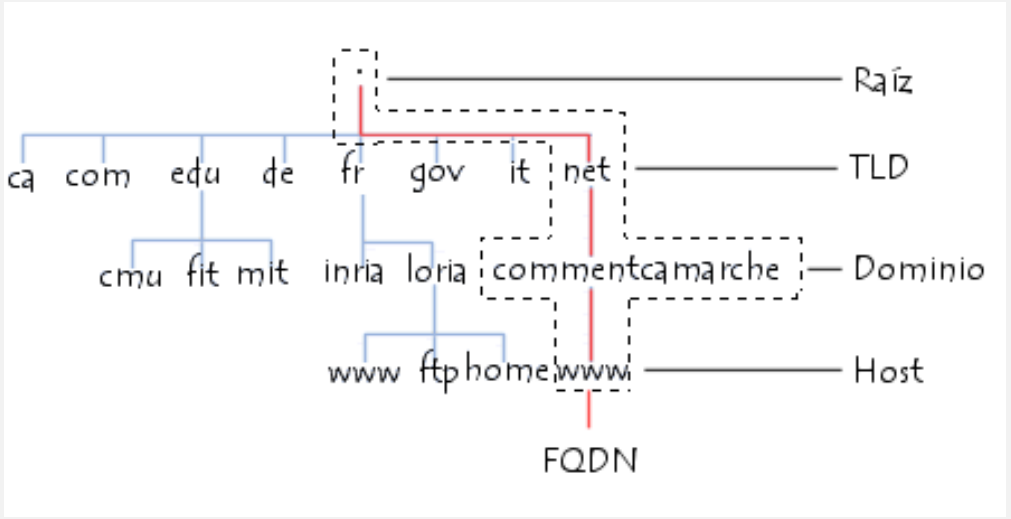
DNS: SERVICIO DE NOMBRES

Redes

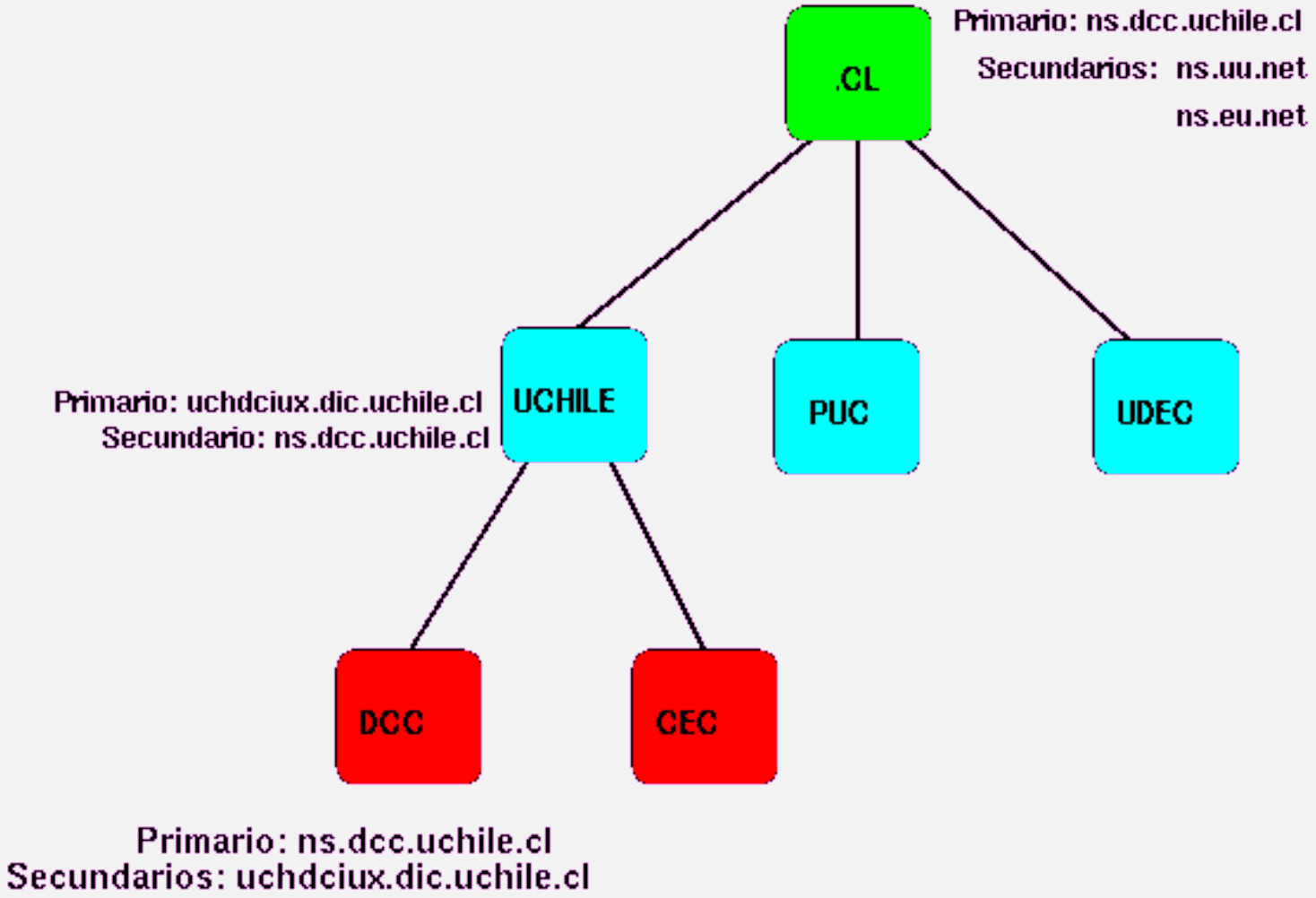
U. de Chile

ÁRBOL DE DOMINIOS (1)

- Servicio de nombres distribuido, basado en UDP
- Redundante
- Sin administración central
- Traducción IP <-> nombre



ÁRBOL DE DOMINIOS (2)



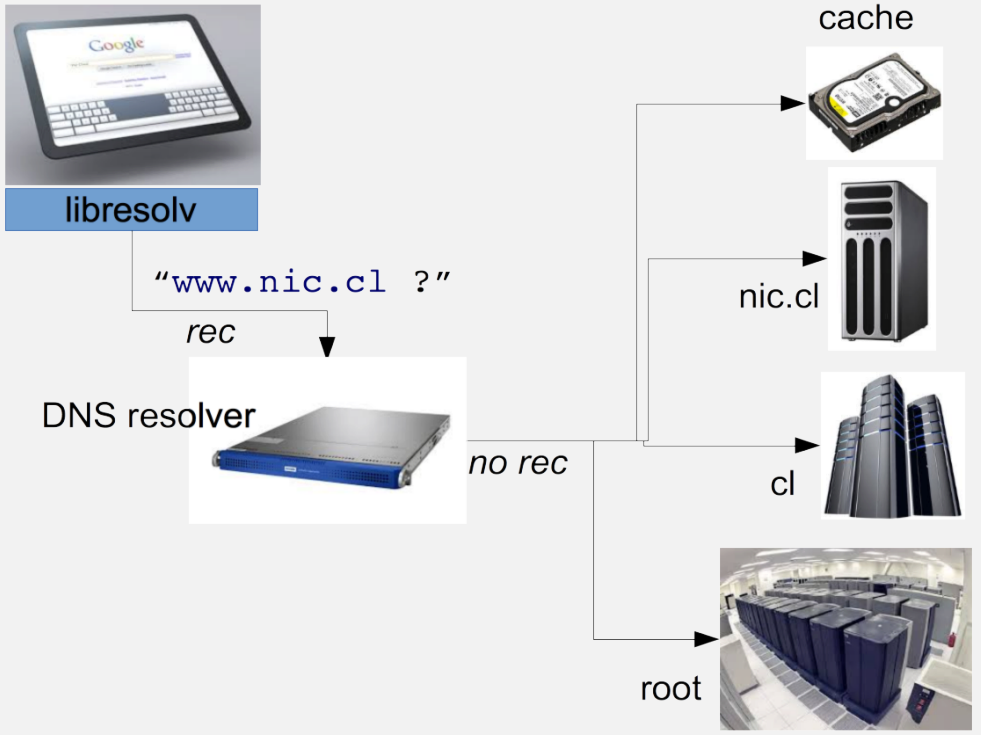
DELEGACIÓN DE AUTORIDAD (1)

- Delegación de Responsabilidad.
  - Servidor Primario: actualiza .
  - Servidor Secundario: informa.
  - Servidor Cache: informa sin seguridad.
- La raíz tiene primario administrado por ICANN
- Un dominio se delega con un record NS .

DELEGACIÓN DE AUTORIDAD (2)

- Las preguntas bajo ese dominio son derivadas
  - Por ej: .CL, .com, etc...
- Servidor: Bind, Cliente: Resolver
- Records NS, A y MX: Nombre -> IP
- Dominio Inverso: IP -> Nombre
  - 83.146.in-addr.arpa

RESOLUCIÓN (1)



RESOLUCIÓN (2)

- Cada dominio tiene una lista de servidores (NS)
- ¿A cual le pregunto?
- Round Robin + prioridad por RTT
- Permite elegir al más rápido, distribuyendo la carga
- Los servidores modifican el orden en las listas de sus respuestas, para aleatorizar la primera pregunta

RESOLUCIÓN (3)

- Pregunta inical: recursiva, al resolver local
- Luego: preguntas no recursivas, al que sabe
- Muchas respuestas parciales que analizar
- El trabajo lo hace mi resolver local
- Lo provee mi red
- O un externo (google: 8.8.8.8, ojo en rede corporativas)

RESOLUCIÓN (4)

- Hoy hay servidores raíz en muchos países
- ¿De qué sirve?
- Principalmente para los errores
- ¡Que pueden ser millones por segundo!
- Usamos anycast (IP compartida) para tener cientos de raíces
- Aunque son solo 13 direcciones IP

RESOLUCIÓN (5)

.CL (1)

- Necesitamos secundarios en todo el mundo
- Pero la lista de servidores no debe ser muy larga
- Utilizamos anycast: una IP se comparte entre computadores en muchas partes del mundo
- Se usa BGP-4 para difundir todos los caminos que llevan a él
- Se simula que es uno solo, pero en realidad son muchos

.CL (2)

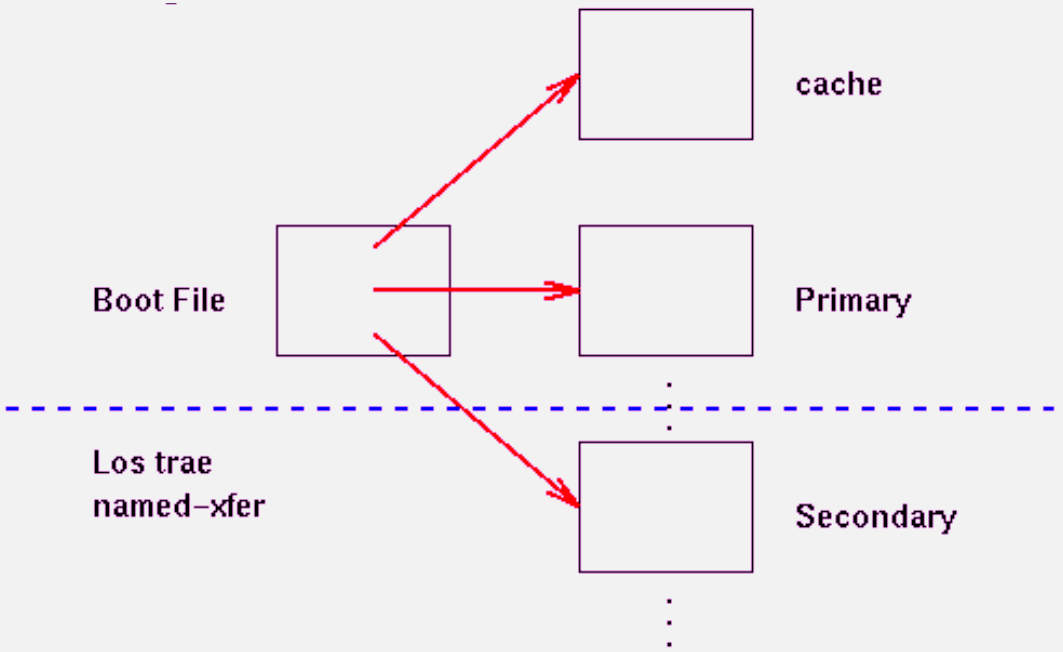

Mapa
Satélite
Relieve
Canada
México
(Mexico)
Venezuela
Colombia
Perú
(Peru)
Brasil
(Brazil)
Bolivia
Argentina
North
Atlantic
Ocean
South
Pacific
Ocean
South
Atlantic
Ocean
Indian
Ocean
Ísland
(Iceland)
Sverige
France
España
(Spain)
Italia
(Italy)
Türkiye
(Turkey)
Россия
(Russia)
Қазақстан
(Kazakhstan)
Монгол Улс
(Mongolia)
中国
(China)
India
Indonesia
Papua New
Guinea
Australia
New
Zealand
Mali
Niger
Tchad
(Chad)
السودان
(Sudan)
(Ethiopia)
Angola
Namibia
South
Africa
Madagasikara
(Madagascar)
(Libya)
(Egypt)
(Saudi
Arabia)
(Iraq)
(Algeria)
2000 mi
2000 km
©2013 Google - Datos del mapa ©2013 MapLink - Condiciones de uso

CONFIGURACIÓN (1)

- El cliente requiere un archivo de configuración.
- El servidor requiere de un archivo de partida y un directorio para sus zonas primarias y secundarias.
- Al configurar un primario: conseguir dos secundarios.
- Definir bien los parámetros del record SOA.

