPLS)	86
6.6 Data Center Networking	87
Data Center Architectures	87
Topología jerárquica	87
Técnica para evitar problemas de escala y falta de throughput	88
6.7 Retrospective: A Day in the Life of a Web Page Request	88
Paso 1: DHCP, UDP, IP y Ethernet. Conexión a la red	88
Paso 2: DNS y ARP. Obtención de MAC gateway	89

Paso 3: Resolución de domino DNS	89
Paso 4: TCP y HTTP	89
Capítulo 8: Security in Computer Networks	90
8.1 What is Network Security?	90
8.2 Principles of Cryptography	90
Symmetric Key Cryptography	91
Cifrado por bloques/ Block ciphers	91
Cipher-Block Chaining	92
Public Key Encryption	92
RSA	93
8.3 Message Integrity and Digital Signatures	94
Cryptographic Hash Functions	94
Message Authentication Code	94
Digital Signatures	95
8.4 End-Point Authentication	96
8.6 Securing TCP Connections: TLS	97
Visión general	98
8.7 Network-Layer Security: IPsec and Virtual Private Networks	99
IPsec and VPNs	99
The AH and ESP Protocols	100
Security Associations	100
The IPsec Datagram	100
Bases de datos IPsec	102
IKE: Key Management in IPsec	102
8.9 Firewalls	102

Capítulo 1: Computer networks and the internet

1.1 What's the internet?

Dos enfoques: Visto como *Nuts and Bolts* (visto como todos los dispositivos) y visto como una infraestructura que provee servicios a las apps

- **Nuts and Bolts**
 - Conecta millones de dispositivos
 - Hosts: sistemas finales, dispositivos que corren apps
 - Los hosts se conectan por medio de **communication links** y **packet switches**
 - Packets: paquetes de información que se envían entre hosts
 - Packet switch: Toma un packet y lo vuelve a enviar por medio de communication links. (Routers y link-layer switches)
 - Analogía del libro: Packets son con camiones, communication links son las calles, packet switches son las intersecciones, end systems son los edificios
 - Los hosts acceden a internet por medio de **Internet Service Providers (ISPs)**
 - Un ISP es una red de packet switches y communication links
 - Todo el internet está regido bajo distintos protocolos que ponen reglas a la hora de enviar información. **Transmission Control Protocol (TCP)** y **Internet Protocol (IP)**
 - Los **requests for comments (RFCs)** tienen detallado todos los estándares y detalles de los protocolos
- **Proveedor de servicios**
 - Distributed apps: solo corren en los hosts/end systems no en packet switches
 - Internet permite el envío de data por medio del **socket interface** que especifica como un programa corriendo en un end system le envie data a otro
- **Protocolos**
 - Definen reglas, mensajes específicos y respuestas para esos mensajes
 - Definen formato y orden de los mensajes enviados y acciones a tomar una vez que lo recibes
 - Toda relación/comunicación entre 2 dispositivos o más está regida por un protocolo

1.2 The network edge

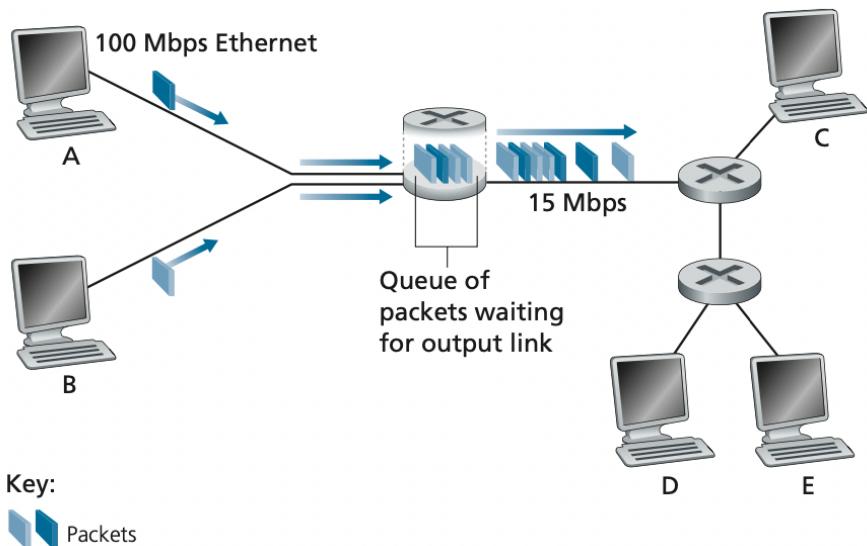
- Hosts = end systems
- Un host puede ser un cliente (Compus, teléfonos) o un servidor (Data center)
- Conexión a internet para casas/edificios
 - **DSL (digital subscriber line)**
 - Comparte cable par trenzado con teléfono
 - Transmite datos por medio del protocolo **DSLAM**, diferentes tipos de datos por diferente frecuencia. Un splitter divide la data para que luego el ISP pueda manejar ambas por separado
 - **Internet por cable**
 - Hace uso del cable de la compañía de televisión

- Salen de las casas por medio de un cable coaxial, llegan a un fiber node que lo transforma a cable de fibra óptica para llegar al **cable modem termination system (CMTS)** que luego te conecta al internet
 - Suelen tener un canal de carga y uno de descarga
 - Se puede sobrecargar si muchos usuarios la usan al mismo tiempo
- **FTTH (fiber to the home)**
 - Similar al cable pero todo fibra óptica
 - Más rápido
- Suelen combinar el wifi access point, router y modem(dsl/cable) en un solo aparato creando un LAN
- Conexión para empresas/universidades
 - Se crean redes locales para tener mejor conexión. LAN(Local area network)
 - Sirve para distancias cortas, locales.
 - Fueron pensadas para empresas pero hoy dia estan en muchos hogares
 - **Ethernet**
 - Conecta los dispositivos por cable al router o switch
 - Mayor velocidad de carga y descarga
 - **Wireless lan: WIFI**
 - Distancias un poco más largas que ethernet
 - Mezcla las tecnologías para crear una ms veli
- Redes de acceso a celular
 - 3G, LTE 4G y 5G
 - Provistas por operador de teléfono
 - Gran amplitud
- Redes de acceso para data center
 - Interconexión de miles de servidores a gran velocidad
- Medios físicos para las conexiones
 - Medios guiados (por cable)
 - Par trenzado, utilizado en telefonía y cables ethernet
 - Cable coaxial, bidireccional, múltiples canales y con banda ancha. Utilizado en televisión
 - Fibra óptica
 - Conduce pulsos de luz
 - Muy veloz pero muy caro
 - Medios no guiados
 - Canales de radio
 - Se mueven por el espacio electromagnético
 - No requiere instalación
 - Problemas de propagación
 - Interferencia
 - Efecto espejo
 - WIFI, 4G, Bluetooth

1.3 The Network Core

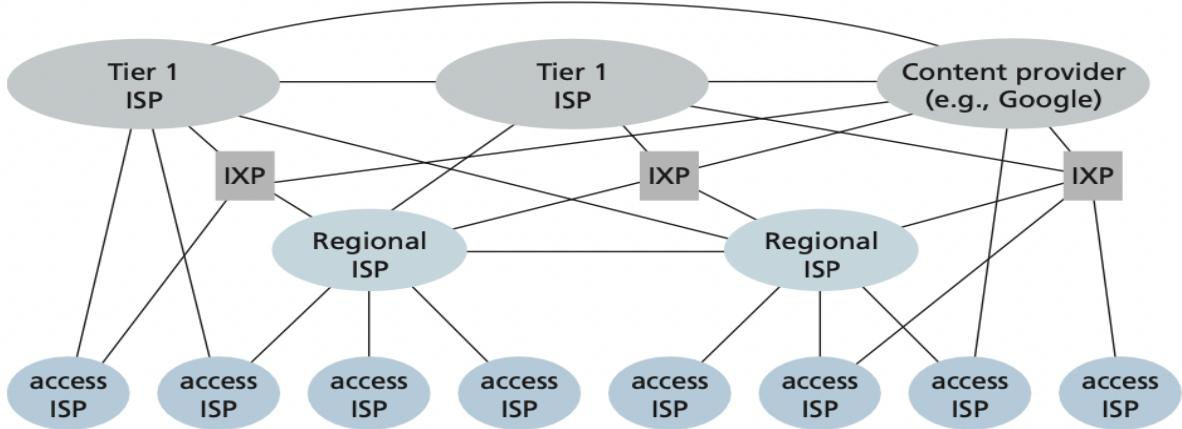
- **Packet switching**

- Toda comunicación o mensaje se envía en forma de **packets**
- Cada packet viaja a través de communication links y packet switches(routers) a una determinada velocidad
- Se envían packets de L bits a una velocidad R bits/sec por lo que el tiempo de transmisión es L/R sec
- Ideal para enviar datos en rafagas
- Recursos compartidos
- Puede sufrir congestión y pérdidas
 - Las manejan los distintos protocolos
- **Store and forward transmission**
 - Es el metodo mas comun a la hora de transmitir packets
 - El packet switch debe recibir por completo el packet antes de poder enviarlo nuevamente
- **Queuing Delays and Packet Loss**
 - El packet switch va recibiendo packets y los va almacenando en una cola para luego enviarlos
 - Si la tasa de llegada es mas rápida que la tasa de salida se van a generar colas de packets y va a haber delay a la hora de enviarlos



- Si la cola está llena, el packet switch deja de recibir packets y estos últimos se pierden

- **Forwarding Tables and Routing Protocols**
 - Cuando un packet llega a packet switch, este mira una parte del packet en donde contiene la IP del destinatario y por medio de un protocolo determina a donde enviarlo
 - El packet switch no lo envía directamente al destinatario sino que lo envía a otro packet switch más cercano al destino
- **Circuit Switching**
 - A diferencia del packet switching, esta forma de transportar data es reservada. Esto significa que al momento de estar enviando, nadie más puede usar ese “canal”
 - Analogía a restaurante con reserva (circuit) y sin reserva (packet)
 - Teléfonos tradicionales
 - Se reserva el canal para poder enviar la voz por un determinado tiempo
 - Estaban literalmente conectados
 - **Frequency division multiplexing (FDM)**
 - Dividir el ancho total de banda para que se puedan enviar más de un mensaje a la vez
 - **Time division Multiplexing (TDM)**
 - Cada conexión recibe un intervalo de tiempo (slots) para poder enviar data
 - Tiene silent periods, cuando nadie está hablando en la llamada no se le puede asignar el recurso a otra conexión
- Ambas maneras de transmisión de datos se usan hoy día, pero la tendencia está yendo hacia el packet switching ya que es más eficiente. El Circuit Switching muchas veces reserva conexiones innecesariamente por lo que la hace mucho más ineficiente.
- **A Network of Networks**
 - Hosts se conectan por distintos medios a una ISP local, ya sea una empresa, universidad o compañía. Pero cómo conectamos a hosts conectados a ISP diferentes? Deben estar conectadas entre sí en una **red de redes**
 - ISP locales son consumidores de una ISP global (proveedor)
 - Hosts: se conectan a internet por medio de una ISP local
 - ISP locales: Se conectan con ISP regionales para que sus clientes estén comunicados entre sí
 - ISP regionales: Están conectadas a IXP y a Tier 1 ISP
 - IXP: puntos neutros de intercambio. Se genera un intercambio de data entre distintas ISP para reducir costos
 - ISP Tier 1: Son las ISP más grandes y globales
 - Data center(Google, Amazon): Crean redes de conexión privadas pero también se conectan a ISP tier 1 y regionales

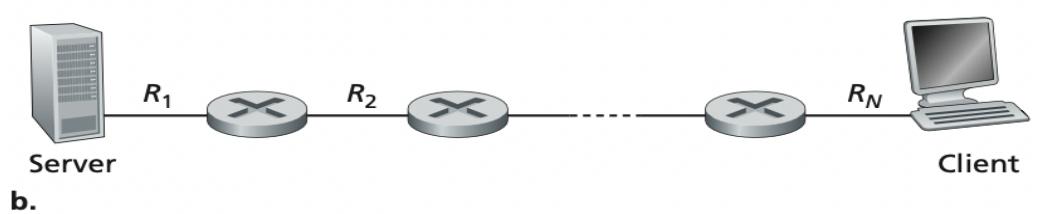
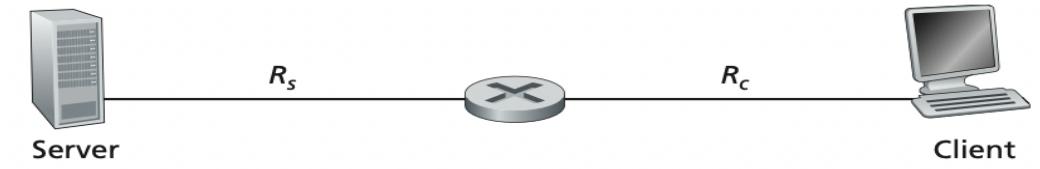


- Peering entre ISP del mismo tipo, es una conexión entre 2 ISP para enviarse data directamente sin pasar por un ISP de mayor envergadura(Acuerdos gratuitos)
- Todo ISP que desee conectarse a un ISP mayor debe pagarle, a excepción del peering
- Las ISP suelen estar conectadas a varias ISP más grandes así tienen mayor cantidad de opciones/caminos/paths a la hora de enviar la data. **Multi-home**

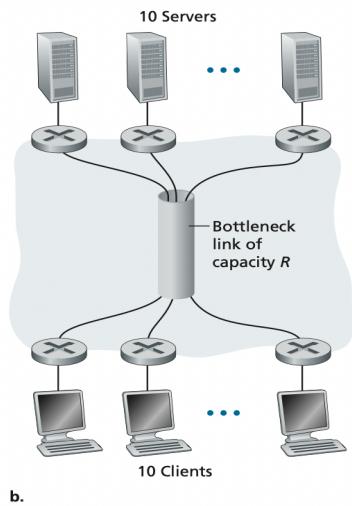
1.4 Delay, Loss and Throughput in Packet-Switched Networks

- Al enviar packets de un host a otro, estos se pueden demorar e incluso perder en el camino
- Distintos tipos de delay: processing, queueing, transmission and propagation
- La suma de los 4 es el total nodal delay
- **Processing Delay:** Tiempo que tarda en examinar el header del packet para ver dónde enviarlo nuevamente. Medido en microsegundos.
- **Queueing Delay:** Tiempo que espera el packet en la cola para luego ser enviado. Un packet puede ser transmitido por el cable si y solo si no hay otro packet siendo enviado al mismo tiempo. Medido en milisegundos.
- **Transmission Delay:** $L = \text{tamaño de packet}$, $R = \text{tasa de transmisión en bits/segundos}$. $L/R = \text{transmission delay}$. Tiempo que tarda el router en pasar los bits al medio de transmisión. No depende de la distancia entre A y B.
- **Propagation Delay:** Tiempo que tarda el packet en el cable. $S = \text{Velocidad de propagación (vel de la luz)} = 2 \times 10^8$ m/s. $D = \text{Distancia entre A y B}$. $D/S = \text{propagation delay}$. No depende ni del tamaño del paquete ni de la tasa de transmisión.
- Si la tasa de arribos de packets es mayor que la tasa de transmisión, se va a generar una cola de packets que crece de forma exponencial.
- **Packet Loss:** Si la cola está llena y llega un nuevo packet este no podrá ser encolado y por lo tanto se perderá.
- **End-to-end Delay:** Tiempo total que tarda un packet en llegar desde un host A hasta B. $d_{end-end} = N \times (d_{proc} + d_{trans} + d_{prop})$. Suponiendo que hay N links (N-1 routers), que no hay queueing delay y que los otros 3 delays son iguales en todos los routers.
- **Traceroute**
 - Te devuelve el camino que haría el packet para llegar a x destino
 - Mide el tiempo de respuesta de cada servidor/host->ping

- TTL: cuantos saltos puede hacer el mensaje especial
 - Manda 3 mensajes por TTL
 - El primero es con TTL = 1 y después va aumentando de a 1
 - Cuando un router recibe uno de estos mensajes le resta 1 al TTL y si llega a 0 le responde al host original el tiempo
- **Throughput (tasa de transferencia)**
 - Cantidad de megabits por segundo a la que está siendo transferida una archivo



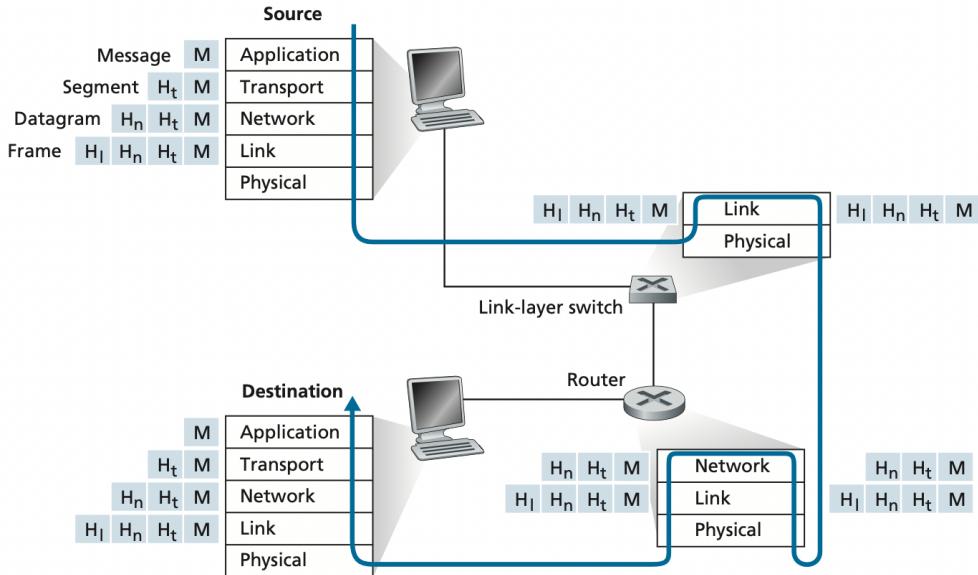
- El Throughput de una conexión es el mínimo (R_1 , R_2 , ..., R_N) y el tiempo de transferencia es el tamaño del packet/Throughput
- Suelen haber cuellos de botella que limitan el throughput
- En la práctica, los cuellos de botella suelen ser R_c o R_s ya que el network core tiene una tasa de transmisión mucho superior a la necesaria justamente para mejorar los tiempos



1.5 Protocol Layers and Their Service Model

- Importante modularizar en layers/capas porque puedes modificarlas por separado sin afectar al sistema entero
- Cada layer provee un **service model** (1) brindando ciertas acciones (2) usando los servicios de los layers adyacentes
- Los distintos layers se pueden implementar tanto en software como hardware o en ambos
- El **internet protocol stack** tiene 5 principales layers: Application, transport, network, link y physical. Todos distribuidos.
- **Application Layer:** Todos los protocolos que sirven como soporte para las apps (HTTP: web docs requests and transfer, SMTP: transfer mail, FTP: transfer files, DNS) **message**
- **Transport Layer:** Protocolos para la transferencia de datos, **segment**
 - TCP (transmission control protocol) : permite que 2 hosts se puedan enviar datos, garantiza que los mensajes se envían en orden correcto y sin errores gracias al "acuse de recibo". Corta los mensajes y provee un sistema de control de congestión.

- UDP(user datagram protocol): Permite conexión remota de 2 hosts ya que tiene suficiente info sobre el destino, no garantiza llegada en orden, no tiene control de flujo
- **Network Layer:** Ruteo de datagramas, recibe el segment y el address del transport layer y lo entrega al destinatario. Determina la ruta que hará cada packet. **Datagrams**
 - **IP(internet protocol):** Responsable del direccionamiento de packets. Busca la mejor ruta para conectar 2 hosts pero no garantiza nada. Puede no llegar, llegar roto, en otro orden. De eso se ocupa el transport layer.
- **Link Layer:** Se ocupa de transferir los datos desde un nodo a otro (dispositivos de redes adyacentes). Recibe información del network layer sobre para dónde mandar el packet y utiliza el physical layer para enviar los datos. **Frame**
 - Utiliza los protocolos WIFI, Ethernet y algunos de cable
- **Physical Layer:** Se ocupa de transferir los bits entre los nodos por medios físicos
 - Coaxial, fibra óptica, ethernet, entre otros.
- **Encapsulamiento:** Cada layer va agarrando información de la layer superior y le agrega información propia para luego pasarselo todo a la layer inferior



- El host envía un **M (message)** en la application layer y se lo pasa a la *transport layer* que le agrega un **header_t**, con info, como error-detection y para qué app es el msj, (**H_t**) que va a ser usado por la *transport layer* pero del destinatario. El **message** con el **H_t** forman el **segment de la transport layer** que le llega al *network layer*, que le agrega su propio **header_n**, con info como destinatario y dirección (**H_n**). El **segment** (**M + H_t**) + **H_n** forma el **datagram** y se lo pasan a la *link layer* que le agrega su propia data/header_l (**H_l**) y le pasan el **frame (datagram + H_l)** a la *physical layer* que se encarga de enviar todos los bits.
- Cada layer recibe información/packet de una layer superior, esta se llama **payload field** y le agrega un header con data propia.

Capítulo 2 - Capa de aplicación

2.1 Principios de las aplicaciones de red

- Un programa/servicio/app corre en distintos end systems/hosts en la capa de aplicación
 - 2 principales: Usuario y servidor(hosteado en data center)
 - No van a “correr” en la red/routers sino en hosts. Para eso están las capas inferiores del internet, sería un caos si todos hacen todo.
- 2 tipos de arquitecturas en la capa de aplicación: P2P y cliente-servidor
- **Cliente servidor:**
 - Muchos hosts(clientes) le piden cosas a un servidor. El server está siempre conectado y tiene una IP fija y conocida
 - Los clientes no se comunican entre sí
 - Muchas veces un único server no soporta todas las requests. Para solucionar esto se lo hostea en grandes data centers donde pueden tener más servidores
- **P2P**
 - No hay un servidor (no se puede caer)
 - Comunicación directa entre **peers**
 - Es auto escalable, si aumentan los usuarios también aumenta los “servidores” que vendrían a ser los mismo usuarios
 - Fácil de mantener pero no hay verificación del contenido enviado
- Comunicación de procesos
 - Procesos corriendo en diferentes hosts se comunican por medio de diferentes mensajes. Siempre pensado como paradigma cliente-servidor
 - El que inicia la comunicación es el cliente, el que espera a ser contactado es el server (Tambien aplica para P2P)
 - La forma en que un proceso puede enviarle un msj a otro y recibir es por medio un **socket**
 - Analogía socket-puerta
 - Un socket es el intermediario entre la capa de aplicación y la de transporte

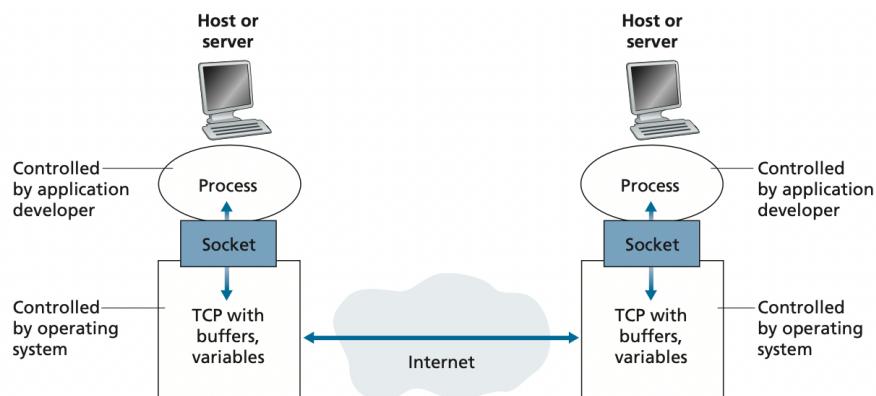


Figure 2.3 ♦ Application processes, sockets, and underlying transport protocol

- Tipos de socket:
 - Datagram socket: Se montan en UDP
 - Stream socket: Se montan en TCP
 - Raw sockets: Envío de paquetes directo con IP

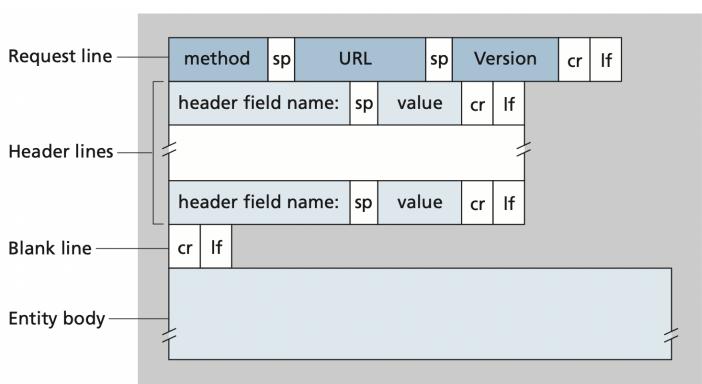
- En necesario conocer la IP del otro host para poder enviar mensajes y tambien el numero de puerto para diferenciarse de otros procesos y mensajes
- **Servicios que ofrece la capa de transporte**
 - **Transferencia de datos confiable**
 - TCP provee un servicio de entrega de data íntegros/confiable(Bancos, servicios)
 - UDP no garantiza la entrega de datos íntegros. La usan para transferir mensajes que toleran pérdidas de data en el medio(Video)
 - **Throughput**
 - Tasa de transferencia de MBPS
 - Es variable en el tiempo
 - Se puede proveer una tasa de transferencia mínima garantizada
 - Algunos procesos requieren tener tasa de transferencia mínima constante (Teléfono) y otros no, llamados aplicaciones elásticas(Mail, envío de archivos)
 - **Delay**
 - Algunas apps necesitan bajo delay/latencia: telefonía y videojuegos
 - **Seguridad**
 - Cifrado y descifrado de mensajes, integridad
- **Servicios de transporte que provee Internet**
 - **TCP(Transmission control protocol)**
 - Lo utilizan FTP, SMTP, HTTP, telefonía, etc
 - Handshaking antes de empezar a enviarse mensajes, orientado a conexión
 - Control de flujo y congestión
 - Provee transferencia de datos confiable: Garantiza la correcta llegada del mensaje, en orden y sin faltantes
 - Se le implemento TLS(Transport layer security) para encriptación de mensajes ya que TCP no provee ningún tipo de seguridad al enviarlos. Se lo utiliza a nivel aplicación, tiene sus propios sockets, APIs y librerías de código
 - **UDP**
 - Provee un servicio minimal sin garantizar control, correcta llegada ni orden
 - Más barato y rápido que TCP
 - Ninguno de los 2 servicios proveen garantías de throughput ni delay mínimo. Sin embargo, muchas aplicaciones tiempo-sensibles como telefonía y videojuegos se las arreglan para funcionar.
- **Protocolos de la capa de aplicación**
 - Definen como los distintos procesos de aplicaciones se comunican entre sí
 - Que tipo de mensajes: request and response
 - Sintaxis y campos
 - El significado de cada campo
 - Reglas para determinar cuándo y cómo responder y enviar
 - Todos los protocolos están definidos en RFC que especifican su uso y funcionamiento
 - Protocolo HTTP(hyper text transfer protocol) para transferencia de páginas web

2.2 La Web y HTTP

- La web fue la primera aplicación de Internet que llamó la atención del público
- Servicio on-demand, lo que quieras cuando quieras
- Le da la base a la mayoría de aplicaciones de internet
- **Visión general de HTTP**
 - Protocolo de web
 - Define la estructura de los mensajes que se intercambian entre el cliente y el servidor.
Como el cliente pide páginas web y cómo el servidor las transfiere
 - Una página web consiste de objetos (texto, imágenes, videos, archivos HTML) que están indexados a una URL
 - Usualmente primero se pide el archivo HTML, que tiene referencias a otros objetos que luego se van pidiendo
 - Usa conexiones TCP para asegurarse de la correcta llegada de la data
 - Es un protocolo stateless, no guarda información de los clientes
- **Conexiones persistentes y no persistentes**
 - Por default, HTTP/1.1 usa conexiones TCP persistentes. Aunque el cliente lo podría modificar.
 - **Conexiones no persistentes:** establezco la conexión (handshake), pido un único archivo y luego de que me lo envíen se cierra la conexión.
 - 2 RTT + transmission time
 - También existe abrir múltiples conexiones TCP en paralelo para reducir el tiempo
 - **Conexiones persistentes:** Se mantiene abierta la conexión luego de pedir un archivo, para luego evitar tener que abrir otra conexión para enviar otros objetos de la página
 - Reduce el tiempo
 - El servidor cierra la conexión luego de un tiempo máximo de no interacción
- **Formato de mensajes HTTP:**
 - Requests message
 - Request line: Método(GET,PUT, POST, DELETE, HEAD)- URL- HTTP version
 - Header lines: host, tipo de conexión, user-agent, idioma
 - Métodos: GET para pedir cosas, POST para pedir cosas pero con formularios, PUT para agregar al archivo(Solo programadores), HEAD es igual que GET pero no pide el archivo pide los headers(Solo programadores, para debug)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

- Response message



- Status line: Version HTTP- Código de status- mensaje de status
- Seis header lines: Tipo de conexión, fecha en la que se creo la respuesta HTTP, servidor, fecha de última modificación del objeto, longitud del objeto, tipo de objeto
- Cuerpo: donde va el objeto pedido
- Status y sus mensajes:

