TCP - UDP

Redes de Computadoras

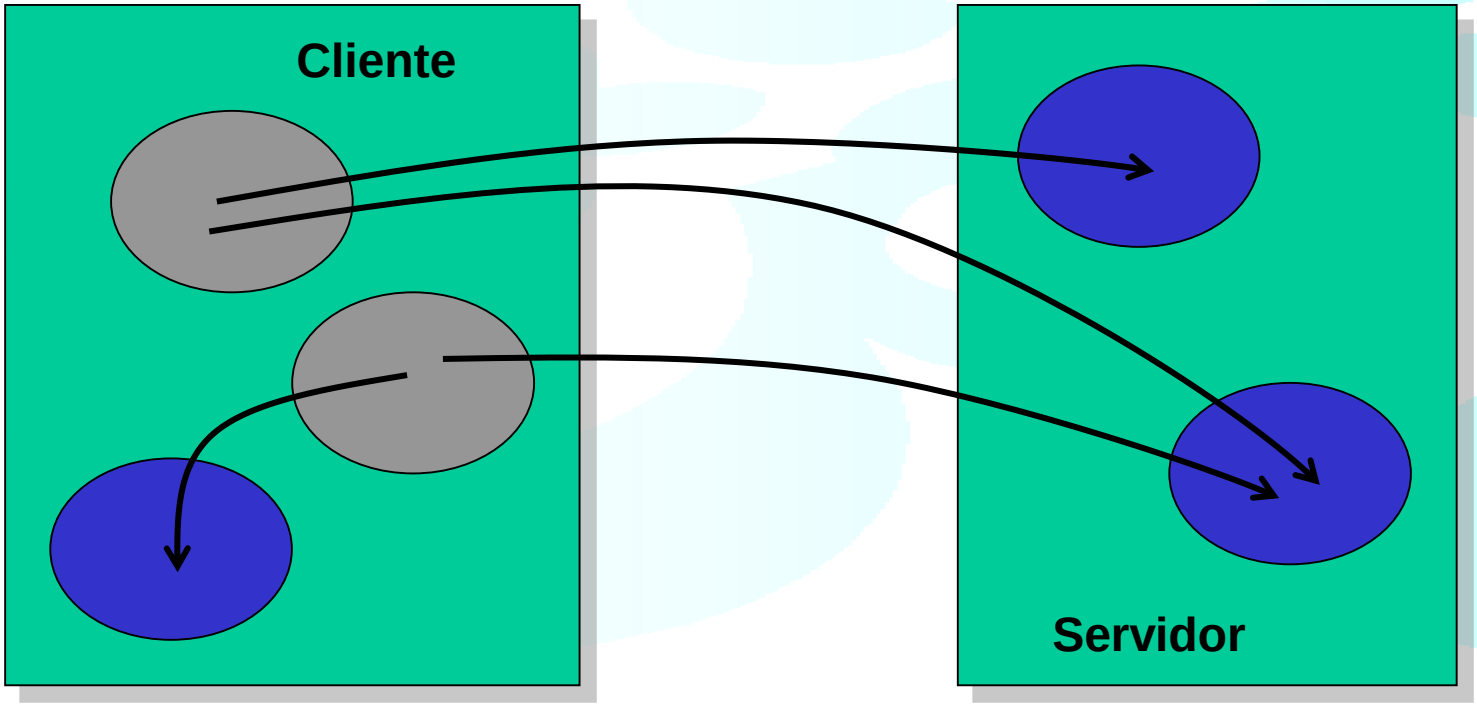
FIUBA

Ing. Marcelo Utard

mutard@fi.uba.ar

Trasmission Layer
IPC Inter Process Communication

El fin último del mecanismo de comunicación entre computadoras es que un proceso pueda comunicarse con otro proceso que corre en otra máquina (o en la misma) de modo transparente.