CONFIGURACIÓN (2)

- Configuración del Servidor
  - Boot File
  - Cache
  - Primary
  - CL Primary



CONFIGURACIÓN (3)

- Boot File

```
;
; sunsite.boot : boot file for name server ns.dcc.uchile.cl
;
directory  /usr/etc/named
cache      .                        sunsite.ca
primary    cl                       cl.zone
primary    dcc.uchile.cl            dcc.uchile.cl.zone
primary    srcei.cl                 srcei.cl.zone
primary    4.83.146.in-addr.arpa    4.83.146.revzone
primary    0.0.127.in-addr.arpa     sunsite.local
;
; Secundarios para todos los subdominios de .CL
;
secondary utfsm.cl                  146.83.198.3         back/utfsm.zone
secondary rdc.cl                    146.155.30.25 146.155.1.155 back/rdc.zone
```

CONFIGURACIÓN (4)

- Cache

```
;
; sunsite.ca : Initial cache data for root domain servers.
;
.                       99999999  IN   NS         A.ROOT-SERVERS.NET.
                        99999999  IN   NS         H.ROOT-SERVERS.NET.
                        99999999  IN   NS         B.ROOT-SERVERS.NET.
                        99999999  IN   NS         C.ROOT-SERVERS.NET.
                        99999999  IN   NS         D.ROOT-SERVERS.NET.
                        99999999  IN   NS         E.ROOT-SERVERS.NET.
                        99999999  IN   NS         I.ROOT-SERVERS.NET.
                        99999999  IN   NS         F.ROOT-SERVERS.NET.
                        99999999  IN   NS         G.ROOT-SERVERS.NET.
;
; Prep the cache (hotwire the addresses).
;
A.ROOT-SERVERS.NET.       99999999  IN   A 198.41.0.4
H.ROOT-SERVERS.NET.       99999999  IN   A 128.63.2.53
B.ROOT-SERVERS.NET.       99999999  IN   A 128.9.0.107
C.ROOT-SERVERS.NET.       99999999  IN   A 192.33.4.12
D.ROOT-SERVERS.NET.       99999999  IN   A 128.8.10.90
E.ROOT-SERVERS.NET.       99999999  IN   A 192.203.230.10
I.ROOT-SERVERS.NET.       99999999  IN   A 192.36.148.17
F.ROOT-SERVERS.NET.       99999999  IN   A 39.13.229.241
G.ROOT-SERVERS.NET.       99999999  IN   A 192.112.36.4
```

CONFIGURACIÓN (5)

- Primary

```
;
; srcei.zone : Authoritative data for srcei.cl.
;
@          IN    SOA     ns.dcc.uchile.cl. hostmaster.dcc.uchile.cl. (
                 96010214     ;Serial
                 43200        ;Refresh (12 horas)
                 7200         ;Retry   (2 horas)
                 2592000      ;Expire  (30 días)
                 43200)       ;Minimum (12 horas)

; Servidores de Nombres para srcei.cl
           IN    NS      ns.dcc.uchile.cl.
           IN    NS      inti.inf.utfsm.cl.
           IN    NS      huelen.reuna.cl.
;
$ORIGIN cl.
srcei      IN   A      164.96.124.4
$ORIGIN srcei.cl.
netscada   IN A 164.96.64.2
news       IN   CNAME srcei.cl.
*          IN   MX     10              srcei.cl.
```

CONFIGURACIÓN (6)

- CL Primary

```
;
; cl.zone : Authoritative data for .CL
;
@               IN      SOA     ns.dcc.uchile.CL.  hostmaster.dcc.uchile.CL. (
                                96051856        ; version (yymmddhh)
                                86400           ; refresh (1 día)
                                14400           ; retry   (4 horas)
                                2592000         ; expire  (30 días)
                                172800 )        ; minimum (2 días)
                IN      NS      ns.dcc.uchile.cl.
                IN      NS      sunsite.dcc.uchile.cl.
                IN      NS      pucmon.puc.cl.
                IN      NS      uchile.cl.
                IN      NS      ns.UU.NET.
                IN      NS      sparky.arl.mil.
                IN      NS      ns.EU.net.
                IN      NS      uucp-gw-1.pa.DEC.COM.
                IN      NS      uucp-gw-2.pa.DEC.COM.
                IN      NS      ns.cec.uchile.cl.
                IN      NS      ns.ict.uchile.cl.
; Estos los comentamos para evitar que algunos mails se vayan para USA.
;               IN      MX      100     relay1.UU.NET.
;               IN      MX      100     relay2.UU.NET.
;
; A record de parche porque no recupero la direccion IP de pucmon
pucmon.puc.cl.  IN      A       146.155.1.155
;
```

CONFIGURACIÓN (7)

- CL Primary (Continuación)

```
;
; Sub dominio Banco de Crédito e inversiones
; encargado: pcousin@bci.cl (Pablo Cousino)
; encargado Tecnico: rleiva@bci.cl (Raul Leiva)
;
BCI             IN      NS      rsnet.bci.cl.
                IN      NS      ns.rdc.cl.
                IN      NS      ns.dcc.uchile.cl.
                IN      MX      10 rsnet.bci.cl.
;
; Sub dominio CONICYT
; encargado wmaldona@uchcecvm.cec.uchile.cl (Waldo Maldonado)
;
CONICYT         IN      NS      daniel.conicyt.cl.
                IN      NS      uchile.cl.
                IN      NS      ns.dcc.uchile.cl.
                IN      MX      10 conicyt.cl.
;
; Sub dominio Orden Ltda.
; encargado: aaraya@tolten.puc.cl (Arnoldo Araya Flores)
;
ORDEN           IN      NS      macha.orden.cl.
                IN      NS      lapa.orden.cl.
                IN      NS      pucmon.puc.cl.
                IN      NS      ns.dcc.uchile.cl.
```

CONFIGURACIÓN (8)

- Configuración de las zonas
  - SOA

Campo	Dominio (CL)	Sub-dominio (udec.cl)	sub-subdominio (dpi.udec.cl)
refresh	1 dia (86400)	18 horas (64800)	12 horas (43200)
retry	4 horas (14400)	3 horas (10800)	2 horas (7200)
expire	30 dias (259200)	30 dias (259200)	30 dias (259200)
min ttl	2 dias (172800)	1 dia (86400)	12 horas (43200)

  - Errores Clásicos
  - Lame Delegations
  - Punto al final
  - SOA mal configurado (expire, ttl)
  - Pocos Servidores de nombres

TRANSPORTE: LAYER 3

UDP / TCP

INTRODUCCIÓN (1)

- Las aplicaciones esperan ciertas características del servicio entregado por la capa de transporte, como:
  - Despacho garantizado de mensajes.
  - Despacho de los mensajes en el orden original.
  - Despacho de a lo más una copia de cada mensaje.
  - Envío de mensajes de largo arbitrario.

INTRODUCCIÓN (2)

- Las aplicaciones esperan ciertas características del servicio entregado por la capa de transporte, como:
  - Sincronización entre emisor y receptor.
  - El receptor debe ser capaz de aplicar control de flujo al emisor.
  - Múltiples aplicaciones por host (ports).

INTRODUCCIÓN (3)

- Hay dos líneas principales de servicios entregados por la capa de transporte (en el modelo de Internet):
  - TCP: Transmission Control Protocol
    - Orientado al flujo confiable de bytes
    - Provee control de flujo, control de errores, orden.
  - UDP: User Datagram Protocol
    - Orientado principalmente a la mensajería.
    - Es asíncrono.
    - No confiable, sin control de flujo ni orden.

INTRODUCCIÓN (4)

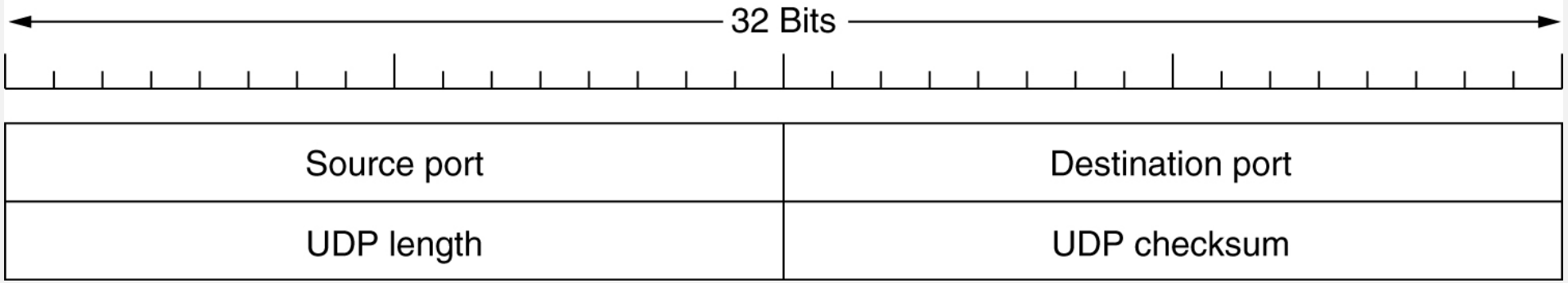
- La capa transporte opera sobre la capa red
- La capa red es la “nube” Internet
- Provee conectividad de paquetes IP
- Entre hosts (computadores/equipos) IP
- Los extremos se identifican en forma única por dirección IP
- Suponemos que esa capa funciona (con errores)

PROTOCOLO UDP (1)

- Protocolo de transporte muy simple.
- Permite mensajes/paquetes entre aplicaciones
- Orientado a los mensajes y a los sistemas que funcionan en base a request/reply. En estos casos, el establecimiento de una conexión es un costo que no vale la pena asumir.
- El control de flujo, orden y recuperación de errores queda en manos de la aplicación
- Control de errores: se descartan los paquetes con errores de transmisión

PROTOCOLO UDP (2)

- El checksum se calcula sobre el header UDP más un *pseudoheader*: campos del header IP como Protocolo, IP origen, IP destino y el largo del payload.
- Checksum permite detectar mayoría de errores de bits en el paquete y descartarlo en caso de error
- En IPv4 era opcional (en cero), pero en IPv6 es obligatorio



MULTIPLEXIÓN POR PUERTOS (1)

- Los servicios de la capa de transporte requieren que el emisor y receptor establezcan un socket. Cada socket se identifica por el par (IP, puerto).
- Las conexiones se identifican por los identificadores de los sockets a ambos lados.
- Estos permite asociar una aplicación con una conexión, mediante una tabla interna propia del S.O.

PID	Socket
1234	1
4563	2
8890	3

Socket	IP	Puerto
1	10.0.45.3	3535
2	10.0.45.4	5050
3	10.0.45.4	8080

MULTIPLEXIÓN POR PUERTOS (2)

- Los puertos son números de 16 bits (por lo que existen 65535 posibilidades).
- Se dividen en dos categorías: Puertos bien conocidos (*well-known ports*, aquellos menores al 1024) y puertos no privilegiados (todos los otros)
- Los puertos bien conocidos están reservados para los servicios estándares. Así, un cliente conoce exactamente a qué puerto dirigir un requerimiento.
- Cada flujo se identifica por la tupla (Dir_{origen}, $Puerto_{origen}$, $Dir_{destino}$, $Puerto_{Destino}$, Protocolo)

MULTIPLEXIÓN POR PUERTOS (3)

- Listado de puertos de ejemplos

Servicio	Puerto	Protocolo	Servicio	Puerto	Protocolo
FTP	21	TCP	POP3	110	TCP
SSH	22	TCP	SunRPC	111	TCP y UDP
Telnet	23	TCP	NNTP	119	TCP
SMTP	25	TCP	NTP	123	TCP y UDP
Whois	43	TCP y UDP	SNMP	161	TCP y UDP
DNS	53	TCP y UDP	BGP	179	TCP
HTTP	80	TCP	HTTPS	443	TCP

TRANSMISIÓN CONFIABLE (1)

- Transmisión Confiable se traduce en el mecanismo para asegurar que todo lo que se envía es recibido, en el mismo orden original.
- Checksum sólo detecta un error
- Cuando un error aparece, la capa debe descartar el paquete, y recuperarse posteriormente de esa situación.