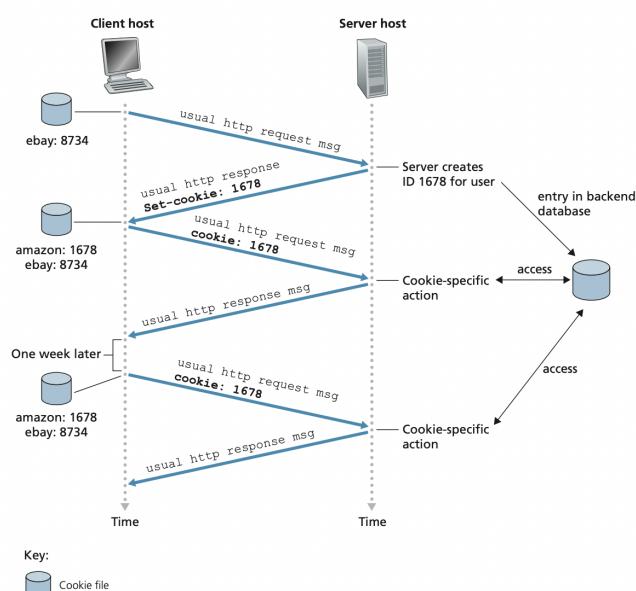
errores

- 200 OK:
Request exitosa
 - 301 Moved Permanently
 - 304 not modified
 - 400 Bad Request
 - 404 Not Found
 - 500 Internal server error
 - 505 HTTP Version Not Supported
- ```

HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

- **Interacción cliente-servidor: Cookies**

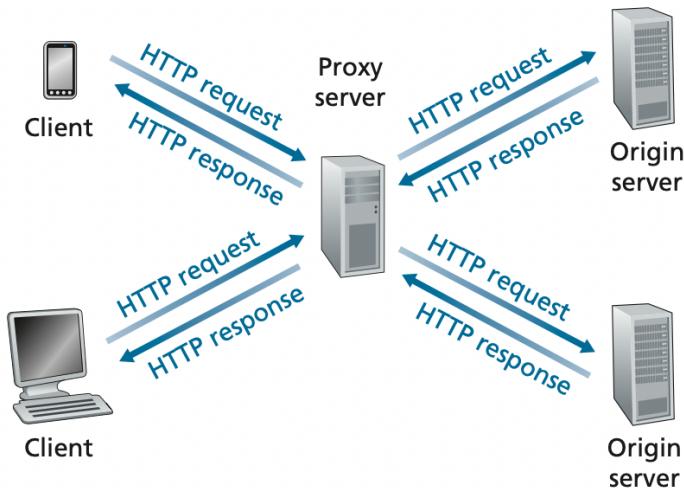
- Las cookies le permiten al servidor guardar información del usuario, sirven para mantener estados
- Páginas comerciales las usan para guardar los carritos y preferencias del cliente
- Tienen 4 componentes:
  - Un header en la response del servidor, set-cookie identificador de usuario
  - Un header en la request del usuario, para asociarlo a su data
  - Un archivo con la cookie
  - Base de datos en el sitio web con la info del cliente
- Se mantienen en el tiempo



- **Web Caching:**

- Un servidor caché o servidor proxy satisfacen requests HTTP de parte del servidor original

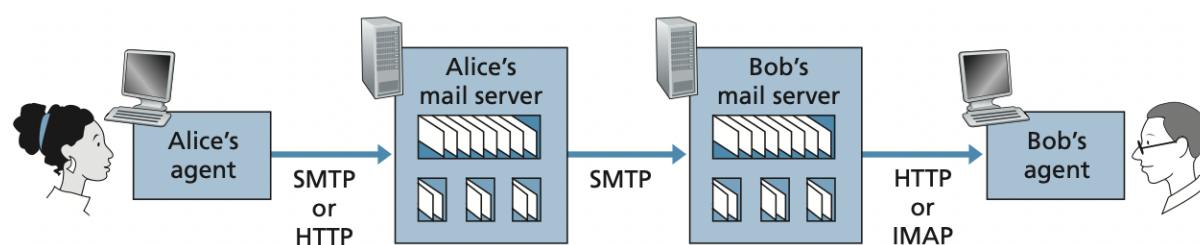
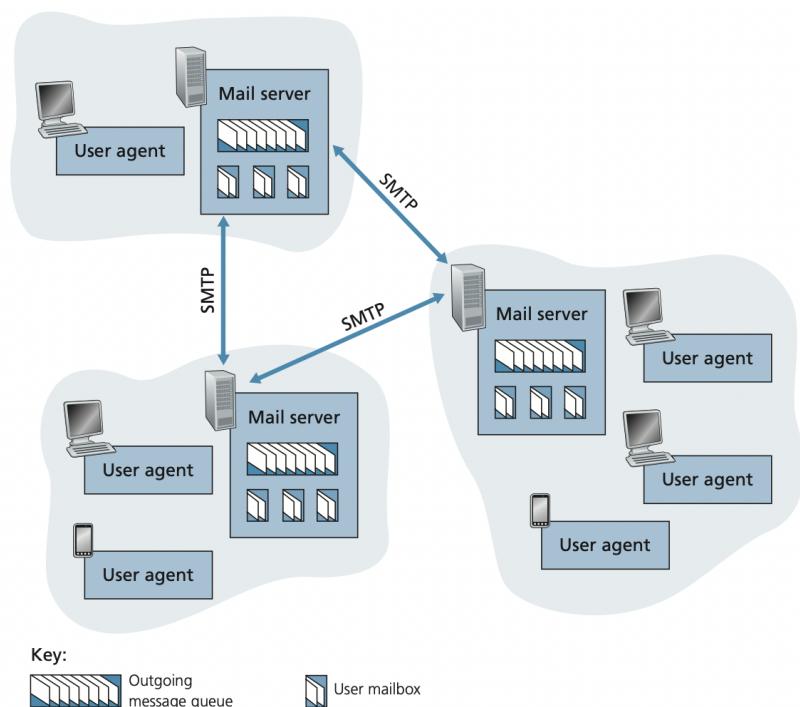
- Tienen su propia memoria y guardan copias de lo que recientemente le fue pedido al servidor original
- Al pedir un objeto por medio de HTTP, primero se establece conexión con la caché y en caso de que no lo tenga, esta se conecta al servidor original, le pide al objeto, lo cachea y lo manda al cliente.
- Toman el rol de servidor para el usuario y de cliente para el servidor
- Suelen reducir el tiempo de respuesta y alivianar el tráfico que recibe el servidor
- Las suelen instalar ISP, universidades y empresas



- **GET condicional:**
  - Soluciona el problema de pedir objetos a la caché que estén desactualizados
  - Cliente pide objeto a cache, cache lo tiene en su memoria, le pregunta al server con un *if-modified-since: fecha en la que le llegó a la caché*, el servidor le puede responder *304 Not Modified* o enviarle el archivo *nuevamente si es que fue modificado*.
- **HTTP/2**
  - Vino a resolver problemas de tiempo que tenía HTTP/1.1
  - HTTP/1.1 implementó múltiples GETs en una misma conexión TCP, el servidor responde FCFS. Esto produce Head of line blocking **HOL**: si el primer objeto es muy pesado, los demás no se van a enviar nunca.
  - Ese problema lo solucionaron abriendo múltiples conexiones TCP en paralelo
  - HTTP/2 soluciona el problema de otra forma ya que al abrir muchas conexiones en paralelo se requieren más sockets y es más complicado manejar el control de flujo
  - Implementaron el **Framing**, divide todos los objetos en frames y los va mandando en una única conexión TCP intercalados entre sí y cuando llegan los unifica de nuevo
    - También permite aplicarle prioridades a los objetos enviados
    - El servidor también puede mandarle objetos que le va a servir más tarde que todavía no pidió
  - HTTP/3 aplica el protocolo QUIC sobre UDP

## 2.3 Electronic Mail

- El protocolo SMTP(Simple mail transfer protocol) es el que regula los mensajes de mail
  - Utiliza conexiones TCP persistentes en la capa de transporte, ya que garantiza la correcta llegada de los mensajes
  - Permite enviar mensajes entre servidores
  - Es un PUSH protocol, te permite enviar cosas pero no pedirlas
- 3 principales componentes
  - User agent: Outlook, Gmail, etc
    - Te permiten escribir y leer mails
  - Servidores de mail: almacenan los mails en buzones y tiene una cola de salida
  - Protocolo SMTP
- Permite conexión asincrónica entre 2 personas
- El servidor de mail actúa de cliente y de servidor dependiendo del caso
  - Cuando recibe un mail, es el servidor
  - Cuando envía un mail, es el cliente
- Envío de mails
  - Usuario A utiliza su user agent para redactar mail
  - User agent de A envía el mail a su servidor vía SMTP y lo encola
  - Servidor A(cliente) abre una conexión TCP con servidor B(servidor) y le envía el mail pero la forma en la que se comunican está establecida por SMTP
    - Handshake, transfiere data y cierra conexión
  - Servidor B guarda el mail en el buzón del usuario B
  - Usuario B utiliza su User agent para leer el mail via IMAP(Internet mail access protocol) o HTTP

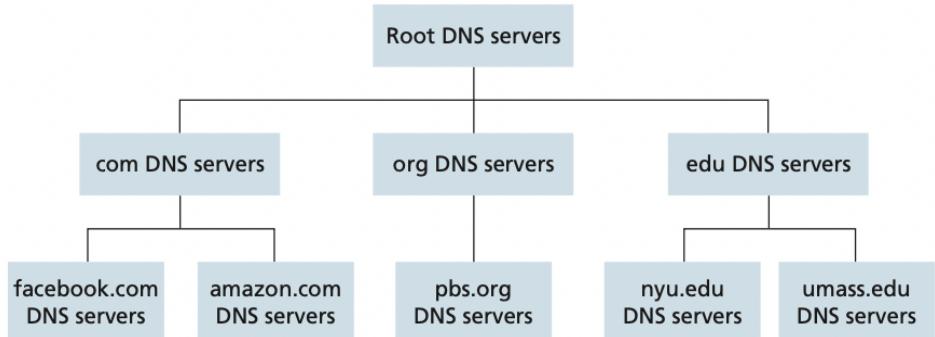


- Es muy importante el uso de servidores de mail ya que sin estos, deberíamos estar conectados al momento de recibir un mail.
- **Interacciones SMTP**
  - Se establece conexión y el cliente dice que servidor es. HELO
  - Cliente dice quien envía: **MAIL FROM** y a quién va el mail: **RCPT TO**
    - El servidor autentifica a ambos mails
  - El cliente comienza a mandar el mail **DATA**. Requiere toda la data en ASCII
  - Una vez terminado debe finalizar con un carácter especial para hacerle saber al servidor
  - Se cierra conexión **QUIT**
  - Comandos SMTP: **HELO, MAIL FROM, RCPT TO, DATA, QUIT**
- **Formato de mensajes mail**
  - Diferentes de las interacciones SMTP
  - Son las que se escriben en el user agent
  - From:, to:, Asunto:, etc

## 2.4 DNS- Domain Name System

- 2 formas de identificar hosts
  - Hostname. Ej: [www.facebook.com](http://www.facebook.com)
  - IP addresses: número de 4 bytes
    - Son jerárquicas ya que se leen de izq a der y están separadas por puntos. Cada número entre puntos significa algo
- **Servicios que provee DNS:**
  - Es una base de datos gigante distribuida por jerarquías
  - Protocolo de aplicación
  - Utiliza conexiones UDP y el puerto 53
  - A grandes rasgos, DNS convierte hostnames/dominios a IPs
  - **Host aliasing:** muchos Hostnames son muy largos, por lo que si busco una abreviación me debería llevar a la misma página. Estos hostnames están asociados a un **Hostname canónico**
  - **Mail aliasing:** DNS permite que un servidor email sea igual que una servidor web ya que son de distintos tipos que DNS clasifica de formas diferentes
  - **Descentralización:** Muchas páginas web están hosteadas en varios servidores con distintas IPs, el servidor DNS tiene asociado su(s) hostname(s) a todos esos servidores. En una consulta DNS por ese hostname, el servidor DNS envía todas esas IPs al cliente pero modificando el orden de las mismas así descentraliza el tráfico. El cliente por lo general agarra la primera, entonces al enviarlas en distintos órdenes evita congestión.
- **Como funciona DNS a grandes rasgos**
  - Usuario busca una página web
  - El usuario, un cliente DNS, le pasa la request a la aplicación de DNS
  - Cliente DNS envía una request al servidor DNS que tiene el hostname buscado
  - El cliente recibe una eventual respuesta con la IP del hostname
  - Una vez con la IP, se inicializa la conexión entre el usuario y el servidor HTTP

- **Problemas de un servidor DNS centralizado**
  - Si falla el servidor, también fallaría el internet entero
  - No podría manejar todo el tráfico de queries
  - No podría estar cerca de todos, tardarían mucho en llegar las queries
  - Imposible de mantener, no escalable
- **DNS distribuido jerárquicamente**
  - No existe un servidor DNS que tenga todos los mapeos
  - Hay 3 tipos de servidor DNS
    - Root servers
      - Hay 1000 distribuidos por todo el mundo, son copias de 13 originales
    - Top Level Domain(TLD)
      - Hay uno por cada dominio. Ej: .com, .edu, .ar, etc
    - Servidores autoritativos
      - Propios de cada organización: universidades, empresas grandes, gobiernos
      - Los que no tienen uno propio, le pagan a otro
    - Hay un cuarto tipo que son los local DNS, cada ISP tiene uno. Este se encarga de “hablar” con los otros servidores DNS y después te dan la IP que buscabas. También puede cachear estas IPs para futuras consultas. Por último, pueden cachear las IPs de algunos TLD con el fin de saltarse el servidor root



- A medida que vas bajando en la jerarquía de DNS, los servidores saben más acerca de en dónde está el mapeo. El root sabe cual es el TLD correspondiente, que a su vez sabe cual autoritativo tiene el mapeo.
- **Como realmente funciona DNS:**
  - Un host(cliente DNS) busca una página en un browser
  - La aplicación DNS extrae ese dominio y se lo pasa a al local DNS server
    - El DNS local puede tener cacheado el mapeo, en este caso le devuelve la IP y se termina el proceso
  - El local DNS se comunica con el Root server y le envía el dominio en cuestión
  - El root le responde con IPs de TLD servers, que saben más acerca de donde está guardado el mapeo
  - El local DNS se comunica con alguno de los TLD que le pasaron y este le responde con las IPs de servidores autoritativos que tienen el mapeo. (Puede pasar que el TLD

no conozca el autoritativo que tenga el mapeo pero si va a conocer uno intermediario que conozca más)

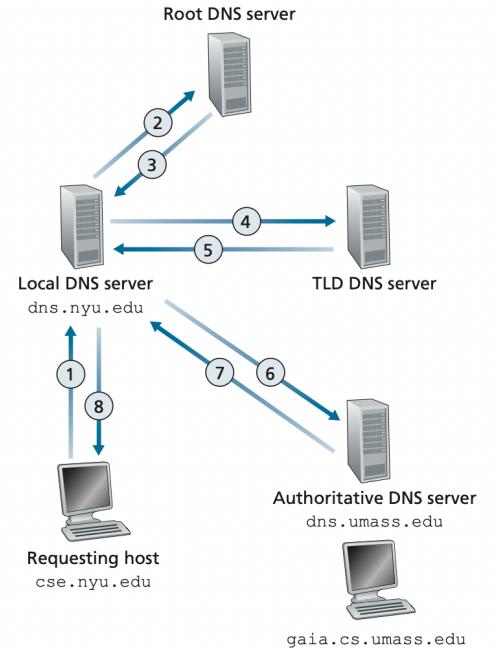
- Una vez conseguida la IP del autoritativo, el local se comunica y el autoritativo le envía la IP correspondiente
- Por último, el servidor DNS local le envía al host la IP en cuestión
- Las consultas pueden ser tanto recursivas como iterativas

- **Mensajes y registros DNS**

- Los servidores DNS almacenan RRs, resource records
- Los RRs son una cuádrupla
  - (Nombre, valor, tipo, TTL)
  - El TTL es el time to live en una caché
- Si es de Tipo = A, el nombre es un dominio y el valor la IP correspondiente
  - Mapean dominios a IPs
- Si es de Tipo = NS, el nombre es un dominio y el valor es un dominio de otro servidor que sabe más acerca del servidor destino
- Si es de Tipo = CNAME, el nombre es un alias y el valor es el nombre canónico
- Si es de Tipo = MX, el nombre es un alias de un servidor de mail y el valor es el nombre canónico del servidor de mail
  - Se puede tener el mismo alias para servidor de mail y servidor común ya que son tipos diferentes

- **Mensajes DNS:**

- Solo hay querys y replies



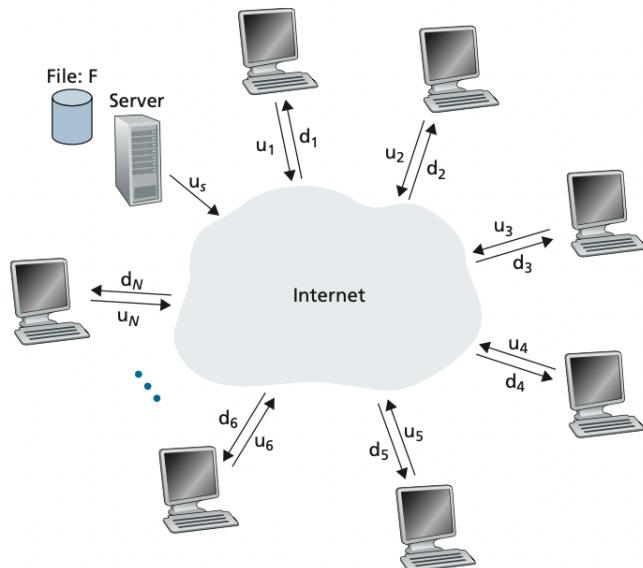
| Identification                                                  | Flags                    | 12 bytes                                   |
|-----------------------------------------------------------------|--------------------------|--------------------------------------------|
| Number of questions                                             | Number of answer RRs     |                                            |
| Number of authority RRs                                         | Number of additional RRs |                                            |
| Questions<br>(variable number of questions)                     |                          | Name, type fields for a query              |
| Answers<br>(variable number of resource records)                |                          | RRs in response to query                   |
| Authority<br>(variable number of resource records)              |                          | Records for authoritative servers          |
| Additional information<br>(variable number of resource records) |                          | Additional "helpful" info that may be used |

- Identificación: número para que el host asocie la query con la reply

- Flags: indican si es query o reply, sí quiero recursión o no, si el server soporta recursión
- Question: el dominio que está buscando su IP, el tipo de query que es: A, NS, MX, CName
- Answer: Respuesta de servidor, pueden ser más de una. Me puede devolver 10 IPs de servidores que estén asociados a mi query
- Additional: Si hice una query MX, en esta sección me podrían pasar la IP del servidor canónico de mail

## 2.5 Peer-To-Peer

- Distribución de archivos entre pares
  - Casi que no requiere mantenimiento ni confianza en unos pocos servidores
  - Autoescalable
  - A cada Peer se le manda una fracción de un archivo para que distribuya
- **Auto Escalabilidad de P2P:**



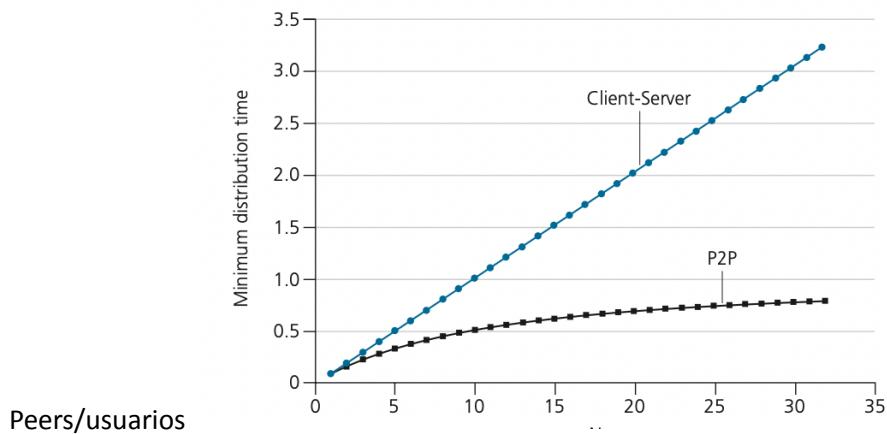
- Comparada con Cliente-Servidor
- Se quiere transmitir un archivo de  $F$  bits a  $N$  Peers,  $u_s$  es la tasa de subida del server,  $u_i$  tasa de subida de los Peers,  $d_i$  tasa de descarga de Peers
- Si usamos una arquitectura **Cliente-servidor** el tiempo que tardaría que todos los Peers tengan el archivo es:
  - El servidor tiene que mandar  $N \cdot F$  bits a una tasa  $u_s$  y los Peers descargan  $F$  bits a una tasa  $d_i$ . Por lo que el tiempo total debe ser igual al máximo entre  $N \cdot F / u_s$  y  $F / d_{\min}$

$$D_{cs} = \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{\min}} \right\}$$

- Notemos que a medida que aumenta N, el tiempo total va a ser mayor ya que con N grande  $N*F/U_s > F/D_{min}$
- Si utilizamos la arquitectura **P2P** el tiempo que tardaría es el siguiente:

$$D_{P2P} = \max \left\{ \frac{F}{U_s}, \frac{F}{d_{min}}, \frac{NF}{U_s + \sum_{i=1}^N U_i} \right\}$$

- El servidor tiene que mandar una vez el archivo como mínimo:  $F/U_s$
- El peer con menor tasa de descarga tiene que poder descargar el archivo:  $F/D_{min}$
- Por último, se tienen que transmitir  $N*F$  bits para que todos puedan descargarlo, pero a diferencia de CS, se divide por la suma de todas las tasas de carga de los Peers
- Aca podemos ver como escala P2P a diferencia de CS a medida que aumentan los



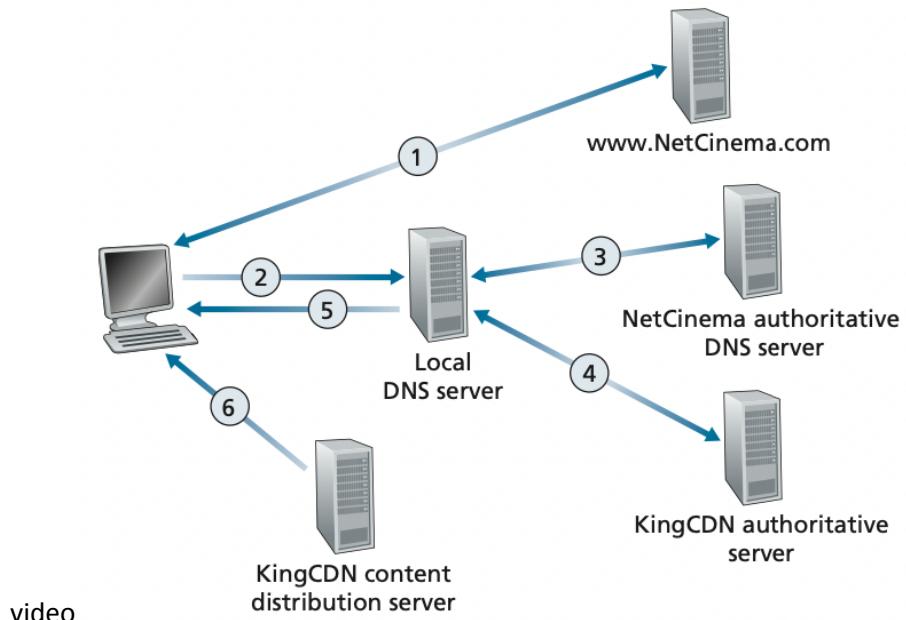
- **BitTorrent:**
  - Conjunto de peers intercambiando chunks se los llama un torrent
    - Descargan y envían chunks de 256 Kbytes
  - Cuando un nuevo peer se une, no tiene chunks para enviar. Pero con el paso del tiempo los va recibiendo. Al unirse, debe registrarse en un **tracker**.
    - El tracker tiene registro de todos los peers que están en un torrent
  - Una vez ya recibido tu archivo, puedes abandonar el torrent(egoísta) o seguir descargando y enviando chunks a otros peer
  - Cuando te conectas, el tracker te envía una lista de 50 peers vecinos con sus respectivas IPs e intenta establecer una conexión TCP con cada uno de ellos. Con los que se logra establecer la conexión, se los llama peers vecinos.
  - Periódicamente los peers se envían listas de todos los chunks que tienen
  - Con esta información, los peers van a definir qué chunks pedir y a quién tiene que mandarle chunks requeridos

- Para saber que chunks pedir, BitTorrent aplica un técnica llamada **rarest first**, en donde los peers piden los chunks que menos aparecen en el torrent así hay una distribución equitativa de los chunks.
  - Para determinar con qué peers intercambiar chunks, se seleccionan a los 4 peers que mayor rate de intercambio tengan con uno. Cada 10 segs se recalcula esto y se pueden modificar los 4. Con los demás peers vecinos no intercambia nada
  - Otra técnica que utiliza es la llamada **optimistically unchoked**, que selecciona un peer al azar cada 30 segs y le empieza a mandar chunks. Puede pasar que entre en el top 4 de peers y viceversa. Esto permite la incorporación de nuevos peers al torrent.
- Tit-for-tat**

## 2.6 Video Streaming and Content Distribution Networks

- Según varias estimaciones, las plataformas de streaming de video como Netflix, Amazon Prime y YouTube acumulan el 80% del tráfico de Internet
- **Internet Video**
  - Contenido ya grabado y almacenado en servidores. Usuarios mandan requests a estos últimos para verlos *on demand*.
  - Un video es una secuencia de imágenes mostradas a un rate constante de 24/30 imágenes por segundo. Una imagen es un array de píxeles y cada píxel es un conjunto de bits que representan color y luminosidad.
  - Los videos pueden ser comprimidos a bits y ser enviados a diferentes rates por lo que facilita mucho su transmisión. A mayor bit rate, mejor la calidad del video
  - La red debe proveer un throughput promedio mínimo para que el usuario pueda ver el video al bit rate deseado.
  - Al enviar todo comprimido, el usuario puede elegir la calidad a la que quiere ver dependiendo del ancho de banda que disponga
- **HTTP Streaming and DASH**
  - En streaming HTTP, el video está guardado en un servidor HTTP como un archivo ordinario
  - Cuando el usuario quiere acceder a él, hace una GET request y el servidor comienza a mandarle el video vía TCP.
  - Una aplicación del cliente va a ir almacenando partes del video en buffers y una vez que supere un umbral va a comenzar a “correr” el video. Los buffers todo el tiempo van a estar guardando partes futuras del video para mostrarlas después.
  - Hoy día, se utiliza otro protocolo de streaming HTTP llamado **DASH, Dynamic Adaptive Streaming over HTTP**. Este consiste en que el cliente va a ir cambiando la calidad de los pedidos de chunks de video dependiendo del ancho de banda que disponga en el momento.
  - El servidor HTTP tiene un **archivo manifiesto** que contiene todas las URLs del video con diferente calidad

- Te permite ver un video de corrido sin que se corte pero va fluctuando la calidad dependiendo de tu ancho de banda
- **Content Distribution Networks**
  - Una compañía que provee video por Internet se enfrenta a varios problemas a la hora de compartir los videos con gente de todo el mundo
    - Clientes que se encuentren lejos van a tener mucho delay
    - Probablemente un mismo video se envie varias veces para un mismo lugar, enviarlo todas esas veces desde el data center principal provocaría en un gasto enorme de ancho de banda
    - Si se rompe el data center, no se pueden transmitir los videos
  - Para solucionar todo esto, se implementan **Content Distribution Networks (CDNs)**
    - Pueden ser privados(Google) o de terceros (Akamai)
    - Estos estan distribuidos en múltiples partes del mundo y guardan en sus servidores copias de los videos
    - Dos filosofías a la hora de placear los servidores
      - Enter Deep: Ponen los servidores en los ISPs
        - Más caro pero más rápido
      - Bring Home: Ponen los servidores en los IXPs
    - Funciona de manera similar a una caché web
    - Como CDNs se interpolan con DNS
      - Usuario cliquea en un link de video
      - Se envía una query DNS por medio del Local DNS a un DNS autoritativo que le devuelve el dominio un CDN
      - Se le envía una query al CDN, que tiene su propia infraestructura de DNS y le devuelve la IP del CDN que eventualmente le va a enviar el video
      - Se establece conexión TCP con este último y comienza la transmisión de



- El CDN tiene una estrategia para determinar cuál es el cluster CDN que más rápido pueda enviarle el video al cliente. Todo el tiempo están haciendo mediciones con el Ping y cuando les llega una query saben a qué cluster direccionarlo

## Capítulo 3 - Capa de transporte

- Fundamental para conectar la capa de aplicación con la de red

### 3.1 Introduction and Transport-Layer Services

- Provee conexión lógica entre 2 procesos de aplicación corriendo en diferentes hosts
  - “Llevarle la carta al cartero”
  - No se encarga físicamente de llevarlo
- Los protocolos de transporte se implementan en end systems, no en routers
- A grandes rasgos, la capa de transporte desde el punto de vista del emisor, recibe un mensaje de la capa de aplicación, le mete un header y genera un segmento. Por último, lo pasa a la capa de red que se ocupa de las demás cosas. Le llegan varios mensajes de app y los multiplexa en uno solo.
- El receptor recibe el segmento, verifica el header, extrae el mensaje y lo multiplexa para enviarlo a las diferentes aplicaciones

### Relación entre la capa de transporte y la de aplicación

- Ejemplo de las cartas entre 2 casas
  - Mensajes de app = cartas
  - Procesos = los que las escriben
  - Hosts/end-systems = las casas
  - Protocolo de transporte = Los que llevan las cartas de la casa al cartero y los que las retiran
  - Protocolo de red = Servicio postal
- El protocolo de transporte únicamente le agrega un header al mensaje y se lo pasa al protocolo de red
- Muchas veces, los protocolos de transporte no pueden garantizar algunos servicios como delay máximo porque los protocolos de red no proveen muchas cosas. Pero en otros casos sí pueden garantizar otros servicios aunque los de red no provea, como por ejemplo la transferencia confiable de datos

### Mirada general sobre los protocolos de transporte

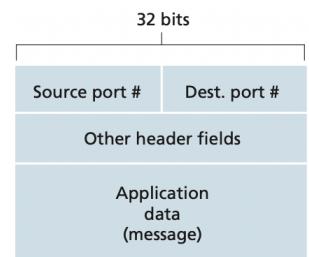
- 2 protocolos de transporte
  - UDP (User Datagram Protocol)
    - No confiable y no requiere conexión entre los 2 hosts
    - Se le pueden agregar funcionalidades en la capa de app para hacerlo más seguro
  - TCP (Transmission Control Protocol)
    - Confiable y requiere conexión entre hosts(handshake). Más complejo

- Provee control de flujo y de congestión
- IP, internet protocol. Es un protocolo de red con un servicio de tipo best effort
  - No garantiza la llegada del paquete, ni que llegue en orden ni la integridad del mismo
  - Por esas razones, es un servicio no confiable
- Los protocolos de transporte tienen que extender el servicio de red hacia los hosts mediante la multiplexación y la demultiplexación

### 3.2 Multiplexación y demultiplexación

- La capa de transporte, en el lado del receptor, recibe segmentos de la capa de red, los demultiplexa y los entrega a la aplicación correspondiente. No se lo pasa directamente a la app sino que pasa por medio de un socket que tiene un número de puerto específico. Si no se mezclarán todos los mensajes. Desde el punto del emisor, la capa de transporte recibe muchos mensajes de las sockets y tiene que juntarlos y crear el segmento. Esto se llama multiplexar.
- Cada socket debe tener un puerto único
- Cada segmento debe tener escrito el puerto de destino y el puerto de origen

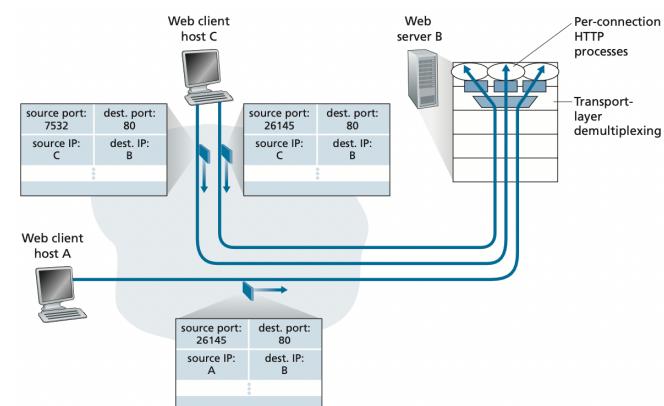
#### Connectionless Multiplexing and Demultiplexing (UDP)



- Como UDP funciona como best effort, cuando el receptor recibe un segmento tiene que analizarlo, demultiplexar y enviarlo al socket correspondiente que debe estar especificado en el mismo segmento
- Puede pasar que 2 hosts diferentes están enviando segmentos al mismo host y al mismo socket específico, es por esto que se necesita especificar el puerto de origen para poder enviar una eventual respuesta
- 2 segmentos con diferente IP y/o puerto de origen pero con mismo IP y puerto de destino van a pasar por el mismo socket
- UDP usa sockets con una tupla como identificador que tienen la IP y el puerto de destino

#### Connection-Oriented Multiplexing and Demultiplexing (TCP)

- TCP usa sockets con una cuatrupla como identificador que tienen IP y puerto de origen y IP y puerto de destino
  - Por lo general, el puerto de origen se elige al azar entre los puertos no reservados, del 1200 en adelante
- A diferencia de UDP, 2 segmentos con diferentes IP y/o puerto de origen pero con misma IP y puerto de destino van a diferentes sockets
- El servidor tiene un “socket de bienvenida” que espera a conexiones TCP en un puerto específico, el cliente TCP crea un socket con el nombre del servidor y el puerto, se establece



conexión activando un flag(bit) en el segmento enviado, el servidor acepta la conexión y se fija en el puerto e IP de origen y en el puerto e IP de destino(las suyas). Queda identificada la conexión y todos los segmentos que tengan esos 4 valores van a ser demultiplexados a ese socket. Ya queda habilitada la conexión para pedir datos.

- El servidor tiene que ser capaz de soportar conexiones TCPs simultáneas e ir creando diferentes sockets para cada una

### 3.3 Connectionless Transport: UDP

- Provee los servicios mínimos e indispensables
  - Multiplexación y demultiplexación
  - Mínimo chequeo de errores
- Es casi como ir directo a IP, no le agrega nada de valor
- Toma el paquete de aplicación, le agrega puerto de origen y destino para la multi y demulti, le agrega la longitud, el checksum y pasa la todo eso (el segmento) a la capa de red. Luego, esta última hace su mejor esfuerzo por entregar el paquete.
- Si el paquete llega al destino, lo demultiplexa con el puerto de destino y lo envía al socket correcto