Trasmission Layer
IPC Inter Process Communication

Modelos de interacción entre procesos aplicativos distribuidos

- **Peer to Peer**
- **Client / Server**
- **Publisher / Subscriber**

Trasmission Layer
Protocolos TCP / UDP

Los protocolos de nivel **transmission** propuestos por DARPA son:

- **TCP** - Transmission Control Protocol
- **UDP** - User Datagram Protocol

TCP y UDP son protocolos “end to end”

TCP y UDP viajan **sobre IP**

Las PDUs de TCP y UDP se encapsulan en IP Datagrams

- PDU de TCP: **TCP Segment**
- PDU de UDP: **UDP Datagram**

Trasmission Layer
Multiplexado con Port Numbers

Para implementar la comunicación IPC, tanto en TCP como en UDP se utiliza el Source y Destination **Port Number**

El trafico se multiplexa entre los distintos procesos aplicaivos utilizando 5 parámetros:

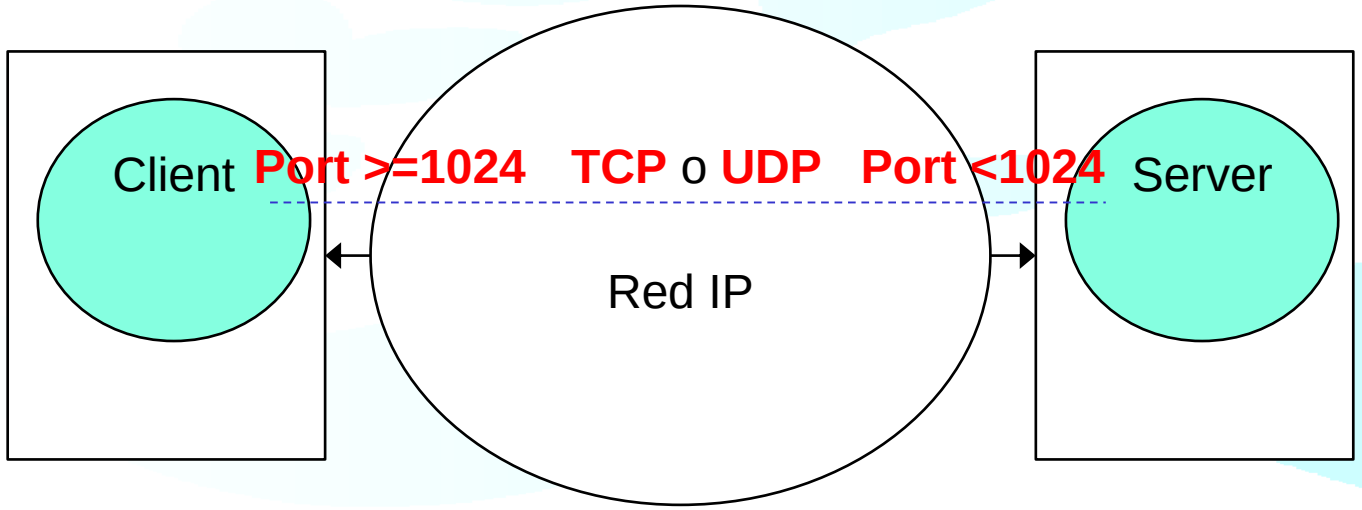
- Source IP **Address**
- Source Transmission Protocol **Port** Number
- Transmission **Protocol** Number (TCP o UDP)
- Destination IP **Address**
- Destination Transmission Protocol **Port** Number

Al par de parámetros (IP Addr, Port Number) se lo denomina **End-Point**

Trasmission Layer

TCP / UDP Well Known Services / Reserved Port Numbers

Ciertas aplicaciones Client/Server
conocidas como "**Well Known Services**“,
tienen **port numbers reservados (<1024)** que usa el **Server**
y los **Client** usan **ephemeral port numbers** (>= 1024)



Trasmission Layer

TCP / UDP Well Known Services / Reserved Port Numbers

Well known service	Protocol	Reserved Port Number
TELNET	**TCP**	**23**
FTP	**TCP**	**20, 21**
DNS	**UDP, TCP**	**53**
HTTP	**TCP**	**80**
HTTPS	TCP	443
SMTP	**TCP**	**25**
POP3	TCP	110
IMAP4	TCP	143
IMAP4 over TLS/SSL	TCP	993
SSH	TCP	22
SNMP	UDP	161, 162