TRANSMISIÓN CONFIABLE (2)

- Esto se logra mediante la combinación de los mecanismos: *acknowledgments (ACK)* y *timeout.*
- Los ACK son pequeños paquetes de control que un receptor envía de vuelta al emisor para notificar la recepción satisfactoria de un paquete previo.
- Estos paquetes de control son sólo el encabezado (*header*) y no contienen datos. Por lo mismo, el receptor puede incluir un ACK en un paquete de datos que vaya en el sentido contrario. A este mecanismo se le conoce como *piggybacking*.

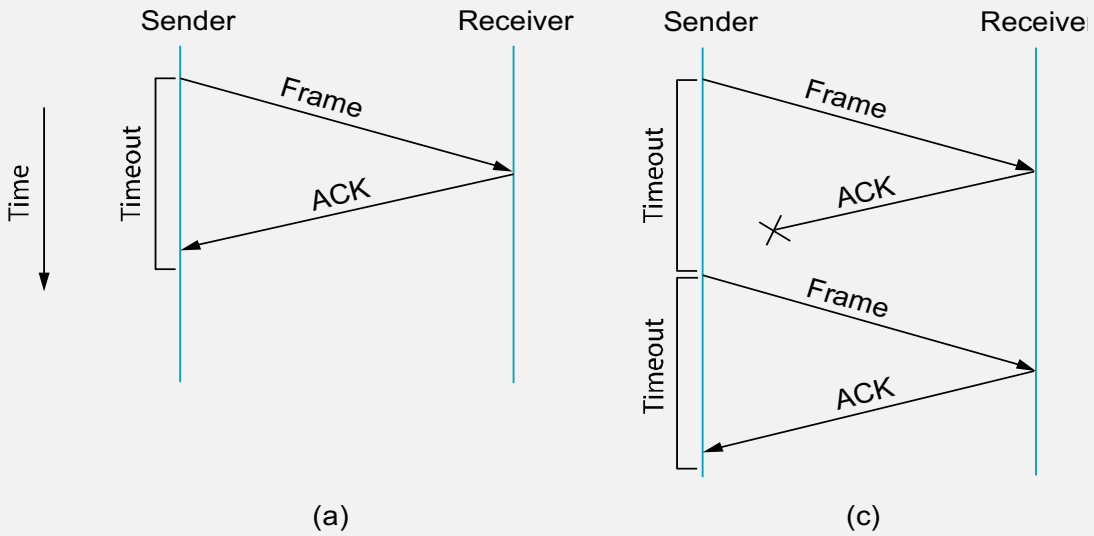
TRANSMISIÓN CONFIABLE (3)

- Si después de una cantidad razonable de tiempo el ACK no se ha recibido, el emisor retransmite el paquete que no fue confirmado. El tiempo máximo de espera por un ACK se conoce como timeout.
- La estrategia que usa ACK's y timeouts para implementar transmisión confiable es llamado *automatic repeat request* (ARQ).
- Revisaremos tres tipos de ARQ.
  - Stop-and-Wait
  - Go-Back N
  - Sliding Window

STOP-AND-WAIT (1)

- La idea es muy simple: Envío un paquete y me quedo esperando por el ACK respectivo antes de enviar otro paquete.
- Si después de cierta espera el ACK no llega, se retransmite.
- Revisemos algunos casos que se pueden producir durante la operación de este mecanismo.

STOP-AND-WAIT (2)



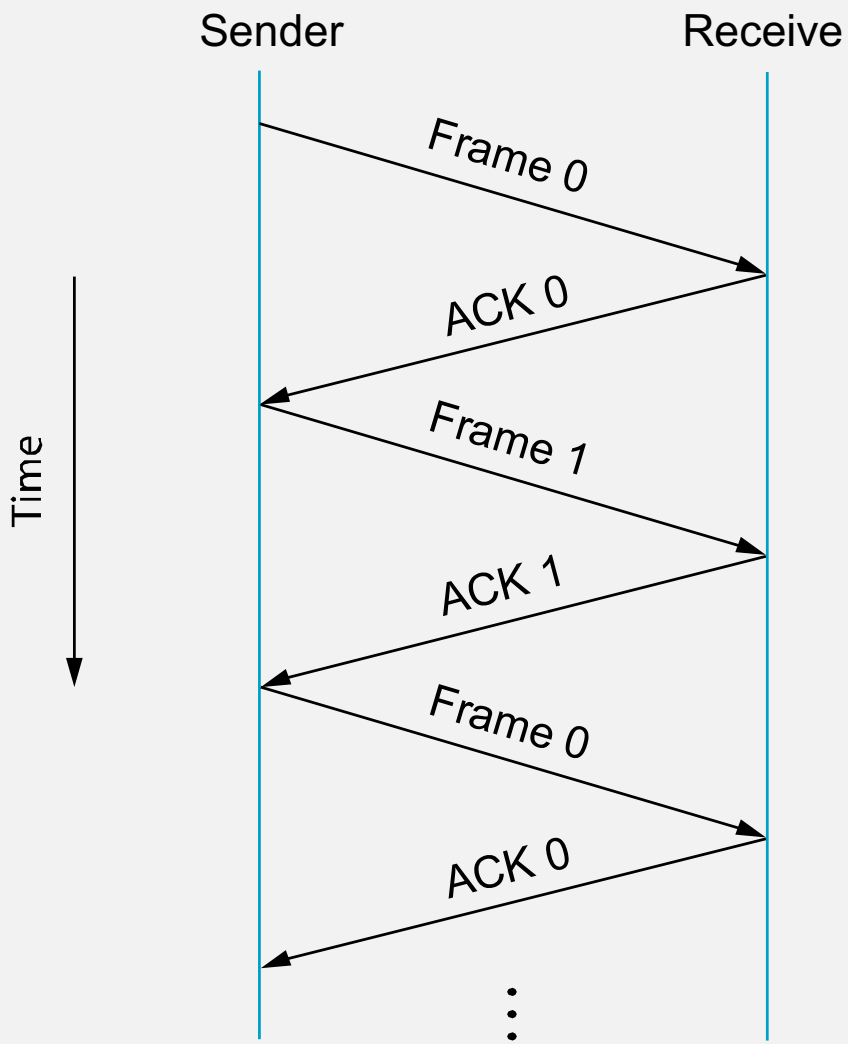
STOP-AND-WAIT (3)

- En (a) se muestra el envío del paquete, y la posterior recepción del ACK.
- En (b) se muestra el envío de un paquete y su posterior pérdida, lo que genera una retransmisión correcta.
- En (c) se muestra el envío de un paquete, su recepción y generación del ACK, el cual se pierde antes de llegar al emisor, lo que genera una retransmisión incorrecta.
- En (d) se muestra un caso parecido a (c), pero el ACK no se pierde, sino que se retrasa. Esto genera una retransmisión incorrecta y un ACK duplicado.

STOP-AND-WAIT (4)

- (b), (c) o (d) indistinguibles
- En los casos (c) y (d) el emisor reenviará el paquete original, pero el receptor creerá que se trata de un nuevo paquete, lo que generará una duplicación.
- En (d) se duplica el ACK, haciendo creer al enviador que se recibió un nuevo paquete OK.
- Por ello, es necesario identificar el frame **Y** el ACK, por lo que se utilizan números de secuencia.
- Para este caso, basta un número de secuencia de 1 bit.

STOP-AND-WAIT (5)



STOP-AND-WAIT (6)

- El protocolo está OK sin desorden de paquetes
- Frente a desorden, puede ocurrir que, en el escenario (d), el segundo frame (retransmisión innecesaria del primero) viaje por el mundo y llegue después, confundiéndose equivocadamente con otro
- Dibuje un escenario que reciba un paquete por otro para verificar este problema
- Frente a desorden, la única defensa es usar más bits en el número de secuencia, disminuyendo la probabilidad de confusión

BDP (1)

- La gran desventaja de Stop-and-Wait es que solo permite que un paquete de datos viaje al mismo tiempo, lo que se puede traducir en un uso ineficiente de la capacidad del enlace.
- ¿Cómo medir esa ineficiencia?
- ¿Cuánto pierdo en ancho de banda al esperar los ACKs?
- ¿Cuánto es el máximo ancho de banda que puedo lograr?

BDP (2)

- Si envío paquetes de 65 Kbytes y el RTT es de 0.5s
- Ancho de banda máximo: 65 Kbytes/0.5s => 130 Kbytes/s
- (pueden multiplicar por 10 para Kbits/s aprox)
- ¡Da lo mismo que tengan una conexión de alta velocidad!
- Habría que lograr que los paquetes fueran más grandes
- ¿qué tan grandes?
- *Tamaño / RTT = ancho de banda disponible*
- *Tamaño = ancho de banda disponible * RTT =* ***BDP***
- ***(Bandwidth-Delay Product)***

BDP (3)

- Otra forma de ver el BDP es: “la cantidad de bytes máxima que caben en el enlace que estoy usando”
- Para lograr el ancho de banda disponible, aún sin errores, necesito mantener el enlace lleno de bytes siempre.
- Ejemplos:
  - Un enlace de 20 Mbps con un RTT de 50 ms. El BDP será de 1 Mbit (~100 Kbytes)
  - Un enlace de 1 Gbps con un RTT 1ms. ¡El BDP será de 1 Mbit también!
  - Un enlace de 1Gbps con un RTT de 100ms. BDP será de 100 Mbits (~10 Mbytes)

BDP (4)

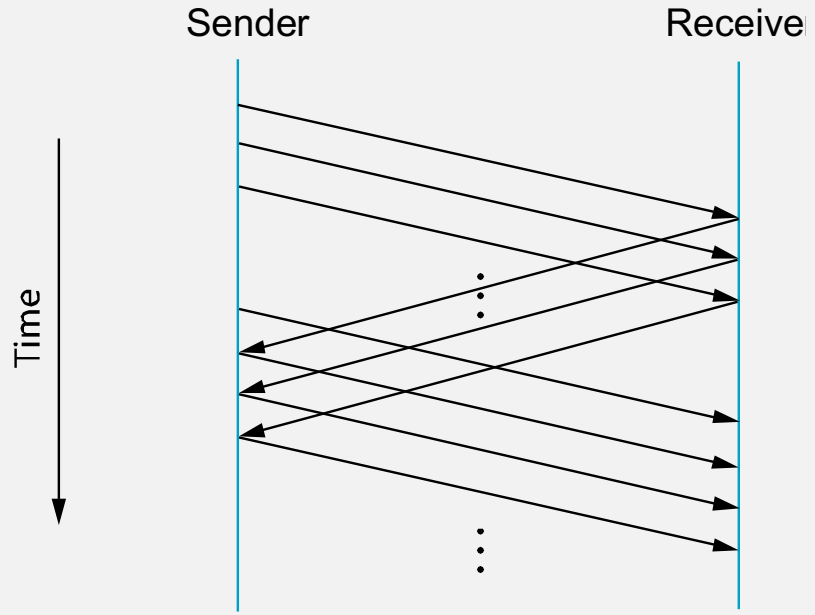
- No puedo suponer que tendré paquetes de esos tamaños
- Pero sí puedo tener varios paquetes en tránsito
- Basta que sumen ese tamaño entre todos
- En Internet, ¡puedo tener RTT enormes y anchos de bandas enormes!

BDP (5)

- El peor comportamiento lo tiene en *Long Fat Networks (LFN)*
- También llamadas “Elefantes”
- Ej: enlace satelital de 100Mbps, RTT de 600 ms
- BDP: ~6 Mbytes
- Es lo que cabría en el enlace hasta que llega el primer ACK
- Deberíamos entonces tener 'BDP' bytes en buffers de transmisión en espera de ACKs

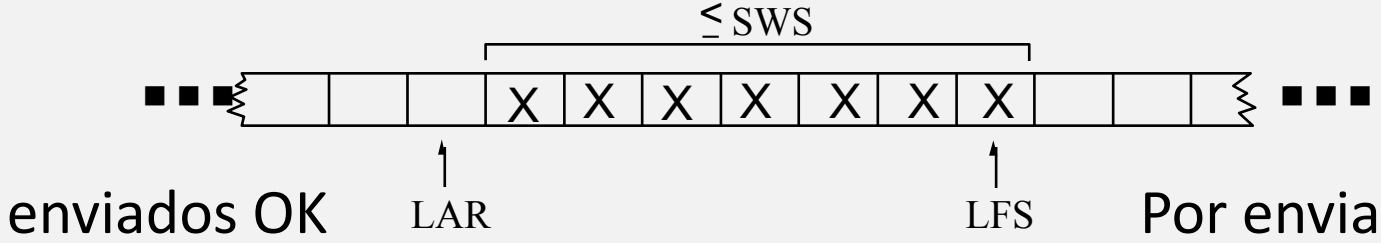
SLIDING WINDOW (1)

- En el caso de BDP 8KB y paquetes de 1KB quisiéramos enviar 8 paquetes a la vez, y poder enviar el noveno en cuanto llegue el ACK del primer paquete.