- No hay handshake, por eso de lo llama connectionless
- DNS utiliza UDP, si el host que hizo la query no recibe respuesta luego de un tiempo, repite la query
- Porque las aplicaciones elegirían UDP antes que TCP sabiendo que no es confiable?
  - La capa de aplicación se puede ocupar de garantizar la confiabilidad de la data, sacándole trabajo a la capa de transporte y por ende menor delay. TCP hace un control de congestión y se ocupa de garantizar la llegada de cada paquete con diferentes métodos, haciendo que la comunicación sea más lenta. **Más rápido**
  - No se establece conexión con el otro host. Nuevamente podemos ver cómo agiliza los tiempos de conexión y envío de data entre 2 hosts. **Más rápido**
  - No mantiene registro de las conexiones a diferencia de TCP. **Menor uso de recursos, soporta más conexiones**
  - UDP usa un header más chico y con mucha menos data que el header de TCP. 8 bytes contra 20
  - El que usa UDP es porque requiere de conexiones más rápidas y baratas pero está dispuesto a que se pueda perder una parte del msj, que lleguen en desorden entre otras cosas
- Que protocolo usa cada aplicación
- UDP no provee control de congestión, pero este es requerido para evitar saturaciones en la red
- Si muchas personas comienzan a descargar video con UDP, se saturaría la red ya que no hay control de congestión y el número de paquetes que lleguen correctamente a su destino será

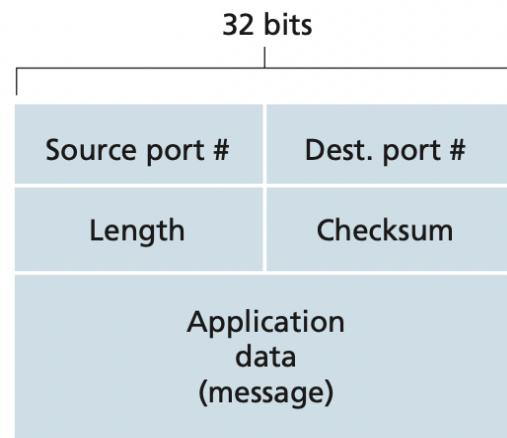
| Application                   | Application-Layer Protocol | Underlying Transport Protocol    |
|-------------------------------|----------------------------|----------------------------------|
| Electronic mail               | SMTP                       | TCP                              |
| Remote terminal access        | Telnet                     | TCP                              |
| Secure remote terminal access | SSH                        | TCP                              |
| Web                           | HTTP, HTTP/3               | TCP (for HTTP), UDP (for HTTP/3) |
| File transfer                 | FTP                        | TCP                              |
| Remote file server            | NFS                        | Typically UDP                    |
| Streaming multimedia          | DASH                       | TCP                              |
| Internet telephony            | typically proprietary      | UDP or TCP                       |
| Network management            | SNMP                       | Typically UDP                    |
| Name translation              | DNS                        | Typically UDP                    |

muy bajo. A su vez, esto provocará que los TCP senders reduzcan su tasa de transmisión por la saturación de la red.

- Es posible que una aplicación envíe datos confiables vía UDP, agregando algunos cheques y controles en la capa de aplicación. Un ejemplo es QUIC. Esto permite que la aplicación envíe datos confiables sin estar restringido a los tiempos que impone TCP por los controles de congestión

#### UDP Segment Structure

- Cada campo tiene 2 bytes
- El número de puerto sirve para saber a qué socket enviar el mensaje
- La longitud es la suma del header + el tamaño del mensaje



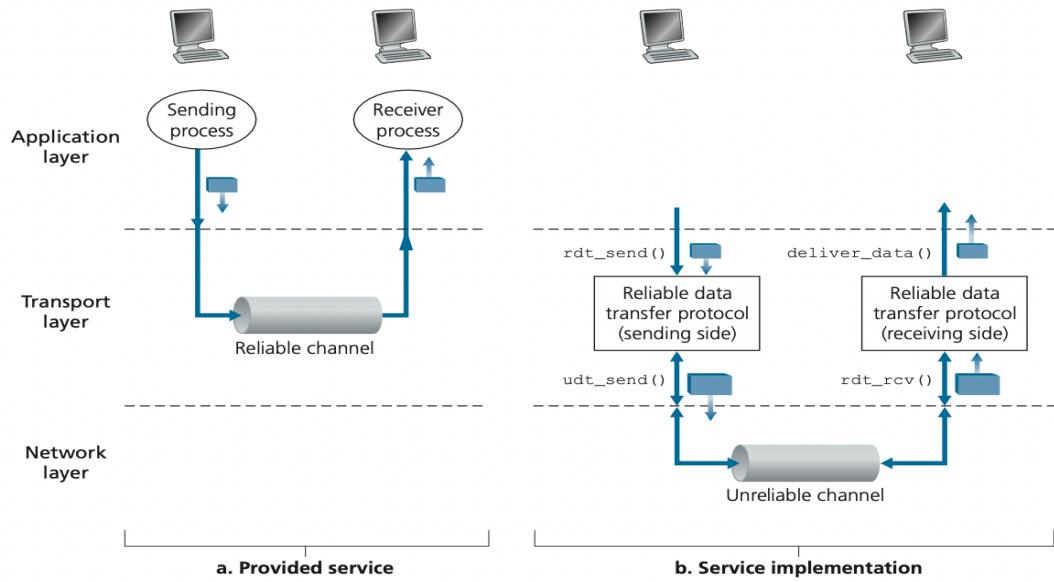
#### UDP Checksum

- Es usado para la detección de errores, se fija si el contenido fue alterado en el camino
- Es la suma de todo el segmento y lo transforma a complemento a 2 (negativo)
- Al recibir el paquete, el receptor suma todos los campos y si todo está correcto y el mensaje no se rompió ni modificó, la suma le debería dar 0
- En caso de detectar un error no hace nada para remediarlo, únicamente puede:
  - Descartar el mensaje
  - Pasar el mensaje al socket con un warning
- No es muy confiable este método para la detección de errores. La suma puede dar bien aunque se hayan modificado los datos.  $01 + 10 = 11$  pero  $10 + 01 = 11$  por lo que los datos pueden estar completamente al revés o mal e igualmente con el checksum no se va a detectar

#### 3.4 Principios de la transferencia de datos confiables

- En esta sección se aborda el problema de la transferencias de datos confiables visto desde un panorama general para luego aplicarlo a cómo funciona realmente TCP

- ¿Cómo garantizar confiabilidad si la capa de red no lo es?

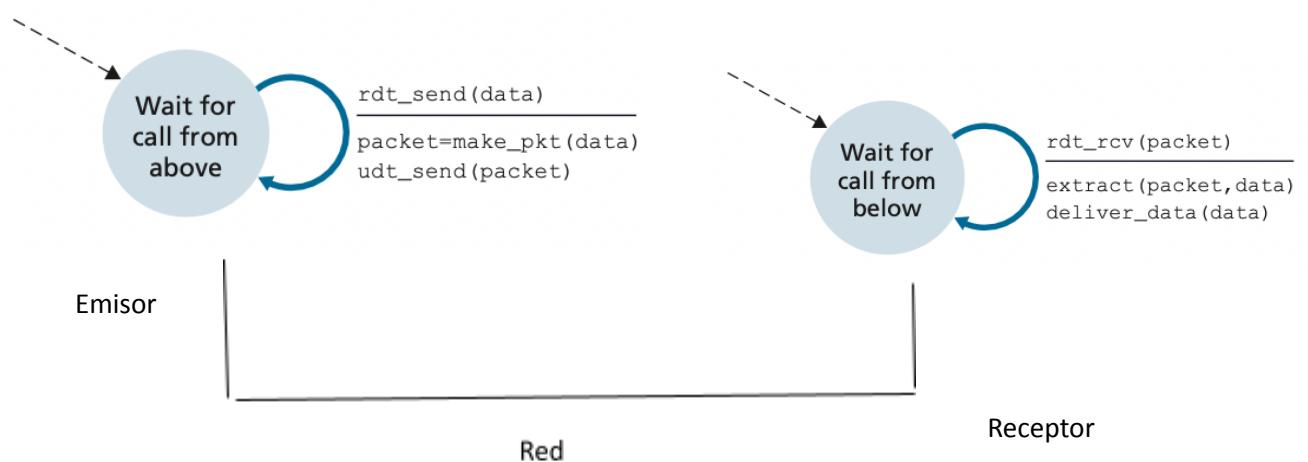


- Vamos a asumir que no se desordenan los paquetes entre sí pero que sí pueden corromperse o perderse
- Funciones que vamos a utilizar a lo largo del capítulo
  - `rdt_send()`: a nivel app para enviar datos (Reliable data transfer)
  - `udt_send()`: enviar data por canal no confiable (Unreliable data transfer)
  - `rdt_rcv()`: invocada cuando llega el paquete
  - `deliver_data()`: invocada para enviar la data al socket correspondiente

### Creando un protocolo de transferencia de datos confiable con máquinas de estado finitas

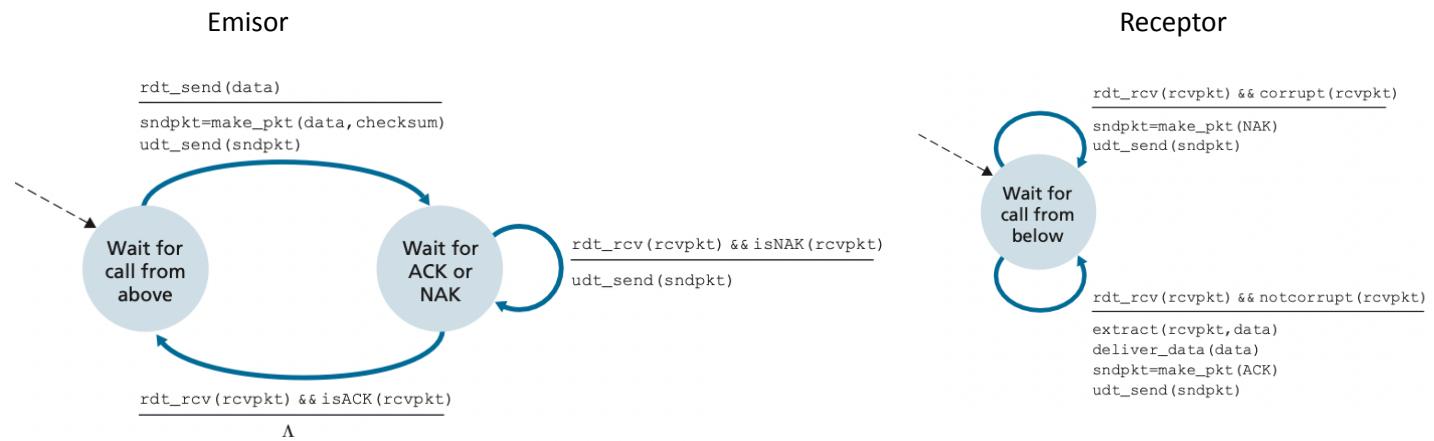
#### Protocolo para un canal perfectamente confiable: rdt1.0

- No hay pérdida de paquetes ni errores de bits



### Protocolo para un canal con errores de bits: rdt2.0

- Los errores se detectan con un checksum
  - Si el paquete llegó bien el receptor envía un ACK (acknowledge)
  - Caso contrario envía un NAK (negative acknowledge) y el emisor retransmite el paquete
- Asumimos que los paquetes llegan

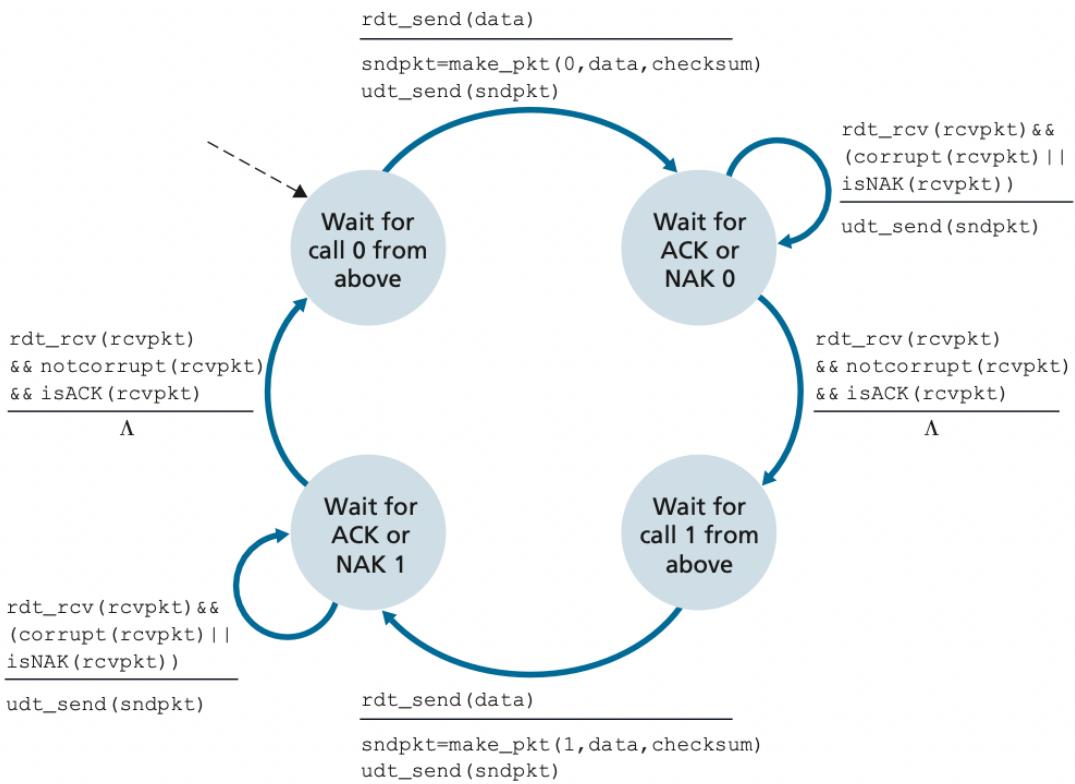


- Una vez que el emisor envía el paquete con la data y el checksum, cambia de estado a esperar ACK o NAK. Una vez en este estado únicamente puede esperar la respuesta, no puede enviar más datos. Es un protocolo de **stop and wait**. Recién va a poder enviar data nueva si le llega un ACK del receptor o una retransmisión si recibe un NAK.
- El receptor una vez que recibe el paquete, verifica si está corrupto o no con el checksum. Si está corrupto envía un NAK y si no estaba corrupto envía un ACK y le pasa la data a la capa de aplicación. En ambos casos, luego de mandar el respectivo mensaje se queda esperando que le lleguen nuevos paquetes.
- Tiene un gran problema, no contempla los ACKs y NAKs corruptos. Puede producir duplicados en caso de que el paquete llegue bien, el receptor envía un ACK pero el receptor recibe un NACK. Para resolver esto se aplican checksums para ACKs y NAKs y números de secuencia a los paquetes

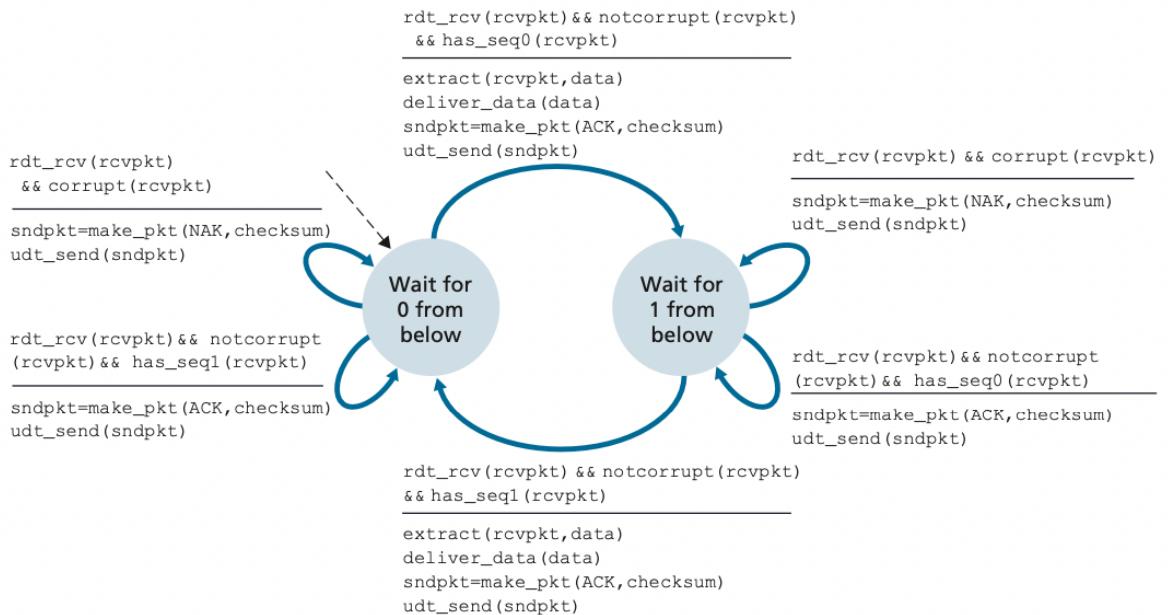
### rdt2.1 soluciona el problema de los ACKs y NAKs corruptos

- Secuencia binariamente los paquetes y le agrega un checksum a los ACKs/NAKs

- Emisor



- Receptor



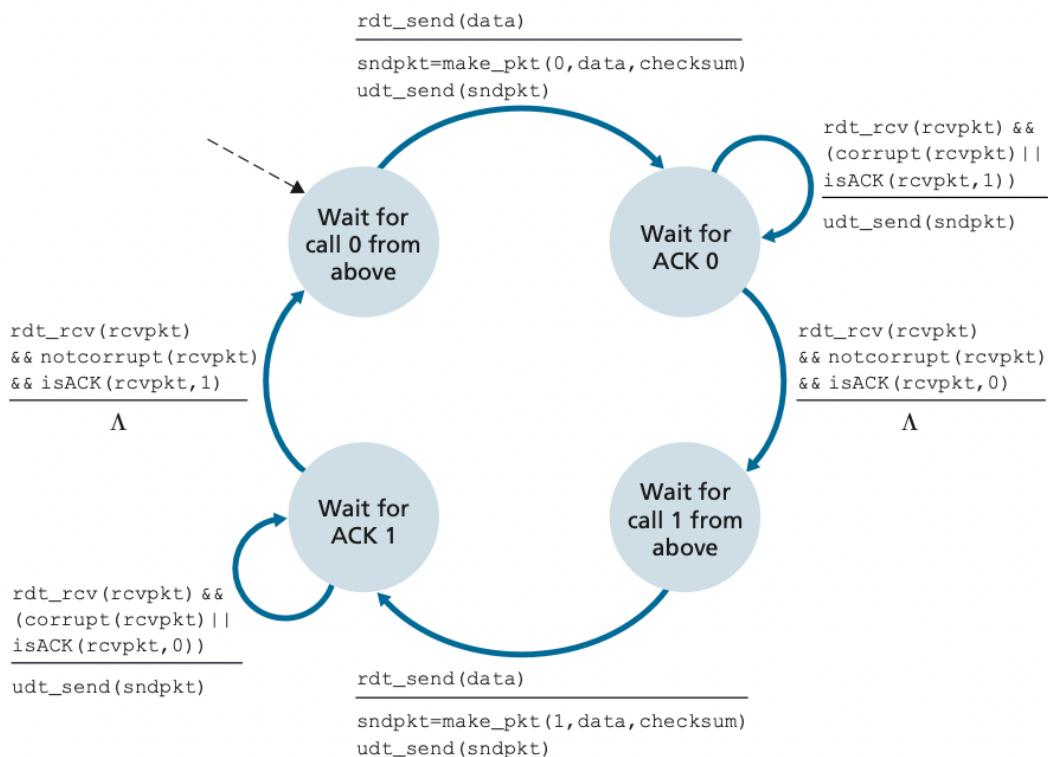
- Un caso interesante en este protocolo es cuando el emisor manda un paquete 0. El receptor estaba esperando el 0, lo recibe correctamente, le pasa la data al socket, envía el ACK y pasa a estar esperando el paquete 1. Ese ACK que envía se corrompe por lo que el emisor va a reenviar el paquete 0 cuando el receptor está esperando el 1. Cuando el emisor recibe el 0, no corrupto pero no coincide el número de

secuencia, envía un ACK directamente ya que ese paquete ya lo tiene. Esto va a pasar hasta que el emisor reciba correctamente el ACK.

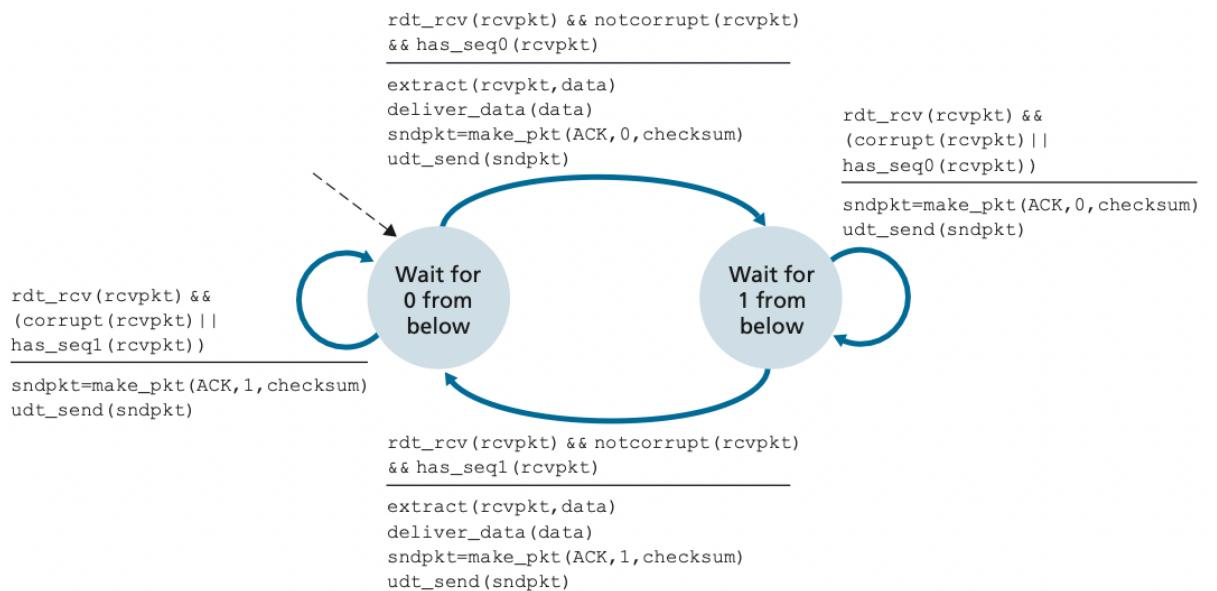
- Breves conclusiones
  - Solo 2 números de secuencia (Emisor)
  - Se debe comprobar corrupción de ACKs y NAKs (Emisor)
  - Se agregan 2 estados más (Emisor)
  - Se debe comprobar si el paquete es duplicado, fijándose si el paquete que llega es el que estaba esperando o no (Receptor)
  - El receptor no sabe si el ACK/NAK llegó correctamente

### rdt2.2 eliminando los NAKs

- El receptor únicamente envía ACKs con el número de secuencia del último paquete que recibió correctamente
- Ejemplo: Si estaba esperando el paquete 1, le llega pero corrupto, va a enviar un ACK 0. Haciendo referencia a que el último paquete correcto que recibió fue el 0 y que el 1 le llegó mal
- Un ACK duplicado en el emisor significa lo mismo que un NAK, debe retransmitir
- Simplifica un poco los estados
- Emisor



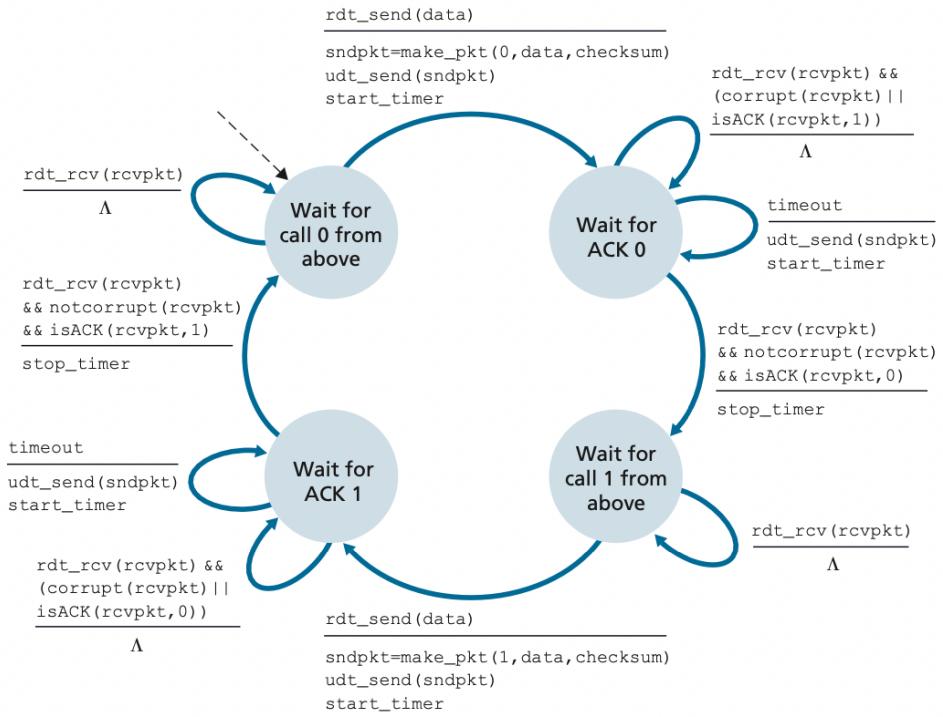
- Receptor



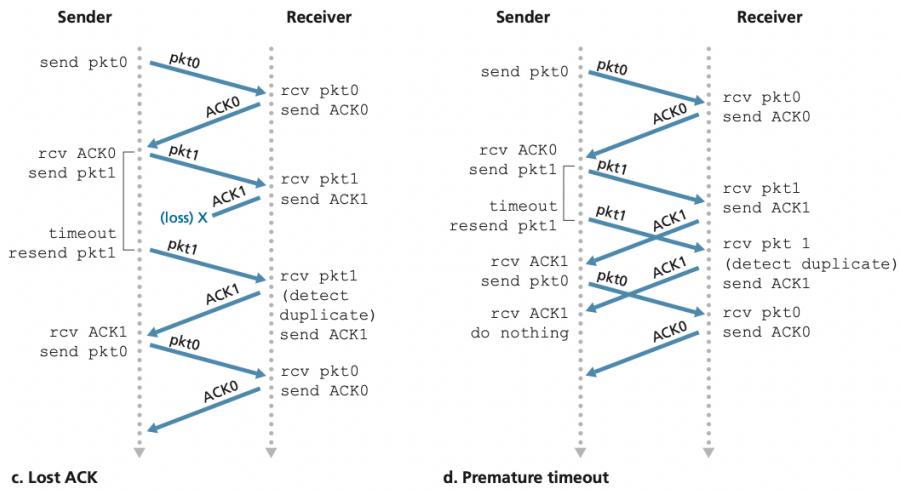
Protocolo de transferencia de datos confiable con pérdida de paquetes y errores en los bits:  
rdt3.0

- Se pueden perder tanto los datos como los ACKs
- No alcanza únicamente con checksums, ACKs y números de secuencia
- Se necesita un timer y luego de cierto tiempo sin recibir respuesta se retransmite el paquete
- Podría pasar que haya una demora grande en el ACK y que el emisor retransmite el paquete generando un duplicado pero eso se soluciona con los números de secuencia
- Si el emisor tiene que retransmitir porque se llegó a un timeout, nunca va a saber si fue porque se perdieron los datos, el ACK o si el ACK se demoró mucho
- El timer lo resetea cada vez que envía un paquete nuevo o cada vez que retransmite un paquete
- Funciona igual que el RDT 2.2 con la única diferencia que le agrega el timer
- Es conocido como un protocolo de alternancia de bit ya que solo tiene 2 números de secuencia, 0 y 1

- Emisor



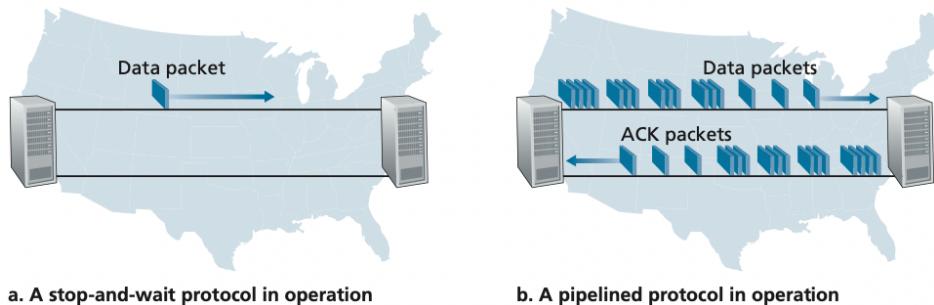
- Receptor es igual que rdt2.2
- Algunos posibles desenlaces



### Pipelined Reliable Data Transfer Protocols

- Rdt3.0 tiene el problema de ser un protocolo de Stop And Wait, se pierde mucho tiempo con este tipo de protocolos
- El protocolo limita las capacidades

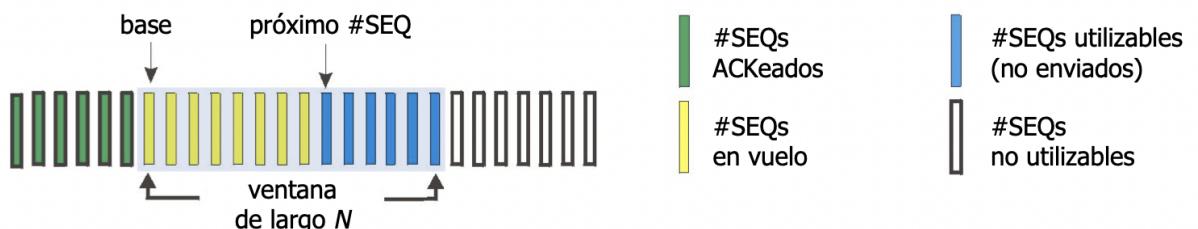
- Se desperdicia mucho ancho de banda



- En stop and wait, la utilización del canal,  $U = (L/R) / (RTT + L/R)$ 
  - $L/R$  = delay de transmisión
  - RTT = round trip time
- Para solucionar el problema, se pueden mandar paquetes en simultáneo sin tener que esperar la confirmación de llegada de cada paquete
- Se debe aumentar la cantidad de números de secuencia, cada paquete debe tener uno único
- Se necesita que tanto el emisor como el receptor tengan buffers disponibles para ir almacenando los paquetes
- Dos protocolos principales modelan este servicio: **Go-Back-N** y **selective repeat**

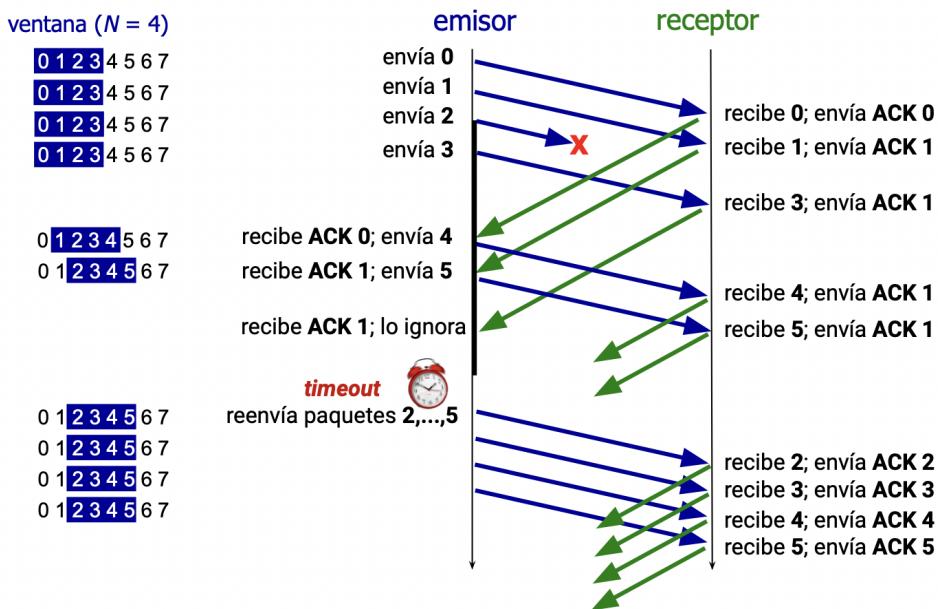
### Go-Back-N (GBN)

- El emisor puede enviar como máximo N paquetes sin esperar por su acknowledgment.
- Se define una base, el número de secuencia del paquete más viejo que no fue ACKeado. También el próximo paquete en ser enviado
- El número de secuencia es un número de K bits en el header del paquete



- A medida que los ACKs van llegando, la ventana se va moviendo hacia la derecha habilitando nuevos números de secuencia
- Utiliza ACKs acumulativos. Si le llega el ACK 5 pero aún no recibe el ACK 4, es tomado como que el paquete 4 le llegó correctamente al receptor ya que este nunca hubiera ACKeado el 5 si no ACKeo el 4
- Cuando la capa de aplicación quiere enviar datos, el protocolo debe fijarse si la ventana está llena o no. Si no está llena, crea un paquete con el número de secuencia correspondiente y lo envía, caso contrario devuelve el mensaje
- Es necesario el uso de uno o varios timers. El timer se inicia cuando se envía un paquete base, en caso de time out se retransmiten todos los paquetes de la ventana.
- Si se recibe un ACK del paquete base, pero sigue habiendo paquete sin ACKear, el timer se reinicia

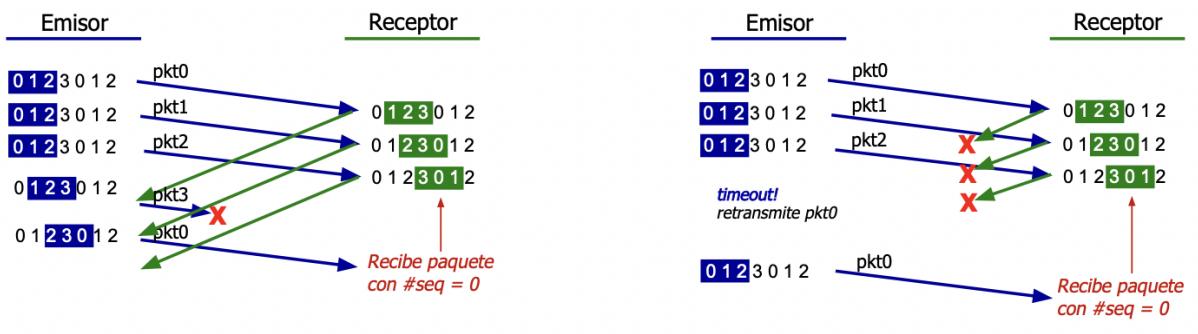
- El receptor siempre envía un ACK para el paquete con el #SEQ más alto (en orden) recibido correctamente
  - Puede generar ACKs duplicados, en caso de que se pierda algún paquete y le llegue uno mayor
  - Debe recordar el #Seq que espera recibir a continuación
  - Cuando recibe un paquete fuera de orden lo descarta y debe enviar un ACK para el paquete con #Seq más alto recibido
  - También se podría buffear el paquete en desorden pero como GBN lo va a retransmitir de todas formas es innecesario buffear
- Ejemplo de cómo funciona



### Selective Repeat (SR)

- En algunos escenarios, GBN sufre de grandes delays. Cuando se usa una ventana muy grande y se pierde uno de los primeros paquetes de la venta, se deben retransmitir todos los siguientes de nuevo
- Selective Repeat evita retransmisiones innecesarias, manteniendo un registro individual de cada ACK recibido. Puede tener ACKeado el paquete  $n+1$  sin tener ACKeado el  $n$
- El emisor inicializa un timer por cada paquete que no fue ACKeado
- El receptor va a responder un ACK por cada paquete que llegue no corrupto sin importar si está en el orden correcto
- En caso de que no esté en orden lo bufferea hasta que se acomoden
- Acciones del emisor
  - Si le piden enviar un paquete y el próximo #seq está disponible, lo manda
  - Timeout. Tiene un timer por cada paquete enviado así poder transmitir únicamente el paquete perdido
  - Reconoce individualmente los ACKs. Si el ACK recibido es el de la base, mueve la ventana hacia la derecha
- Acciones del receptor

- Al recibir el paquete n correctamente, manda un ACK n y dependiendo si está en orden o no lo manda a aplicación o lo bufferea
- Si recibe un paquete que no pertenece a la ventana [Base; Base + N-1] lo descarta
- Si recibe un paquete que pertenece a [Base; Base + N-1] pero que ya lo recibió previamente, debe ACKearlo de nuevo porque significa que el ACK se perdió en el camino y el emisor no puede mover la ventana
- El emisor y el receptor no siempre van a tener las mismas ventanas. Problema de sincronización. Si los números de secuencia se repiten, el receptor puede no distinguir entre si el paquete que llegó es una retransmisión de un paquete ya ACKeado pero que se perdió el ACK o un paquete nuevo
- Es necesario que el tamaño de la ventana sea la mitad o menos de la cantidad de números de seq para evitar el problema de la repetición
- Para el receptor ambos casos son iguales



- Cuando se llega al quinto paquete, que tiene #Seq 0 el receptor no tiene forma de saber si se trata de una retransmisión o un paquete nuevo
- Al comienzo de la sección, asumimos que los paquetes no pueden ser reordenados en el camino. Tiene sentido si A y B están unidos por un cable físico. Pero si lo que los une es la red pierde el sentido. Podría pasar que al receptor le llegue un paquete antiguo que ni él ni el emisor están esperando que le llegue (ya fue retransmitido). Esto podría generar un problema de duplicados ya que el receptor podría buffearlo como si fuese un paquete futuro.
- Para resolver esto, no se pueden repetir los números de secuencia hasta estar seguros de que el paquete le llegó o que eventualmente desapareció de la red. En TCP se toma un tiempo de 3 minutos para decir que el paquete no existe más.

### 3.5 Connection-Oriented Transport: TCP (Transmission Control Protocol)

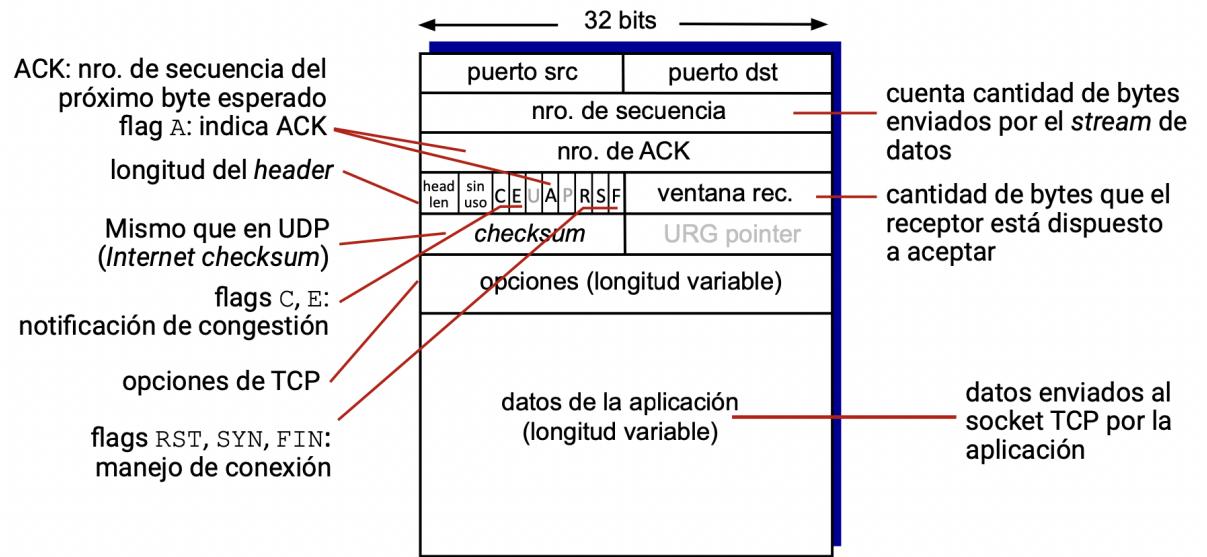
#### Conexión TCP

- Es un protocolo orientado a la conexión ya que antes de que se empiecen a enviar datos debe haber un “handshake” entre ambas partes donde se presetean e inicializan variables y otras cosas
- Conexión lógica entre dos end systems. Point-to-point

- Provee un servicio **full-duplex**, A le puede mandar cosas a B y viceversa. Flujo de datos bidireccional
- Cuando un cliente TCP quiere enviar datos a otro, primero debe establecer una conexión de 3 pasos. Mientras se establece la conexión, el cliente TCP va almacenando la data a enviar en buffers y luego de establecer la conexión se procede a colocarle el respectivo header, formando un segmento TCP y luego enviarla a la capa de red

### Estructura del segmento TCP

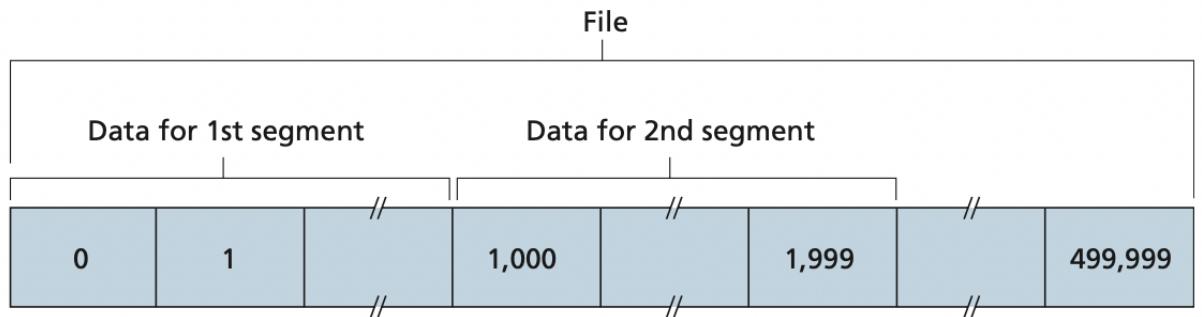
- MSS = maximum size of a segment
- Cuando se manda un archivo grande, se divide en varios segmentos de MSS tamaño
- El header de TCP tiene 20 bytes a diferencia del de UDP que tiene 8
- El header TCP contiene **puerto de origen y destino**, un **checksum**. Al igual que UDP
- También tiene un campo 32 bits(4 bytes) del **número de secuencia**, otro campo de 32 bits del **número del ACK**. Un campo de 16 bits de la **ventana del receptor**, cuantos bytes el receptor está dispuesto a recibir. Un campo de 4 bits con el **tamaño del header**. Un **campo opcional** con longitud variable y una serie de flags de 6 bits en total. El bit ACK que indica la validez. **RST**, **SYN** y **FIN** son los flags utilizados para el manejo de conexión y **CWR** y **ECE** para el manejo de congestión



### Números de secuencia y de ACKs

- Esenciales para la transferencia confiable de datos
- El número de secuencia es el índice en el stream de bytes del primer byte de los datos de app enviados

- El número de secuencia inicial es elegido al azar para minimizar la posibilidad de repeticiones



- Nro de seq 0 nro de seq 1000
- El número de ACK que A pone en el segmento es el número de secuencia del próximo byte que espera recibir de B
  - Son acumulativos
  - El manejo de segmentos fuera de orden queda a discreción de la implementación, no hay ninguna regla

### Round-Trip Time Estimation and Timeout

- No es fácil poner un tiempo de timeout correcto. Si es muy corto, van a haber retransmisiones innecesarias y si es muy largo se va a desperdiciar tiempo
- SampleRTT es el tiempo transcurrido entre la transmisión del segmento y la recepción de ACK. No tiene en cuenta retransmisiones. Va fluctuando segmento a segmento.
- Se utiliza la siguiente fórmula para medir el Estimated RTT
  - Estimated RTT =  $(1 - \alpha) * \text{Estimated RTT} + \alpha * \text{SampleRTT}$
  - Es un promedio ponderado entre el promedio viejo y la última medición
  - Se suele utilizar  $\alpha = 0.125$  dándole más importancia al promedio viejo que la última medición
  - El **timeout (RTO)** se calcula:  $\text{RTO} = \text{Estimated RTT} + 4 \text{ DevRTT}$
  - DevRTT vendría a ser como un desvío del Estimated RTT y se lo utiliza como un margen de seguridad para cuando hay fluctuaciones en el muestreo. Se calcula con la siguiente fórmula
    - $\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$
    - Con  $\beta = 0.25$

### Transferencia de datos confiable

- TCP planta un sistema de confiabilidad sobre la capa de red/IP no confiable
- TCP garantiza que la data va a llegar no modificada, sin duplicarse, sin espacios y en orden
- 3 eventos principales desde el punto de vista del emisor
  1. Recepción de datos de app
    - a. Genera un segmento e inicializa el timer si no está corriendo
  2. Expira el timer
    - a. Retransmite segmento con menor número de secuencia no ACKeado
    - b. Reinicia el timer
  3. ACK recibido

- a. Actualiza ventana
  - b. Reinicia timer
- TCP utiliza un único timer que está asociado al segmento con menor número de secuencia no ACKeado
  - Frente a un timeout, retransmite el paquete descrito en el renglón de arriba
- Cuando ocurre un timeout para cierto ACK, se duplica el RTO para la retransmisión pero no se tiene en cuenta para calcular futuros RTOS con las fórmulas explicadas previamente.
  - Es uno de los tantos servicios de **control de congestión** que provee TCP. Quiere decir que la ruta que está tomando el paquete está sobrecargada por lo que para la retransmisión le da un poco más de tiempo

### Fast Retransmit

- Generación de ACKs por parte del receptor

| Evento                                                                             | Acción                                                                                        |
|------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| Arribo de segmento en orden con #SEQ esperado; los bytes hasta #SEQ están ACKeados | <i>Delayed ACK</i> : esperar hasta 500 ms para el siguiente segmento; si no llega, enviar ACK |
| Arribo de segmento en orden con #SEQ esperado; otro segmento tiene ACK pendiente   | Envío inmediato de <b>ACK acumulativo</b> que contemple ambos segmentos                       |
| Arribo de segmento fuera de orden con #SEQ mayor al esperado ( <i>gap</i> )        | Envío inmediato de <b>ACK duplicado</b> indicando #SEQ del siguiente byte esperado            |
| Arribo de segmento que cubre parcial o totalmente un gap                           | Envío inmediato de ACK (siempre y cuando el segmento comience en la parte inferior del gap)   |

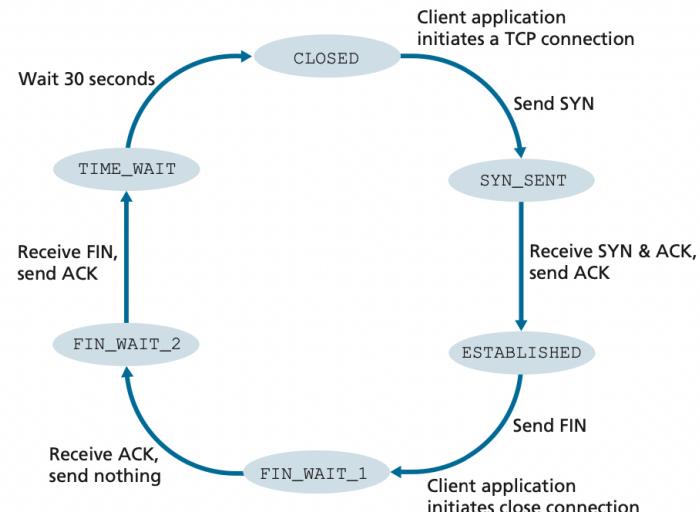
- El delayed ACK es por si te llegan varios paquetes juntos, puedes enviar menos ACKs ya que son acumulativos
- Muchas veces el emisor envía muchos paquetes en simultáneo, si un paquete se llega a perder va a recibir muchos ACKs duplicados y hasta que no haya un timeout no lo va a poder retransmitir
- Para solucionar esto, si el emisor recibe 3 ACKs repetidos se reenvía el segmento sin ACK con el #SEQ más bajo y así evita esperar al timeout
  - Asume que se perdió
- TCP funciona como una combinación de Go-Back-N y selective-Repeat
  - Muchas implementaciones de TCP bufferean paquetes fuera de orden
  - Usa ACKs acumulativos

## Control de flujo

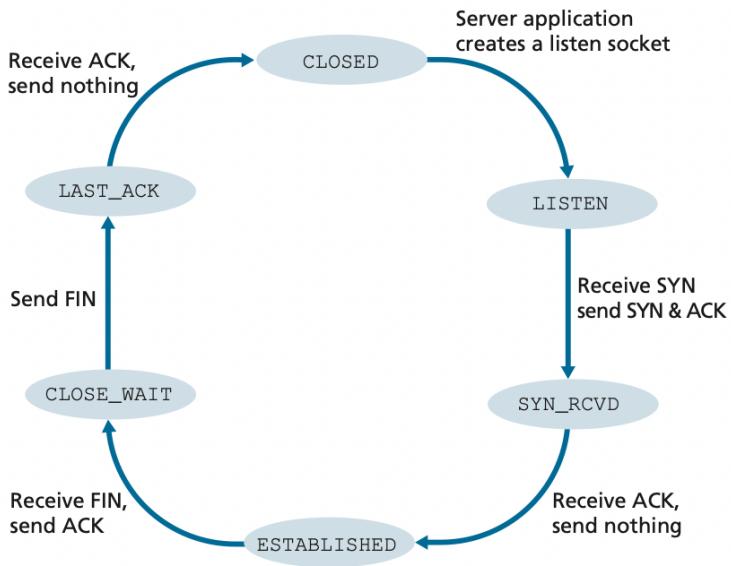
- Muchas veces las aplicaciones no usan instantáneamente la data que llega, por lo que TCP la almacena en buffers. También pasa que las apps muchas veces leen a una velocidad menor a la que se va llenando el buffer. Estas 2 acciones puede provocar que se sobrecargue el buffer y se comience a perder data
- Para evitar esto, TCP provee un servicio de control de flujo
- Es un mecanismo mediante el cual el receptor controla al emisor de modo tal que no sature los buffers transmitiendo demasiado rápido
- El receptor anuncia en el campo **Receiver Window(rwnd)** en el header del segmento, el espacio disponible en el buffer
- Se tiene que cumplir esta desigualdad ya que los buffers no se pueden saturar
  - Last Byte Rcvd – Last Byte Read <= Tamaño Buffer
  - rwnd = Tamaño Buffer – [LastByteRcvd – LastByteRead]
- El emisor limita la cantidad de segmentos en vuelo dependiendo del rwnd
  - LastByteSent – LastByteAcked <= rwnd
- Cuando el rwnd = 0, A no le envia mas data a B pero si debe enviarle segmentos con 1 solo byte para que B los vaya ACKeando y eventualmente se libere espacio del buffer y rwnd > 0

## Establecimiento de conexión TCP

- Antes de comenzar a intercambiar datos, se establece un handshake de 3 pasos
  - Acuerdo de conexión donde se establecen algunos parámetros
- 1. El cliente manda un segmento **SYN**, en dónde está activado su bit. El cliente inicializa un número random de secuencia
- 2. El servidor recibe el SYN y le manda un **SYN ACK** segment diciendo que acepta la conexión. Bit de SYN y ACK activados. Se inicializa un número de secuencia del servidor y crea buffers
- 3. Por último, el cliente manda un segmento **ACK** confirmando la recepción. Inicializa las variables que le pasa el servidor, crea buffers y apaga el bit de SYN
- Ahora ya se pueden enviar datos
- Si el establecimiento de conexión fuera de 2 pasos, habían varios problemas como conexiones semiabiertas o duplicación de datos
- Ambos pueden cerrar la conexión enviando un segmento con el flag **FIN** encendido y el otro responde con un ACK. Ahí se cierra la conexión para el que envió el FIN, luego de un tiempo pasa a la inverso
- Ciclo de los estados de conexión desde el punto de vista del cliente:



- Ciclo de los estados de conexión desde el punto de vista del servidor:



- Si el servidor no tiene abierto o escuchando el puerto de destino, cuando le llegue un segmento con ese puerto no lo va a recibir sino que va a enviar un segmento TCP especial con el flag **RST** que significa que no tiene tal socket activo y que no se lo vuelva a reenviar
- Si no recibis respuesta al SYN es que nunca llegó

## Capítulo 4: The network Layer: Data Plane

### 4.1 Overview of Network Layer

#### Forwarding and Routing

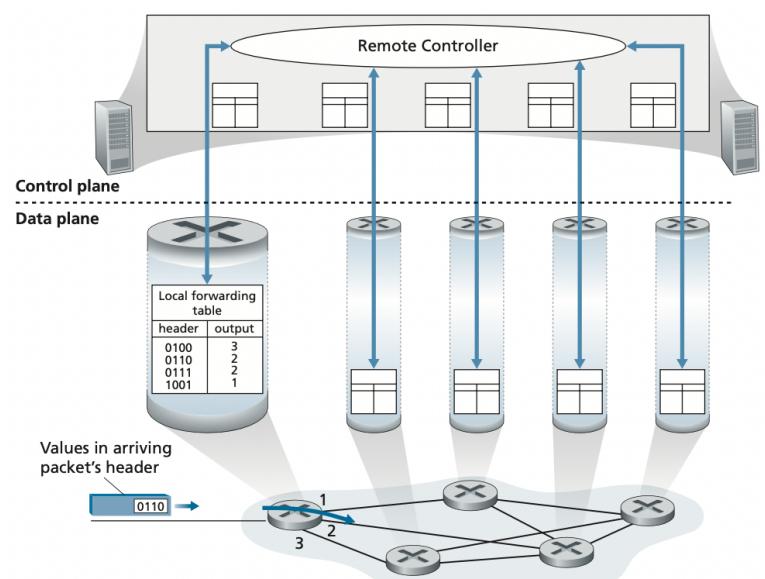
- Plano de datos(local) vs plano de control(global)
  - Plano de datos: Determina cómo se redirecciona un datagrama desde un puerto de entrada hacia uno de salida