Trasmission Layer

UDP *User Datagram Protocol*

User Datagram Protocol (**UDP**)

provee un mecanismo de **transporte extremo a extremo no conectado**

con un **overhead mínimo**

UDP ofrece a las aplicaciones un acceso directo a IP,

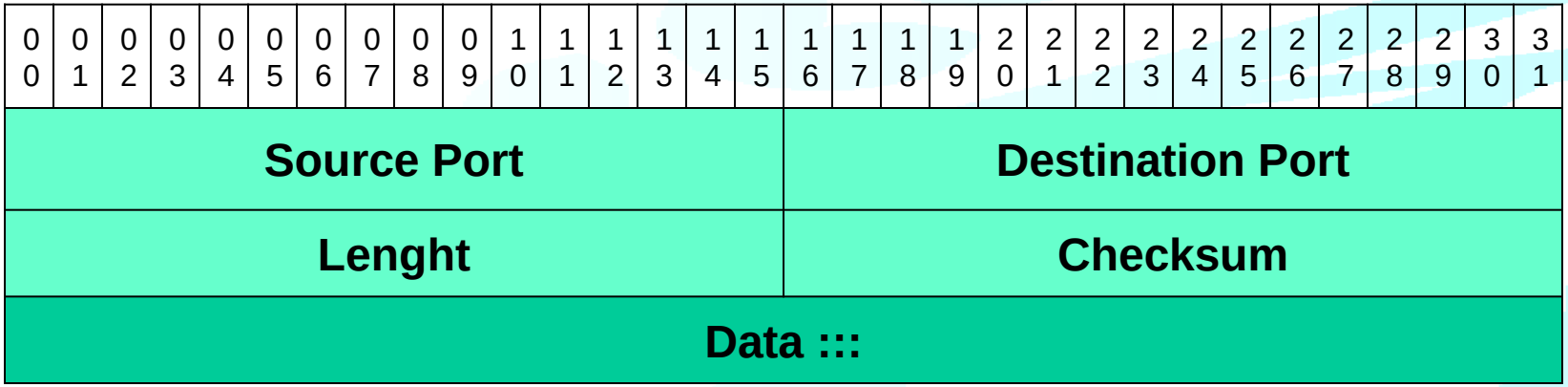
con **multiplexado de trafico IPC** (**Port Numbers**)

El programa aplicativo sobre UDP será el responsable de asegurar la confiabilidad y el control de flujo.

Trasmission Layer

UDP Header

Formato del paquete UDP:



El uso del CheckSum es opcional y sirve para el chequeo de la integridad del header y de los datos del paquete. El checksum se calcula agregando un pseudo-header que contiene la source y destination address, así como el protocol number y el UDP packet length, afín de poder a su vez verificar sin lugar a dudas que el paquete llego a su correcto destinatario.

Trasmission Layer

UDP *User Datagram Protocol*

El Datagrama UDP **se encapsula sobre IP**.

El **tamaño máximo de un Datagrama UDP**
esta dado por el payload máximo de un Datgrama IP
aproximadamente **64 KB** (2^16 menos el header IP).

En el caso que sea necesario hacer fragmentación a nivel IP,
el Datagrama UDP puede ser transportado por varios Datagramas IP.
Solo el 1er fragmento IP transporta el header UDP.

Trasmission Layer
TCP Transmission Control Protocol

TCP provee un servicio de **extremo a extremo**
de **transmisión confiable**,
orientado a la conexión (**connection oriented**),
de flujo de bytes (**byte stream oriented**).

TCP considera que las capas inferiores no ofrecen servicios confiables de transporte o transmisión de datos de extremo a extremo.

TCP
Funcionalidades

- **Multiplexado** de trafico IPC con **Port Numbers**
- **Transmisión Byte Oriented Full duplex**
- **Segmentación**
- **Control de secuencia**
- **ARQ Sliding Window**
- **Control de errores**
- **Control de Flujo explícito**
- **Control de Congestión implícito**

TCP
Encapsulado sobre IP

TCP es **Byte Stream Oriented**, **Full Duplex**.