SLIDING WINDOW (2)

- El mecanismo de *Sliding Window* define lo siguientes elementos:
  - El emisor le asigna un *número de secuencia* a cada paquete que envía **SeqNum**.
  - El emisor mantiene tres variables. **SWS**=*send window size* (debiera aproximar BDP), que define el límite superior de paquetes sin confirmar que se pueden transmitir; **LAR**=*last ack received*, que indica el número de secuencia del último ACK recibido; **LFS**=*last frame sent*, que indica el número de secuencia del último paquete enviado.
  - El emisor mantiene el invariante:



SLIDING WINDOW (3)

- Cuando se recibe un ACK, el emisor “mueve” LAR a la derecha, lo que permite enviar otro paquete.
- El emisor tiene un timer asociado a cada paquete enviado (aprox RTT).
- Si expira antes de la recepción del respectivo ACK, debe retransmitir. Esto se traduce en que el emisor debe ser capaz de mantener SWS paquetes en caso de que los necesite para retransmisión.

SLIDING WINDOW (4)
Go-back-N

- El receptor mantiene una ventana de tamaño 1:
  - ExpectedSeq == paquete esperado
  - Todo paquete distinto a ese es descartado
  - Pero siempre enviamos ACK para ExpectedSeq-1
- El emisor retransmite la ventana completa si hay un timeout (*Go-Back-N)*

SLIDING WINDOW (5)
Go-back-N

- Al recibir un ACKn con n > LAR+1
  - Puedo dar por recibidos OK todos los paquetes enviados desde LAR+1 hasta n (no me habrían enviado este ACK si no los hubieran recibido bien)
  - Se define ACKn ⟹ ACKi i <= n
- Al recibir un paquete con número de secuencia > ExpectedSeq
  - Puedo dar por perdidos los paquetes entre medio
  - Enviar NACK para ExpectedSeq y apurar la retransmisión

SLIDING WINDOW (5.5)
Go-back-N

- Varios ACKs duplicados:
  - Puedo dar por perdido el paquete y apurar el timeout
  - Se define: 3 ACKs duplicados ⟶ timeout

SLIDING WINDOW (6)
Go-back-N

- Mucho más eficiente que Stop&Wait si la ventana de emisión es suficiente para llenar el BDP de la conexión
- Frente a pérdidas: retransmite la ventana completa, problemas si es muy grande y hay muchos errores
- En enlaces de alto BDP y alta pérdida necesitamos una mejor solución: aprovechar los paquetes transmitidos OK y no re-transmitirlos.
- Soporta mejor la pérdida de ACKs que de datos

SLIDING WINDOW (7)
Selective Repeat

- Ahora el receptor mantiene ventana también, con tres variables:
  - **RWS**=receive window size, que define el límite máximo de paquetes desordenados que puede aceptar.
  - **LAF**=largest acceptable frame, que define el número de secuencia más grande que puede aceptar.
  - **LFR**=last frame received, que define el número de secuencia del último paquete recibido tal que todos los paquetes con secuencia <= LFR ya se recibieron OK.
- El receptor mantiene el siguiente invariante:

$$LAF - LFR \leq RWS$$

SLIDING WINDOW (8)
Selective Repeat

- Cuando un paquete con número SeqNum llega, el receptor hace lo siguiente:
  - Si SeqNum<=LFR o SeqNum>LAF, se considera fuera de la ventana de recepción y se descarta.
  - Si LFR<SeqNum<=LAF, entonces el paquete está dentro de la ventana de recepción y es aceptado.
  - SeqNum <= LFR es una retransmisión
  - SeqNum > LAF es un paquete que no puedo recibir ya que mi ventana de recepción es muy pequeña

SLIDING WINDOW (9)
Selective Repeat

- Cuando un paquete con número SeqNum llega, el receptor hace lo siguiente (Continuación):
  - El receptor envía el ACK para el paquete recibido (SeqNum) si SeqNum <= LAF
  - Si SeqNum > LAF, envío ACK para LRF
  - Si SeqNum == LRF+1, ajusta la ventana hasta el próximo paquete no recibido.
  - ***Ahora un ACKn no implica Acks a < n***

SLIDING WINDOW (10)

- Notas
  - El tamaño de la ventana de emisión se calcula en base al BDP, ¡pero ahora sirve que sea más grande!
  - El tamaño de la ventana de recepción puede ser cualquiera.
    - Si RWS=1, el receptor no aceptará paquetes desordenados.
    - Si RWS=SWS, el receptor puede aceptar cualquier paquete que el emisor envíe.
    - Si RWS>SWS, no tiene mucho sentido, pues no pueden haber más de SWS paquetes desordenados.

NÚMEROS DE SECUENCIA (1)

- Originalmente consideramos que no habían restricciones para el número de secuencia, pero en la práctica eso no es posible: debe ser un número finito.
- Por lo tanto, los números de secuencia se reutilizan de cuando en cuando, lo que agrega un nuevo problema: cómo diferenciar un paquete “antiguo” de uno “nuevo” si tienen el mismo número.
- Ello nos obliga a estar seguros que el número máximo de secuencia sea mayor que el número de paquetes en viaje.
  - Por ejemplo, en Stop-and-Wait tenemos como máximo dos números de secuencia y sólo uno en viaje a la vez (si no hay desorden)

NÚMEROS DE SECUENCIA (2)

- Supongamos el caso de que SWS<=MaxSeqNum-1, donde MaxSeqNum es número de secuencias disponibles (de 0 a MaxSeqNum-1).
  - Si RWS=1, el valor es suficiente.
  - Si RWS=SWS, no es suficiente. Veamos un ejemplo
    - Sea MaxSeqNum=7, SWS=RWS=7.
    - El emisor envía los paquetes del 0 al 6, que son recibidos satisfactoriamente, pero todos los ACK se pierden.
    - El receptor espera ahora los paquetes 0..6, pero el emisor considera perdidos los paquetes 0..6 y los retransmite.
    - Al recibir el segundo bloque de paquetes, el receptor cree que está recibiendo la segunda generación de los paquetes, cuando en realidad son copias de los recibidos previamente, ¡y los acepta!

NÚMEROS DE SECUENCIA (3)

- Piense ahora en un Selective Repeat y un escenario en que tenemos MaxSeqNum=8, SWS=RWS=7
- Transmitimos 0..6 llegan bien y perdemos todos los ACK
- ¿Qué pasa ahora?
- ¿Funciona?

NÚMEROS DE SECUENCIA (3)

- La fórmula general está dada por Stop-and-Wait: que los números de secuencia alternen entre las dos mitades del espacio de números.
- Con ello podemos ver que SWS<(MaxSeqNum+1)/2
- Eso permite que caben dos ventanas y queda un número más disponible para separarlas.
- Todo número recibido fuera de LRF+SWS se considera “anterior”: enviamos ACK y descartamos el paquete

Protocolo TCP (1)

- Protocolo de transporte que ofrece un servicio confiable, orientado a la conexión y al flujo de bytes.
- Las aplicaciones no tienen que preocuparse de ordenar o retransmitir los datos, ni de dividir los datos en pedazos.
- Las conexiones son bidireccionales, con un flujo en cada dirección.

DISEÑO (1)

- El protocolo de ventana de corredera es esencial en el funcionamiento de TCP, pero la implementación supone una serie de retos.
- La estructura presentada originalmente suponía la operación sobre un enlace punto a punto, bajo ciertas condiciones controladas, pero TCP opera en base a canales lógicos sobre Internet.
- TCP requiere el establecimiento explícito de la conexión entre las partes. Durante ese proceso, se intercambia información esencial para la operación del algoritmo. Ello incluye la reserva de recursos, que deben ser liberados explícitamente mediante el cierre de la conexión

DISEÑO (2)

- Un enlace físico punto a punto tiene generalmente un RTT fijo, pero una conexión TCP deberá lidiar con RTT variables (y con variaciones durante la misma). Por ello, los valores de timeout deberán ser adaptivos
- Los paquetes pueden sufrir variaciones en el orden durante su viaje en Internet, lo que no pasa en un enlace punto a punto. El problema a resolver es cuan desordenados y retrasados se pueden admitir los paquetes. Se define un MSL (maximum segment life) en 120 segundos, como el valor máximo que un paquete puede vivir en Internet.

DISEÑO (3)

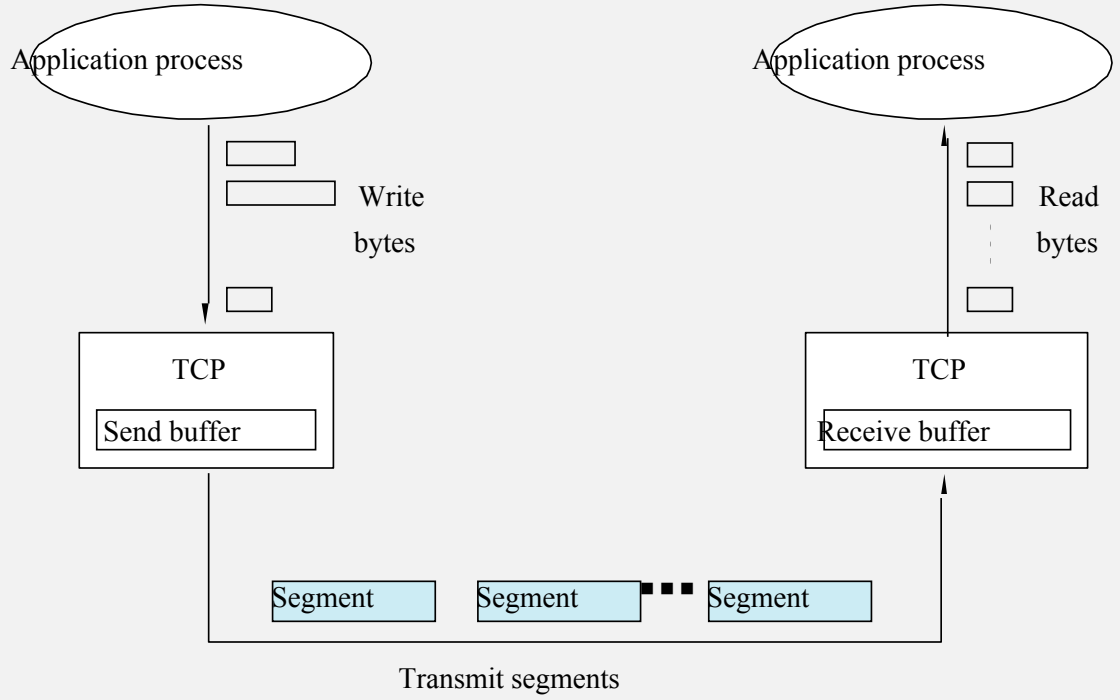
- En los enlaces punto a punto el BDP es fijo, pero en un canal lógico a través de Internet tenemos capacidad y RTT variable. Por lo que la asignación de recursos a cada conexión debe ser capaz de variar y adaptarse.
- En un enlace punto a punto, el emisor no puede enviar datos más rápido de lo que le permite el ancho de banda y sólo un host inyecta datos, por lo que no se puede saturar el enlace por error. En TCP el emisor no sabe por cuales enlaces va a atravesar. Puede estar directamente conectado a un enlace de gran capacidad pero en el trayecto cruzar por un enlace de mucha menor capacidad. Para peor, muchos otros hosts pueden estar inyectando datos en dicho enlace. Entonces el control de congestión es esencial.