    - **Forwarding:** Cuando un paquete llega a un input link de un router, este debe enviarlo al output link correcto para que pueda seguir la ruta deseada (Hardware)
    - Usa un tabla de ruteo para saber a donde mandar cada paquete
      - Concepto de prefijo más largo
  - Plano de control: Determina cómo se encamina un datagrama entre distintos routers a lo largo de la red
    - **Routing:** Decidir que ruta debe tomar el datagrama para llegar a destino (software)
    - Con algoritmos de ruteo tradicionales, implementados en los routers
    - Software-defined networking (SDN): implementado en servidores remotos

#### Control Plane: Traditional Approach

- Los algoritmos de ruteo determinan los valores las tablas de ruteo
- Los algoritmos de ruteo de cada router se comunican entre sí por medio de un protocolo para ir modificando las tablas de ruteo (Se ve en el capítulo 5)

#### Control Plane: SDN Approach

- Se separa el plano de datos y el de control. El de datos sigue estando en el router al igual que en el tradicional, pero el plano de control pasa a estar manejado por un controlador externo
- El controlador remoto es quien maneja las tablas de ruteo, se comunica con los router vía SDN(software-defined networking)

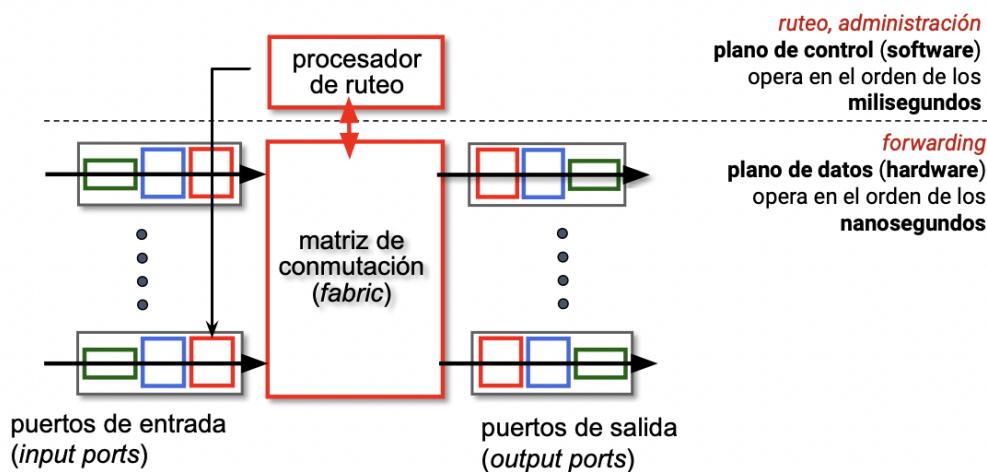


#### Modelo de servicios de Red

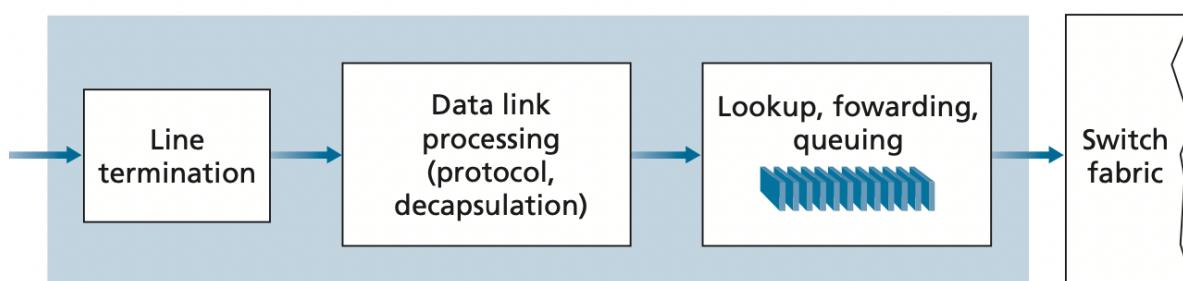
- Ofrece un servicio best-effort

- Básicamente no garantiza nada. No garantiza llegada del paquete, ni delay mínimo, ni encriptación ni llegada en orden. Igualmente funciona bastante bien
- Es simple y de fácil adopción
- Su combinación con protocolos de capas superiores mostró muy buenos rendimientos (DASH)

## 4.2 What's inside a Router?



- Puertos de entrada: Usa las 3 capas más bajas de internet. Cuando le llega un paquete le entra por la capa física, luego hace uso de funciones de la capa de enlace para desencapsular el paquete y pasarlo a la capa de red. Una vez en esta última “caja” se consulta a la tabla de ruteo para saber a qué puerto de salida mandar el paquete. En este punto se puede producir encolamiento.
  - Destination-based forwarding: se fija únicamente en la IP de destino
  - Generalized forwarding: Se fijan en otros campos del header para forwardear los paquetes



- Fabric: conecta los puertos de entrada con los de salida
- Puertos de salida: Almacena los paquetes recibidos del fabric y los va transmitiendo a medida que le agrega los headers de la capa de enlace y física
- Procesador de ruteo: Se encarga de las funciones del plano de control. Puede usar algoritmos tradicionales o SDN para mantener las tablas de ruteo
- Diferenciar los puertos del router con los de los sockets. Los de los router son **puertos físicos**

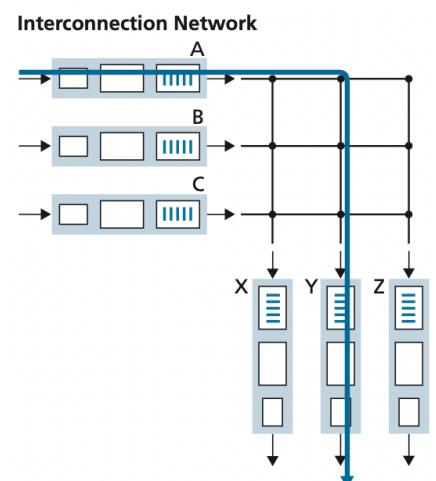
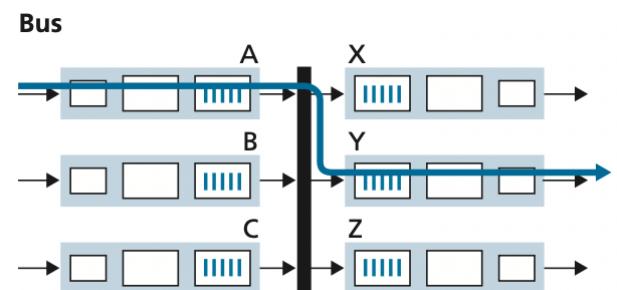
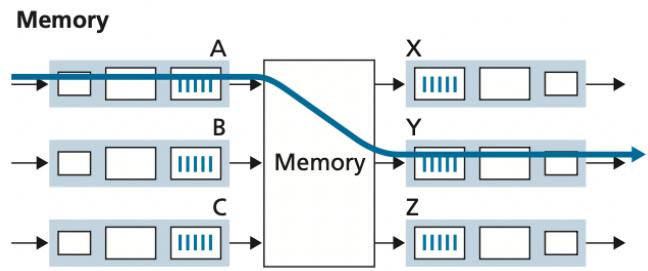
## Input Port Processing and Destination-Based Forwarding

- Es imposible tener una tabla de ruteo con todas las IPs existentes, son más de 4 billones
- Para solucionar esto, las tablas de ruteo funcionan con la regla del **longest prefix matching**
- Se compara la IP con los prefijos de la tabla (**Look-up table**), y el que más coincide es al puerto al que se va a enviar el paquete
- Se utilizan memorias especiales (**TCAM**) para almacenar estas tablas ya que se requiere que se encuentre el prefijo más largo rápidamente. Se obtienen un solo ciclo de clock
- En el puerto de entrada también ocurren procesamiento de la capa de enlace y física, un mínimo control de checksum y del TTL

| Prefix                     | Link Interface |
|----------------------------|----------------|
| 11001000 00010111 00010    | 0              |
| 11001000 00010111 00011000 | 1              |
| 11001000 00010111 00011    | 2              |
| Otherwise                  | 3              |

## Switching

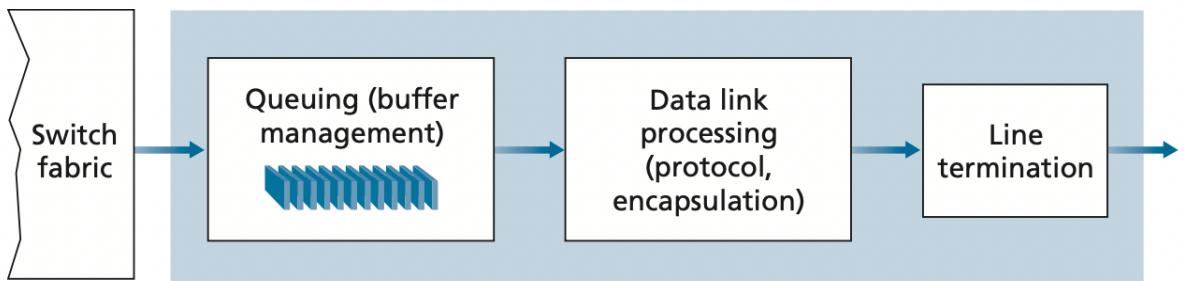
- **Switching via memory**
  - Los routers de primera generación
  - El puerto de entrada copia el paquete a memoria, se extrae el header y se evalúa a qué puerto enviarlo y se lo vuelve a copiar en el buffer de ese puerto
  - El switching se hace vía CPU
  - No se puede forwardear a más de un paquete a la vez, muy lento
- **Switching via a bus:** El puerto de entrada mete el paquete a un bus compartido sin intervención del procesador
  - Se pone una etiqueta interna con el puerto de salida en el paquete y se lo envía a todos los puertos. Únicamente lo va a aceptar el puerto marcado en la etiqueta
  - Únicamente puede viajar un paquete a la vez por el bus
- **Switching via an interconnection network**
  - Se usan  $2N$  cables para conectar  $N$  cables de entrada con  $N$  cables de salida
  - Permite enviar paquetes en paralelo siempre y cuando no tengan el mismo puerto de salida. En ese caso van a tener que esperar
  - Cada puerto de entrada está conectado a todos los puertos de salida



- Al llegar paquetes, el controlador cierra el punto de cruce apropiado y sube el paquete a bus

### Output Port Processing

- Se agarra el paquete proveniente del fabric y lo transmite al link de salida. En el proceso tiene que encolar/desencolar otros paquetes a enviar y hacer las funciones necesarias de la capa de enlace y física

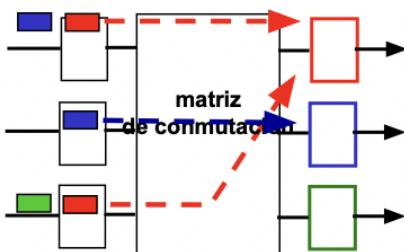


### ¿Dónde ocurre el encolamiento?

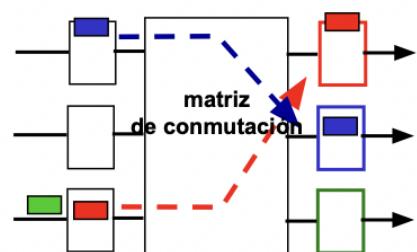
- Puede ocurrir tanto en los puertos de entrada como en los de salida
  - Depende de diferentes tasas de transmisión
- Acá es donde se pueden perder los paquetes

### Input Queuing

- Ocurre cuando la tasa de llegada de paquetes es mas rápida que la tasa que tiene el fabric para transmitirlos hacia los puertos de salida
- También ocurre cuando 2 puertos de entrada tienen que enviar paquetes a un mismo puerto de salida. Los paquetes, en un mismo puerto, que estén esperando a que se envíe el primer paquete, tienen que esperar sin importar si tienen otro puerto de salida que no se está usando. Esto produce HOL (head of line blocking)



**contención:** sólo puede transferirse un paquete rojo (mismo puerto)  
el paquete rojo inferior queda **bloqueado**



el paquete verde experimenta  
**HOL blocking**

### Output Queueing

- Ocurre cuando llegan paquetes al puerto de salida más rápido de lo que los puede transmitir (Poner en el cable)
- Política de drop: decidir qué paquete descartar en caso de que se llene la cola

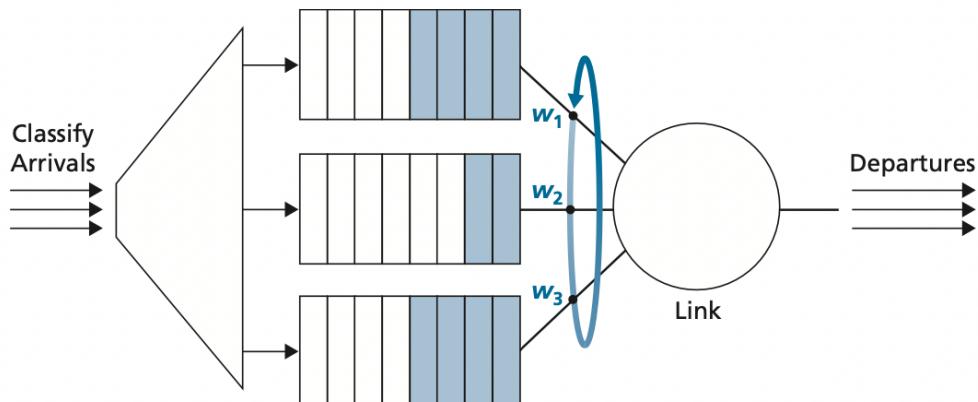
- Drop-tail: Descartar el paquete que llega último cuando no hay lugar
- Descartar un paquete intermedio
- También tienen diferentes algoritmos para manejar las queues y avisar al control de congestión
- Política de scheduling: Determinar que paquetes de la cola se transmiten primero

#### Tamaño de los buffers

- No siempre mayor tamaño de buffer es mejor, ya que a mayor tamaño mayor queue
- Muchos estudios para determinar el mejor tamaño para el buffer

#### Packet Scheduling

- First-in-First-Out
  - Si la queue está llena y llega un nuevo packet se lo puede descartar o eliminar un paquete del medio de la queue para hacerle lugar
- Priority Queuing
  - Hay diferentes queues para las diferentes prioridades de cada packet
  - Un paquete de prioridad baja se envia si no hay paquetes de mayor prioridad esperando para enviarse
- Round Robin y Weighted Fair Queuing (WFQ)
  - Hay diferentes prioridades con sus respectivas queues
  - Transmite uno de prioridad 1, uno de prioridad 2, uno de prioridad 3, etc
  - Va transmitiendo un paquete a la vez de mayor a menor prioridad pero no descuida los de prioridad baja como lo hace priority queuing



**Figure 4.16** ♦ Weighted fair queuing

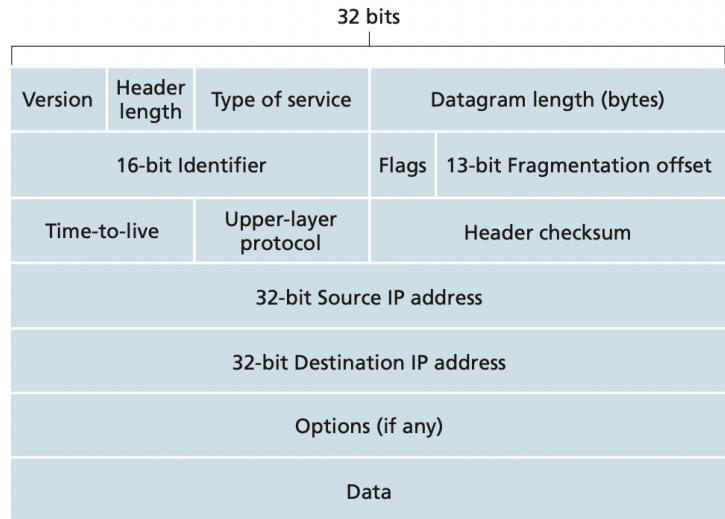
#### Neutralidad de la red

- Un ISP puede decidir en base a diferentes parámetros, que prioridad asignarle a cada paquete a la hora de ser enviado. Puede ver el puerto de destino, su IP entre otras
- Estados Unidos definió 3 reglas principales
  - **No Blocking**
  - **No Throttling**
  - **No Paid Prioritization**

### 4.3 The Internet Protocol (IP): IPv4, Addressing, IPv6 and More

#### Formato del datagrama de IPv4

- Número de versión ya sea IPv4 o IPv6
- Header Length: utilizada para determinar si se usaron campos opcionales
- ToS: Prioridades y control de congestión
- Datagram length: tamaño total del datagrama. 16 bits por lo que el tamaño máximo es 65000 bytes. No suelen tener más de 1500 bytes
- Identificador, flags y offset: utilizados para manejar el tema de la fragmentación
- TTL: Número de hops que puede hacer, decremente de a 1 por cada router
- Upper layer protocol: Especifica si es TCP o UDP
- Checksum: para la detección de errores. Debe ser modificado en cada router ya que el TTL cambia todo el tiempo y los campos opcionales también pueden modificarse. Se checksumea el header únicamente
- IP de destino y de origen
- Opcionales: no se usan mucho. Complica las cosas ya que no hay un tamaño fijo, por esta razón se lo eliminó en IPv6
- Data: el segmento, Application data + Transport Header
- El header del datagrama tiene 20 bytes en total, sin contar los opcionales. El header de TCP también tiene 20 bytes



#### IPv4 Addressing

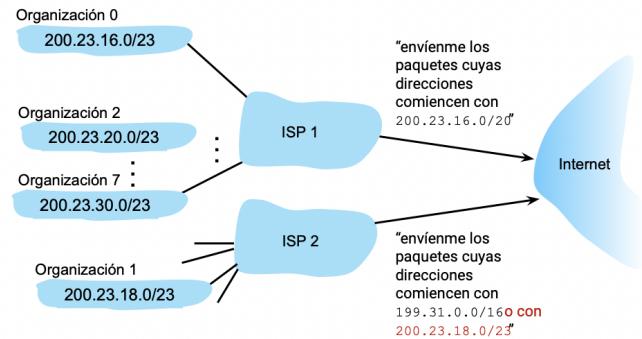
- La conexión entre un host y el link físico se lo llama **interfaz**
- Un host suele tener una única interfaz
- Un router tiene 2 o más interfaces ya que necesita recibir data y poder enviarla
- Cada interfaz debe tener su IP
- Se suelen crear subnets, varios hosts conectados a un router con la misma interfaz
  - Cada subnet está asociada con una máscara: 223.1.1.0/24
  - CIDR: Classless inter domain routing
  - Formato A.B.C.D/X donde X es el número de bits de la máscara
  - Comparten los primeros 24 bits
  - Los hosts de una subnet se pueden comunicar entre sí sin pasar por un router, están conectadas por un switch o wifi
  - Antes de CIDR, las máscaras únicamente podrán ser de 1, 2 o 3 bytes. Esto produce que muchas direcciones queden en desuso

- Todas las subnets de una misma organización comparten un prefijo en la dirección IP. Esto facilita el ruteo de datagramas ya que puertas afuera solo se fijan en ese prefijo para reenviar los paquetes

### Obtaining a Block of Addresses

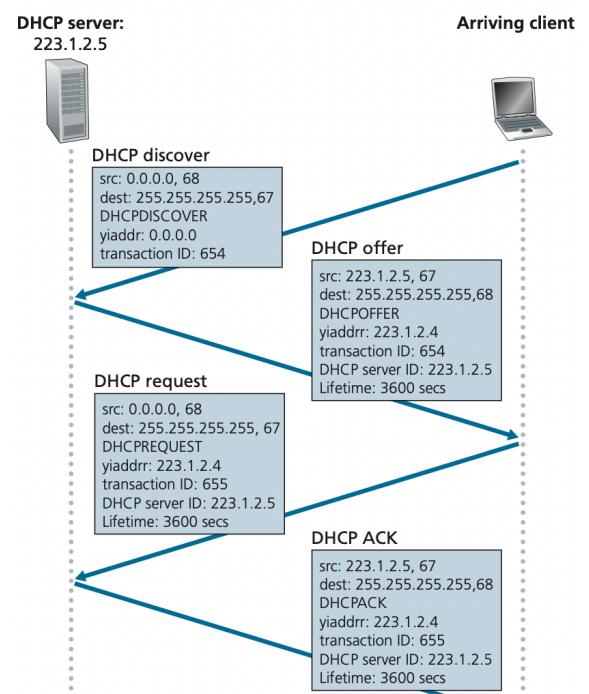
- Los ISPs reciben un bloque de IPs de ICANN con libertad en los últimos N bits
- Las ISPs también usan máscaras para todos sus clientes y le “piden” a internet que le envíen todos los datagramas con IP de destino A.B.c.d/16
  - Si un cliente se cambia de ISP, el nuevo proveedor puede agregar sus direcciones IPs viejas a su nueva tabla de ruteo así no modificar toda la estructura del cliente
- La dirección IP para mandar un datagrama Broadcast es 255.255.255.255
- A su vez, los ISPs también tienen que poder pedir un bloque de direcciones para poder asignarlas. De esto se ocupa a ICANN que también mantiene los root DNS
- Las IPs de una organización son fijas mientras que las de una casa no lo son

|                |                |                                            |
|----------------|----------------|--------------------------------------------|
| ISP's block:   | 200.23.16.0/20 | <u>11001000 00010111 00010000 00000000</u> |
| Organization 0 | 200.23.16.0/23 | <u>11001000 00010111 00010000 00000000</u> |
| Organization 1 | 200.23.18.0/23 | <u>11001000 00010111 00010010 00000000</u> |
| Organization 2 | 200.23.20.0/23 | <u>11001000 00010111 00010100 00000000</u> |
| ...            | ...            | ...                                        |
| Organization 7 | 200.23.30.0/23 | <u>11001000 00010111 00011110 00000000</u> |



### The Dynamic Host Configuration Protocol (DHCP)

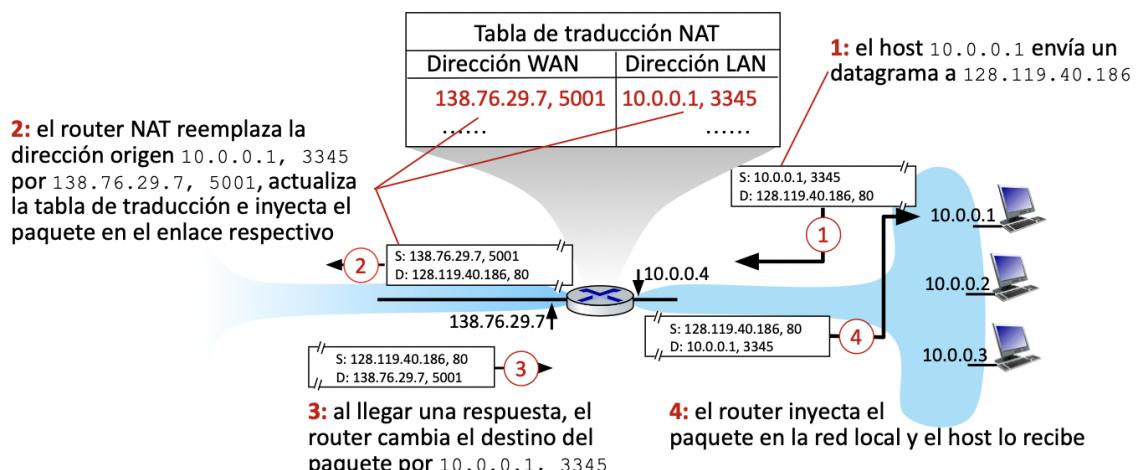
- Este protocolo se utiliza para asignar direcciones IPs dinámicas, es decir que no las tenes para siempre. También se pueden asignar manualmente pero no se hace
  - Son reutilizables
- Además de darte una IP, te da información acerca del router, el DNS local y la máscara de la subnet
- Debe haber un servidor DHCP por cada subnet
- Un nuevo host para conseguir su dirección IP sigue estos 4 pasos
  1. **DHCP server discover:** El host lo envía broadcast ya que no conoce la dirección del router
    - a. Utiliza UDP, IP dst: 255.255.255.255 y port dst: 67. El host no tiene IP por lo que IP scr: 0.0.0.0 y un src port al azar
    - b. Solo lo va a recibir aquel que esté escuchando en ese puerto
  2. El servidor responde con un **DHCP offer:** Responde diciendo que es el servidor y le manda una IP para que use en el campo YIADDR
    - a. Lo envía broadcast ya que el host no tiene IP aun



3. **DHCP request:** El host responde al servidor diciendo que aceptó la IP enviada. Puede pasar que varios servidores DHCP le envíen el offer, es por esto que debe responder con el request
  - a. Va broadcast para hacerle saber a los otros servidores DHCP que aceptó una y que liberan la IP que habían reservado
4. **DHCP ACK:** El servidor confirma la IP y confirma algunos parámetros
  - Una vez terminado estos 4 pasos el host ya puede utilizar la IP por un tiempo determinado
  - Existe un mecanismo para renovar la misma IP luego del tiempo de expiración

### Network Address Translation (NAT)

- Hay suficientes direcciones IPs? Hay  $2^{32}$  combinaciones
- ICANN reservo el último RR en 2011
  - Sin embargo hay nuevos hosts todo el tiempo
  - Las direcciones son renovables, no son para siempre
  - La técnica NAT ayuda a solucionar este problema
- El router tiene una IP única frente a Internet, IP pública
- Todos los hosts tienen una dirección IP privada que es utilizada únicamente en la subnet
  - Frente al exterior, son visibles por la dirección pública del router
  - Todos los datagramas salen con IP src del router y se diferencian por el puerto
  - Y todos los datagramas que llegan tienen la IP del router como destino
- Muchas subnets tienen las mismas IPs y máscaras pero como todos los router tienen una IP única no hay problema ya que las subnets no son visibles al exterior
- Nataar tiene muchas ventajas:
  - El ISP solamente tiene que asignar un IP
  - Posibilidad de cambiar direcciones de hosts en la red sin notificar al mundo exterior
  - Posibilidad de cambiar de ISP sin cambiar las direcciones de los dispositivos
  - Seguridad: los dispositivos en la red no son directamente visibles/direccionables
- El router debe reemplazar el par (IP src, port src) por el par (IP pública, port new) y almacenar en una tabla de traducción los mapeos
  - Tabla WAN-LAN, Wide Area Network-Local Area Network
- También deben hacer el reemplazo de pares a la hora de recibir paquetes del exterior
- Los hosts/servidores externos responden a la IP publica y puerto nuevo



- NAT tuvo sus controversias:

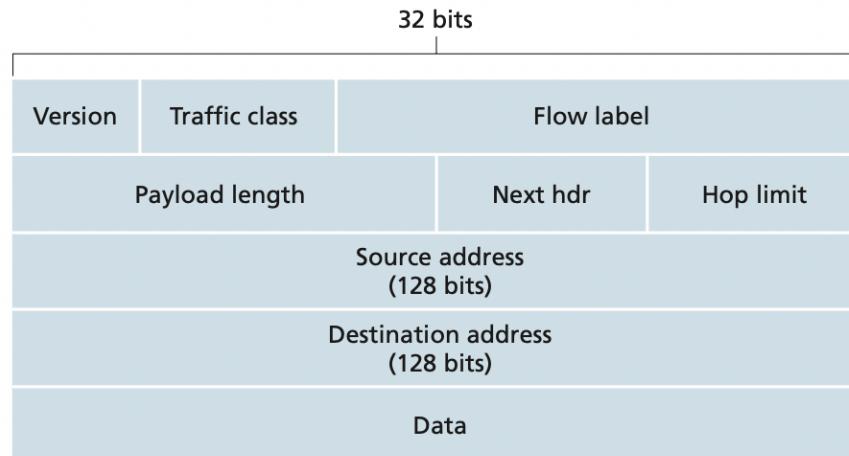
- Los routers no deberían manipular información de niveles superiores (puertos)
- El problema de la escasez de direcciones IP debería resolverse con IPv6
- Complicaciones al tener un servidor atrás de un NAT
- En teoría, los routers son dispositivos que únicamente usan hasta la capa de red pero con NAT están haciendo cosas de la capa de transporte
- Sin embargo NAT llegó para quedarse

## IPv6

- Viene a solucionar el problema de la escasez de direcciones IPs de IPv4
- Imprime direcciones de 128 bits en vez de 32
- Mejora otros aspectos de IPv4 como el control de flujo y las prioridades

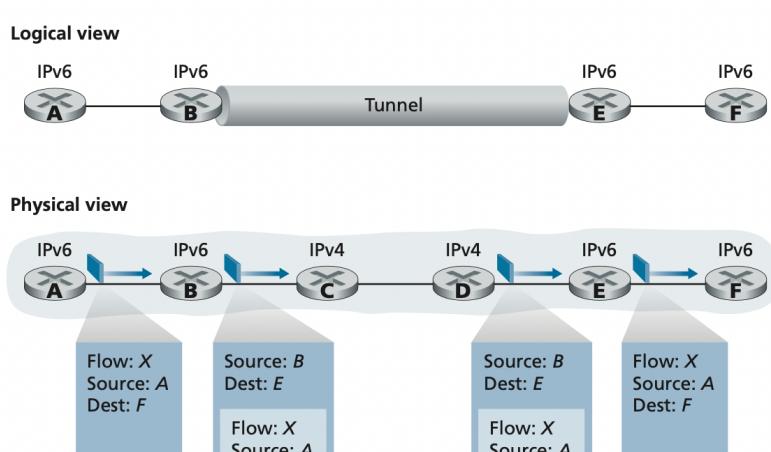
### Formato del datagrama IPv6

- Tamaño fijo de header, sin campos opcionales. Ahora el header es de 40 bytes
  - Permite un procesamiento más veloz
- Flow label: no muy bien definido, se usa para manejar mejor los envíos de video
- Traffic class: 8 bits destinados al manejo de prioridades
- Payload length: longitud del segmento (data + T header)
- Next header: TCP o UDP
- Hop limit: Por cuantos routers puede pasar, decrementa de a 1
- IP de destino y origen: cada una de 128 bits
- Data
- Campos como el checksum, flags de fragmentación y los opcionales no están en el datagrama de IPv6
  - No se permite fragmentación
  - El checksum era medio redundante



### Transición de IPv4 a IPv6: Tunneling

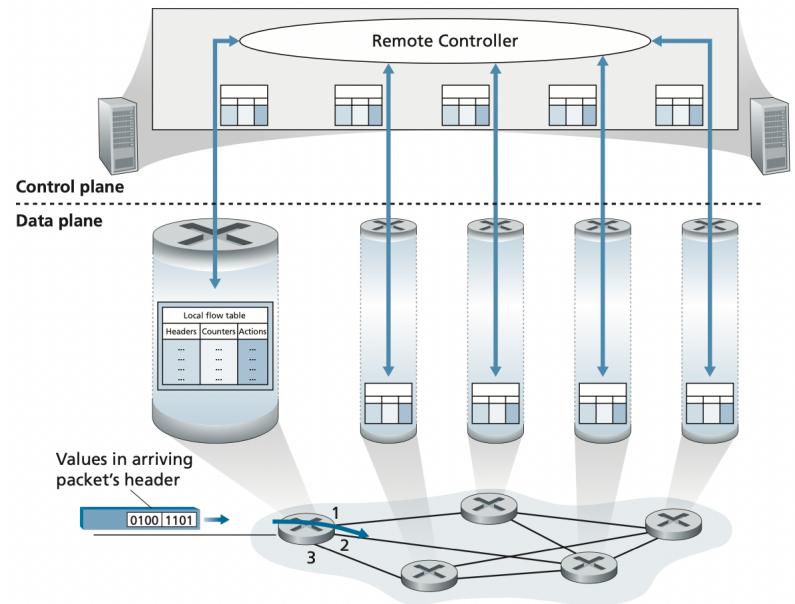
- Todo el internet está basado en IPv4, como transicionar todo a IPv6?
- Los nuevos routers pueden manejar datagramas de ambos protocolos pero los routers viejos únicamente de IPv4



- No sirve poner una fecha límite para el cambio, ya fue probado años atrás y fracaso
  - Se implementó una técnica llamada **tunneling**
    - Cuando un router IPv6 quiere enviarle algo a otro IPv6 pero en el camino hay IPv4, el router IPv6 encapsula el datagrama IPv6 dentro de uno IPv4 para que lo pueda procesar y enviar hasta llegar a destino donde se lo desencapsula
  - IPv6 nos mostró que es muy complicado cambiar protocolos de las capas inferiores, son los cimientos de internet. Se comenzó a implementar en 1990s y recién ahora el 40% de routers lo implementa
    - Si es más fácil modificar protocolos de aplicación, de hecho están en constante cambio

## 4.4 Generalized Forwarding and SDN

- Software defined networking
  - OpenFlow: standard de match plus action
  - Match plus action: Mirar algunos campos del header (match) y luego enviar el datagrama, modificarlo, descartarlo, copiarlo, etc(action)
  - También tiene un set de contadores que se van actualizando a medida que recibe paquetes
  - A diferencia del destination based forwarding, acá se miran varios campos y no solo la IP de destino
  - Las tablas de flujo determinan las reglas del match plus action del router



**Figure 4.28** ♦ Generalized forwarding: Each packet switch contains a match-plus-action table that is computed and distributed by a remote controller

## Match

- Se pueden matchear con campos tanto de la capa de enlace, de red y de transporte
  - Se utilizan tablas de flujo o flow/forwarding tables

| Ingress Port | Src MAC | Dst MAC | Eth Type | VLAN ID       | VLAN Pri | IP Src | IP Dst | IP Proto        | IP TOS | TCP/UDP Src Port | TCP/UDP Dst Port |
|--------------|---------|---------|----------|---------------|----------|--------|--------|-----------------|--------|------------------|------------------|
| Link layer   |         |         |          | Network layer |          |        |        | Transport layer |        |                  |                  |

## Action

- Cada entrada de la tabla tiene un número de acciones a hacer si el paquete matchea con ciertos valores
    - Si hay más de 1 acción, se realizan en el orden especificado en la lista
  - Acciones más comunes:

- **Forwarding:** reenviar el datagrama a cierto puerto de salida, broacastearlo. También se lo puede encapsular y enviar al controlador
- **Dropping:** Descartar el datagrama, se puede explicitar o las entradas que no tengan acción indican que debe ser descartado
- **Modify-field:** Varios de los campos del datagrama pueden ser modificados

### Ejemplos de Match-plus-action

Simple Forwarding:

| s1 Flow Table (Example 1)                                |            | s2 Flow Table (Example 1)            |            |
|----------------------------------------------------------|------------|--------------------------------------|------------|
| Match                                                    | Action     | Match                                | Action     |
| Ingress Port = 1 ; IP Src = 10.3.*.* ; IP Dst = 10.2.*.* | Forward(4) | Ingress port = 2 ; IP Dst = 10.2.0.3 | Forward(3) |
| ...                                                      | ...        | Ingress port = 2 ; IP Dst = 10.2.0.4 | Forward(4) |
|                                                          |            | ...                                  | ...        |

### Firewall

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|-------------|---------|---------|----------|---------|----------|--------|--------|---------|--------|------------|------------|--------|
| *           | *       | *       | *        | *       | *        | *      | *      | *       | *      | *          | 22         | drop   |

Descartar todos los datagramas destinados al puerto TCP 22 (ssh)

### 4.5 Middleboxes

Middleboxes: “any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host”

La abstracción de match-plus-action permite unificar distintos dispositivos de red

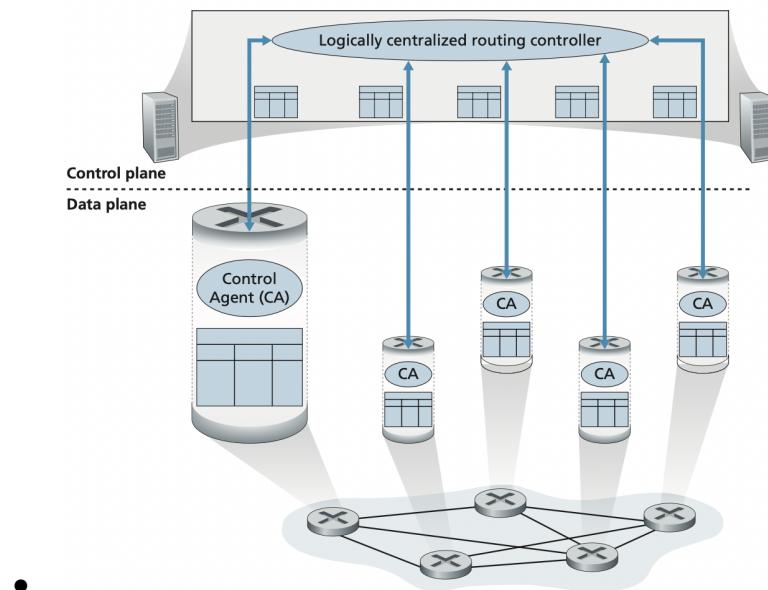
- Router (no es un middlebox)
  - match: longest prefix
  - action: reenviar a puerto de salida
- Switch
  - match: dirección destino nivel enlace
  - action: reenviar o floodear
- Firewall
  - match: dirección IP y puertos TCP/UDP
  - action: permitir o descartar
- NAT
  - match: dirección IP y puerto
  - action: reescribir dirección IP y puerto

Antes se solía respetar la separación de las capas, hoy dia estan entrelazadas y hay dispositivos que hacen cosas que no le pertenece a su capa

## Capítulo 5: The network Layer: Control Plane

### 5.1 Introduction

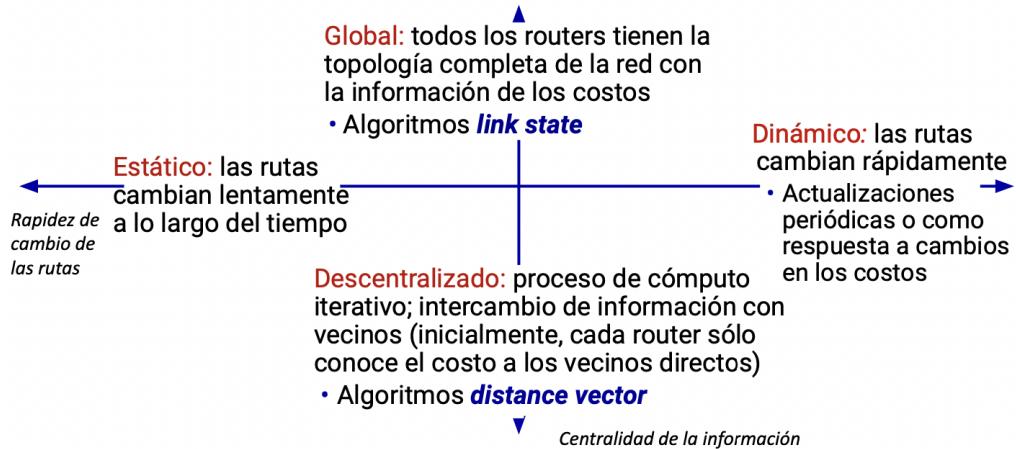
- Vamos a estudiar cómo se crean, mantienen e instalan las forwarding tables, tanto las destination based como las generalized
- Plano de control = routing
- Plano de datos = forwarding
- Per-router control: Cada router tiene su respectivo algoritmo/componente de ruteo, y este algoritmo/componente se comunica con los de los otros routers para ir seteando y modificando las tablas
- Logically centralized control: Un algoritmo global se ocupa de setear las tablas de todos los routers. SDN



### 5.2 Routing Algorithms

- Cómo encontrar el camino más barato para ir desde el emisor hacia el receptor
- Un grafo  $G = (N, E)$  tiene  $N$  nodos/routers y  $E$  relaciones que representan los enlaces físicos entre 2 routers
- Para 2 nodos cualquiera,  $c(x,y)$  es el costo entre esos 2 nodos
- Si el par  $(x,y)$  no pertenece a  $E$ ,  $c(x,y) = \infty$
- Si el par  $(x,y)$  pertenece a  $E$ ,  $x$  e  $y$  son vecinos

- Clasificación de los algoritmos de ruteo: por Centralidad de la información y por rapidez de cambio de las rutas



TD4 2022

11

- También se los puede clasificar por si son sensibles a los cambios de costos de un link o no, es decir si en algún momento un link se está usando mucho debería cambiar su costo o no

### Link-State (LS) Routing Algorithm

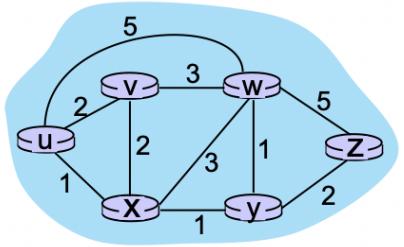
- En este tipo de algoritmos, todos los nodos conocen la topología de la red y los costos de los enlaces
  - Se logra vía broadcast
- Algoritmo de Dijkstra:** Computa caminos de costo mínimo desde un nodo hacia los demás y crea una tabla de forwarding
  - Es iterativo: luego de k iteraciones se conoce el camino de costo mínimo hacia k nodos
  - $D(v)$ : menor costo desde el nodo inicial hasta el nodo v
  - $p(v)$ : nodo predecesor a lo largo del camino del origen a v
  - $N'$ : Conjunto de nodos para los que ya se conoce el camino mínimo
- Ejemplo práctico:
  - Inicialización: para todo nodo  $a$ , si es vecino de  $u$ ,  $D(a) = C_{u,a}$ , sino es infinito
  - Encontrar  $a$  que no esté en  $N'$  tal que  $D(a)$  de la iteración anterior es mínimo (busco costo mínimo en la última fila que calculé)
  - Agregar  $a$  a  $N'$  (agrego nodo con costo mínimo a  $N'$ )
  - Actualizar  $D(b)$  para todo  $b$  vecino de  $a$  que no esté en  $N'$ :  $D(b) = \min(D(b), D(a) + c_{a,b})$  (me fijo si me conviene o no pasar por este nodo intermedio que fue agregado a  $N'$  desde cada nodo para llegar a destino, y anoto el costo mínimo en la tabla)
    - “ $u$ ” indica que el último nodo antes de llegar a destino (nodo predecesor) es  $u$
    - Notar que en la segunda iteración (paso 2), se conoce el camino a 2 destinos