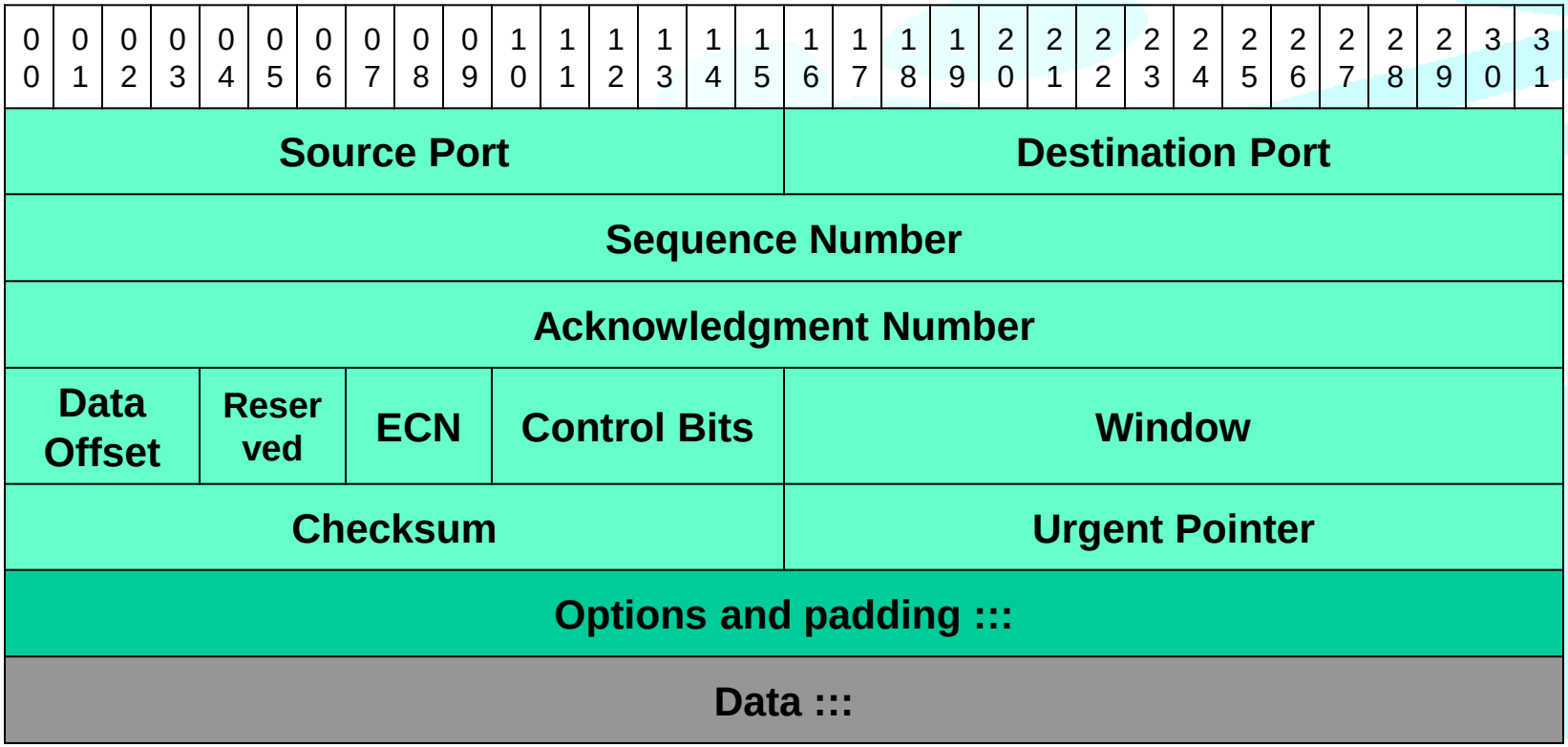
El Data Stream es enviado en **Segmentos TCP**.

Cada Segmento TCP **se encapsula sobre IP**.

El **tamaño máximo de un Segmento TCP**
esta dado por el payload máximo de un Datagrama IP
aproximadamente **64 KB** (2^16 menos el header IP).

En el caso que sea necesario hacer fragmentación a nivel IP,
el Segmento TCP puede ser transportado por varios Datagramas IP.
Solo el 1er fragmento IP transporta el header TCP.

TCP
Header


0 0 0 1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 1 0 1 1 1 2 1 3 1 4 1 5 1 6 1 7 1 8 1 9 2 0 2 1 2 2 2 3 2 4 2 5 2 6 2 7 2 8 2 9 3 0 3 1
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Reserved
ECN
Control Bits
Window
Checksum
Urgent Pointer
Options and padding :::
Data :::

TCP
Header

- **SN Sequence Number**
  Nro de secuencia del 1er byte contenido en el segmento

- **ACK Acknowledge Flag/Bit**
- **AN Acknowledge Number**
  Nro de secuencia del próximo byte que se espera recibir

- **W Advertised Window**
  Cantidad máxima de bytes que se pueden seguir recibiendo a partir del AN