INTRODUCCIÓN (2)

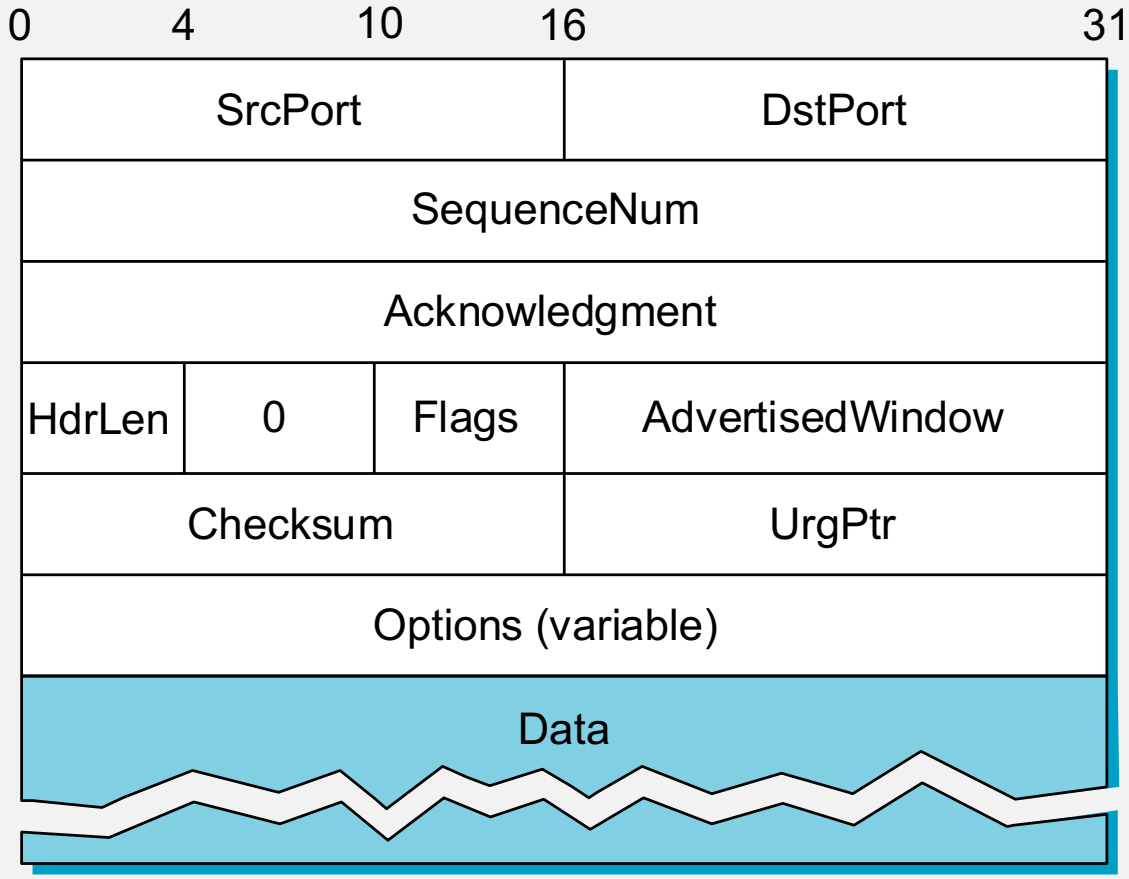
- Incluye control de flujo en cada dirección (de manera de que el receptor pueda limitar la cantidad de datos despachada por el emisor).
- Incluye control de congestión, que le permite ajustar la velocidad con que se envían datos para evitar saturar la red.

FORMATO DEL SEGMENTO (1)

- TCP es orientado al flujo de bytes, pero para transmitir utiliza bloques de bytes llamados segmentos, donde cada uno transporta un "*pedazo*" del flujo de bytes.

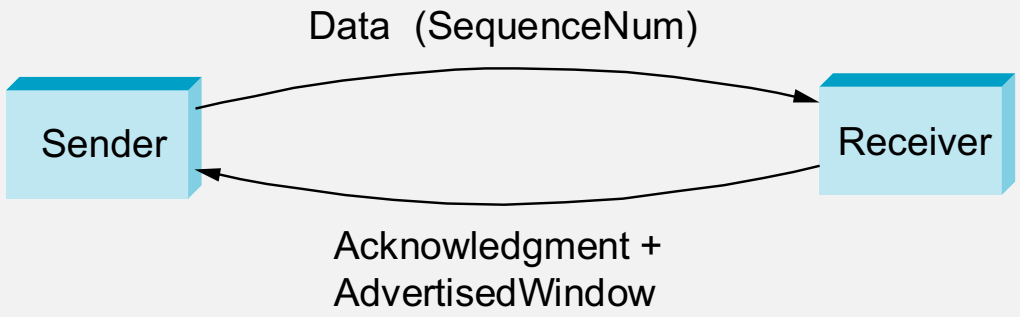


FORMATO DEL SEGMENTO (2)



FORMATO DEL SEGMENTO (3)

- SrcPort y DstPort son los puertos que identifican completamente el flujo: <SrcIP, SrcPort, DstIP, DstPort> identifican una conexión.
- Acknowledgment, SequenceNum y AdvertisedWindow están relacionados con el protocolo de ventana de corredera. SequenceNum indica la secuencia del primer byte en el segmento. Ack indica que el emisor está esperando ese número de secuencia. AdvWin indica la cantidad de espacio disponible en el receptor.



FORMATO DEL SEGMENTO (4)

- El parámetro Flags, se refiere a 6 bits usados para diferentes propósitos y cuyo orden es URG, ACK, PUSH, RESET, SYN y FIN.
  - SYN se usa para indicar el inicio de una conexión.
  - FIN se usa para indicar el fin de una conexión.
  - ACK indica que el contenido del campo Ack es válido y debe considerarse.
  - URG indica que el segmento contiene datos urgentes, lo que se traduce que el campo UrgPtr indica inicio de los datos no urgentes (que se puede entender como que los datos urgentes empiezan en el inicio del payload del segmento y terminan en el puntero)

FORMATO DEL SEGMENTO (5)

- Flags
  - PUSH significa que el emisor ha invocado la operación de push, por lo que el receptor debe “priorizar” el procesamiento de este segmento.
  - RESET significa que el receptor ha decidido cancelar la conexión por alguna razón.
  - Checksum se calcula sobre el header TCP, los datos TCP y el pseudoheader.
- HdrLen indica el largo del header TCP, medido en “palabras” de 32 bits.

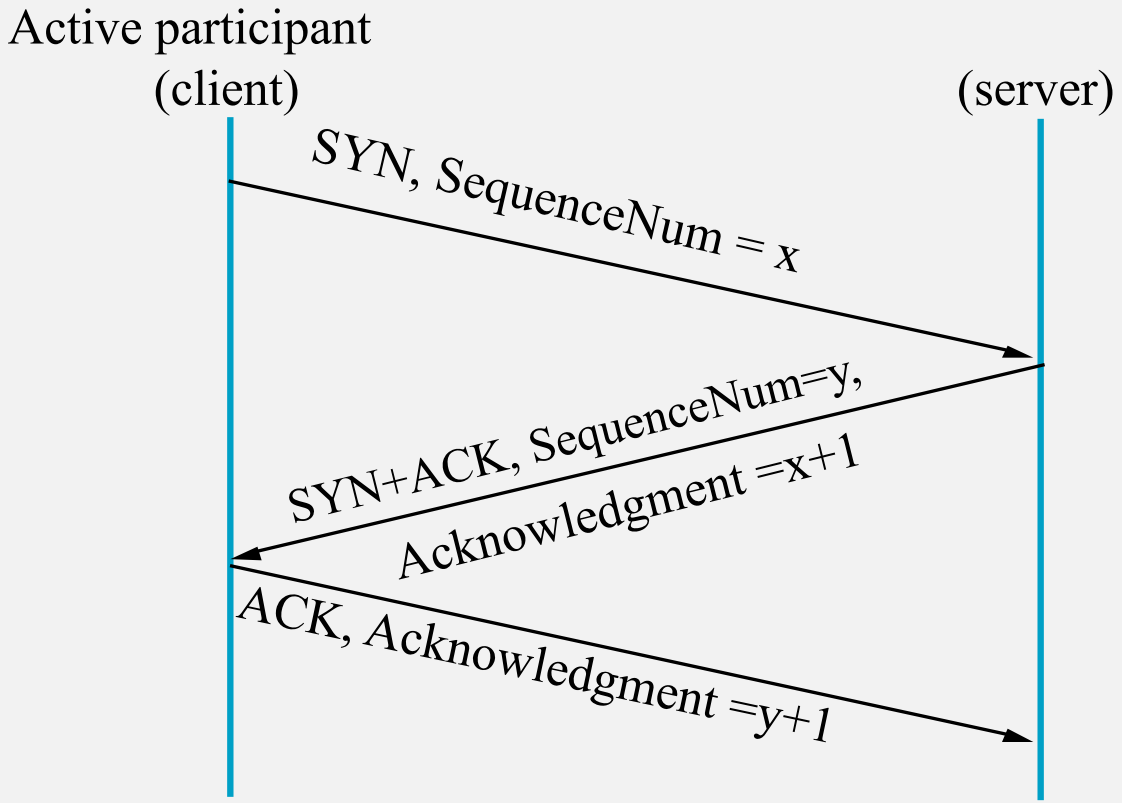
ESTABLECIMIENTO DE LA CONEXIÓN (1)

- Una conexión tiene un cliente, que realiza una “apertura” activa y un servidor que está pasivamente esperando un requerimiento.
- Sólo después del establecimiento de la conexión se transmiten datos.
- En cuanto el cliente ha terminado de enviar datos, inicia una ronda de mensajes para cerrar.
- Aún así, es posible que una de las partes cierre (no desee enviar más datos) pero el otro lado mantendrá su “dirección” del flujo abierta para continuar enviando datos.

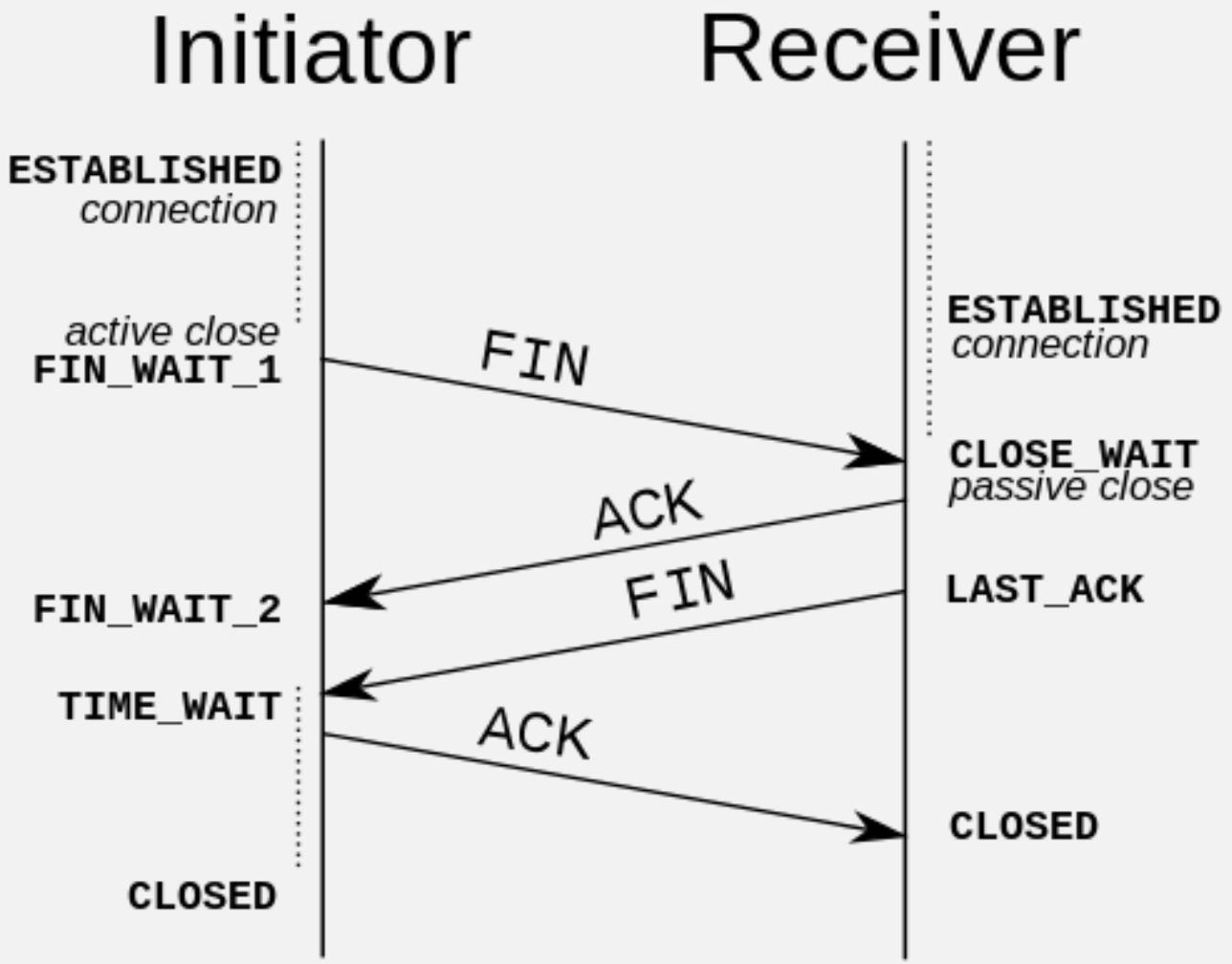
ESTABLECIMIENTO DE LA CONEXIÓN (2)

- Negociación de tres vías
  - El proceso de creación de una conexión en TCP se llama three-way handshake.
  - El objetivo es que las partes se pongan de acuerdo en ciertos parámetros que gobernarán la conexión, algunos de ellos definitivos y otros no.
    - Se fijan los números de secuencia en cada sentido, el tamaño de las ventanas y otras opciones.
    - El objetivo de fijar los números de secuencia (y no partir cada conexión con un número fijo) es evitar que un segmento retrasado de una conexión anterior pueda confundir o interferir con el actual estado.
  - Para los dos primeros segmentos enviados (SYN y SYN+ACK) existe retransmisión.