```

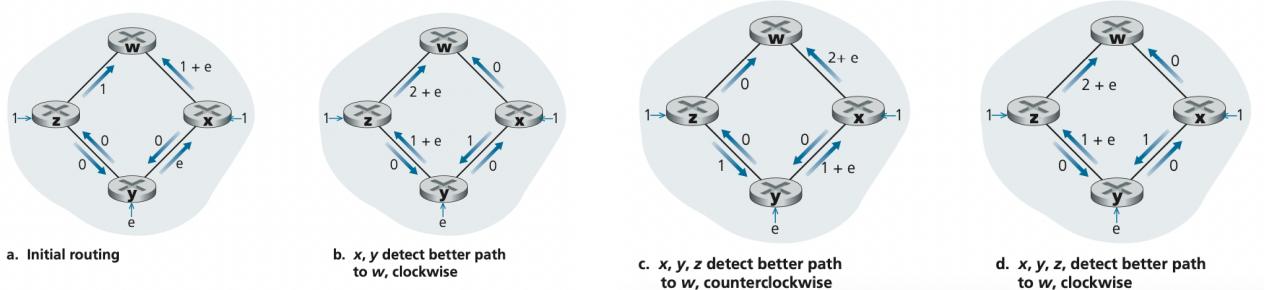
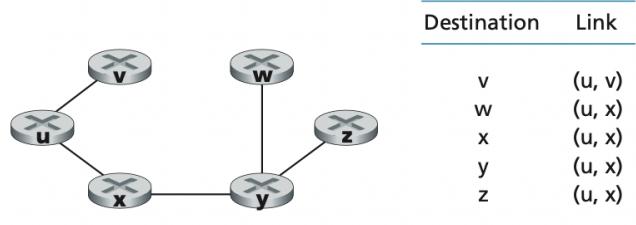
1 Initialization:
2 N' = {u}
3 for all nodes v
4 if v is a neighbor of u
5 then D(v) = c(u,v)
6 else D(v) = ∞
7
8 Loop
9 find w not in N' such that D(w) is a minimum
10 add w to N'
11 update D(v) for each neighbor v of w and not in N':
12 D(v) = min(D(v), D(w)+ c(w,v))
13 /* new cost to v is either old cost to v or known
14 least path cost to w plus cost from w to v */
15 until N' = N

```

| step | $N'$ | $D(v), p(v)$ | $D(w), p(w)$ | $D(x), p(x)$ | $D(y), p(y)$ | $D(z), p(z)$ |
|------|------|--------------|--------------|--------------|--------------|--------------|
| 0    | u    | 2, u         | 5, u         | 1, u         | $\infty$     | $\infty$     |
| 1    | ux   | 2, u         | 4, x         |              | 2, x         | $\infty$     |
| 2    | uxy  | 2, u         | 3, y         |              |              | 4, y         |
| 3    | uxyv |              | 3, y         |              |              | 4, y         |
| 4    |      |              |              |              |              | 4, y         |



- Cuando LS termina tenemos para cada nodo su predecesor para llegar con costo mínimo, y para cada predecesor tenemos su predecesor. De esta forma creamos una tabla de ruteo para el nodo
- **Complejidad computacional:** N nodos, E aristas
  - Hay n iteraciones; en cada una se revisan los nodos w que no estén en N'
  - se busca un total de  $n^*(n-1)/2$  nodos → complejidad  $O(n^2)$
  - Se puede implementar en tiempo  $O(e + n \log n)$  (con Fibonacci heaps)
- **Complejidad en mensajes**
  - Cada router debe hacer broadcast de su estado a los otros routers
  - Algoritmos de broadcasting eficientes permiten disseminar un mensaje atravesando  $O(n)$  enlaces
  - Agregando para los n routers, tenemos  $O(n^2)$  mensajes en total
- Oscilaciones:



- Si los costos de los links dependen del tráfico pueden ocurrir oscilaciones
- La solución es que los routers ejecuten el algoritmo en momentos diferentes y no estén sincronizados. Esto se logra randomizando el momento de ejecución del algoritmo

### The Distance-Vector(DV) Routing Algorithm

- **Iterativo:** proceso continúa hasta que no se intercambia más información entre vecinos.
- **Asíncrono:** no requiere que todos los nodos operen al unísono entre sí
- **Distribuido:** cada nodo recibe alguna información de uno o más de sus vecinos conectados directamente, realiza un cálculo y luego distribuye los resultados de su cálculo a sus vecinos.

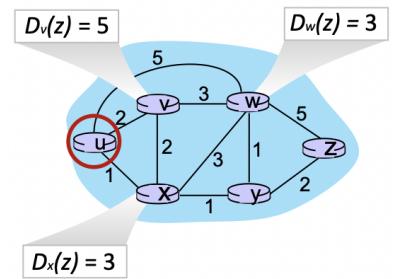
- **Ecuación de Bellman-Ford:**  $d_x(y) = \min_v \{ C_{x,v} + d_v(y) \}$ 
  - $d_x(y)$ : costo del camino de menor costo del nodo  $x$  al nodo  $y$
  - $\min_v$ : mínimo tomado sobre todos los vecinos  $v$  de  $x$
  - $C_{x,v}$ : costo del enlace de  $x$  a  $v$
  - $d_v(y)$ : costo del camino de menor costo del nodo  $v$  al nodo  $y$
  - Intuición: viajó de  $x$  a  $v$  y tomó el camino mínimo de  $v$  a  $y$  (como hay que empezar viajando a una vecino  $v$ , el menor costo de  $x$  a  $y$  es el mínimo de  $C_{x,v} + d_v(y)$  entre todos los vecinos  $v$ )

- Ejemplo:

$$\begin{aligned} d_u(z) &= \min \{ C_{u,v} + d_v(z); C_{u,x} + d_x(z); C_{u,w} + d_w(z) \} \\ &= \min \{ 2 + 5; 1 + 3; 5 + 3 \} \\ &= 4 \end{aligned}$$

el nodo que ofrece el mínimo ( $x$ ) es el next hop en el camino mínimo estimado hacia el destino ( $z$ )

- La ecuación de Bellman-Ford sugiere la forma de comunicación vecino a vecino que tendrá lugar en el algoritmo DV → te dice el next-hop



- Idea:

- Cada cierto tiempo, los nodos envían a sus vecinos sus propias estimaciones de caminos mínimos hacia los demás nodos  $D_x(y)$
- Cuando  $x$  recibe una estimación, actualiza la propia a partir de la ecuación de Bellman-Ford:  $D_x(y) = \min_v \{ C_{x,v} + D_v(y) \}$  para cada nodo  $y$  en  $N$
- Si las estimaciones de  $x$  fueron modificadas al aplicar la ecuación,  $x$  manda vuelve a mandar sus estimaciones actualizadas a cada uno de sus vecinos
- Mientras que los nodos sigan intercambiando sus estimaciones, cada costo estimado  $D_x(y)$  converge al costo mínimo real,  $d_x(y)$

## Distance-Vector (DV) Algorithm

At each node,  $x$ :