TCP
Header

- **SYN Synchronize Flag/Bit**
- **FIN Final Flag/Bit**

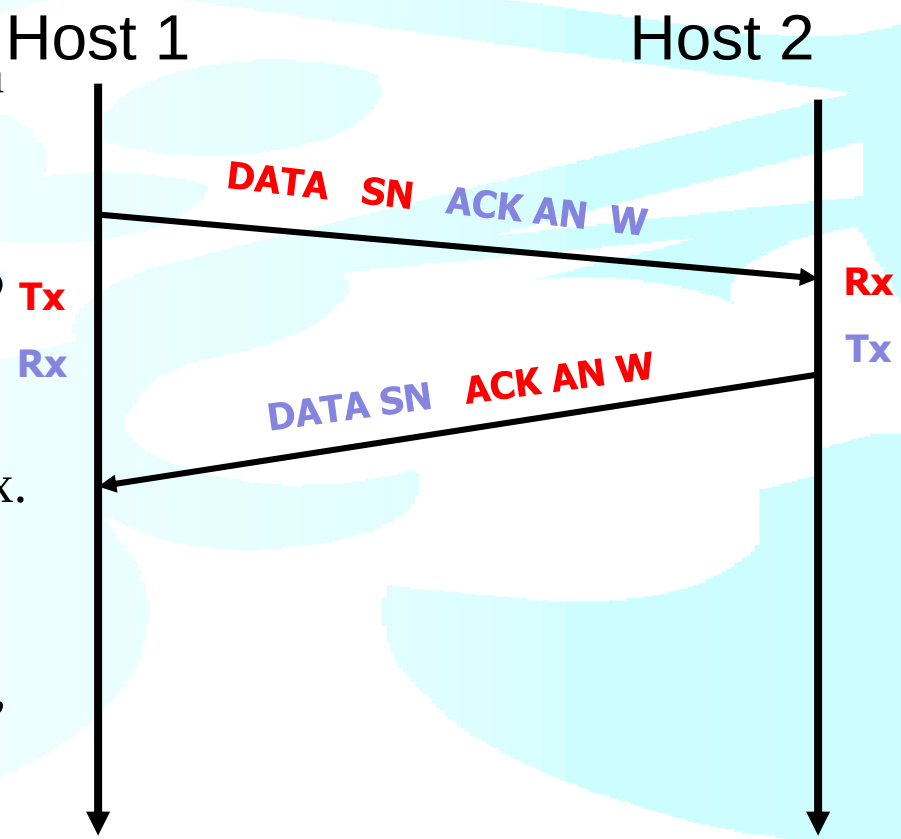
- **PSH** Push Flag/Bit
- **RST** Reset Flag/Bit

- **URG** Urgent Data Flag/Bit
- Urgent Pointer (Out of Band Data)

- **MSS Maximum Segment Size Option Field**
  (MSS = MTU - TCPHdrLen – IPHdrLen)