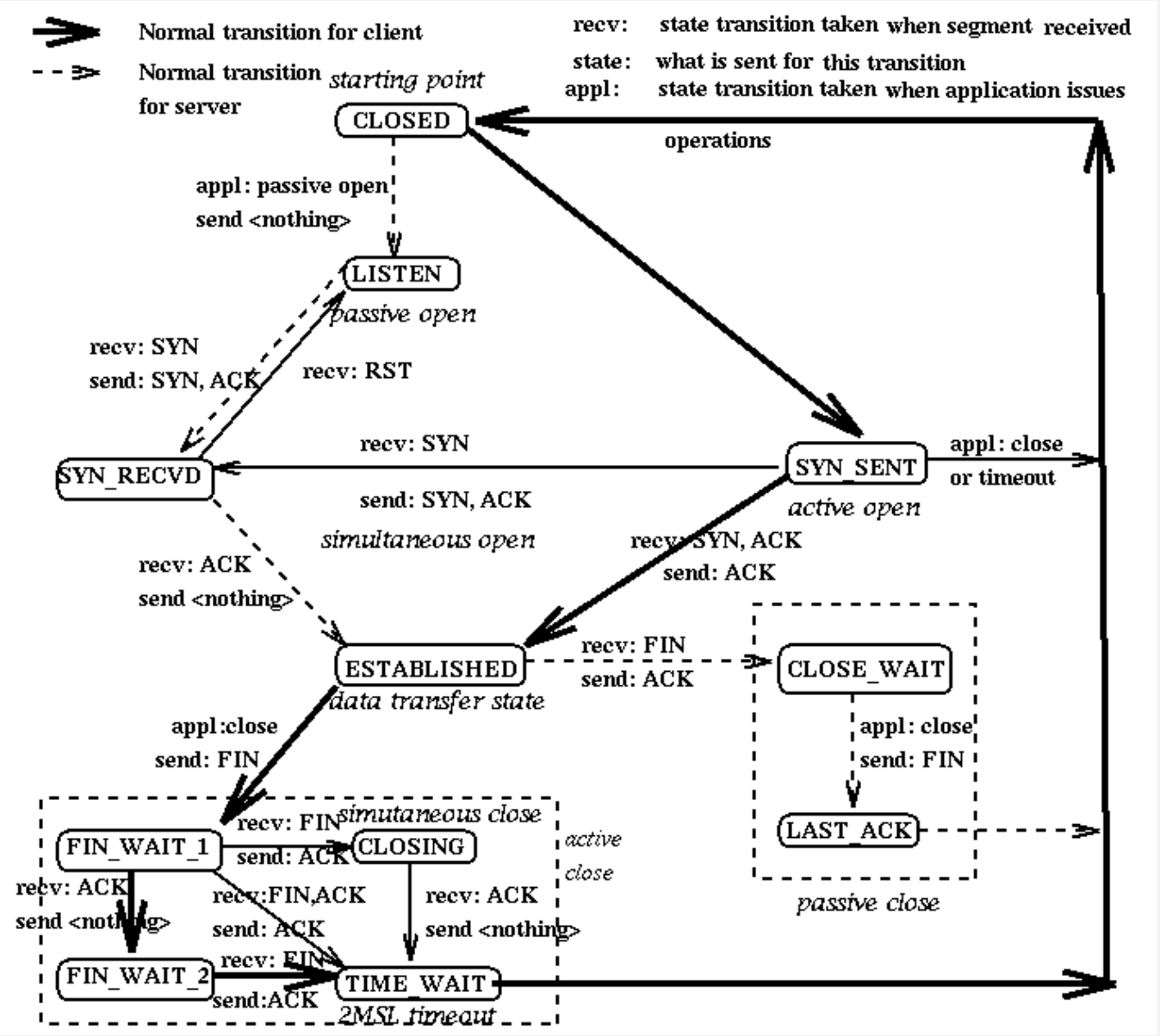
ESTABLECIMIENTO DE LA CONEXIÓN (3)



FIN DE LA CONEXIÓN



Estados TCP

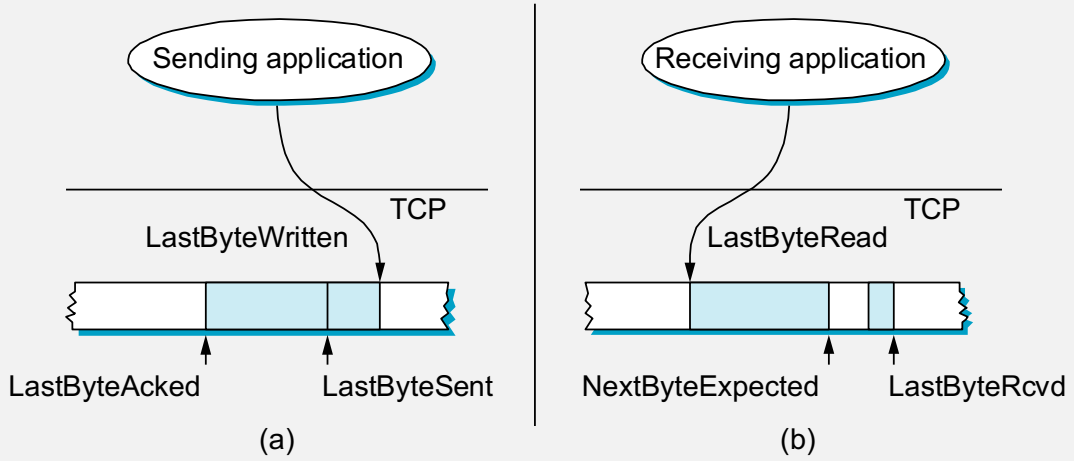

Normal transition for client
Normal transition for server
recv: state transition taken when segment received
state: what is sent for this transition
appl: state transition taken when application issues operations
starting point
CLOSED
appl: passive open
send <nothing>
LISTEN
passive open
recv: SYN
send: SYN, ACK
recv: RST
recv: SYN
send: SYN, ACK
simultaneous open
SYN_RECVD
SYN_SENT
appl: close
or timeout
active open
recv: SYN, ACK
send: ACK
recv: ACK
send <nothing>
ESTABLISHED
data transfer state
recv: FIN
send: ACK
CLOSE_WAIT
appl: close
send: FIN
LAST_ACK
passive close
appl:close
send: FIN
recv: FIN
send: ACK
simutaneous close
FIN_WAIT_1
CLOSING
active close
recv: ACK
send <nothing>
recv:FIN,ACK
send: ACK
recv: ACK
send <nothing>
recv: FIN
send:ACK
FIN_WAIT_2
TIME_WAIT
2MSL timeout

Estados TCP

- La cuadrupla: <SrcIP, SrcPort, DstIP, DstPort> identifica una conexión
- Eso permite tener más de un cliente conectado al mismo servidor, incluso desde la misma IP de origen
- Los S.O. tratan de no re-utilizar los ports demasiado rápido para no confundir paquetes de otras conexiones
- Una conexión TCP requiere 3+4 paquetes para la conexión a lo menos, y dos RTT: ¡para enviar un solo paquete de datos no es muy buena idea!

SLIDING WINDOWS Y TCP (1)

- Los tamaños de las ventanas no son fijos, sino variables y se establecen inicialmente en la negociación previa a la conexión (utilizando el campo *AdvertisedWindow* del header TCP.



SLIDING WINDOWS Y TCP (2)

- Cada punta mantiene un buffer de envío y recepción. Por simplicidad estudiaremos un solo sentido.
- En el emisor tenemos las siguientes variables: LastByteAcked, LastByteSent y LastByteWritten. Se cumple que LastByteAcked <= LastByteSent y que LastByteSent <= LastByteWritten.
- En el receptor tenemos: LastByteRead, NextByteExpected y LastByteRcvd. Se cumple que LastByteRead <= NextByteExpected y que NextByteExpected <= LastByteRcvd+1.

CONTROL DE FLUJO (1)

- Se definen *MaxSendBuffer* y *MaxRcvBuffer*, para especificar el tamaño de la ventana de envío y recepción respectivamente.
- Recordando "sliding window", el tamaño de la ventana de envío especifica la cantidad de datos que se pueden enviar sin esperar por un ACK.
- El receptor debe cumplir que

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}$$

para evitar saturar su buffer de recepción.

- Para ello anuncia una ventana de

$$\text{AdvertisedWindow} = \text{MaxRcvBuffer} - ((\text{NextByteExpected} - 1) - \text{LastByteRead})$$

CONTROL DE FLUJO (2)

- El valor anterior representa el espacio libre en el buffer de recepción. En la medida que se reciben datos, los receptor los confirma.
- El valor de LastByteRcvd se incrementa, lo que puede producir que la ventana se achique.
  - Esto dependerá de que tan rápido pueda la aplicación “consumir” los datos.
    - Si LastByteRead se incrementa con la misma tasa de LastByteRcvd, entonces la ventana conserva tu tamaño.
    - Si no, la ventana anunciada se achicará hasta llegar a cero.

CONTROL DE FLUJO (3)

- Por otro lado, el emisor debe respetar que:

$$LastByteSent - LastByteAcked \leq AdvertisedWindow$$

- por lo que se define una ventana efectiva que limita cuantos datos se pueden enviar.

$$EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)$$

- El valor efectivo tiene que ser mayor a cero para que salgan datos desde el origen. El emisor debe cumplir con:

$$LastByteWritten - LastByteAcked \leq MaxSendBuffer$$

CONTROL DE FLUJO (4)

- Para no saturar el buffer de emisión. Si un proceso intenta escribir y bytes, pero se cumple que:

$$(LastByteWritten - LastByteAcked) + y > MaxSendBuffer$$

- TCP bloqueará al emisor y no permitirá generar más datos.
- Si el receptor libera espacio y lo anuncia mediante un aviso incluído en un ACK, entonces el origen podrá retomar el envío de datos.
- ¿Qué pasa cuando la ventana de recepción llega a cero? ¿Es el receptor quién anuncia posteriormente la habilitación de espacio?
  - La respuesta es NO. El receptor sólo anuncia la ventana en respuesta a segmentos originados en el emisor.
  - Es por ello que cuando se llega a un anuncio de Ventana=0, el emisor envía a intervalos regulares segmentos con payload de 1 byte y el mismo número de secuencia, para generar la respuesta.
  - Esto sigue el principio de emisor inteligente/receptor tonto y que justifica la no existencia de los NAK.

SILLY WINDOWS SINDROME (1)

- Si un emisor se encuentra detenido a la espera de espacio y le llega un anuncio de que se desocupó MSS/2 bytes, ¿qué se debería hacer?
  - Esperar a que hayan suficientes bytes para llegar un segmento
  - Enviar cuanto antes los datos en espera
- Originalmente se actuaba agresivamente, enviando datos en cuanto hubiera espacio en el receptor. Esto derivó en el problema de la ventana tonta. Si llega una serie de anuncio de ventana de 1 byte, se enviarán segmentos con 1 byte de datos (claramente ineficiente)

SILLY WINDOWS SINDROME (2)

- Para resolver el problema, Nagle propuso el siguiente algoritmo.

```
When the application produces data to send
If both the available data and the windows >= MSS
     send a full segment
else
     if there is unACKed data in flight
          buffer the new data until an ACK arrives
     else
          send all the new data now
```

NÚMEROS DE SECUENCIA (1)

- En TCP, así como lo vimos en el protocolo de ventana original, existen números de secuencia que pueden reutilizarse.
- Descubrimos que el espacio de números debía al menos duplicar el tamaño de ventana posible. En TCP se cumple, pues los números de secuencia son de 32 bits y las ventanas de a lo más 16 bits (64K). *Ver Window Scaling y Timestamps* para ver cómo tener ventanas de 1 Gbyte.
- Aún así, es importante considerar que los números de secuencia pueden dar la vuelta en una conexión, por lo que para distinguir diferentes encarnaciones del mismo segmento, suponemos que no pueden sobrevivir en Internet más allá de MSL segundos (Maximum Segment Lifetime, actualmente en 120 segundos).

NÚMEROS DE SECUENCIA (2)

- Esto pasará en la medida de la rapidez con que utilicemos números de secuencia.

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

VENTANA DE EMISIÓN

- Con el objetivo de mantener el enlace en máximo uso, los 16 bits de la ventana podrían resultar insuficientes.
- Se ha creado un mecanismo opcional llamado Window Scaling que permite definir una ventana más allá del límite de los 16 bits, para nodos que lo soporten.