```

1 Initialization:
2 for all destinations y in N:
3 $D_x(y) = c(x, y)$ /* if y is not a neighbor then $c(x, y) = \infty$ */
4 for each neighbor w
5 $D_w(y) = ?$ for all destinations y in N
6 for each neighbor w
7 send distance vector $\mathbf{D}_x = [D_x(y) : y \text{ in } N]$ to w
8
9 loop
10 wait (until I see a link cost change to some neighbor w or
11 until I receive a distance vector from some neighbor w)
12
13 for each y in N:
14 $D_x(y) = \min_v \{c(x, v) + D_v(y)\}$
15
16 if $D_x(y)$ changed for any destination y
17 send distance vector $\mathbf{D}_x = [D_x(y) : y \text{ in } N]$ to all neighbors
18
19 forever
```

- cada nodo (lo de abajo es un loop):
  - **Espera**: cambios en costos de enlaces locales o mensaje estimaciones updateadas de algún vecino
  - **Recomputa** estimaciones a partir de la información recolectada y updatea su forwarding table
  - **Notifica** a sus vecinos si la estimación a algún destino sufrió cambios
- **Descentralizado**: un nodo sólo tiene los costos de los enlaces hacia sus vecinos, y la información que recibe de estos vecinos.
- **Iterativo y asincrónico**: cada iteración se genera por:
  - Cambios en los costos locales
  - Actualización de los vecinos
- **Distribuido y auto-regulado**: cada nodo notifica a sus vecinos sólo cuando sus estimaciones cambian:
  - No se producen acciones si no se reciben notificaciones

### Ejemplo:

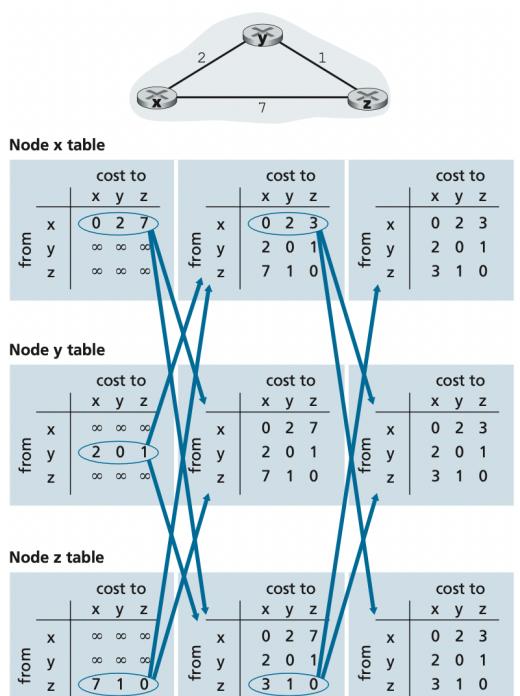
1º columna ( $t_0$ ) : Inicialización.

- Los nodos sólo tienen estimaciones hacia sus vecinos directos.
- Envían su vector de distancias (DV) local a sus vecinos

2º columna ( $t_1$ ) :

- los nodos reciben DVs de sus vecinos
- Computan su nuevo DV local en base a eso. por ejemplo, el nodo  $x$ :

$$D_x(x) = 0$$



$$\begin{aligned}
 D_x(y) &= \min \{ C_{x,y} + D_y(y); C_{x,z} + D_z(y) \} \\
 &= \min \{ 2 + 0; 7 + 1 \} = 2 \\
 D_x(z) &= \min \{ C_{x,y} + D_y(z); C_{x,z} + D_z(z) \} \\
 &= \min \{ 2 + 1; 7 + 0 \} = 3
 \end{aligned}$$

- Envían el nuevo DV local a sus vecinos (si es que hubo cambios)

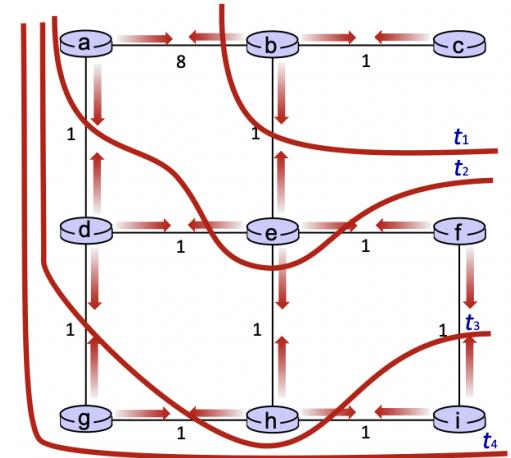
3º columna ( $t_2$ ) :

- los nodos reciben DVs actualizados de sus vecinos
- Computan su nuevo DV local en base a eso.
- Este intercambio y cómputo de DVs va a seguir hasta que no se envíen más DVs actualizados.

### Distance-Vector Algorithm: difusión de información

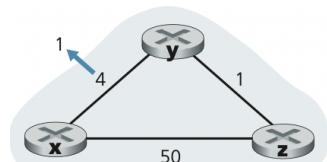
El estado de cada nodo se propaga a través de la red de manera iterativa:

- $t_0$ : El estado de  $c$  en  $t_0$  sólo se ve en  $c$
- $t_1$ : El estado de  $c$  en  $t_0$  se propagó a  $b$  y puede influir en los cálculos de hasta 1 hop de distancia (i.e., en  $b$ )
- $t_2$ : El estado de  $c$  en  $t_0$  puede influir en los cálculos de hasta 2 hops de distancia (i.e., en  $b, a$  y  $e$ )
- $t_3$ : El estado de  $c$  en  $t_0$  puede influir en los cálculos de hasta 3 hops de distancia (i.e., en  $b, a$  y  $e$ , y también en  $d, h$  y  $f$ )
- 4: El estado de  $c$  en  $t_0$  puede influir en los cálculos de hasta 4 hops de distancia (i.e., en  $b, a, e, d, h, f, g, i$  y  $h$  e  $i$ )



### Cambios en los costos de los enlaces

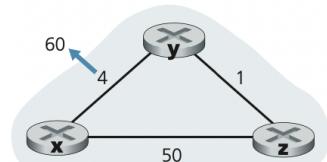
- **Ejemplo:** costo de  $y$  a  $x$  pasa de 4 a 1 (**disminución del costo**)
  - $t_0$ :  $y$  detecta un cambio de costo, actualiza su DV y notifica a sus vecinos
  - $t_1$ :  $z$  recibe el mensaje de  $y$ , computa el nuevo costo mínimo a  $x$  y envía su DV actualizado a sus vecinos
  - $t_2$ :  $y$  recibe la actualización de  $z$  pero sus costos mínimos no sufren cambios;  $y$  **no envía** nuevos mensajes
  - El cambio de costo se propaga rápidamente a través de la red (en 2 iteraciones)



- **Ejemplo:** costo de  $y$  a  $x$  pasa de 4 a 60 (**aumento del costo**)

initialmente:  $D_y(x) = 4; D_y(z) = 1; D_z(y) = 1; D_z(x) = 5$

- $y$  nota que el enlace a  $x$  tiene costo 60 pero  $D_z(x) = 5$  (desactualizado):  $y$  concluye que puede llegar a  $x$  con costo 6 vía  $z$ . Luego informa de esto a  $z$ .



$$D_y(x) = \min \{ C_{y,x} + D_x(x); C_{y,z} + D_z(x) \} = \min \{ 60 + 0; 1 + 5 \} = 6$$

- z observa que el camino a  $x$  vía  $y$  tiene nuevo costo 6: z actualiza su costo a  $x$  y lo pone en 7, vía  $y$ . Luego notifica a  $y$  de este nuevo costo.
$$D_z(x) = \min \{ C_{z,x} + D_x(x); C_{z,y} + D_y(x) \} = \min \{ 50 + 0; 1 + 6 \} = 7$$
- y observa que el camino a  $x$  vía  $z$  tiene nuevo costo 7: y actualiza su costo a  $x$  y lo pone en 8, vía  $z$ . Luego notifica a  $z$  de este nuevo costo.
  - z observa que el camino a  $x$  vía  $y$  tiene nuevo costo 8: z actualiza su costo a  $x$  y lo pone en 9, vía  $y$ . Luego notifica a  $y$  de este nuevo costo.
  - y sigue así.... hasta que z empiece a ver el otro enlace con buenos ojos
- problema de **cuenta a infinito** → cambio de costo viaja muy lentamente a través de la red.  
En este caso toma 44 iteraciones, pero si el cambio es mayor, tardaría muchísimo más
- Para resolver el problema de cuenta a infinito se implementó **poisoned reverse**
  - Si z usa a  $y$  para llegar a  $x$ , z le va a decir a  $x$  que  $D_z(x) = \infty$
  - Z le miente a  $y$  para qué y no use a  $z$  para llegar a  $x$  ya que se estaría usando a sí mismo como intermediario
  - Como  $y$  cree que  $D_z(x) = \infty$ , nunca intentará enrutar a  $x$  a través de  $z$  (nunca se usará a sí mismo), siempre y cuando  $z$  continúe enrutando a  $x$  a través de  $y$

### Comparación entre LS y VS

- DV: cada nodo habla sólo con sus vecinos, pero les proporciona estimaciones de su menor costo hace todos los nodos en la red
- LS: requiere información global (cada nodo necesita saber el costo de cada enlace en la red). Cada nodo necesita comunicarse con todos los demás
- Complejidad en mensajes
  - LS:  $n$  routers,  $O(n^2)$  mensajes
  - DV: intercambio de mensajes entre vecinos; tiempo de convergencia variable
- Ante cambios de costos:
  - LS: el nuevo costo del enlace debe ser enviado a todos los nodos
  - DV: informa sólo si el DV de alguno de los nodos adjuntos a ese enlace se ve modificado
- Velocidad de convergencia
  - LS: complejidad linearítmica (Puede tener oscilaciones)
  - DV: tiempo de convergencia variable (Puede haber ciclos; Cuenta a infinito)
- Robustez: resiliencia ante fallas/hacks en los routers
  - LS:
    - El router puede informar costos incorrectos de sus enlaces (solamente)
    - Cada router sólo computa su propia tabla de ruteo (pero como los ruteos se calculan por separado, le proporciona un grado de solidez)
  - DV:
    - El router puede informar costos incorrectos de **caminos/rutas** (fenómeno de *black-holing*)
    - Las tablas de los routers son usadas por otros routers (En cada iteración, el cálculo de un nodo en DV pasa a su vecino): los errores pueden propagarse a través de toda la red

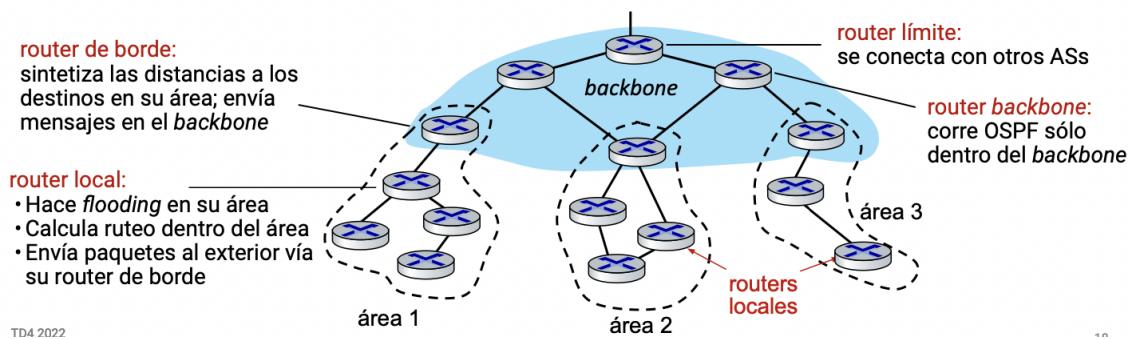
### 5.3 Intra-AS Routing in the Internet: OSPF

- Red como colección de routers interconectados (todos ejecutan el mismo algoritmo de enrutamiento para calcular las rutas a través de toda la red). Modelo simplista xq:
  - *Escala*: se requiere mucha memoria para almacenar información de enrutamiento en cada router de Internet. Se necesita reducir la complejidad del cálculo de rutas en una red tan grande como Internet
  - *Autonomía Administrativa*: Internet es una red de ISPs y cada ISP es una red de routers, que opera y administra su red como desea, pudiendo conectarse a otras redes externas
- Para solucionar estos problemas, los routers bajo el mismo control administrativo (mismo algoritmo de enrutamiento) se agrupan en **autonomous systems (ASs)**
  - Generalmente, todos los routers de una ISP y los enlaces que los conectan constituyen un AS. Todo router en el mismo AS debe correr el mismo protocolo intra-AS
  - Algunos ISP dividen su red en varios ASs interconectados
  - mismo algoritmo de enrutamiento → ***intra-autonomous system routing protocol***
- Un AS se identifica por su número de AS globalmente único (asignado por ICANN)

#### Open Shortest Path First (OSPF)

- Es un protocolo de ruteo intra AS
  - Es abierto y público
- Es un Link state protocol, estilo Dijkstra
  - Cada router conoce la topología del AS. Para eso hace un flooding de la red con mensajes vía IP. Con esta data arma la tabla de forwarding
  - Cuando percibe un cambio en un link envia su nueva información a todos los routers
  - Los costos de los enlaces los determina el administrador de la red
    - Se pueden setear todos a 1 para buscar la menor cantidad de hops o setearlos inversamente a su capacidad
- Los anuncios OSPF están contenidos en mensajes OSPF que son transportados directamente por IP, por lo que el protocolo debe implementar mecanismo de confiabilidad (transferencia de mensajes confiable, verificación de enlaces operativos) por su cuenta
- Provee algunos servicios
  - Seguridad: los mensajes son autenticados, solo los routers confiables pueden enviar data
  - Cuando hay múltiples caminos con el mismo costo, se puede usar cualquiera
  - Soporte para jerarquías
- Un AS se puede configurar jerárquicamente en 2 áreas: local y backbone
  - Cada área ejecuta su propio algoritmo de enrutamiento y cada router en un área transmite su LS a los demás router en esa área
  - áreas donde sí se conocen entre todos, pero para conectarse con routers de otra área, saben a qué router dentro del área dirigirse (al router borde → enrutan los paquetes fuera del área) para salir a través de un gateway (enlaces de salida de cada área)

- *backbone*: routers especiales que interconectan las áreas (área troncal)
  - contiene todos los enrutadores de borde de área en el AS
  - enruta el tráfico entre las otras áreas del AS
  - no sabe costos para llegar a cada router del área, sino sólo al router borde; por eso el router borde sintetiza las distancias a los destinos en su área (costo promedio)
- enruteamiento entre áreas: paquete debe ir al router borde de su área, luego se enruta a través del área troncal al router borde del área de destino y luego se enruta al destino final.
- 2 funciones
  - que cada router pueda anunciar su existencia al resto
  - poder routear



## 5.4 Routing Among the ISPs: BGP

- ruta que sigue un paquete para llegar a destino está determinada por el protocolo intra-AS → OSPF
- para enrutar un paquete a través de ASs → protocolo de enruteamiento inter-AS.
- Protocolos para mandar paquetes entre diferentes ASs
  - **Inter-autonomous system routing protocol**
  - Todos los AS deben usar el mismo protocolo para poder comunicarse entre sí
  - Usan el **Border Gateway Protocol (BGP)**
    - Es el protocolo más importante de Internet junto con IP
    - Es el pegamento de los ISPs
    - Descentralizado y asincrónico

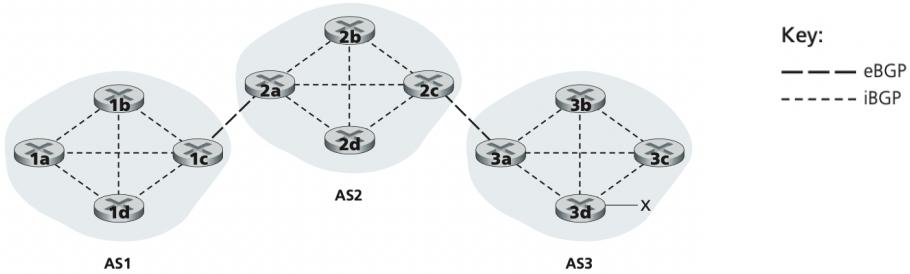
### The Role of BGP

- Cada router de un AS tiene una forwarding table. Para los destinos que están dentro del mismo AS, las entradas en la tabla están determinadas por el protocolo intra-AS. Pero, ¿los destinos que están fuera del AS? BGP
  - Los paquetes no se enrutan a una dirección de destino específica, sino a prefijos CIDRizados, donde cada prefijo representa una subred o una colección de subredes.
  - La tabla tendrá entradas de la forma  $(x, I)$ , donde  $x$  es un prefijo (como 138.16.68/22) e  $I$  es un número de interfaz para una de las interfaces del enrutador.
- BGP proporciona a cada router un medio para:

- Obtener información de accesibilidad de prefijos de los AS vecinos: que cada subred/AS anuncie su existencia al resto de los routers de internet (y sus destinos alcanzables).
  - “I exist and I am here”
- Determinar las “mejores” rutas a los prefijos

### Advertising BGP Route Information

- **Routers internos:** aquellos que no tienen comunicación con otros AS, se conectan a otros routers internos y a hosts
- **Routers gateway:** aquellos que sí tienen comunicación con otros AS
  - Corren eBGP y iBGP
- Conexiones BGP van vía TCP en el puerto 179
- **eBGP, External BGP connection:** conexión de 2 gateway routers, de diferentes AS
- **iBGP, internal BGP connection:** conexión entre 2 routers de un mismo AS
  - No siempre son links físicos



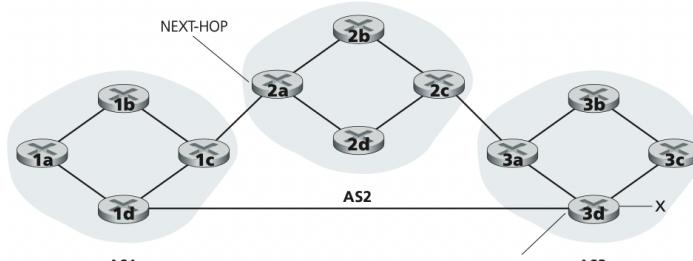
- ¿Cómo publicar información de accesibilidad para el prefijo x en todos los routers?
  - ASs no se envían mensajes entre sí, sino que lo hacen los routers (routers intercambian información de enrutamiento a través de conexiones TCP semipermanentes mediante el puerto 179)
  - Cuatro tipo de mensajes BGP:
    - **OPEN:** abre una conexión TCP a un *peer* remoto y autentica al *peer* emisor
    - **UPDATE:** anuncia un nuevo camino (o actualiza/elimina uno anterior)
    - **KEEPALIVE:** mantiene la conexión abierta en ausencia de UPDATEs; también sirve como ACK de OPEN
    - **NOTIFICATION:** reporta errores en el mensaje anterior; también empleado para cerrar la conexión
  - Cada una de esas conexiones TCP se las llama conexiones BGP
- ¿Cómo publicar información de accesibilidad para el prefijo x en todos los routers en AS1 y AS2?
  - El gateway router 3a envía un mensaje eBGP "AS3 x" al gateway router 2c. Este envía el mensaje iBGP "AS3 x" a todos los demás enrutadores en AS2.
  - El gateway router 2a envía el mensaje eBGP "AS2 AS3 x" al gateway router 1c. Este usa iBGP para enviar el mensaje "AS2 AS3 x" a todos los enrutadores en AS1.
- Puede haber más de una ruta para llegar al mismo destino

## Determining the Best Routes

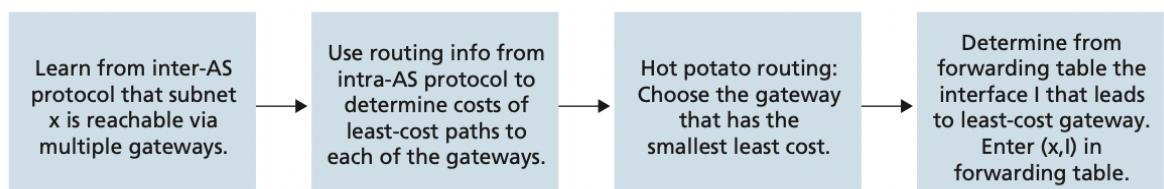
- ¿Cómo elegir entre tantos caminos existentes?
- Atributos BGP
  - AS-PATH: lista de AS por lo que ha pasado el anuncio (cuando se pasa un prefijo a un AS, el AS agrega su ASN a la lista existente en AS-PATH)
    - para no caer en un loop infinito, si un router ve que su AS está contenido en el AS-PATH del anuncio, lo descarta
  - NEXT-HOP: dirección IP del gateway router del AS próximo
- BGP route se escribe de la forma: NEXT-HOP;AS-PATH;prefijo de destino

## Hot Potato Routing

- Algoritmo de enrutamiento BGP
- Elige la ruta con menor costo hacia el NEXT-HOP
- Elige la ruta que menor costo interno tiene para llegar a algún gateway router
  - Busca llegar al gateway con menor coste posible
- En este caso 1B tiene que enviar algo a x, tiene 2 posibles rutas
  - Ir a 1c (solo tiene que hacer un salto internamente), luego a AS2 y por último a AS3
  - Ir a 1d (tiene que hacer 2 saltos internamente) e ir directo a AS3
  - Como este algoritmo se fija en la menor cantidad de saltos posibles dentro del AS va a elegir ir vía 1C y en su tabla de ruteo va a poner (x, interfaz 1 que llega a 1C)



- Cuando se asume que se tiene una table de forwarding para cada router, se agregan los costos de las rutas y se eligen las más baratas. Los routers utilizan tanto el protocolo de enrutamiento inter-AS (BGP) como el protocolo de intra-AS (OSPF).
- Algoritmo egoísta: router intenta reducir el costo en su propio AS mientras ignora los otros componentes de los costos de extremo a extremo fuera de su AS.
  - un paquete es análogo a una “hot potato”, que como te está quemando la mano, deseas pasárselo a otra persona (otro AS) lo más rápido posible
  - El router busca obtener paquetes de su AS con el menor costo posible sin preocuparse por el costo del resto de la ruta fuera de su AS al destino.
- Dos routers en el mismo AS pueden elegir dos rutas AS diferentes al mismo prefijo.



## Route-Selection Algorithm

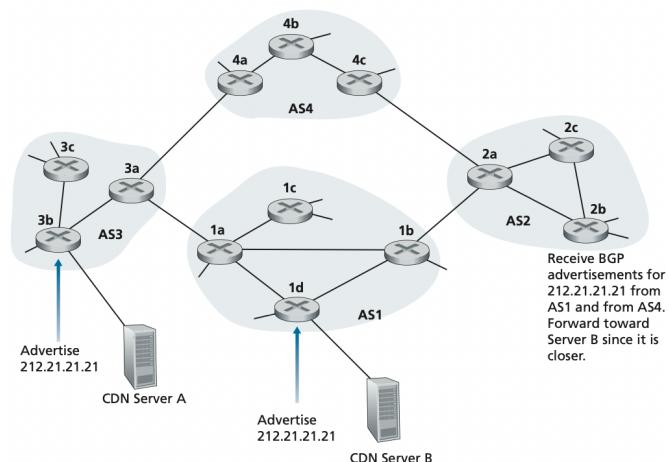
- En la práctica, BGP usa un algoritmo más complejo que hot potato pero que igualmente es este último es usado
- Si hay más de una ruta posible, BGP aplica las siguientes reglas
  1. Se seleccionan las rutas con los valores de preferencia local más altos (atributo que el administrador de red del AS le asigna a cada ruta)
  2. De las rutas restantes (todas con el mismo valor de preferencia local más alto), se selecciona la ruta con el **AS-PATH más corto** → algoritmo DV
  3. De las rutas restantes, se utiliza el *hot potato routing*, es decir, se selecciona la ruta con el enruteador **NEXT-HOP más cercano**
  4. Si aún queda más de una ruta, el enruteador usa identificadores BGP para seleccionar la ruta
- BGP no es un algoritmo egoísta: primero busca rutas con rutas AS cortas

## IP-Anycast

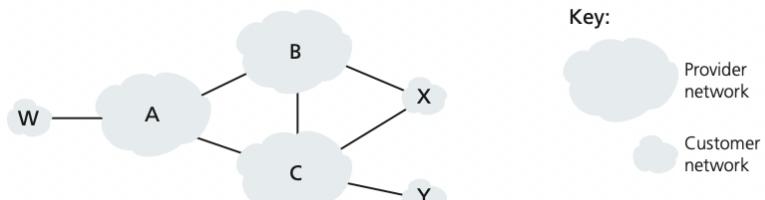
- Para (1) replicar el mismo contenido en diferentes servidores en diferentes ubicaciones geográficas dispersas y (2) hacer que cada usuario acceda al contenido desde el servidor más cercano → algoritmo de selección de ruta BGP
- ¿Cómo una CDN podría usar IP-anycast?

Durante la etapa de configuración de IP-anycast, la empresa CDN asigna la misma dirección IP a cada uno de sus servidores y utiliza BGP estándar para anunciar esta dirección IP desde cada uno de los servidores.

El router BGP que recibe múltiples anuncios de ruta para esta dirección IP, los trata como si fuesen diferentes rutas a la misma ubicación física (cuando en realidad son diferentes ubicaciones físicas). Al configurar su forwarding table, cada router utilizará localmente el algoritmo BGP para elegir la "mejor" ruta a esa dirección IP
- Cuando un cliente solicita un video, la CDN devuelve al cliente la dirección IP común utilizada por los servidores dispersos geográficamente. Cuando el cliente envía una solicitud a esa dirección IP, los routers de Internet envían el paquete de solicitud al servidor "más cercano", según lo define el algoritmo de selección de ruta BGP.
- DNS utiliza IP-anycast para dirigir las consultas de DNS al servidor DNS raíz más cercano.
  - 13 servidores Root originales, los demás son copias y se tratan con IP-Anycast



## Routing Policy



- Cuando un router selecciona una ruta a un destino, la política de enrutamiento del AS puede prevalecer sobre todas las demás consideraciones
- W, X e Y → ISPs de acceso. Todo el tráfico que ingresa está destinado a esa red, y todo el tráfico que sale debe haberse originado en esa red
  - X → ISP de acceso multihomed, está conectado al resto de la red a través de dos proveedores diferentes
- A, B y C → Backbone provider networks
- ¿Cómo se evitará que X reenvíe tráfico entre B y C? Esto se puede lograr fácilmente controlando la forma en que se anuncian las rutas BGP.
  - X funcionará como una red ISP de acceso si anuncia (a sus vecinos B y C) que no tiene caminos a ningún otro destino excepto a sí mismo.
  - al ISP B, si le tiene que mandar algo a C, quizás le convendría pasarlo por X; pero X al ser un ISP cliente, esto no le conviene (no es su negocio). Al no anunciarle a B que tiene rutas a C y viceversa, se ahorra este problema
  - política de anuncio de ruta selectiva para implementar relaciones de enrutamiento cliente/proveedor.
- A le anuncia a B que puede llegar a w a través de A
  - B instala la ruta AW en su base de información de enrutamiento.
  - B le anuncia a x (su cliente) que puede llegar a w a través de B (ruta BAW)
  - ¿debería B anunciar la ruta BAW a C?
    - B no le anuncia a C que puede llegar a w a través de B, salvo que tengan un convenio o algo, porque no es su cliente, sino su "competencia" (no tendría por qué asumir la carga y el costo de transportar tráfico entre A y C)
- Cualquier tráfico que fluya a través de la red troncal de un ISP debe tener un origen o un destino (o ambos) en una red que sea cliente de ese ISP; de lo contrario, el tráfico estaría viajando gratis en la red del ISP.

### ¿Por qué existe ruteo inter-dominio e intra-dominio?

- **Política**
  - inter-dominio: importante decidir quién rutea a través de la red y cómo lo hace
  - intra-dominio: mismo control administrativo; la política pierde relevancia
- **Escala**
  - El ruteo jerárquico ahorra espacio de tablas y reduce el tráfico en las redes
- **Rendimiento**
  - inter-dominio: la política se prioriza más que el rendimiento

- Una ruta larga o costosa que cumple con ciertos criterios políticos puede ser preferida a una corta y barata que no los satisface
- intra-dominio: puede focalizar en optimizar las rutas

## Capítulo 6: The Link Layer and LANs

### 6.1 Introduction to the Link Layer

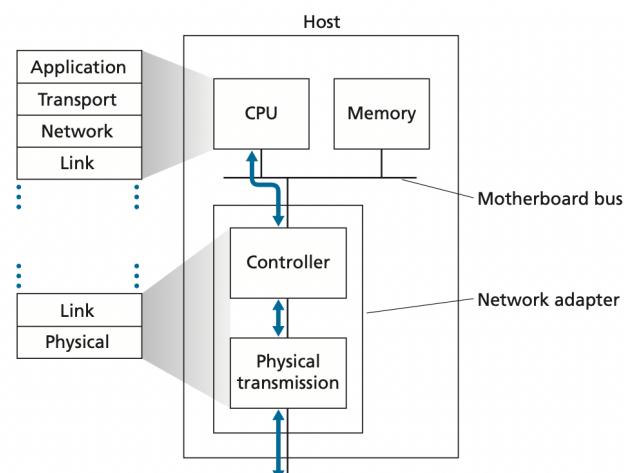
- Nodos: Hosts y routers
- Tipo de paquete: frame -> encapsula datagrama
- Los datagramas pueden ser transmitidos por distintos protocolos de enlace sobre distintos enlaces
- Tipos de enlace: cable, inalámbrico, LANs
- Transmitir datagramas por un enlace desde un nodo a otro físicamente adyacente

#### Servicios que provee la capa de enlace

- Framing:
  - Encapsular datagramas en frames, agregándoles un header y un trailer
  - Poner direcciones MAC en los headers
- Link access:
  - Protocolos y reglas de acceso al canal en caso de que sea compartido
- Reliable delivery:
  - Se suele implementar en medio que tienen grandes tasas de error como por ejemplo los medios Wireless
  - Checksums, ACKs, etc -> para poder detectar errores más rápidamente y re-transmitir. Se pueden corregir algunos errores de bits
- Detección y corrección de errores:
  - Mecanismos para detectar errores de bits y corregirlos
  - Eventualmente se pueden corregir
- Control de flujo:
  - Ritmo de transmisión entre nodos adyacentes

#### Donde se implementa la capa de enlace?

- Implementación del nivel de enlace
  - presente en todos los hosts y dispositivos
  - Se implementa en la placa de red (NIC, Network Interface Controller) o en chip
    - implementa en capa de enlace y capa física
  - Se conecta a los buses del hosts
  - combinación de hardware, software y firmware

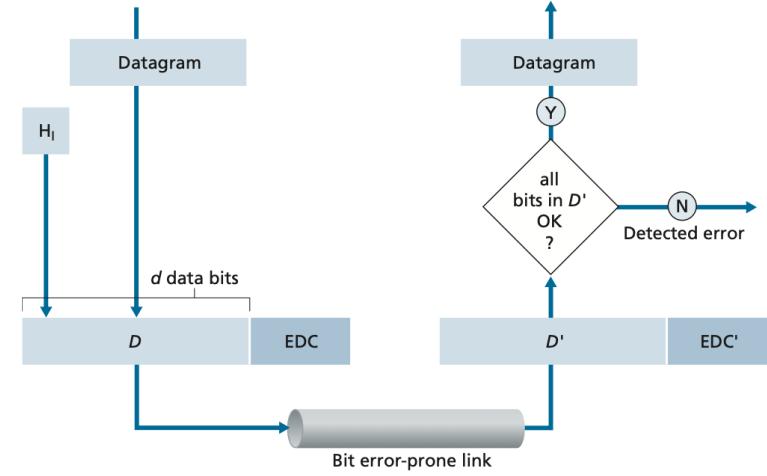


- Desde el lado **emisor** el controlador encapsula datagrama en un frame; agrega bits para control de errores, entrega confiable, control de flujo, etc y finalmente envía el frame al communication link
- Desde el lado **receptor** el controlador recibe el frame, verifica si hay errores, extrae datagrama y lo pasa a la capa superior

## 6.2 Error-Detection and Correction

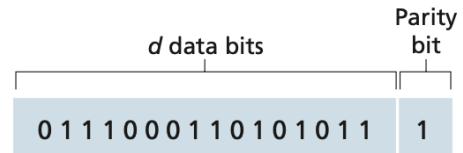
### Techniques

- Funcionamiento de detección de errores:
- El nodo emisor envía un frame al nodo receptor. Este contiene
  - datos ( $D$ ) a proteger por el control de errores → suele incluir datagrama + campos del header
  - bits de detección/corrección de errores EDC (ej: redundancia) al final del frame
  - ej copiar:  $D = 1010$ , le agrego  
 $EDC = 1010 \rightarrow$  Si  $D$  no coincide con  $EDC$ , es porque hay error
- el nodo receptor se debe fijar su  $D'$  (lo que recibió) es igual a  $D \rightarrow$  nos preguntamos si fue detectado algún error, NO si ocurrió algún error.
- La detección de errores puede fallar → el receptor puede no darse cuenta de que la información recibida contiene errores de bits.
  - errores pueden quedar enmascarados (como en checksum)
  - cuánto + bits tenga el campo EDC, mayor será la efectividad de la detección de errores pero a su vez será más caro de enviar y computar
- 3 métodos de detección de errores: control de paridad, checksum y CRC



### Control de paridad

- **Paridad de un bit:** receptor detecta errores de un solo bit
  - me fijo la cantidad de 1 en mi datagrama
    - si tengo cantidad par pongo un 0 en el bit de paridad
    - si tengo cantidad impar pongo un 1 en el bit de paridad
  - Ventajas: Barato y fácil
  - Desventaja:
    - alta proba (50%) de no detectar errores. En caso de que haya un número par de errores no detecta
    - errores por interferencia no son de un bit, sino de ráfagas de bits (no se altera solo un bit) → no me garantiza mucho este método



- **Paridad bidimensional:** receptor puede detectar y corregir errores de un sólo bit → como sabes paridad en filas y columnas, podes ubicar el bit que falló
  - Ordena los bits en una matriz y pone bits de paridad por cada fila y columna
  - detectar y corregir errores → **FEC (Forward Error Correction)** → reduce la cantidad de retransmisiones y al corregir en el receptor, evita tener que esperar el RTT necesario para que el emisor reciba el NAK y para que retransmita el paquete y llegue al receptor.
  - Ejercicio: Dado  $D = 110011010010101$ , ¿cuáles serían los valores del campo que contienen los bits de paridad para el caso de un esquema de paridad bidimensional? La respuesta debe minimizar la cantidad de bits de paridad utilizados

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |

→ siempre va a coincidir

→ 9 bits de paridad

$$\# \text{bits de paridad} = R(k) = n/k + k + 1$$

donde  $n$  es la cantidad de bits y  $k$  es la cantidad de columnas

Busco  $k$  que minimice  $R(k)$  → derivo e igualo a 0

$$R'(k) = -n/k^2 + 1 = 0$$

$$n/k^2 = 1$$

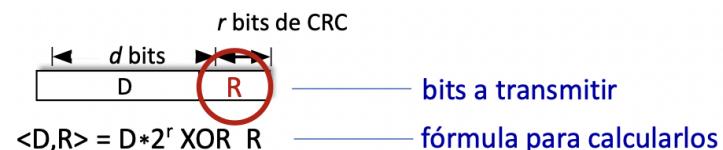
$$n = k^2 \rightarrow \sqrt{n} = k$$

## Checksumming Methods

- Emisor
  - interpreta el contenido del segmento UDP (incluyendo header + direcciones IP) como una tira de enteros de 16 bits
  - *checksum*: suma en complemento a uno del contenido del segmento
  - Este valor se coloca en el campo *checksum* del header UDP
- Receptor: Calcula el checksum del segmento recibido y comprueba si coincide con el del campo del header
  - si no coincide: error!
  - si coincide: no se detecta error, pero podría haberlo
- Detección de errores en la capa de enlace se implementa en software → Se utiliza checksum porque es importante contar con un esquema de detección de errores simple y rápido
- Detección de errores en la capa de enlace se implementa en hardware y pueden resolver cosas más complejas muy rápido. Por eso se usa CRC en la capa de enlace

## Cyclic Redundancy Check (CRC)

- Método de detección de errores + poderoso
- $D$ : bits de datos (interpretados como un número en base 2)
- $G$ : "patrón" de bits (generador) de largo  $r+1$   
→ viene dado por el método
- A los datos, les concatenamos  $R$  (algo que vamos a calcular a partir de  $G$ )
  - bits a transmitir =  $d$  bits de  $D$  +  $r$  bits de CRC
  - para componer el vector completo:  $\langle D, R \rangle = D * 2^r \text{ XOR } R \rightarrow$



bits a transmitir

fórmula para calcularlos

XOR es la suma sin carry (es decir, bit a bit, si coinciden pones un 0, sino un 1)

- objetivo: elegir r bits de CRC, R, tales que  $\langle D, R \rangle$  sea divisible por G (módulo 2)
  - el receptor conoce G: divide  $\langle D, R \rangle$  por G y detecta errores si obtiene un resto no nulo, caso contrario la data es aceptada por el receptor.
  - puede detectar todos los errores en ráfagas de a lo sumo r bits
  - muy usado en la práctica (ej: Ethernet y Wifi)
- Pasos: a los datos le agrego r ceros al final (esto es lo mismo que hacer  $D * 2^r$ ) , lo divido por el generador y obtengo el resto (R). calculo  $\langle D, R \rangle = D * 2^r \text{ XOR } R$  y esos son los bits que le transmito a B. B se fija si  $\langle D, R \rangle$  es múltiplo de G para detectar errores

### 6.3 Multiple Access Links And Protocols

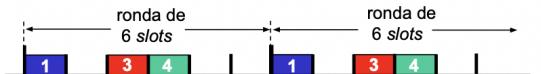
- Conexiones de tipo Point-to-point → Point to point protocol (PPP)
  - Fácil de enviar cosas, no hay colisiones
- Conexiones tipo Broadcast
  - Múltiples nodos conectados en simultáneo
  - Múltiples emisores y receptores
  - Se le dice broadcast ya que cuando un nodo envía un paquete, le llega a todos pero únicamente lo debería agarrar el receptor
  - Un ejemplo son las Ethernet o wireless LANs
  - Tiene el **problema del acceso múltiple**
    - ¿Quién manda y cuando?
  - Como humanos, desarrollamos ciertas reglas para la comunicación
    - Todos tienen la posibilidad de hablar
    - No hablar hasta que no se te haya hablado
    - No monopolices la charla
    - Levanta la mano si tienes una pregunta
    - No interrumpas
    - No te duermas en una charla
  - Las computadoras tienen multiple access protocols que son los que rigen las "conversaciones" entre ellas
  - Como todos los nodos pueden enviar frames, existe la posibilidad de que 2 o más manden al mismo tiempo y se produzca una **colisión**
  - Cuando ocurre esto, la información de los frames se pierde y ningún nodo receptor la puede recibir. También pasa que el canal se desperdicia durante ese tiempo
  - Si muchos nodos quieren transmitir frecuentemente, se van a producir múltiples colisiones y se va a desperdiciar mucho ancho de banda del canal
  - Para solucionar esto, se diseñaron protocolos de acceso compartido de varios tipos
    - Channel partitioning protocols → slots de tiempo o frecuencias
    - Random access protocol → Se permiten colisiones, pero provee mecanismos para solucionarlas
    - Taking turns protocols → Por turnos
  - Idealmente un protocolo debería cumplir con las siguientes ideas
    1. Si hay un único nodo conectado, debe poder hacerlo a tasa R bps

2. Cuando M nodos quieren transmitir, deben poder hacerlo a tasa promedio R/M
3. Descentralización. No debe haber un master node, punto de falla único. Ni tampoco debe haber sincronización ni relojes
4. Simplicidad. Debe ser fácil de implementar

## Channel Partitioning Protocols

### TDMA (Time Division Multiple Access)

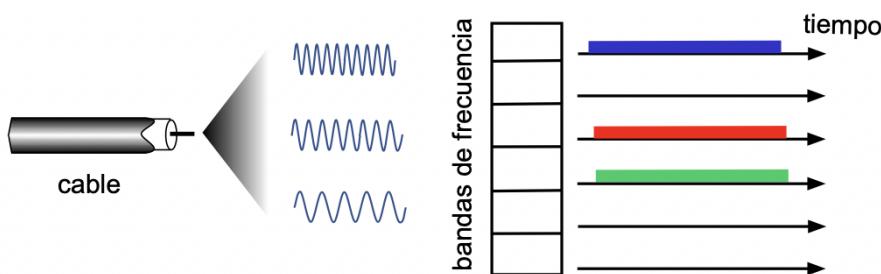
- Acceso al canal en “rondas” y cada “ronda” se divide en N (cant. nodos) slots
- Cada nodo tiene un slot de longitud fija en cada ronda (longitud: tiempo de transmisión de un frame/paquete)
- Los slots que no se usan quedan inactivos (idle)
- Ejemplo: LAN de 6 nodos donde 1,3,4 tienen paquetes para enviar; los slots 2,5,6 quedan inactivos



- ventajas:
  - no hay interrupciones ni colisiones → cuando le toca a un nodo, transmite con todo el ancho de banda del canal
  - es justo con la división del tiempo
  - fácil de implementar
- Desventaja:
  - ineficiente → pierdo capacidad de canal. Incluso si hay un sólo nodo que tiene paquetes para enviar, lo debe hacer a una tasa R/N bps (total de todos los slots)
  - cada uno tiene que esperar su turno
  - Sincronización (cada uno tiene que saber cuando le toca)

### FDMA (Frequency Division Multiple Access)

- El espectro del canal se divide en bandas de frecuencia (cada una con ancho de banda R/N). A cada nodo se le asigna una banda fija → crea N canales más pequeños de R/RN bps a partir de un único canal de R bps
- Puede haber intervalos de tiempo inactivos en cada banda
- Ejemplo: LAN de 6 nodos donde 1,3,4 tienen paquetes para enviar; las bandas 2,5,6 permanecen inactivas



- ventajas:
  - evita colisiones
  - no necesito sincronización
  - justo con la división del ancho de banda
  - fácil de implementar
- Desventaja:
  - ineficiente → un nodo está limitada a un ancho de banda de R/N bps

### CDMA (Code Division Multiple Access)

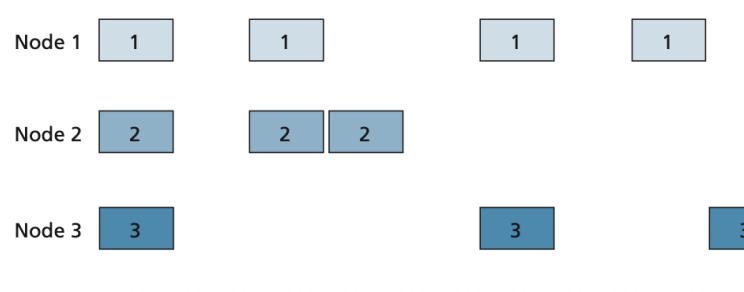
- Asigna un código diferente a cada nodo, quien lo usa para codificar los bits que manda
- Esto permite que varios nodos puedan transmitir simultáneamente y sus respectivos receptores reciban correctamente los bits (suponiendo que el receptor conoce el código del remitente)

### Random Access Protocols

- El nodo que quiere transmitir lo va a hacer a R bps
- No hay coordinación previa entre los nodos
- Si se produce colisión (si 2 o más nodos transmiten a la vez) → cada nodo involucrado retransmite su frame pero no de inmediato. *"Espera un delay random antes de retransmitir"*
  - como cada nodo elige un delay aleatorio independiente, es posible que elijan delays distintos y se evite la colisión cuando retransmiten
- Los protocolos de acceso aleatorio determinan
  - cómo detectar colisiones
  - cómo recuperarse ante colisiones (ej: vía retransmisión con retrasos)
- Protocolos de ejemplo:
  - ALOHA, slotted ALOHA
  - CSMA, CSMA/CD (Ethernet), CSMA/CA (WiFi)

### Slotted ALOHA

- Asumimos ciertos supuestos
  - Todos los frames son de L bits
  - El tiempo se divide en slots de L/R secs
  - Los nodos sólo pueden transmitir al inicio de un slot
  - Los nodos están sincronizados, saben cuando arranca cada slot
  - Si hay una colisión, todos los nodos la detectan
- Operatoria del protocolo
  - Cuando un nodo tiene un frame para enviar, espera al próximo slot y lo envía
  - Si no hay colisión, el nodo puede enviar otro frame
  - Si hay colisión, el nodo retransmite el frame con una probabilidad p en el próximo slot
  - Todos los nodos involucrados en una colisión "tiran una



Time

“moneda” para ver si transmiten en el próximo slot o no

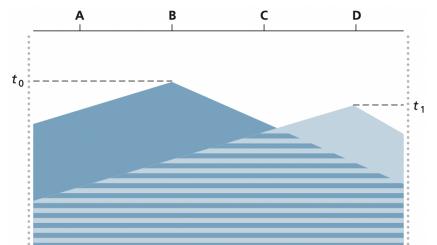
- Ventajas
  - Si solo hay un nodo, puede transmitir todo el tiempo a un rate R
  - Es descentralizado ya que los nodos detectan las colisiones y tiran la moneda para ver sin retransmiten
  - Sencillo y simple de implementar
- Desventajas
  - Colisiones y slots desperdiciados
  - Slots vacíos (cuando tiran la moneda y no les sale retransmitir)
  - Sincronización de relojes
- Eficiencia de Slotted Aloha
  - Eficiencia: fracción de slots exitosos en el largo plazo (muchos nodos, cada uno con muchos frames a transmitir)
  - Supongamos N nodos con muchos frames a enviar; cada uno transmite en un slot con probabilidad p
    - Probabilidad de que un nodo tenga éxito en un slot:  $p * (1 - p)^{N-1} \rightarrow$  proba de que le toque transmitir \* proba de que a todos los otros no les toque transmitir (para que no haya colisión)
    - Probabilidad de que cualquier nodo tenga éxito:  $N * p * (1 - p)^{N-1}$
    - Eficiencia máxima: encontrar  $p'$  que maximiza  $N * p' * (1 - p')^{N-1}$
    - Para muchos nodos, tomar el límite de  $N * p' * (1 - p')^{N-1}$  cuando  $N \rightarrow \infty$
  - **EFICIENCIA MÁXIMA:  $1/e = 0.37$**  → En el mejor de los casos, el canal se utiliza el 37% del tiempo
  - Con  $p = 1/n$  minimizó el número de colisiones, lo resolví en la guía
  - Con  $p = 1$  maximizar el número de colisiones

## Aloha

- Los slots en los nodos no están sincronizados. Apenas un nodo tenga un frame para transmitir, lo envía al canal.
  - si hay colisión → inmediatamente (después de que se transmita completamente su frame colisionado) transmite el frame con probabilidad p.
  - Si no hay colisión → espera un tiempo de transmisión de frames
- **EFICIENCIA MÁXIMA:  $1/2e = 0.185$**
- Es exactamente la mitad de eficiencia ya que al calcularla tengo que considerar que ningún nodo manda un frame en el intervalo  $[t_0-1, t_0]$  ni tampoco en el intervalo  $[t_0, t_0+1]$
- **ALOHA y Slotted ALOHA** → la decisión de un nodo de transmitir (o no) se toma independientemente de la actividad de los otros nodos. Es decir, un nodo no presta atención a si otro nodo está transmitiendo cuando comienza a transmitir, ni deja de transmitir si otro nodo comienza a interferir con su transmisión.

## Carrier Sense Multiple Access (CSMA)

- 2 principales reglas



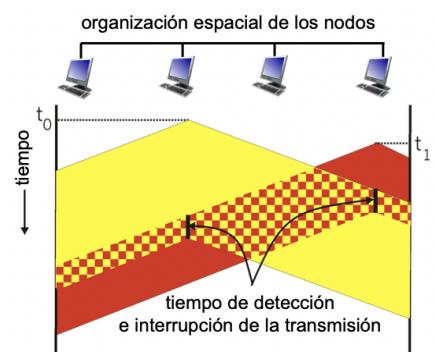
- Carrier sensing: Escuchar antes de transmitir, si un nodo está transmitiendo no transmiso. Espero a que termine
- Collision detection: Si estoy transmitiendo y detectar una colisión, paro de transmitir y espero un tiempo random para volver a transmitir
- Si todos los nodos siguen la regla de carrier sensing, ¿por qué se producirían colisiones?
  - Por el channel propagation delay. Los nodos tardan en enterarse que otro nodo está transmitiendo
- CSMA no hace uso del collision detection
  - Tanto B como D continúan transmitiendo hasta que terminan de enviar el frame
  - Al receptor le llega ruido, se mezclan ambos mensajes

### Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

- Los nodos que utilizan este protocolo, cuando detectan una colisión dejan de enviar su frame
- Esto hace que mejore la performance ya que hay menos tiempo desperdiciado
- El algoritmo CSMA/CD de Ethernet
  1. La NIC recibe un datagrama de la capa de red y crea un frame
  2. Luego, escucha el canal
 

Si está inactivo: comienza la transmisión del frame

Si está ocupado: espera a que quede inactivo y luego transmite
  3. Si la NIC transmite el frame completo sin colisiones, termina
  4. Si la NIC detecta otra transmisión en curso, aborta la transmisión y envía una señal de interferencia (cuando se detecta colisión → *jamming signal*)
  5. Luego de abortar, la NIC entra en exponential backoff. Espera un tiempo random, si fuese fijo ambos volverían a transmitir al mismo tiempo
    - Al cabo de la m-ésima colisión, la NIC elige un K al azar del intervalo  $[0, 1, 2, \dots, 2^m - 1]$ . La NIC espera  $K * 512$  tiempos de bit y se vuelve al paso 2 (escuchar el canal)
    - 512 tiempos de bits → tiempo que tarda (según la tasa de transmisión del enlace) en pasar 512 bits
    - A más colisiones, más largo el intervalo de backoff (tiempo muerto) → nodos esperan + tiempo para transmitir
- Notar que en el paso 5 debe esperar un tiempo aleatorio (en lugar de fijo) porque de lo contrario, si 2 nodos transmitieran frames al mismo tiempo y luego esperan el mismo tiempo para volver a transmitir, continuarán colisionando por siempre.
- **EJERCICIO:** Despues de la 5ta colisión en CSMA/CD
  - ¿Cuál es la probabilidad de que un nodo elija K=4 ?
    - En la 5ta colisión, la NIC elige un K entre  $[0,1,\dots,2^5 - 1] = [0,1,\dots,31]$
    - Prueba de que salga K = 4 →  $1/32$



- ¿A cuánto tiempo de espera equivale ese resultado en un link Ethernet de 10Mbps ?
  - $10.000.000 \text{ bits} \quad \text{-----} \quad 1 \text{ seg}$
  - $512 \text{ bits} \quad \text{-----} \quad x = \frac{1}{10.000.000/512} = \frac{512}{10.000.000} = 0.0000512 \text{ segs}$
  - $K = 4 \rightarrow \text{tiempo de espera} = \frac{512}{10.000.000} * 4 = 0,0002048 \text{ seg} = 204,8 \text{ micro segs}$
- **Eficiencia de CSMA/CD:** fracción de tiempo a largo plazo durante el cual los frames se transmiten en el canal sin colisiones
  - $t_{prop}$  = máximo delay de propagación entre dos nodos de la LAN
  - $t_{trans}$  = tiempo de transmitir un frame de tamaño máximo
$$\text{Efficiency} = \frac{1}{1 + 5t_{prop}/t_{trans}}$$
  - La eficiencia tiende a 1:
    - Cuando  $t_{prop}$  tiende a 0 (*abortedo instantáneo ante colisiones*)
    - Cuando  $t_{trans}$  tiende a infinito (*transmisión muy prolongada → el canal va a estar siendo productivo la mayor parte del tiempo*)
  - Mejor rendimiento que ALOHA siendo simple, barato y descentralizado

### Taking-Turns Protocols

- Aloha y CSMA cumplen con que si solo hay un nodo activo, este tiene la totalidad del canal a su disposición. Pero no cumplen que si hay M nodos, cada uno tenga R/M bps
  - Esta fue la motivación para la creación de los protocolos por turnos
- Vamos a hablar de 2 protocolos por turnos: Polling y token passing

### Polling

- Requiere que haya un nodo maestro que sondea entre los nodos con un round robin
- Le va dando a cada nodo una cantidad máxima de frames a enviar. Una vez que termina, se lo pasa al siguiente y así sucesivamente
- El nodo maestro puede detectar inactividad en el canal, le saca el turno al que estaba inactivo y se lo pasa al siguiente
- Ventajas
  - Elimina las colisiones
- Desventajas
  - Centralizado, endeble ante una falla en el nodo maestro
  - Tiene un polling delay. El tiempo que se pierde en decirle al nodo que le toca enviar
  - Si hay un único nodo, no va a transmitir a tasa R bps ya que el nodo maestro va a ir haciendo round robin entre los inactivos

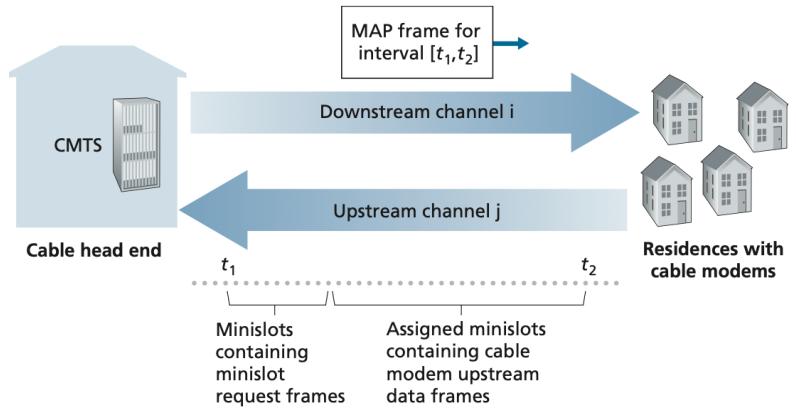
### Token-passing protocol

- No hay un nodo maestro, los nodos se van pasando un token en un orden determinado
- Cuando un nodo recibe el token, se lo queda únicamente si tiene frames para enviar y envía una cantidad de frames máxima predeterminada por el protocolo. Caso contrario lo pasa automáticamente
- Ventajas

- Descentralizado
  - Simple
  - Eficiente
- Desventajas
  - Único punto de falla, si el nodo que tiene el token se cae puede causar que se caiga toda la LAN ya que no va a pasar el token
  - Delay a la hora de pasar el token

### DOCSIS: The Link-Layer Protocol for Cable Internet Access

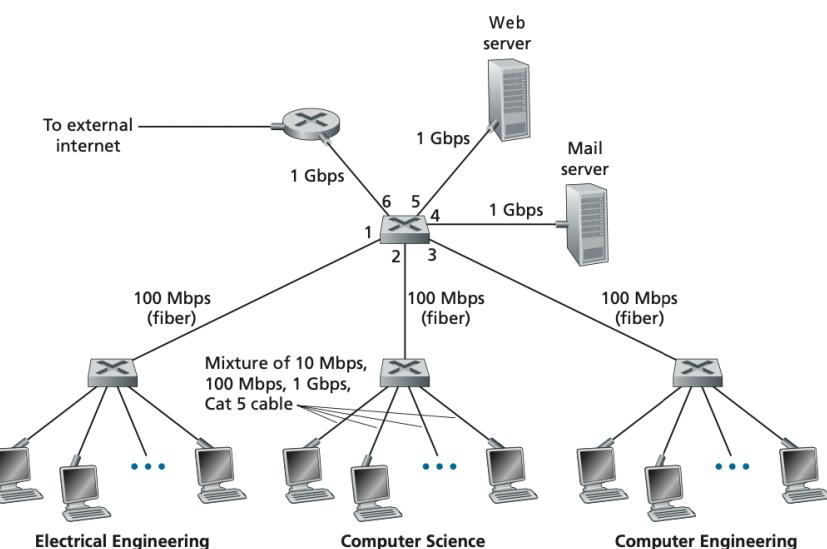
- Cable modem termination system (CMTS) conecta a miles de usuarios
- Usa FDM para dividir el cable tanto de bajada como de subida
- Los frames que envía el CMTS le llegan a los modems, pero al ser un canal end-to-end no hay problema de colisiones ni nada del estilo
- El cable de subida es un poco más interesante ya que deben ponerse de acuerdo todos los modems
- El canal está dividido en mini-slots de tiempo y el CMTS le va diciendo a cada módem cuando transmitir. Esto evita las colisiones. El CMTS se le manda un mensaje de tipo MAP al módem que le corresponde transmitir
- ¿Cómo sabe el CMTS qué módems quieren transmitir?
  - Hay un periodo de tiempo en el que los módems envían mini-slots requests
  - Pero en ese periodo, pueden haber colisiones de esos mensajes
    - Los módems lo solucionan igual que en CSMA/CD
- Podemos ver como un servicio como lo es el Internet usa múltiples protocolos de acceso al medio. Hace uso de FDM, TDM, random access protocols y centrally allocated time slots, todos al mismo tiempo.



### 6.4 Switched Local Area Networks

#### Area Networks

- Ejemplo de switched local network
- 3 deptos
- 2 servers



- 1 router con 4 switches
- Los switches solo manejan frames no datagramas
- No tienen IP, usan otra dirección. Tampoco usan algoritmos como OSPF para rutear

## Link-Layer Addressing and ARP

### MAC Address

- ¿Por qué se necesitan direcciones MAC si ya tenemos las IP?
  - Para poder direccionar en mi LAN
  - Son capas diferentes, en teoría enlace no debería tocar nada de red
- Dirección IP de 32 bits
  - Son dinámicas, van cambiando dependiendo a que router te conectes
- Dirección MAC de 48 bits
  - Son fijas, suelen estar grabadas en la ROM
  - Se expresan en hexadecimal. Ejemplo: 1A:2F:BB:76:09:AD
  - Espacio de direccionamiento es de  $2^{48}$
  - Se usan para direccionar frames dentro de una LAN de una interfaz a otra
  - Los switches no tienen dirección MAC
  - Cada MAC es única, no se repiten. De todas formas, si se llegasen a repetir no sería un problema ya que es muy poco probable que 2 interfaces con la misma MAC estén en la misma LAN
  - El IEEE es el encargado de asignarlas y las reparte por segmentos
  - Analogía
    - Dni: direc MAC
    - Domicilio: IP
  - El espacio de direcciones es chato, a diferencia de IP que es jerárquico
  - Dirección MAC broadcast FF-FF-FF-FF-FF-FF

### Address Resolution Protocol (ARP)

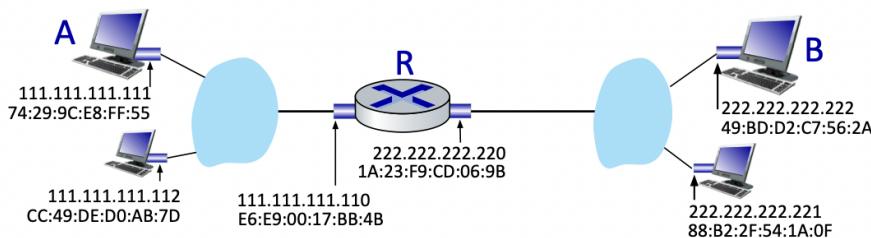
- Es el encargado de traducir direcciones IP a MAC
- Para enviar un datagrama se necesita la IP de destino
- Para enviar un frame se necesita saber la MAC de destino
- Solo puede conseguir las MAC de los nodos de la misma LAN

| IP Address      | MAC Address       | TTL      |
|-----------------|-------------------|----------|
| 222.222.222.221 | 88-B2-2F-54-1A-0F | 13:45:00 |
| 222.222.222.223 | 5C-66-AB-90-75-B1 | 13:52:00 |

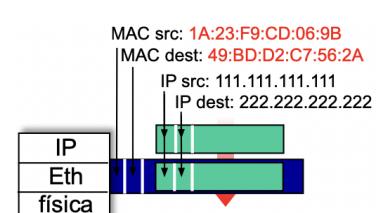
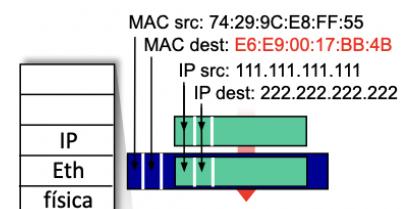
- Para conseguir la dirección MAC de un nodo, el sender crea un **ARP packet** que incluye varios campos
  - IP origen
  - MAC origen
  - IP destino

- MAC objetivo. En la ARP request/query va vacía. Ya que es donde irá la respuesta
- Encapsula el **ARP query** en un frame y lo manda broadcast hacia la subnet ya que no se conoce la MAC de destino
- Lo reciben todos y se lo pasan al **ARP module**. Cada uno de los modules se fija si es su propia IP la que está en el paquete y en caso de que coincida se manda la **ARP request** con el campo de MAC destino completado y se lo envía al que había hecho la request
- Cuando le llega el paquete, actualiza la tabla ARP y ya le pude enviar frames
- ARP es un protocolo mitad de red y mitad de enlace ya que maneja tanto direcciones IP como MAC
- ARP es un protocolo plug-and-play, no requiere ser configurado. Las tablas se llenan automáticamente
- Para llenar la tabla, se broadcaster a una ARP query en la capa de enlace con la dirección IP del host que quería enviarle el frame. Únicamente me va a responder el que tenga la IP de destino y me envía su MAC

### Sending a Datagram off the Subnet



- Notar que el router (R) tiene 2 direcciones IP y 2 direcciones MAC → una por cada interfaz
- Supuestos
  - A sabe la IP de B → con DNS o preguntándole
  - A sabe la dirección IP del gateway R → por DHCP
  - A sabe la dirección MAC de R → por ARP
  - R sabe la dirección MAC de B → por ARP
- **Pasos para enviar datagrama de un nodo A a otro nodo B vía R (B está en otra subnet)**
  1. A arma datagrama con dirección origen A y dirección destino B
  2. Lo encapsula en un frame con la MAC de origen propia y la dirección MAC de destino de la interfaz que me permite conectarme con el router (NO pongo como dirección MAC destino la de B, porque está en otra subnet → nadie lo aceptaría y se perdería)
  3. Envía el frame a toda su subnet
  4. Dirección MAC de R matchea con la de destino del frame. R recibe el frame, extrae el datagrama y lo entrega a IP
  5. R determina cuál de las 2 interfaces tiene que enviar el datagrama extraído a través de su *forwarding table* (le dice que debe hacerlo a través de la interfaz 222.222.222.220)



6. R encapsula el datagrama en un frame con MAC de origen propia (de la interfaz que se conecta con el host B) y la dirección MAC del host B (como destino)
  7. R envía el frame a la subnet 2
  8. Dirección MAC de B matchea con la de destino del frame. B recibe el frame y extrae el datagrama
- Emisor encapsula datagrama IP dentro de un Ethernet Frame y lo pasa a la capa física. Receptor recibe el frame y extrae el datagrama IP a la capa de red

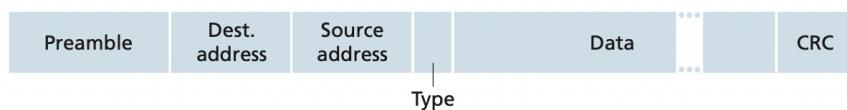
## Ethernet

- Es la tecnología más utilizada para direccionar dentro de una LAN
  - Simple y barata
  - Primera/pionera en el mercado
  - Se mantuvo vigente en cuanto a velocidad de transmisión (hoy por hoy llega a los 400 Gbps y se está trabajando en estándares de hasta 1.6 Tbps)
  - Único chip, múltiples velocidades
- Topología



- **Bus:** mismo cable que conecta todos los nodos (hasta mediados de los 90's)
  - Broadcast LAN → todos los frames transmitidos son procesados por todos los nodos conectados a esa LAN
  - desventaja:mismo dominio de colisión para los nodos
- **Switched:** Los nodos se conectan mediante switches de nivel 2 (disposición vigente hoy en día)
  - Ventaja: Los nodos no colisionan entre sí (xq cada cable solo se comparte entre 2 )

## Ethernet Frame Structure



- Preámbulo de **8 bytes**
  - 7 bytes de 10101010 seguidos por un último byte: 10101011 → para sincronizar los relojes (de lectura y escritura) de los interlocutores
  - al mandarte los dos 1 al final, te estoy avisando que empiezan los datos
- MAC destino y MAC origen → direcciones físicas de **6 bytes** cada una
  - Si el adaptador recibe un frame destinado a su MAC o bien a la MAC de broadcast (e.g., en ARP), entrega los lo que está en el campo de "datos" del frame al protocolo de red respectivo
  - Si recibe un frame con otra dirección MAC, el adaptador descarta el frame

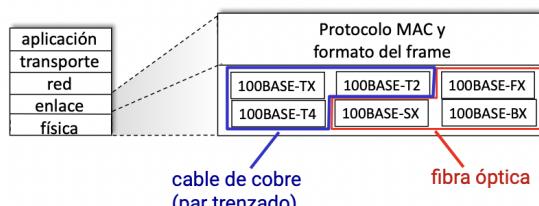
- Tipo de protocolo de nivel superior → **2 bytes**
  - Principalmente IP pero existen otros (e.g., Novell IPX o AppleTalk)
  - Utilizado para demultiplexar en el receptor (para saber a qué protocolo de red enviar la data)
- Datos (payload) de **46 a 1.500 bytes** → acá va el datagrama
  - si es más grande, IP lo fragmenta
  - si es más chico, se agrega basura → capa de red usa el campo de longitud del header del datagrama IP para eliminar el “relleno”
- CRC: 32 bits de redundancia de CRC (**4 bytes**)
  - El receptor descarta el frame si detecta errores

### Servicio Ethernet

- Proveen un servicio **no orientado a conexión** a la capa de red: no existe un proceso de *handshaking* entre interlocutores
- Brindan un servicio **poco confiable** a la capa de red → nodo receptor utiliza CRC para detección de errores, pero no envía ACKs o NAKs
  - Cuando detecta un error, simplemente lo descarta. El emisor no sabe si el frame llegó sin errores o no.
  - Flujo de datagramas pasados a la capa de red puede tener lagunas (puede ser significativo o no dependiendo si la aplicación corriendo en el host receptor es TCP o UDP)
  - Los datos en los frames descartados se recuperan sólo si el emisor emplea protocolos confiables (ej: TCP) en niveles superiores
    - Ethernet retransmite datos sin saber si está transmitiendo un datagrama nuevo con datos nuevos o un datos que ya se han transmitido alguna vez
- Protocolo de acceso al medio (MAC): **CSMA/CD con exponential backoff**

### Tecnologías Ethernet

- **Standards Ethernet IEEE 802.3**
  - Existen múltiples *standards* de Ethernet → 10BASE-T, 10BASE-2, 100BASE-T (el número del principio indica la velocidad)
  - Distintas velocidades: 2 Mbps, 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps, 40 Gbps
  - Todos comparten el protocolo MAC y el formato del frame
  - Distintos medios de nivel físico: fibra o cable coaxil



- En una LAN Ethernet switcheada (predominante hoy en día), **no** sería necesario un protocolo MAC ya que no hay colisiones. El switch coordina sus transmisiones y nunca envía más de un frame a la misma interfaz en ningún momento. Además, los switches suelen ser full-duplex por lo que un nodo y el switch pueden enviarse frames al mismo tiempo sin interferencia.

## Link-Layer Switches

- El trabajo de un switch es recibir un frame de un host y retransmitirlo hacia donde corresponde. Almacena en buffers y reenvía frames ethernet.
  - Inspecciona las MAC y determina si enviarlo a 1 host o broadcast
- Los switches son **transparentes** para los hosts, no saben que existe.
  - No tienen dirección IP ni MAC
- Los switches son **Plug-and-Play**
  - No requieren de configuración
- Se utiliza el protocolo Ethernet en cada enlace:
  - Sin colisiones: *full-duplex* (cualquier interfaz puede enviar y recibir al mismo tiempo)
- **Switching:** pueden coexistir transmisiones de A hacia A' y de B hacia B' sin colisiones. Pero no de A hacia A' y de C hacia A'.

## Forwarding and Filtering

- **Filtering:** función del switch que determine si un frame debe ser reenviado o descartado
- **Forwarding:** función del switch que determina a qué interfaz se debe dirigir el frame (en base a la dirección MAC, no IP) y luego lo mueve a dicha interfaz.
- Cada switch tiene una **switch table**: tabla de *forwarding* pero en switches. Cada entrada contiene 3 campos
  - Dirección MAC : para host/routers de la LAN (no necesariamente todos tienen una entrada)
  - Interfaz: que conduce a dicha dirección MAC
  - Timestamp: horario en la que la entrada fue puesta en la tabla

| Address           | Interface | Time |
|-------------------|-----------|------|
| 62-FF-F7-11-89-A3 | 1         | 9:32 |
| 7C-BA-B2-B4-91-10 | 3         | 9:36 |

- **Algoritmo de forwarding en switches**

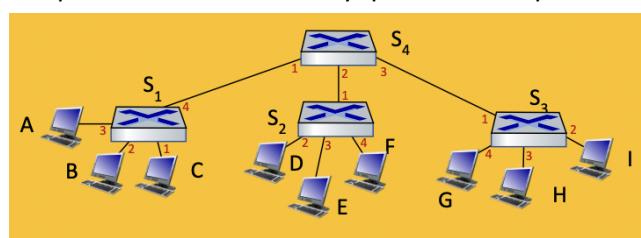
Al recibir un frame

1. Registrar la MAC y la interfaz por la que entró el frame del emisor
2. Indexar la tabla de *forwarding* empleando la MAC destino
  - Si todavía no hay una entrada en la tabla con dicha dirección, *flooding* (reenviar el frame a todas las interfaces excepto a la interfaz por la que llegó) → switch broadcastea el frame. **Flooding**
  - Si ya existe una entrada en la tabla asociando la MAC destino con la interfaz de entrada del frame, se descarta el frame → **filtering** (no hay necesidad de reenviar el

- frame a otra interfaz, ya que el host/router con la dirección MAC de destino se encuentra en la misma interfaz por la que vino el frame)
- Si ya existe una entrada en la tabla asociando la MAC destino con una interfaz diferente a la de entrada del frame, se reenvía el frame por la interfaz indicada en la tabla.
  - Siempre que la tabla esté completa y sea precisa, el switch reenvía frames hacia los destinos sin ningún broadcast.
  - ¿Cómo se generan y administran las entradas de la tabla? Autoaprendizaje (se crea de forma automática, dinámica y autónoma sin intervención de nadie)

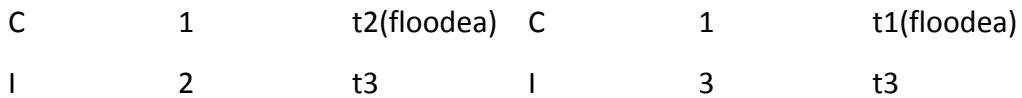
### Self-Learning

- Los switches infieren qué hosts pueden alcanzarse a través de cuáles interfaces
- Funcionamiento:
  1. La tabla del switch está vacía
  2. Cada vez que recibe un frame, el switch almacena en una entrada de su tabla: (1) dirección MAC del nodo emisor, (2) interfaz por la que llegó el frame y (3) la hora. De esta manera, el switch aprende la ubicación del emisor
  3. El switch elimina una entrada de la tabla si no recibe frames con esa dirección MAC origen después de un tiempo (*aging time*). De esta manera, si una PC se reemplaza por otra PC (con un adaptador distinto → distinta dirección MAC), la dirección MAC de la PC original eventualmente se eliminará de la tabla del switch.
- Si todos los nodos de la LAN envían frames, la tabla estará completa.
- Los switches únicamente conocen las interfaces de los hosts que le envían frames
- Los switches pueden interconectarse
- Ejercicio: Supongamos que C envía un frame a I y que éste le responde.



Mostrar las tablas de *forwarding* y el reenvío de paquetes en los 4 switches de la LAN

| S1    |          |        | S2    |          |             |
|-------|----------|--------|-------|----------|-------------|
| Direc | Interfaz | Tiempo | Direc | Interfaz | Tiempo      |
| C     | 1        | t0     | C     | 1        | t2(floodea) |
| I     | 4        | t5     |       |          |             |
| S3    |          |        | S4    |          |             |
| Direc | Interfaz | Tiempo | Direc | Interfaz | Tiempo      |



- Una vez que el frame que envía C llega a S1, este hace flooding (a todas sus interfaces, menos a la interfaz por la que llegó: 2,3 y 4)
  - agrego esa entrada a la tabla de s4
- cuando llega la entrada a s4, como no tenía ninguna entrada, hace flooding (a todas sus interfaces, menos a la interfaz por la que llegó: (2 y 3))
  - agrego esa entrada a la tabla de s2 y s3

### Properties of Link-Layer Switching

- **Eliminación de colisiones:** En una LAN switcheada, no se desperdicia ancho de banda por colisiones, ya que los switches almacenan frames y nunca transmiten más de un frame en un segmento a la vez → mejor rendimiento que LANs con enlaces broadcast
- **Enlaces heterogéneos:** Como el switch aísla un enlace de otro, cada enlace puede operar a diferentes velocidades y ejecutarse en diferentes medios → ideal para mezclar equipos viejos con nuevos.
- **Administración:** facilita administración de red
  - si una adaptador anda mal y no deja de mandar frames, el switch puede desconectar internamente a dicho adaptador)
  - un corte de cable solo desconecta al host conectado a esa interfaz para comunicarse con el switch (en lugar de que se caiga toda la red)
- **Switch Poison:** Un atacante se conecta a la LAN y empieza a mandar frames con direcciones MAC truchas para llenar la tabla. De esta forma, el switch va a floodear todos los frames legítimos ya que la tabla está llena de MACs inexistentes y el atacante va a poder acceder a todos esos frames

### Switches vs Routers

- Ambos son Store-and-Forward y tiene tablas de ruteo
- Router examina header de red, mientras que Switch de enlace
- Switches
  - Ventajas
    - Plug and play → auto aprendizaje
    - Filtering y forwarding
    - Solo procesan hasta capa de enlace, más rápido
  - Desventajas
    - Está restringido a spanning tree, para evitar loops
    - LANs grandes requieren tablas de ARP grandes y enlentece el tráfico

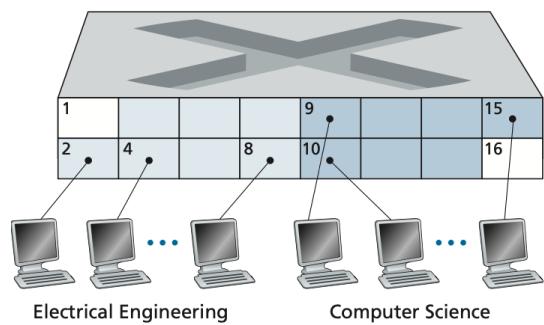
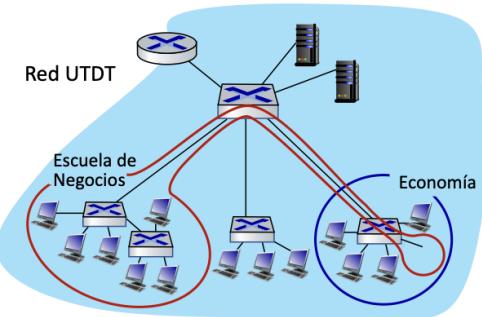
|                   | Hubs | Routers | Switches |
|-------------------|------|---------|----------|
| Traffic isolation | No   | Yes     | Yes      |
| Plug and play     | Yes  | No      | Yes      |
| Optimal routing   | No   | Yes     | No       |

- Susceptibles a broadcast storms, si un host envía infinitos frames broadcast colapsa la LAN
- Routers
  - Ventajas
    - Jerarquía evita loops
    - No está restringido a un spanning tree, puede haber múltiples rutas de un host hacia otro. Se crean múltiples topologías ricas
    - Provee firewall
    - Algoritmos de ruteo muy eficientes
  - Desventajas
    - Necesita configuración
    - Mayor delay ya que procesan hasta capa de red

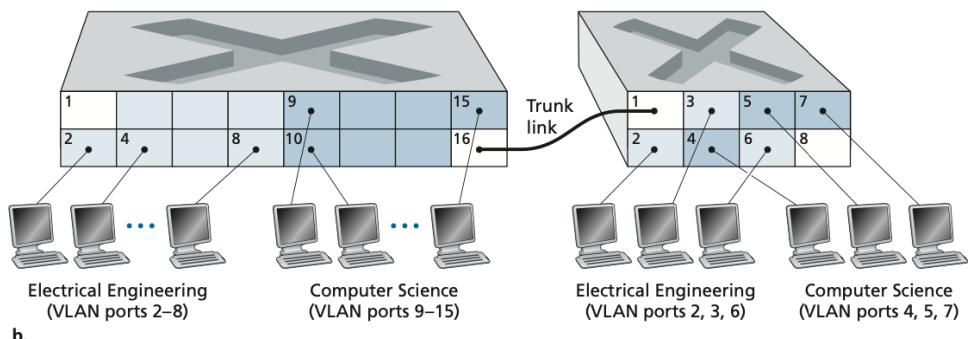
## Virtual Local Area Networks (VLANs)

### Motivación

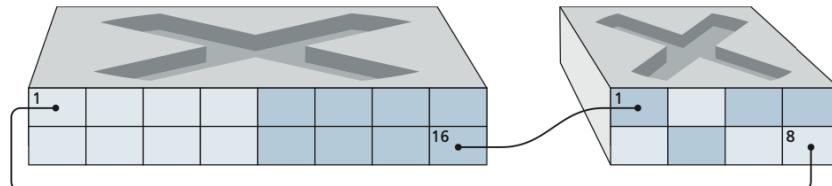
- En las LANs comunes hay un dominio de broadcast único
  - Todo el tráfico broadcast atraviesa toda la LAN
    - Esto genera problemas de seguridad (no quiero que todos vean los frames broadcast), de tráfico y de eficiencia
    - Uso ineficiente de los switches, si cada departamento pequeño tiene uno propio. Se podrían unificar en uno solo
  - Problemas de administración
    - Si un usuario de cambia de oficina pero quiere seguir conectado a la red de su departamento, habría que tirar un nuevo cable
- Para solucionar esto, se implementaron las VLANs
- Los switches con soporte de VLANs pueden configurarse de forma tal que definan múltiples LANs virtuales sobre una única infraestructura de LAN física
- Los hosts pertenecientes a una VLAN se comunican entre sí como si tuvieran un switch propio y no se dan cuenta de que en realidad comparten el switch con otros
- Port-Based VLAN
  - Los puertos del switch se agrupan vía software de forma tal que dicho dispositivo físico opere como múltiples dispositivos virtuales
  - EE y CS frames están aislados entre sí, aunque usen el mismo switch. (Más barato)
  - Si un estudiante de CS se conecta al puerto 8, se puede configurar vía software para que ese puerto pertenezca a la VLAN de CS
  - Separación de tráfico: los frames desde/hacia los puertos 1-8 sólo pueden alcanzar los puertos 1-8
    - También se pueden definir VLANs a partir de las MACs



- Membresía dinámica: los puertos pueden ser asignados dinámicamente
- Forwarding entre VLANs: vía router (como si fuesen switches diferentes)
  - Los fabricantes venden hardware combinado
- VLANs entre múltiples switches
  - CS tiene oficinas en 2 edificios diferentes y quieren tener acceso a la misma red, ¿cómo hacemos ahora?
  - **Trunking:** mecanismo para interconectar 2 switches VLAN
    - Se usa un único puerto especial (trunk port) para conectar los 2 switches
    - Los frames que van entre trunk ports tienen un formato especial. Lo regula el protocolo 802.1Q y tiene algunos campos de más en el header



- Otra forma sería conectar cada VLAN del switch 1 con su VLAN del switch 2 pero NO es escalable ya que para N VLANs necesitas N cables y en cambio con el trunk port solo se necesita 1 puerto por switch.

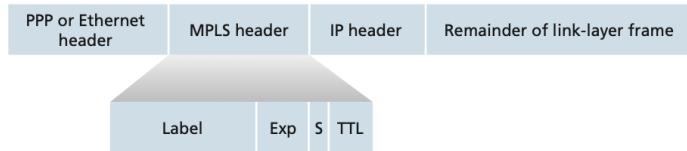


## 6.5 Link Virtualization: A network as a Link Layer

- No lo vimos en clase
- host ven los enlaces/cables simplemente como un canal de la capa de enlace que conecta 2 o más hosts
- desde el punto de vista de Internet, MPLS es una tecnología de la capa de enlace que sirve para interconectar dispositivos IP

### Multiprotocol Label Switching (MPLS)

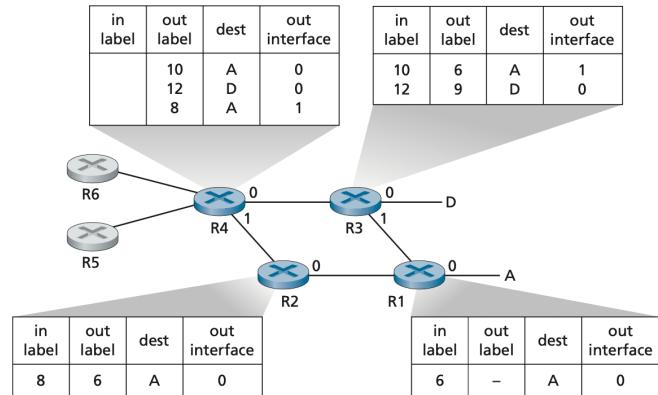
- Objetivo: forwarding de IP de alta velocidad entre la red de routers compatibles con MPLS, utilizando una etiqueta de longitud fija (en lugar de shortest prefix matching)
  - búsqueda más rápida utilizando un identificador de longitud fija, pero el datagrama IP aún conserva la dirección IP
  - Funciona de la mano con IP, usando direccionamiento y enrutamiento de IP
  - técnicas de Virtual Circuit (VC) en una red de datagramas enrutados.
- Formato frame de la capa de enlace que es manejado por un router compatible con MPLS



- Header MPLS entre el header de la layer-2 (Ethernet) y el header de la layer-3 (IP)
- contiene: label + 3 bits reservados para uso experimental + 1 bit S (para indicar el final de una serie de header MPLS “apilados”) y un campo TTL
- frames MPLS sólo se pueden transmitir entre **routers compatibles con MPLS**
  - Label-switched router
  - reenvía paquetes a la interfaz de salida basándose únicamente en el valor de la etiqueta (no inspecciona dirección IP) → busca etiqueta del frame MPLS en su tabla de forwarding y lo dirige a la interfaz correspondiente
  - Forwarding table MPLS distinta a IP forwarding tables
- ¿Cómo sabe un router si su vecino es compatible con MPLS? ¿Cómo completa un router su tabla de forwarding?

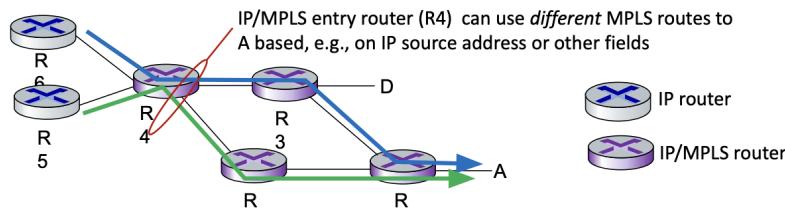
Ejemplo: routers R1 a R4 son compatibles con MPLS, R5 y R6 son routers IP estándar

1. R1 le avisa a R2 y R3 que puede llegar a A y que los frames MPLS que le lleguen por la interfaz 6 serán enviadas a dicho destino
2. R3 le avisa a R1 que puede llegar a A y a D (y las interfaces)
3. R2 le avisa a R1 que puede llegar a A (y la interfaz )
4. R4 tiene 2 caminos MPLS para llegar a A: por la interfaz 0 con la etiqueta MPLS de salida 10 ó por la interfaz 1 con la etiqueta 8



En este ejemplo, los dispositivos IP R5, R6, A y D están conectados entre sí a través de una infraestructura MPLS

- Ventaja (además de mayor velocidad): capacidad de gestión de tráfico.
  - protocolos de ruteo IP sólo especificarían una única ruta (la de menor costo), determinada sólo por la dirección de destino
  - MPLS permite reenviar paquetes a lo largo de rutas que no serían posibles utilizando protocolos IP (por ejemplo, en el ejemplo de abajo R4 tiene 2 rutas MPLS a A) → las decisiones de reenvío de MPLS pueden diferir de las de IP
  - **Ingeniería de tráfico:** Usar direcciones de origen y destino para enrutar flujos al mismo destino de manera diferente → forzar que una parte del tráfico vaya por una ruta y otra parte por la otra ruta (al mismo destino)
  - redirige flujos rápidamente si falla el enlace: rutas de respaldo calculadas previamente



IP router  
IP/MPLS router

## 6.6 Data Center Networking

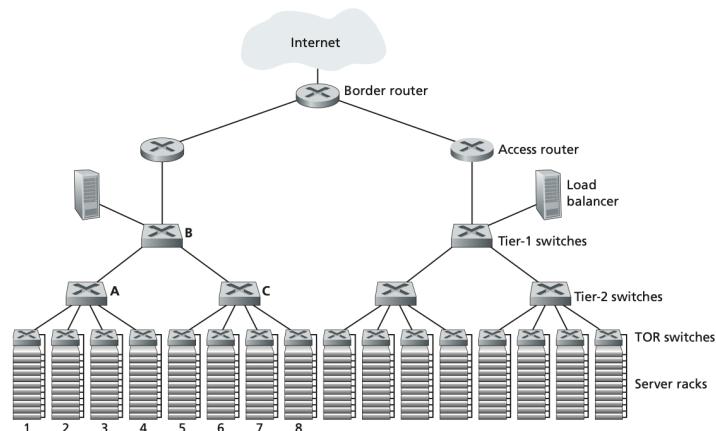
- Un datacenter no está únicamente conectado a Internet, tiene una compleja red que interconecta miles hosts internamente
- Cumplen varios propósitos
  - Proveen y almacenan contenido como páginas web, video streaming entre otras
  - Comercio electrónico
  - Cloud computing
- Algunos desafíos que enfrentan son: manejo de múltiples app, balanceo de cargas y confiabilidad

## Data Center Architectures

- Los diseños de los mismos son secretos de empresa ya que muchas veces es lo que los diferencia de la competencia
- Un datacenter grande cuesta alrededor de 12 millones USD por mes
  - Gastos de hardware, infraestructura(ventiladores), electricidad entre otros
- Manejan 2 tipos de tráfico
  - Dentro del datacenter
  - Clientes externos con datacenter → entra por el border router