Bandwidth	BW Delay Product (Delay = 100ms)
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
FDDI (100 Mbps)	1.2 MB
STS-3 (155 Mbps)	1.8 MB
STS-12 (622 Mbps)	7.4 MB
STS-24 (1.2 Gbps)	14.8 MB

ENVÍO DE SEGMENTOS

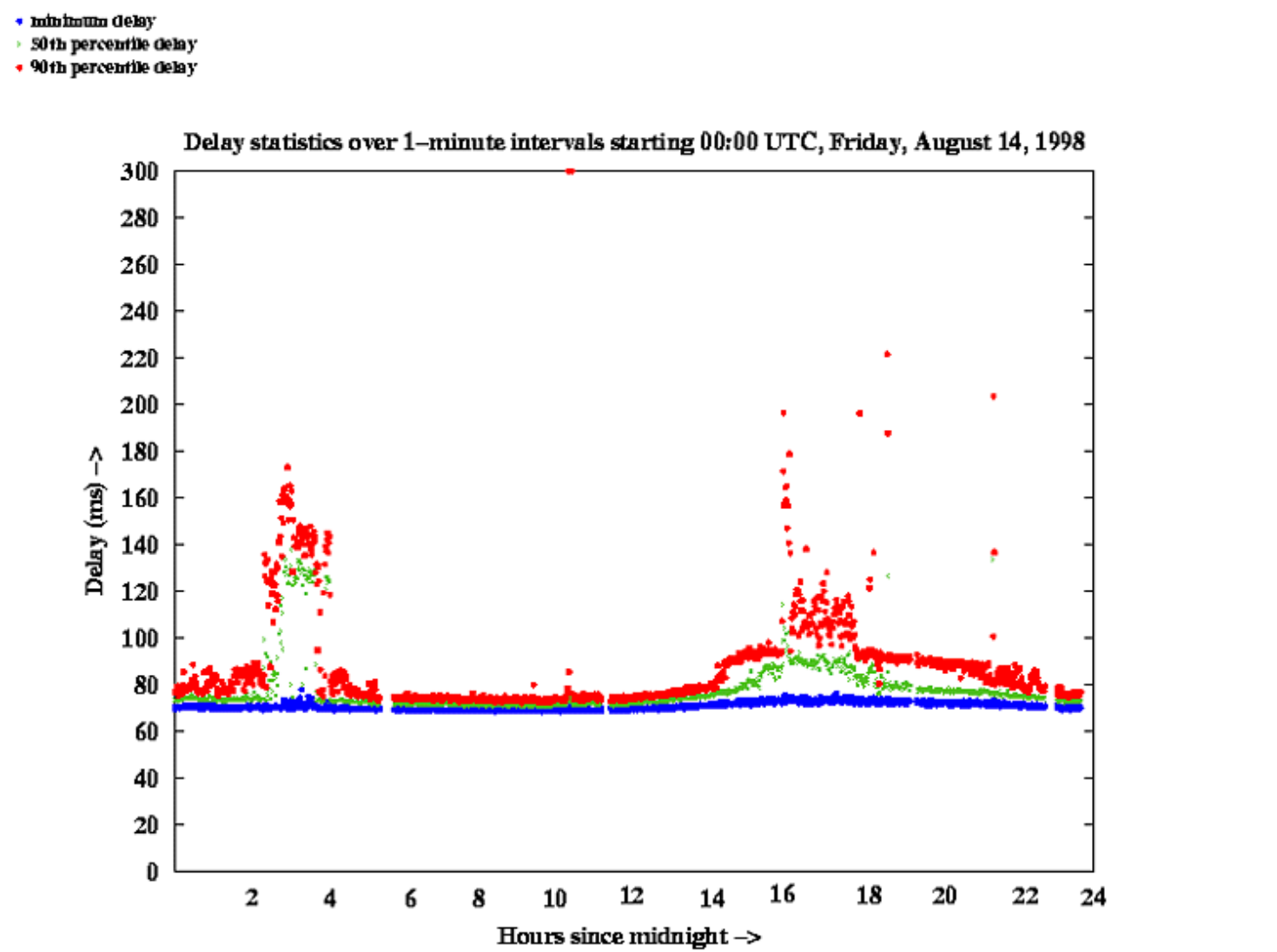
- TCP mantiene una variable llamada MSS (Maximum Segment Size), que se calcula en base a la MTU local, menos el tamaño del header IP menos el tamaño del header TCP, para evitar la fragmentación.
  - Si se soporta MTU Path Discovery, se utiliza el valor descubierto en vez de la MTU local.
- ¿En qué momento envío un segmento?
  - Tan pronto como ha juntado bytes para llegar un segmento.
  - Cuando el proceso explícitamente lo ha solicitado, a través de la opción PUSH.
  - Cuando se acaba cierto tiempo de espera, donde se envía lo que hubiere estado esperando para despacho.

MTU Path Discover

- Envío Segmentos con el bit de No Fragmentar
- Si recibo ICMP “Need to Fragment”
  - Adapto el MSS al nuevo tamaño (viene en el ICMP, o intento con uno más pequeño)
  - Si timeout: intento con uno más pequeño

- Siempre mantengo el bit de No Fragmentar
- Me adapto en caso de cambio de rutas
- Pero MTU Path nunca crece
- En IPv6 eliminamos fragmentación en ruta por esto mismo

Timeouts (1)



Timeouts (2)

- El Timeout controla el tiempo de espera que debiera ser mayor que el RTT
- Pero el RTT tiene varianza muy alta en caso de congestión (colas en routers)
- En ese caso, incluso 2*RTT promedio puede ser poco timeout
- Debemos estimar la varianza junto con el RTT promedio
- Y en qué intervalos? (historia reciente vs largo plazo)
- Timeout = RTT+K*MDEV

Timeouts (3)

- ¿Por qué la varianza es alta?
- -> Sin congestión: RTT es constante
- -> Congestión hace que las colas de espera en los routers crezcan: mientras más RAM, más varianza
- -> Congestión extrema genera pérdidas, cuando se acaba la RAM
- Mientras más routers en el camino, mayor probabilidad de varianzas altas

Timeouts (4)
Ver: RFC6298

- DIFF = RTT_sample – RTT;
- RTT = RTT + alfa*DIFF;
- MDEV = (1-beta)*MDEV + beta*abs(DIFF);
- TIMEOUT = RTT + 4*MDEV
- Alfa = 1/8 típicamente y beta = 1/4

=> ¿historia 8 veces más fuerte que instante?

- Problema: si tuve que retransmitir un segmento, ¿cómo sé cual es su RTT?
- Karn: no actualizar RTT en ese caso
- Duplicar el timeout (max 120 s)

Timeouts (5)

- Inicialmente: TIMEOUT = 3s
- Al recibir primer ACK:
  - RTT = RTT_sample
  - MDEV = RTT/2
- Opción de timestamp en TCP permite aceptar ACKs duplicados recalculando RTT y MDEV
- Pero, con Go-Back-N:
  - Si se pierde un paquete de la ventana,
  - Recibo ACK para el anterior múltiples veces
  - ACKs duplicados => pérdida (fast retransmit)

Ventana Congestión (1)

- En 1988 Van Jacobson descubrió la congestión: si todos retransmitían sus ventanas todo empeoraba
- En TCP suponemos pérdida == congestión
- Algoritmo AIMD (Additive Increase, Multiplicative Decrease), frente a congestión:
  - *Multiplicative Decrease*: cada retransmisión decrementa la ventana de transmisión a la mitad y multiplica el timeout por dos. Converge a un segmento muy rápido.
  - *Additive Increase*: cuando comienzo a recibir los ACKs, incremento la ventana de a un segmento por RTT!

Ventana Congestión (2)

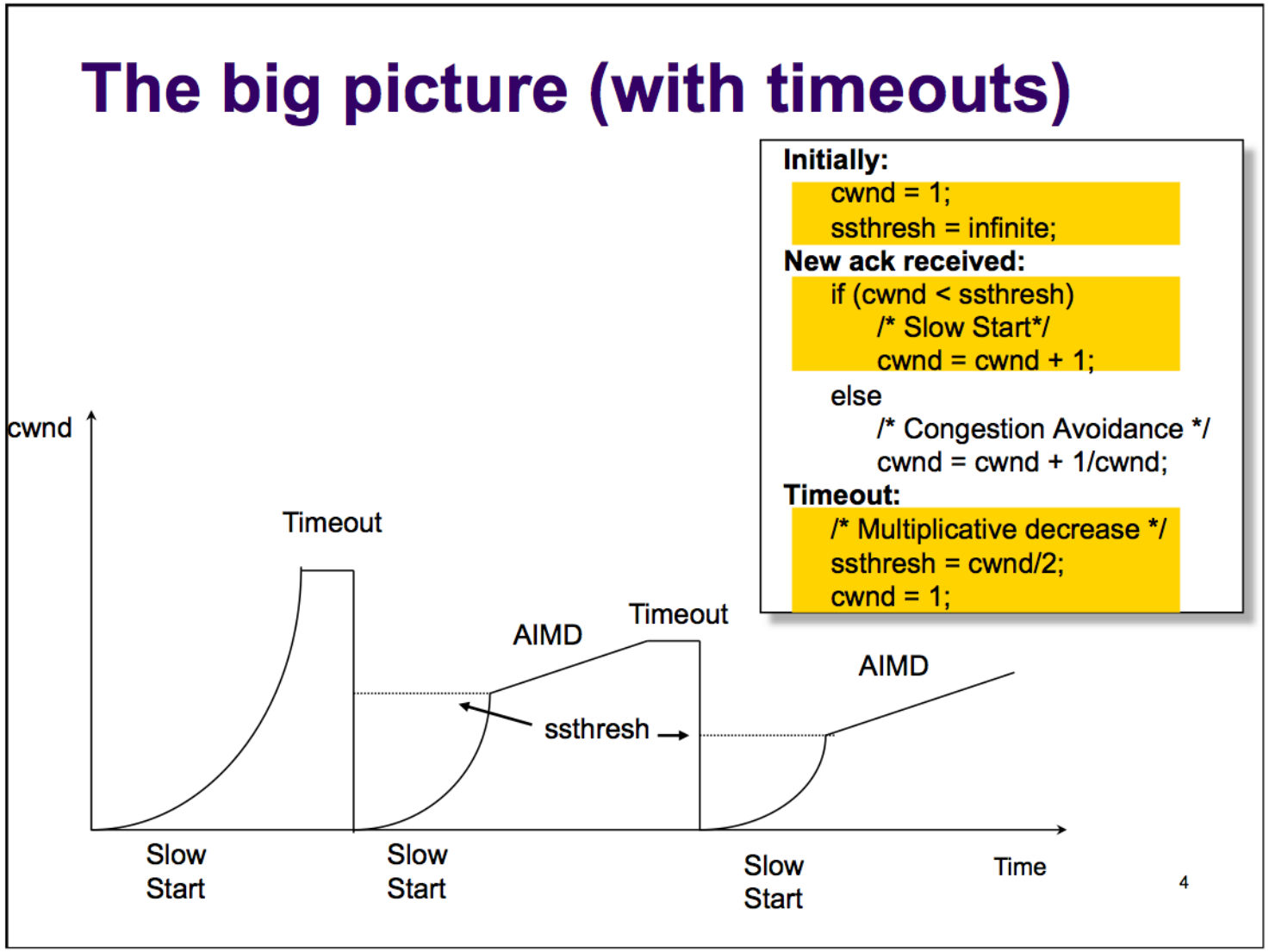
- *Slow Start*: Cuando no hay congestión, la ventana crece en un segmento por ACK recibido
- Esto no es nada de *Slow:* Cada ACK corre la ventana en un segmento y la agranda en otro => genera el envío de dos segmentos, que generan dos ACKS los que generan cuatro segmentos… En realidad es exponencial
- Aquí hablamos en segmentos, pero recuerden que en TCP los contadores siempre son en bytes
- Usaremos dos ventanas en el emisor: la normal y la de congestión. Parten iguales, pero la de congestión manda
- Pueden haber paquetes transmitidos, sin ACK, fuera de la ventana de congestión que NO se retransmiten

Ventana Congestión (3)

- *Slow Start*/AIMD en pseudo-código:

Init:
```
cwnd = 1;
   ssthresh = infinite;
```
ACK:
```
if(cwnd < ssthresh) /* slow start */
    cnwd = cwnd + 1; /* en segmentos */
else /* congestion avoidance: Additive
                                  Increase */
    cwnd = cwnd +1/cwnd;
```
TIMEOUT:
```
/* Multiplicative decrease */
    ssthresh = cwnd/2;
    cwnd = 1;
```

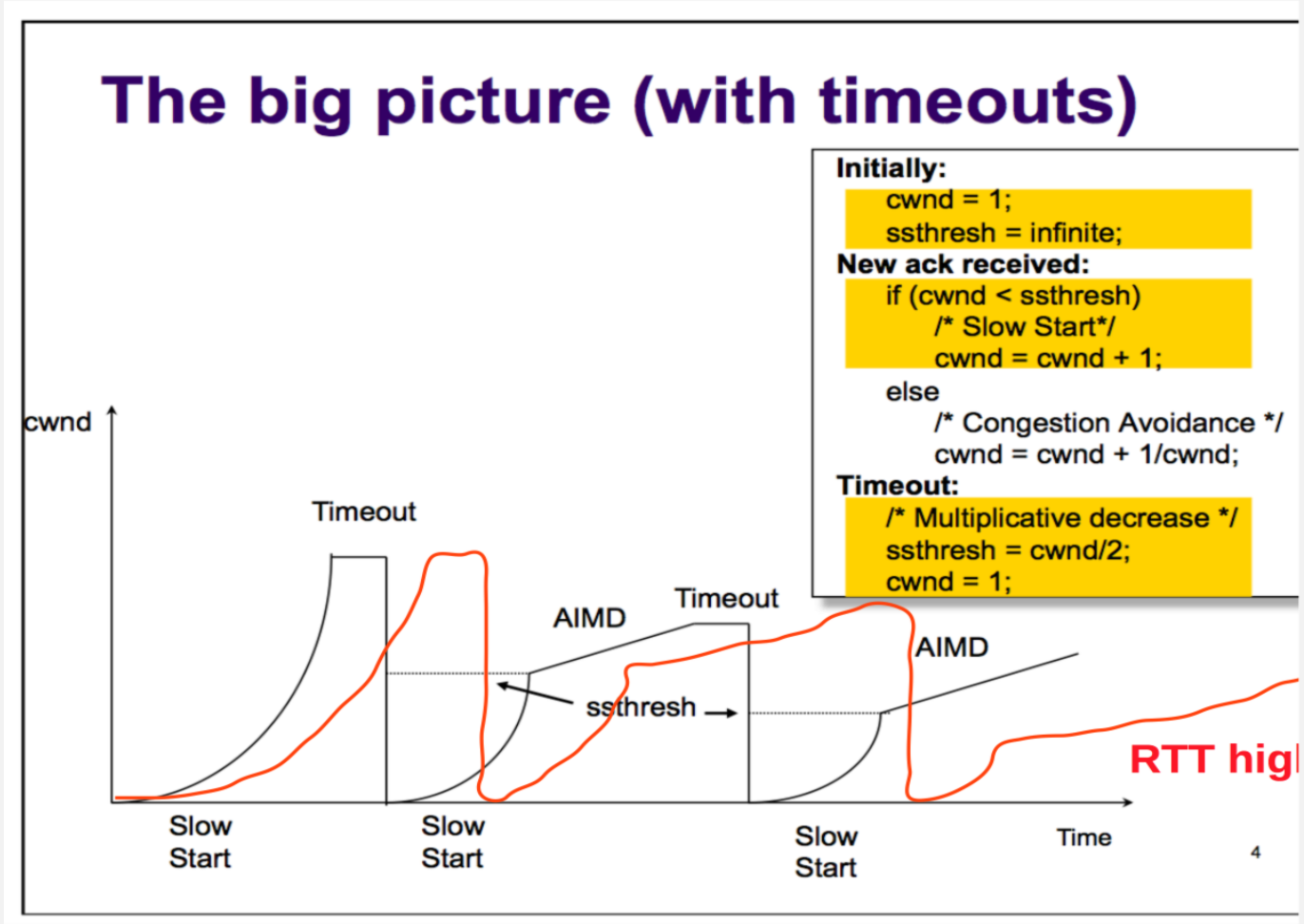
Ventana Congestión (4)



Ventana Congestión (5)

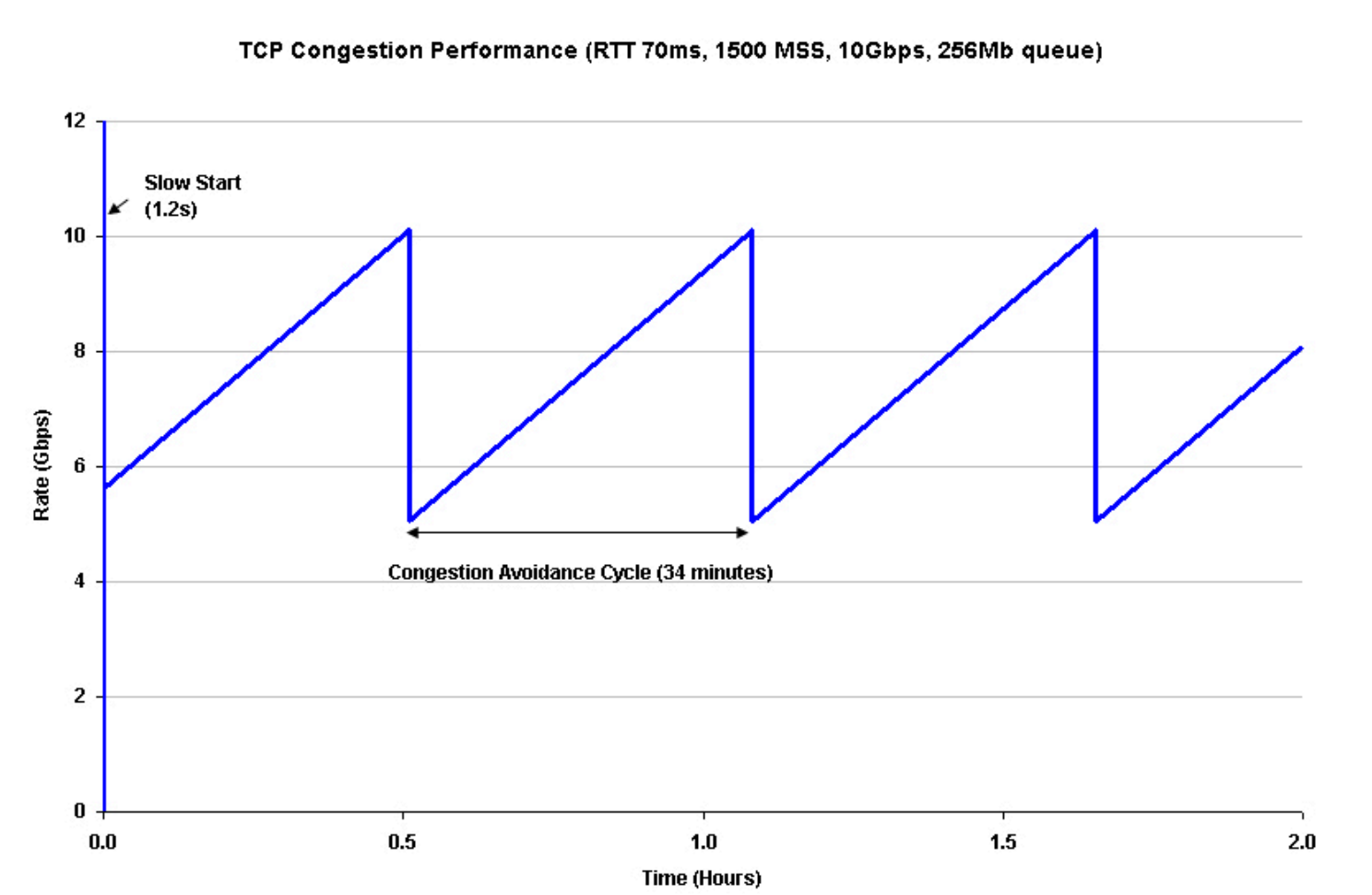
- *Fast Retransmit:* 3 ACKs duplicados => timeout
- *Fast Recovery:* hacer cwnd = ssthresh en vez de 1
- Ventana inicial + grande
- SACK: Selective Repeat en vez de Go-back-N
- Varios “sabores” de TCP (Reno, Vegas, BIC, etc.)
- En linux se puede elegir
- Pero aún tenemos varios problemas pendientes: ¿es posible hacer un TCP para todos los escenarios posibles?
- Mucha investigación: DCTCP, …, TFRC:
- Throughput ~= (1/RTT)*sqrt(3/2p)

Ventana Congestión (6)
El problema del RTT



Ventana Congestión (7)

Ancho de Banda se ve como una "sierra" en el tiempo:



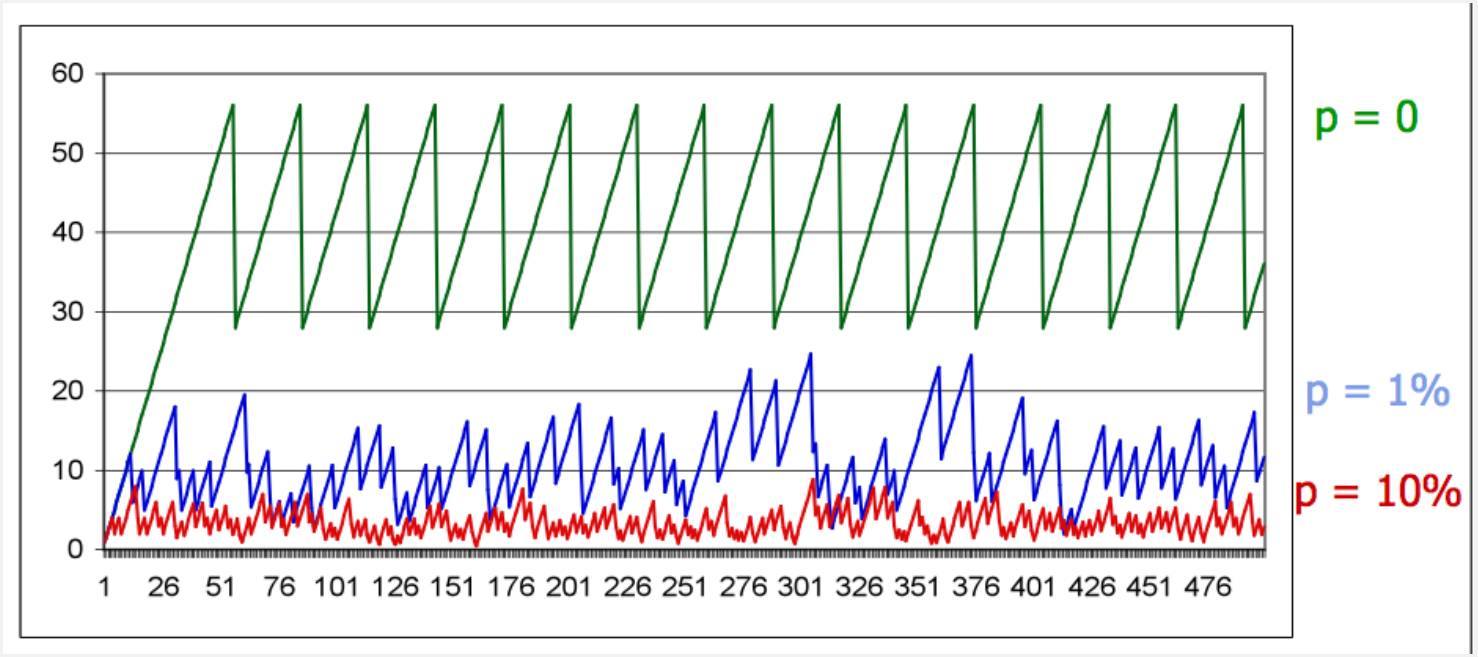
Ventana Congestión (8)

- Enlace de 1.5 Mbps, 70ms, en 0.5s llega a máximo, después de 150 Kbytes
- Enlace de 10Gbps, 70ms, demora 34 mn (sin pérdida!), después de 2 Tbytes!
- Enlace de 100 Mbps, 250ms, 10mn!
- -> TCP entre Chile y USA demora al menos 5 minutos en estabilizarse
- -> Imposible sacar ancho de banda completo
- -> Al sacar RTT de la ecuación dañamos la *fairness*
- -> Se usan varias conexiones simultáneas en esos casos: ¿trampa? (ej: speedtest.net)

Ventana Congestión (9)

- TCP es un buen ciudadano: fairness, *TCP-friendly*
- Pero si la pérdida es física (señal inalámbrica, por ejemplo) el resultado es nefasto => stop-and-wait con timeouts enormes
- Discusión hoy: *cross-layer information,* se trata de que TCP mire el layer físico para ver si los errores son locales, saltándose IP
- En caso límite, conviene matar una conexión y empezar otra vez, cuando la señal es fuerte

Ventana Congestión (10)
Pérdidas Físicas (ruido)



Ventana Congestión (11)

- El manejo de congestión en TCP se considera aún un problema de investigación
- No hemos encontrado mejores soluciones genéricas que Van Jacobson 1988-1990
- Muchas propuestas específicas
- Timeout, Backoff, ventana de congestión han operado increíblemente bien por más de 20 años!

Ventana Congestión (12) Bibliografía

- Van Jacobson, 1988:
- RFCs: rfc4614, rfc6077, rfc7323
- Material pirateado para este curso:
- http://www.potaroo.net/ispcol/2005-06/faster.html
- http://inst.eecs.berkeley.edu/~ee122/fa09/
- http://www.pcvr.nl/tcpip/tcp_time.htm