## Topología jerárquica

- Rack de servidores: 20/40 blades
- Switch TOR (top of the rack)
  - Hay 1 por rack. Conexiones entre racks y entre racks y Tier 2
- Switches Tier-2
  - Conectan 16 TORs aprox
- Switches Tier-1
  - Conectan aprox 16 Tier-2
- Border Router
  - Brindan conectividad con el exterior
- Load Balancer
  - Recibe requests de clientes externos y dirige el tráfico interno del datacenter para que no haya sobrecargas
  - Vendría a ser como un policía de transito
  - También funciona como NAT, traduce la IP pública del datacenter a las privadas del host/blade adecuado
  - Evita que los clientes se conecten directamente a los blades. Brinda mayor seguridad
  - Por ultimo, devuelve los resultados al cliente (esconde la complejidad del datacenter)



## Técnica para evitar problemas de escala y falta de throughput

- Si la única forma de que un rack se conecte con otro es vía un tier-2 o tier-1 por un único camino, se va a generar mucho tráfico y enlentecer la red



- Para solucionar eso, hay redundancia de conexiones entre racks, TORs y Tier 1 y 2
  - Múltiples rutas posibles
  - Mayor velocidad
  - Mayor confiabilidad

## 6.7 Retrospective: A Day in the Life of a Web Page Request

- En esta sección vamos a ver todos los pasos necesarios para que un cliente, Bob, consiga una página web

### Paso 1: DHCP, UDP, IP y Ethernet. Conexión a la red

- Conecta la compu con un ethernet al switch y al router. Este último está conectado al ISP de la escuela.
  - El ISP de la escuela es comcast.net y tambien provee el servicio DNS
- Bob utiliza el protocolo DHCP para obtener una IP propia
  1. DHCP request, viaja en UDP hacia el puerto 67/68. A su vez se encapsula en un datagrama IP, con ip origen: 0.0.0.0 y ip destino: 255.255.255.255 (broadcast)
  2. El datagrama se encapsula en un frame Ethernet con MAC origen: 00:16:D3:23:68:8A y MAC destino: FF:FF:FF:FF:FF:FF (broadcast). El frame se lo envía a toda la LAN, esperando que lo reciba el servidor DHCP
  3. El frame va al switch y este lo broadcastea a todas las otras salidas
  4. El servidor recibe el paquete, y demultiplexa de Ethernet a IP, luego a UDP y por último a DHCP
  5. El servidor arma un paquete DHCP ACK que contiene la IP del cliente, la del gateway router y la del servidor DNS. También se envía cuál es la máscara y algunos otros datos. Encapsula el mensaje en UDP, IP y por último ethernet con MAC destino: 00:16:D3:23:68:8A
  6. Se manda el frame al switch, vía unicast. El switch ya conoce donde está Bob vía self-learning.
  7. Bob demultiplexa el frame, ya tiene su IP, la del DNS y la del Gateway. Actualiza tabla de forwarding. A este punto, Bob ya está conectado a la red

### Paso 2: DNS y ARP. Obtención de MAC gateway

8. Bob consigue la IP de google vía DNS. Se encapsula la query en UDP hacia el puerto 53 y se lo envió al servidor DNS
9. Lo encapsula en IP y luego en Ethernet con MAC de destino el gateway. NO la tengo

10. La consigo vía ARP. Creo ARP query con IP destino: la del gateway y encapsula la query en ethernet con MAC destino FF:FF:FF:FF:FF:FF (broadcast)
11. El gateway recibe el Frame y se da cuenta que la IP del datagrama coincide con la suya así que lo sigue demultiplexando. Arma el ARP reply con su MAC y envía el frame a Bob
12. Bob recibe el frame y actualiza su tabla ARP→ IP: gateway IP a MAC: gateway MAC
13. Bob ya puede iniciar la resolución del dominio de google ya que tiene la IP y MAC del gateway y la IP del servidor DNS.

#### **Paso 3: Resolución de dominio DNS**

14. El gateway envía la DNS query al ISP mirando su tabla de forwarding
15. El ISP demultiplexa el frame y se da cuenta que se lo tiene que enviar al servidor DNS ya que tiene su IP en el datagrama como destino
16. Le llega la query al servidor DNS, y resuelve el dominio. Por suerte la tenía cacheada. Crea el DNS reply con la IP de google, lo encapsula y se lo envía a Bob
17. Finalmente, Bob ya tiene la IP de google luego de demultiplexar el paquete enviado por el servidor DNS

#### **Paso 4: TCP y HTTP**

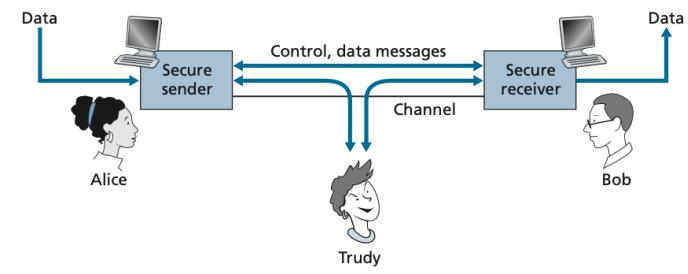
18. Bob crea un socket TCP hacia google para luego mandar el HTTP GET. Pero antes debe establecer la conexión TCP con google. Se envía un TCP SYN al puerto 80 (HTTP) y se encapsula IP con IP destino: la conseguida por DNS y luego en ethernet con MAC destino: gateway, para que se lo envíe al google
19. El gateway, usando sus forwarding tables (completadas por protocolos de ruteo como BGP) envía el paquete a google
20. Google recibe el TCP SYN y envía un TCP SYN ACK
21. Bob recibe el SYN ACK, la conexión queda establecida
22. Bob envía el HTTP GET, vía el socket, TCP, IP y ethernet
23. El servidor de google recibe el GET y envía un HTTP response cuyo body contiene el HTML de la página solicitada
24. Bob recibe el response y finalmente puede ver la página solicitada en el buscador de google

## **Capítulo 8: Security in Computer Networks**

### **8.1 What is Network Security?**

- Principios básicos de la seguridad:

- Confidencialidad: solo el receptor y el emisor deben poder entender el contenido de los mensajes. El emisor encripta los mensajes y el receptor los desencripta. De esta forma un intruso no lo puede entender
  - Integridad del mensaje: Ambos se quieren asegurar que el contenido de los mensajes enviados no fue alterado ya sea por un agente malicioso o por accidente
  - Autenticación: Tanto el emisor como el receptor buscan confirmar la identidad del otro
  - Disponibilidad: los servicios deben ser accesibles y estar disponibles para los usuarios
- Alice y Bob desean comunicarse de manera segura
  - Trudy, con malas intenciones, puede interceder en la comunicación y hacer algunas de estas cosas
    - Sniffing o snooping: interceptar mensajes
    - Inyectar mensajes a la comunicación
    - Spoofing: Impersonar otros actores (hacerse pasar por un servidor)
    - Hijacking: tomar control de la comunicación, eliminando a un interlocutor y tomar su rol
    - Denegar el servicio



## 8.2 Principles of Cryptography

- El texto plano luego de ser encriptado se lo llama ciphertext
- Alice encripta haciendo  $K_a(M)$  y Bob desencripta haciendo  $K_b(K_a(M))$  y obtiene  $M$ 
  - Donde  $M$  es el mensaje original
  - $K_a$  y  $K_b$  son las claves y  $K_a(M)$  es el mensaje encriptado

## Symmetric Key Cryptography

- **Cifrado Cesar:** a cada letra del mensaje se la reemplaza por la letra que está  $k$  posiciones delante. Con  $k = 3$ , la A se reemplaza por la D
  - No es muy efectivo, solo hay 25 combinaciones totales
- **Cifrado monoalfabético:** Se establece un mapeo random para cada letra pero a diferencia de César, no sigue un patrón regular
  - Mejor que Cesar, hay 26! posibles mapeos
- Ambos son métodos de sustitución, por lo que con un poco de análisis estadístico del texto cifrado es fácil obtener el texto plano. (Letras más usadas, terminaciones comunes)
- Diferentes ataques:
  - Ataque de texto cifrado: El intruso solo tiene el mensaje encriptado
    - Va a aplicar métodos de fuerza bruta → ir probando claves
    - Aplica análisis estadístico
  - Ataque de texto plano conocido: el intruso conoce algunos mapeos de (letra, letra cifrada) → algún nombre particular, fecha.

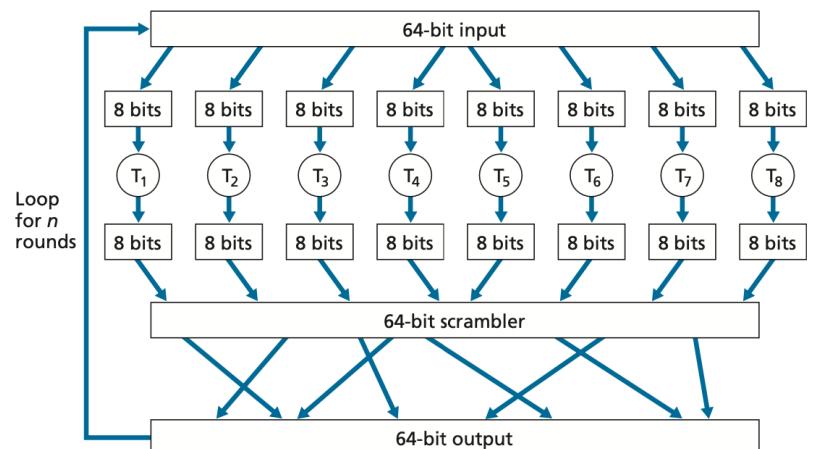
- Ataque de texto plano elegido: el intruso puede obtener el ciphertext de un texto plano a elección (oráculo)
- **Cifrado polialfabético:** Mejora del cifrado monoalfabético, se usan múltiples cifrados monoalfabéticos y dependiendo de la posición se usa cierto alfabeto. La ventaja que presenta es que la misma letra va a aparecer codificada de maneras diferentes.
  - Se usan 2 cifrados cesar y un patrón cíclico que determina para cada posición que k usar. Por ejemplo: C1, C2, C2, C1, C2.
  - Tanto el emisor como el receptor deben conocer los mapeos de alfabetos y el patrón utilizado

|                               |                                                     |
|-------------------------------|-----------------------------------------------------|
| <b>Plaintext letter:</b>      | a b c d e f g h i j k l m n o p q r s t u v w x y z |
| <b>C<sub>1</sub>(k = 5):</b>  | f g h i j k l m n o p q r s t u v w x y z a b c d e |
| <b>C<sub>2</sub>(k = 19):</b> | t u v w x y z a b c d e f g h i j k l m n o p q r s |

### Cifrado por bloques/ Block ciphers

- El mensaje a ser encriptado se procesa en bloques de k bits y cada bloque se encripta de manera independiente
- Se arma una tabla de mapeo en donde cada input de k bits(plaintext) se mapea a un output de k bits(ciphertext)
- Un mismo input siempre recibe el mismo output
- 2 inputs diferentes no reciben el mismo output
- Hay  $2^k$  inputs posibles, por lo que hay  $2^k$  combinaciones para los outputs
- Tanto el emisor como el receptor debe conocer la tabla de mapeo
- Se suelen usar bloques de 64 bits por lo que es imposible hacer fuerza bruta para descifrar
  - El mayor problema de esto es mantener la tabla de mapeo con  $2^{64}$  inputs values
- Se soluciona haciendo divide and conquer y otras técnicas más
- Algunos algoritmos de cifrado en bloques famosos DES(utiliza claves de 56 bits y bloques de 64) y AES(usa claves de 128, 192 o 256 bits y bloques de 128). Siguen la idea de la foto

| input | output | input | output |
|-------|--------|-------|--------|
| 000   | 110    | 100   | 011    |
| 001   | 111    | 101   | 010    |
| 010   | 101    | 110   | 000    |
| 011   | 100    | 111   | 001    |



- Son algoritmos estandarizados y conocidos. Más que utilizar tablas predeterminadas usan funciones
- La clave determina el mapeo de las mini tables (T1, T2, Tn)

### Cipher-Block Chaining

- Los block cipher encriptan el mismo input al mismo output → peligroso
  - Un atacante puede descifrarlos
- Lo que se hace aca es para cada bloque de ciphertext(generado con cifrado por bloques) se genera un numero random y se la hace un XOR → de esta forma si hay 2 bloques con mismo input, no tendrán el mismo output ya que el número random muy difícilmente coincida
- El emisor cifra los bloques y le envía al receptor los bloques cifrados junto con los números random (estos últimos no van cifrados)
- Para desencriptar los bloques, el receptor necesita los bloques cifrados, los números random y la clave simétrica

Problema de la criptografía simétrica: Las 2 partes tienen que ponerse de acuerdo en la clave → ¿cómo se la pasan?

Para resolver este problema surgió la criptografía asimétrica, con un enfoque totalmente distinto

### Public Key Encryption

- Cada host tiene una clave pública  $K_b^+$  que la conoce todo el mundo y una clave privada  $K_b^-$  que solo la conoce el host
- Si Alice le quiere enviar un mensaje a Bob, lo encripta con la clave pública de Bob y se lo envía → Bob lo va a recibir y lo va a desencriptar con su clave privada.  $K_b^- (K_b^+ (M)) = M$
- Dada  $K_b^+$  la clave pública, debe ser computacionalmente inviable derivar  $K_b^-$
- También se debe poder encriptar y desencriptar de la forma inversa  $K_b^+ (K_b^- (M)) = M$
- Con este método de encriptación, cualquiera puede enviarle mensajes a Bob ya que todos conocen su clave pública → vamos a requerir alguna forma de autenticación (firma digital)

### RSA

- Propiedades de módulo
  - $[(a \text{ mod } n) + (b \text{ mod } n)] \text{ mod } n = (a+b) \text{ mod } n$
  - $[(a \text{ mod } n) - (b \text{ mod } n)] \text{ mod } n = (a-b) \text{ mod } n$
  - $[(a \text{ mod } n) * (b \text{ mod } n)] \text{ mod } n = (a*b) \text{ mod } n$
  - $(a \text{ mod } n)^d \text{ mod } n = a^d \text{ mod } n$
- El mensaje a enviar es una secuencia de bits que puede ser representada como un entero
  - En RSA, cifrar un mensaje equivale a cifrar un número entero
- Generación de claves:
  - Elegir 2 números primos p y q grandes
  - Calcular  $n = p*q$  y  $\phi = (p-1)*(q-1)$

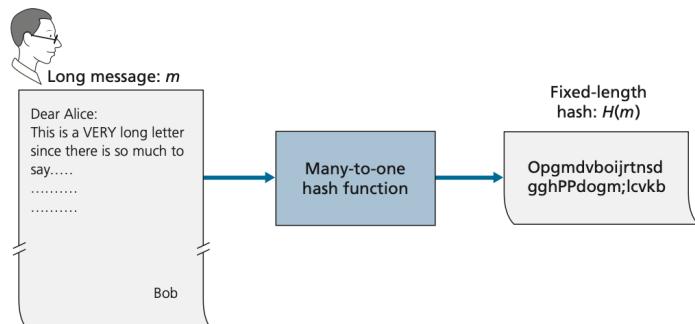
- Elegir  $e < n$  de modo tal que no posea factores comunes con  $z$  ( $e$  y  $z$  se dicen coprimos)
  - Elegir  $d$  de modo tal que  $ed-1$  sea divisible por  $z$  (es decir,  $ed \bmod z = 1$ )
  - La clave pública es el par  $(n,e) = K_b^+$ ; la privada, el par  $(n,d) = K_b^-$
- Cifrado y descifrado
  - Dadas las claves  $(n, e)$  y  $(n, d)$
  - Para cifrar el mensaje  $m$ , se hace la siguiente operación matemática
    - $c = m^e \bmod n$  y se lo envía al receptor
  - El receptor para descifrar hace  $m = c^d \bmod n$  y obtiene el mensaje original
  - Notar lo siguiente  $\rightarrow m = (m^e \bmod n)^d \bmod n$
- ¿Por qué RSA funciona?
  - Mucha teoría matemática y modular por detrás, no nos importa mucho
- Seguridad en RSA
  - Dada una clave pública  $(n, e) \rightarrow$  cuan difícil es computar  $d$ ?
  - Necesitamos factorizar  $n$  en primos  $\rightarrow$  es un problema muy difícil computacionalmente. A día de hoy no existen algoritmos tradicionales que resuelvan el problema en tiempos polinomiales
  - Los semiprimos (producto de primos) suelen ser más difícil de factorizar
- RSA en la práctica
  - La exponenciación (modular) en RSA es costosa
  - Los algoritmos simétricos (como AES) permiten encriptar considerablemente más rápido
  - En la práctica, la criptografía de clave pública se utiliza para establecer una conexión segura y衍生 una clave compartida de sesión para cifrar los datos posteriores con un método simétrico

### 8.3 Message Integrity and Digital Signatures

- Para autenticar un mensaje necesito verificar:
  - Que el mensaje lo mando realmente quien dice haberlo mandado
  - Que el mensaje no haya sido alterado

#### Cryptographic Hash Functions

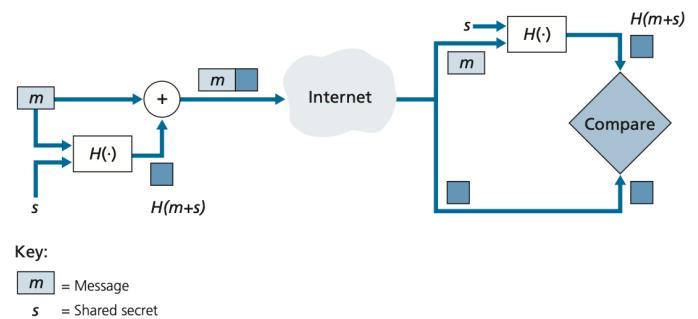
- Una función de hash toma un input  $m$  y computa un string de longitud fija  $H(m)$  conocido como hash
- Una propiedad que debe tener la función es que debe ser imposible computacionalmente que dos inputs  $x$  e  $y$  tengan el mismo hash  $\rightarrow H(x) \neq H(y)$
- Un checksum no cumple con la propiedad de arriba ya que 2 inputs pueden generar el mismo output



- 2 funciones de hash famosas y utilizadas hoy día → MD5 y SHA-1
- Dado un digest  $x$ , es computacionalmente inviable encontrar un  $m$  tal que  $H(m) = x$  (i.e. resistente a la preimagen)

### Message Authentication Code

- Alice crea un mensaje  $m$  y calcula  $H(m)$ . Le envía a Bob un mensaje extendido  $(m, H(m))$
- Bob recibe  $(m, h)$  y calcula  $H(m) \rightarrow$  si  $h(m) == h$  Bob concluye que el mensaje llegó de forma correcta
- Esta técnica no es muy útil ya que Trudy puede interceptar el mensaje y reemplazarlo por  $(m', H(m')) \rightarrow$  luego cuando Bob verifique le va a dar correcto
- Para solucionar esto los locutores necesitan un secreto compartido  $s$ , llamado authentication key
- Alice crea un mensaje  $m$ , calcula  $H(m+s)$  y envia  $(m, H(m+s))$ 
  - $H(m+s)$  es conocido como MAC(message authentication code)
- Bob recibe  $(m, h)$  y conociendo  $s$  calcula  $H(m+s) \rightarrow$  si  $H(m+s) == h$  Bob concluye que el mensaje llegó de manera correcta
- MAC no requiere viajar encriptado
- Tiene un problema: ¿cómo se comparten el secreto  $s$ ?



### Digital Signatures

- Mismo objetivo que una firma común → el que firma esta de acuerdo/verifica con lo que está firmando
- Debe ser posible probar que un documento firmado por  $x$  realmente lo firmó  $x$  y que solo  $x$  pudo haberlo firmado → **Verifiable y no falsificable**
- Una forma para Bob de firmar un mensaje  $m$  es cifrando con su clave privada  $K_b^-$  y enviando  $(m, K_b^-(m))$  a Alice que luego lo desencripta con  $K_b^+(K_b^-(m))$  y obtendrá  $m$ .
- Si luego de desencriptar obtiene  $m$ , Alice estará segura de que Bob fue quien lo envió ya que el es el único que tiene  $K_b^-$ . También puede asegurarse de que  $m$  no fue modificado ya que  $K_b^+(K_b^-(m'))$  no devolverá  $m$

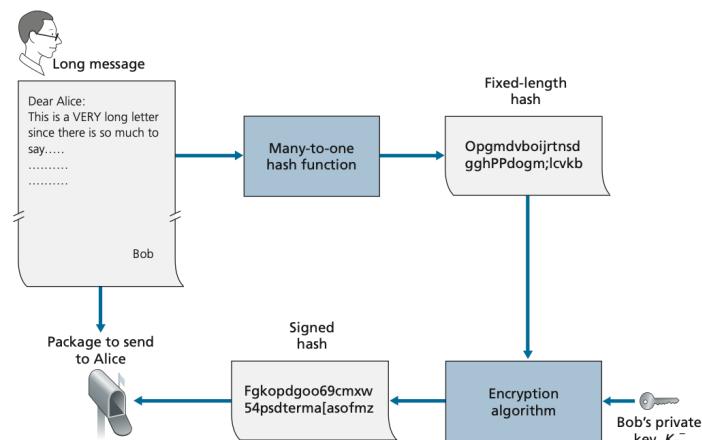
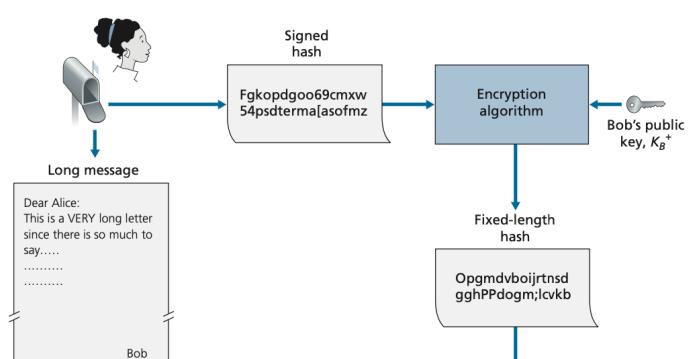


Figure 8.11 ♦ Sending a digitally signed message



- El cifrado y descifrado con RSA es muy costoso → no tiene sentido cifrar todo el mensaje
- Una forma más eficiente es que Bob le aplique una función de hash a  $m$ , cifre  $H(m)$  con su clave privada y le envié a Alice  $(m, K_b^-(H(m)))$ 
  - $H(m)$  es mucho más chico que  $m$ , requiere menos esfuerzo calcular  $K_b^-(H(m))$
- Cuando Alice recibe el par  $(m, K_b^-(H(m)))$ , desencripta  $K_b^-(H(m))$  con  $K_b^+$  y obtiene  $H(m)$ . Luego le calcula la función de hash a  $m$  y si es igual a la recibida Alice se asegura de que el mensaje no fue modificado y que Bob fue quien realmente lo envió

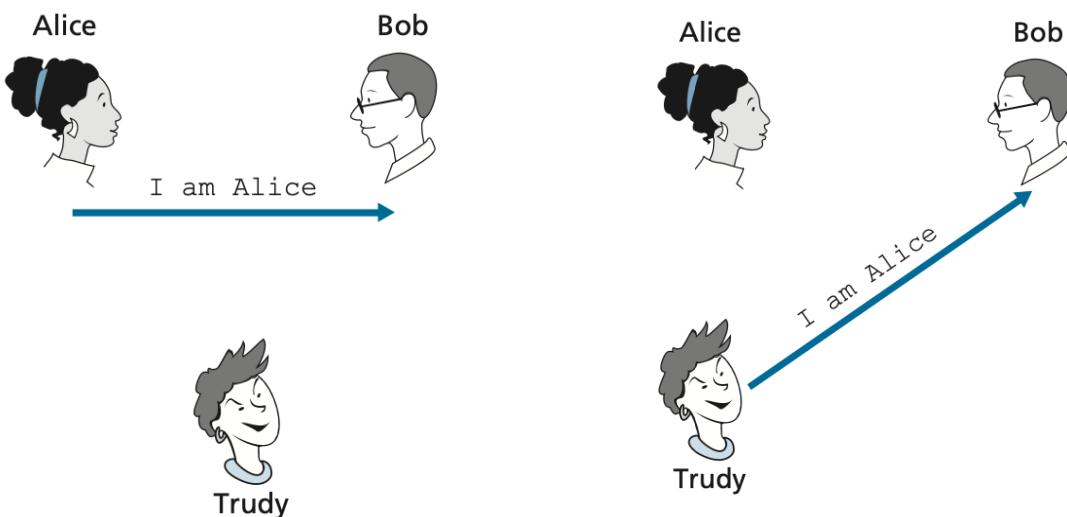
### Public Key Certification

- Es necesario poder verificar que la clave pública de Bob es realmente la de Bob
- Este trabajo lo suelen hacer Certification Authorities (CAs)
- Las CAs verifican tu identidad y te entregan un certificado con tu clave pública y algunos datos propios como la IP entre otros. Este certificado está firmado por la CA.
- Cuando Bob le envía un mensaje a Alice del formato  $(m, K_b^-(H(m)))$ , también le envía su certificado firmado por la CA para garantizar que la clave pública es la real
  - Alice desencripta el certificado con la clave pública de la CA y obtiene la clave pública de Bob que la utiliza para desencriptar  $K_b^-(H(m))$  y luego compararlo con el digest de  $m$
  - De esta forma, Alice está segura de que Bob fue quien envió el mensaje y que este no fue modificado

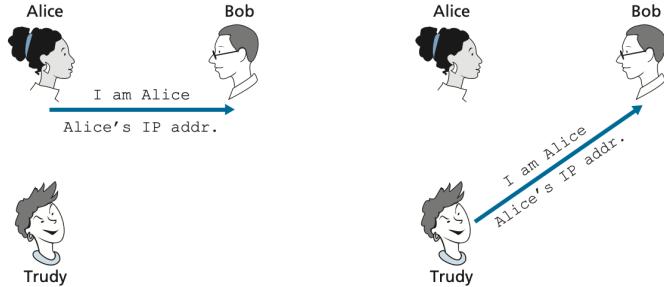
### 8.4 End-Point Authentication

- Proceso por el cual una entidad prueba su identidad frente a otra
- hace foco en autenticación en vivo, no autenticar un mensaje viejo
- Los humanos se autentifican viéndose la cara, por la voz o por el DNI
- Protocolos de autenticación → corren antes que cualquier otro protocolo como TCP
  - Primero autenticar identidad, luego arrancas la comunicación
- Evoluciones de ap
- ap1.0      Idea

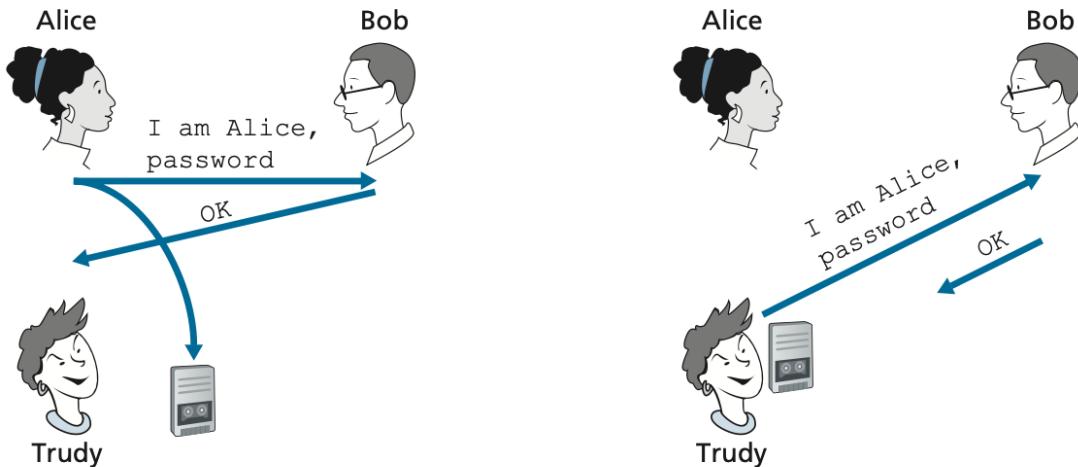
Falla



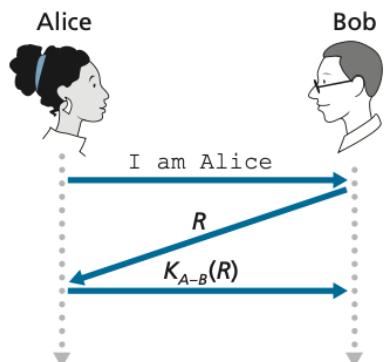
- ap2.0: Alice envía un datagrama con su IP diciendo que es Alice. Fácil de romper



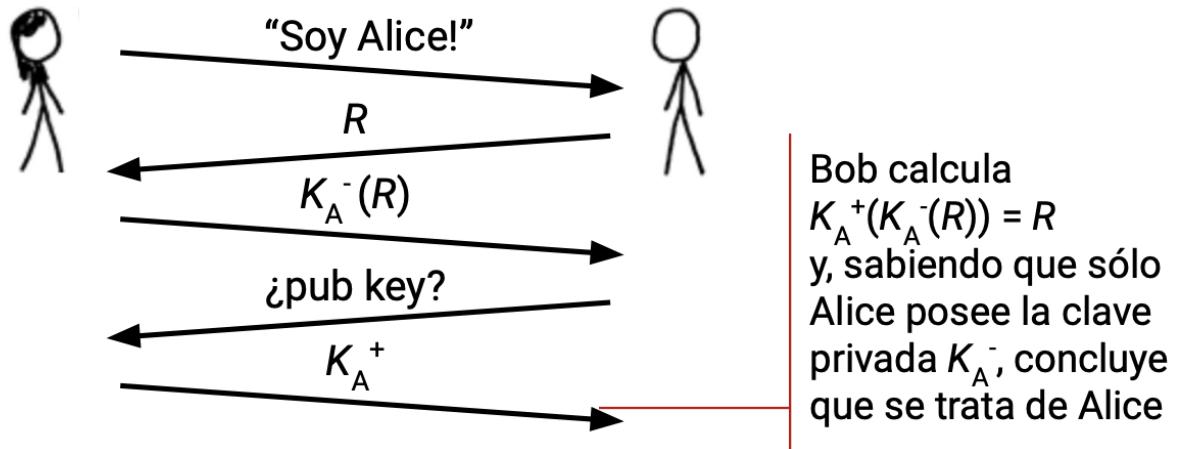
- ap3.0: Alice envía un datagrama con su IP, diciendo soy Alice y una clave. Un intruso puede sniffear el mensaje y luego autenticarse como Alice



- ap3.1: Alice encripta la clave, un intruso no va a poder saberla. Pero de todas formas, un intruso va a poder sniffear el paquete y luego reenviarlo más tarde, sin saber cual es la clave → ataque de replay
- ap4.0: Uso de un **nonce**(número random utilizado por única vez) para descartar ataques de replay. Alice se autentifica con Bob, Bob le envía un nonce y Alice se lo envía encriptado con su clave simétrica. Bob, al recibirlo, se fija si coincide con el nonce enviado. Si coincide, auténtica a Alice ya que solo ellos 2 tienen la clave y con el nonce evita ataques de replay. Luego, descarta el nonce.

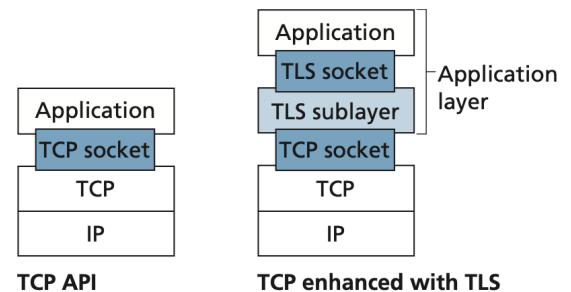


- ap5.0: igual al 4.0 solo que en vez de claves simétricas se usa la clave pública y privada de Alice



## 8.6 Securing TCP Connections: TLS

- Transport layer security
- Protocolo de seguridad para la capa de transporte → se implementa en app, https puerto 443
- Es soportado por todos los navegadores y servidores
- Te das cuenta que estas usando TLS cuando en el browser dice https en vez de http.
- TLS provee a TCP
  - Confidencialidad: Vía cifrado simétrico
  - Integridad: via hashes
  - Autenticación: vía criptografía de clave pública



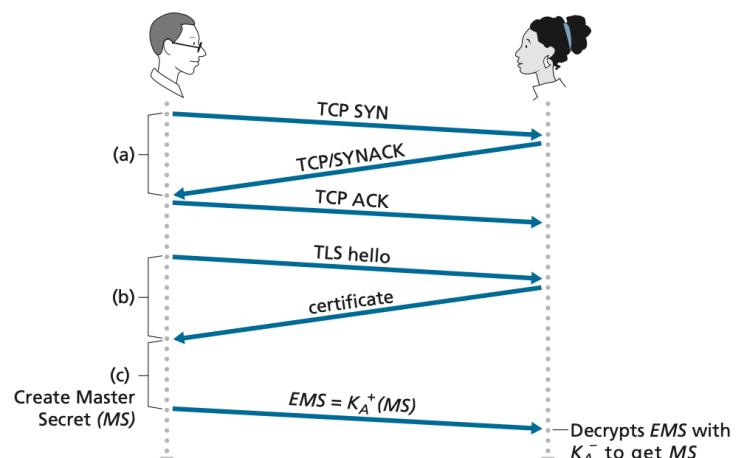
### Visión general

- T-TLS (toy tls) para ver un approach a como funciona
- Ambos tienen 3 fases: Handshake, key derivation and data transfer
- Comunicación entre Bob (cliente) y Alice(servidor) que tiene el par de claves pública y privadas junto a un certificado de identidad

### Handshake

- Se establece conexión TCP normal
- Luego Bob envía un TLS hello y Alice le responde con su certificado firmado por la CA, de esta forma Bob sabe la clave pública de Alice
- Por último, Bob crea un master secret MS, lo encripta con la clave pública de Alice recibida en el certificado y se la envía a Alice.

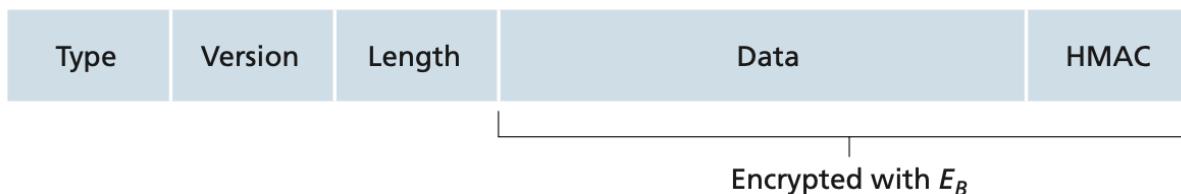
### Key Derivation



- El MS podría ser utilizado como clave simétrica, pero no es del todo seguro
- Se utiliza el MS para generar 4 claves → ambas partes generan las mismas
  - $E_B$  = clave para la data enviada desde Bob a Alice
  - $M_B$  = Clave HMAC para la data enviada de Bob a Alice
  - $E_A$  = clave para la data enviada desde Alice a Bob
  - $M_A$  = Clave HMAC para la data enviada de Alice a Bob

### Data Transfer

- TLS parte el stream de bytes que TCP le da a app en varios registros
- El emisor agarra el mensaje a enviar, le suma la clave HMAC  $M_B$  y lo hashea obteniendo un HMAC real → luego la data a enviar, junto con el HMAC calculado se encripta con la clave simétrica  $E_B$
- Cuando Alice lo recibe lo desencripta con  $E_B$ , a la data le suma  $M_B$  y le calcula la el HMAC. Si coincide con el del segmento, está segura de que no fue modificado en el camino



- Type, version y length no van cifrados pero el cálculo del HMAC da erroneo si estos son modificados
  - Type se usa para avisar si el mensaje es handshake, envío de data o cierre de conexión
- Posibles fallas → Un intruso reordena los segmentos TCP cambiando los #seqs del header TCP sin encriptar
- TLS lo soluciona implementando números de secuencia y nonces → son tenidos en cuenta para el cálculo del HMAC

### Connection Closure

- Se avisa en el campo de type y es tenido en cuenta para el cálculo del HMAC
- Un intruso podría crear un segmento TCP para cerrar la conexión, pero sería muy complicado que cumpla con todos los requisitos

### Una mirada más completa

#### TLS real handshake

- Reduce los TTL y es más seguro
- El cliente envía una lista de los algoritmos que soporta
- El servidor elige un algoritmo simétrico, uno de clave pública y algoritmo HMAC y se lo envía al cliente junto con un certificado
- El cliente verifica el certificado, y genera un Pre master secret (PMS) y se lo envía cifrado con la clave pública al servidor

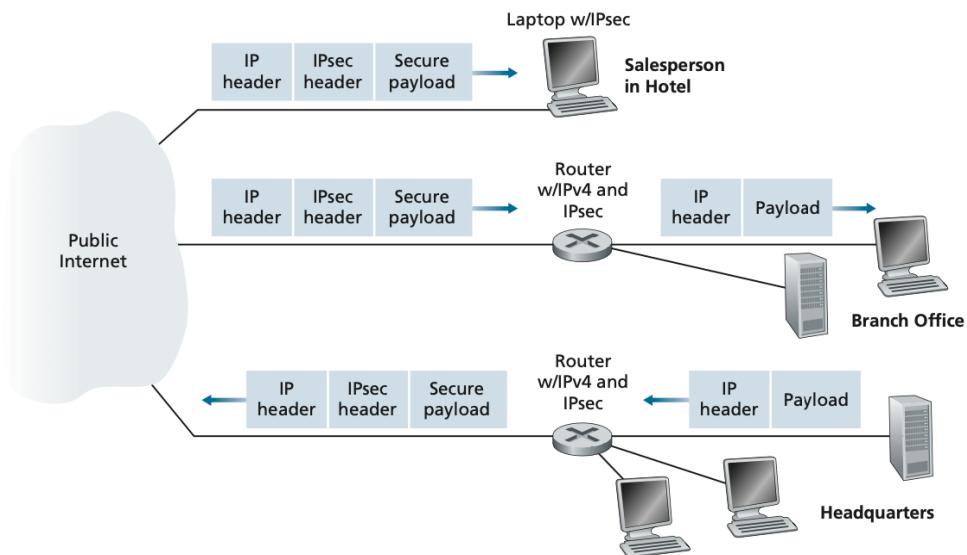
- Tanto el servidor como el cliente, usando el PMS crean el MS y crean las 4 claves
- El cliente y el servidor se envían HMACs de todo el handshake
  - Para corroborar que en ningún paso haya habido algún problema
  - En caso de que hayan inconsistencias se cierra la conexión
- En TLS, los nonces son usados para evitar ataques de replay y los número de secuencia para evitar ataques de reordenamiento

## 8.7 Network-Layer Security: IPsec and Virtual Private Networks

- Seguridad en la capa de red
- Se usa IPsec para crear VPNs (virtual private networks)
- Ofrece confidencialidad (cifrando el payload del datagrama), autenticidad(verificando emisor/receptor) e integridad

### IPsec and VPNs

- Las instituciones quieren que sus empleados puedan comunicarse entre sí de manera segura → antes se creaban redes privadas, muy caras ya que requieren mucha infraestructura
- Hoy día, se implementan VPNs, una forma segura de conectar a 2 personas
- Los empleados en la sede central no usan IPsec para comunicarse entre sí, pero si quieren comunicarse con un vendedor en un hotel si lo hace → convierte el datagrama IP a IPsec
- El datagrama IPsec tiene el mismo header que IP por lo que cualquier router lo puede aceptar → el payload tiene un header IPsec y el mensaje encriptado



### The AH and ESP Protocols

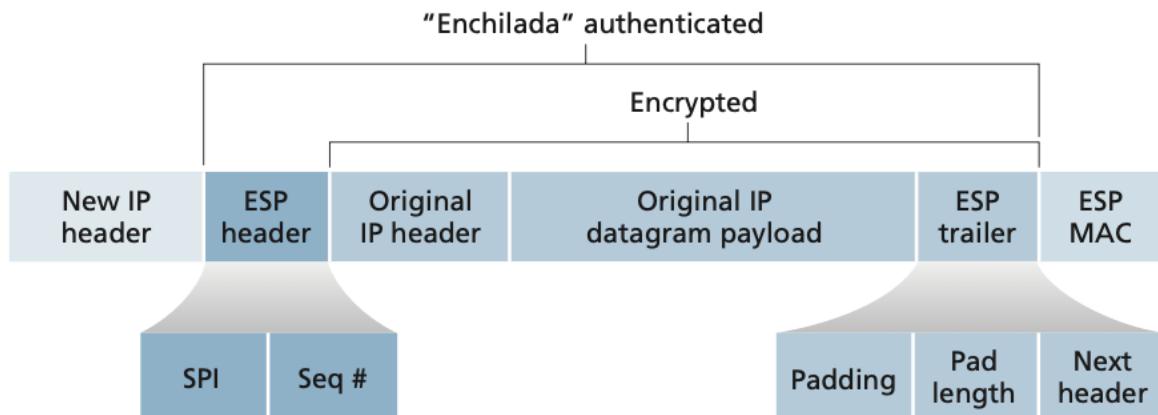
- Authentication Header (AH)
  - Provee autenticación e integridad pero no confidencialidad
- Encapsulation Security Payload(ESP)
  - Provee autenticación, integridad y confidencialidad

## Security Associations

- Antes de enviar los datagramas IPsec, el emisor crea una conexión lógica unidireccional con el receptor llamada security association (SA)
- Si ambos se quieren enviar datagramas se crean 2 SAs
- Es orientado a conexión a diferencia de IP
- El router emisor mantiene información de estado
  - Identificador de SA de 32 bits (SPI) security parameter index
  - IP de origen de la SA (del router, no del host)
  - IP destino de la SA (del router, no del host)
  - Tipo de cifrado utilizado
  - Clave de cifrado
  - Algoritmo de integridad
  - Clave de autenticación
- Cuando el router tiene que enviar un datagrama IPsec, se fija en esta información para ver cómo construirlo

## The IPsec Datagram

- 2 tipos de paquetes
  - Modo transporte: procesa el payload del datagrama, encriptado y autenticación
  - Modo túnel: procesa el datagrama completo y lo encapsula en un nuevo datagrama con header distinto
- Cuando un router recibe un datagrama IP y lo quiere convertir a IPsec sigue estos pasos
  1. Le appendea al final un ESP trailer
  2. Encripta el datagrama original con el trailer
  3. Le agrega un ESP header → el paquete pasa a llamarse **enchilada**
  4. Le agrega a la **enchilada** un integrity check value (ICV). Es un ESP MAC generado a partir de una clave compartida
  5. Por último, usa todo lo anterior como payload y lo mete en un datagrama nuevo con su correspondiente header IP



- El datagrama IPsec no es más que un datagrama IP común pero con algunos campos más en el payload
- En el header IP original, la IP de origen y destino son la de los hosts pero la IP origen y destino del header nuevo son de los routers del final del tunel/conexion SA
- El número de protocolo del IP original se mantiene en TCP, UDP pero el del header nuevo pasa a ser 50 → indica que se está usando el protocolo ESP
- Los routers en el medio van a manejar el datagrama como si fuese una datagrama mas del mordon
- Trailer ESP
  - Padding: bytes de relleno para poder cifrar por bloques → necesita una longitud fija
  - Pad length: cantidad de bytes usados en padding
- ESP header
  - SPI: para que el receptor sepa cómo manipular el datagrama
  - #Seq: para protección de ataques de replay
- ESP MAC
  - Le agrega la MAC key a la enchilada y le calcula la función de hash
- Cuando el router del final de la SA recibe el paquete hace lo siguiente
  1. Como ve que es para el puerto 50 (ESP) le aplica el debido tratamiento
  2. Usa el SPI para ver cómo tratar el datagrama y ver a que SA pertenece
  3. Calcula el MAC de la enchilada y verifica que de igual que el del ESP MAC
  4. Si da bien, esta seguro de que no hubieron modificaciones en el camino
  5. Chequea el #Seq
  6. Desencrypta el payload usando el algoritmo asociado a la SA
  7. Elimina el padding y extrae el datagrama IP original
  8. Envía el datagrama al destino

## Bases de datos IPsec

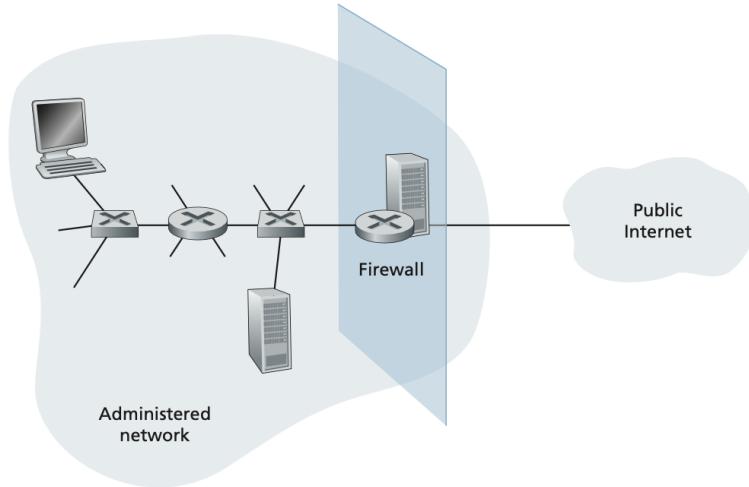
- Security Policy Database (SPD):
  - Dado un datagrama el emisor sabe si usar o no IPsec
  - Permite determinar cuál SA utilizar
  - Cada entrada en la SPD se define por selectores: conjunto de campos de headers IP y de nivel superior
  - Mapean tráfico saliente a una SA
  - SPD: qué hacer
- Security Association Database (SAD)
  - El estado de las SAs se almacena en la SAD
  - Al enviar un datagrama IPsec, el emisor accede a la SAD para determinar cómo procesar dicho datagrama
  - Cuando el datagrama llega al receptor, éste toma el SPI en el header e indexa la SAD con dicho valor para posteriormente procesar el datagrama según indique la SA
  - SAD: cómo hacerlo

## IKE: Key Management in IPsec

- Si bien es posible definir las SAs y sus parámetros de forma manual, esto se vuelve impráctico para VPNs con cientos de endpoints
- De esto se encarga el protocolo IKE
- Cada IPsec entity tiene un certificado que incluye su clave pública
- Deriva claves a partir de un secreto compartido generado vía Diffie-Hellman (DH)
- Se monta sobre UDP (puerto 500)
- La autenticación se lleva a cabo a través de un secreto pre compartido (PSK) o con una infraestructura de clave pública (PKI)
- IKE tiene dos fases
  - Fase 1: establecimiento de una SA bidireccional
    - Esta SA es distinta a las SAs de IPsec
    - Es una SA del protocolo ISAKMP (Internet Security Association and Key Management Protocol)
  - Fase 2: se usa ISAKMP para negociar de forma segura el par de SAs de IPsec
- Vía IKE se intercambian mensajes para definir algoritmos de cifrado, claves, SPIs, etc.
- Permite utilizar el protocolo AH o el ESP

## 8.9 Firewalls

- Combinación de software y hardware que permiten aislar de Internet a la red interna de una organización
- Todo el tráfico de la red interna hacia internet y viceversa pasa por el Firewall
- Solo el tráfico autorizado, definido en su política, tiene permitido pasar
- El firewall es inmune a ataques
- Funciones de los firewalls
  - Prevenir acceso/manipulación de datos privados
  - Permitir acceso autorizado a la red interna a un conjunto de usuario
  - Prevenir ataques DoS (denial of service)
- Tres tipos de firewalls
  - Filtrado sin estado (stateless)
  - Filtrado con estado (stateful)
  - Gateways de app
- **Filtrados stateless**
  - El firewall filtra paquete a paquete, tomando decisiones a partir de:
    - Direcciones IP origen y destino



| acción | IP src              | IP dst              | proto | puerto src | puerto dst | flags |
|--------|---------------------|---------------------|-------|------------|------------|-------|
| allow  | 222.22/16           | fueras de 222.22/16 | TCP   | > 1023     | 80         | *     |
| allow  | fueras de 222.22/16 | 222.22/16           | TCP   | 80         | > 1023     | ACK   |
| allow  | 222.22/16           | fueras de 222.22/16 | UDP   | > 1023     | 53         | ---   |
| allow  | fueras de 222.22/16 | 222.22/16           | UDP   | 53         | > 1023     | ----  |
| deny   | *                   | *                   | *     | *          | *          | *     |

- Puertos origen y destino y protocolo de transporte (TCP/UDP)
- Tipo de mensaje 3t ICMP
- Flags de TCP
- Usan una tabla de match-plus-action
- Ejemplo: bloquear datagramas entrantes y salientes con puerto de transporte xx (origen o destino)
- **Filtrado stateful**
  - A diferencia del filtrado stateless que se fija en cada paquete de forma aislada a los demás, el filtrado stateful mantiene un estado de las conexiones TCP y usa esta información para la toma de decisiones
  - Más específicamente, registra el inicio y cierre de conexiones TCP y determina si los paquetes entrantes son consistentes
  - El filtrado stateless acepta cualquier paquete con el flag ACK activado, sin importar si es un de una conexión existente o de un intruso. En cambio, el filtrado stateful se fija si hay una conexión TCP abierta con alguien dentro de la red. En caso de que haya una conexión consistente, lo deja entrar.
  - La columna de conexión indica si se debe verificar la conexión o no
    - En este caso, todos los paquetes TCP entrantes se les debe verificar

| acción       | IP src             | IP dst             | proto | puerto src | puerto dst | flags | conexión |
|--------------|--------------------|--------------------|-------|------------|------------|-------|----------|
| <b>allow</b> | 222.22/16          | fuera de 222.22/16 | TCP   | > 1023     | 80         | any   |          |
| <b>allow</b> | fuera de 222.22/16 | 222.22/16          | TCP   | 80         | > 1023     | ACK   | X        |