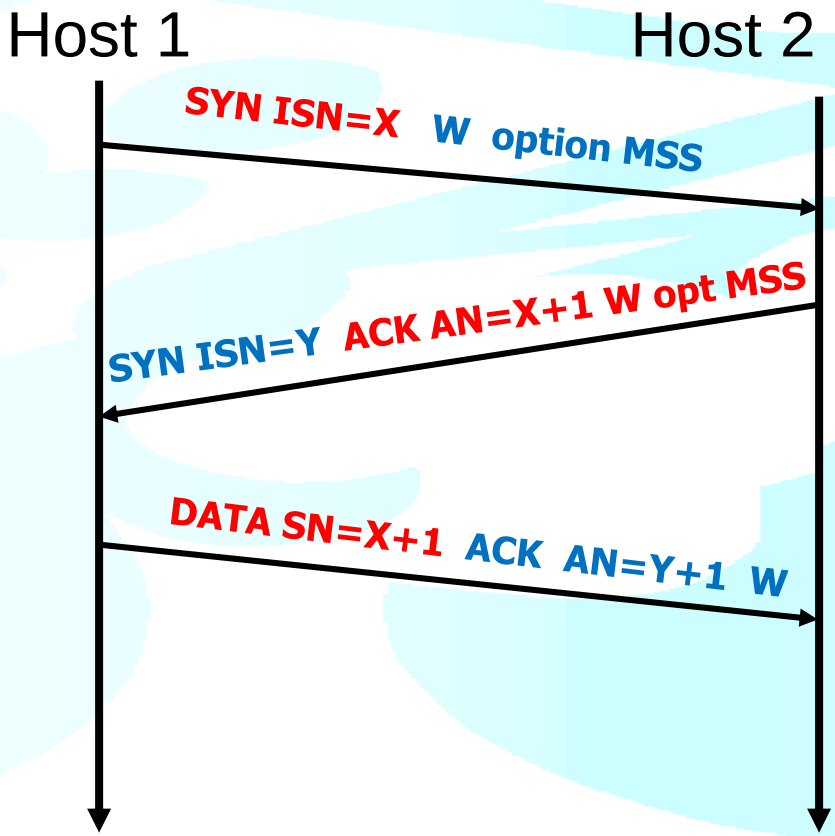
TCP/IP
Full Duplex

- TCP es un servicio de transmision de datos en modo **Full duplex**
- Hay **2 flujos de datos**, 1 en cada sentido. El "control" de cada flujo es **independiente** del otro.
- Cada Entidad TCP tiene un **Transmisor** Tx y un **Receptor** Rx. Cada flujo de datos va del Tx al Rx.
- Junto con los Datos en un sentido, se envían "**piggybacked**" parámetros del flujo contrario.



TCP
Establecimiento de la Conexión

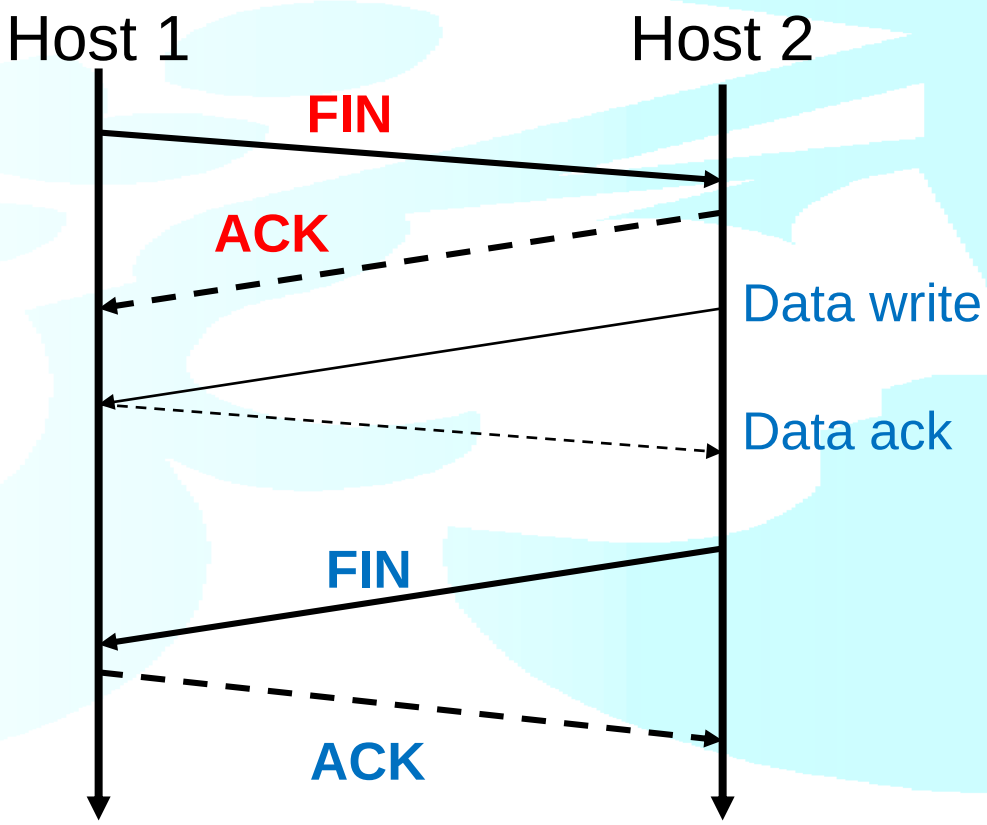
- El establecimiento de la conexión lógica permite **sincronizar las variables de estado** de las entidades protocolares TCP (autómatas finitos)
- **3 way handshake**
- Se define el **ISN** número de secuencia inicial
- Se envía el **MSS**.
- Solo en el 1er segmento el ACK bit no esta seteado.



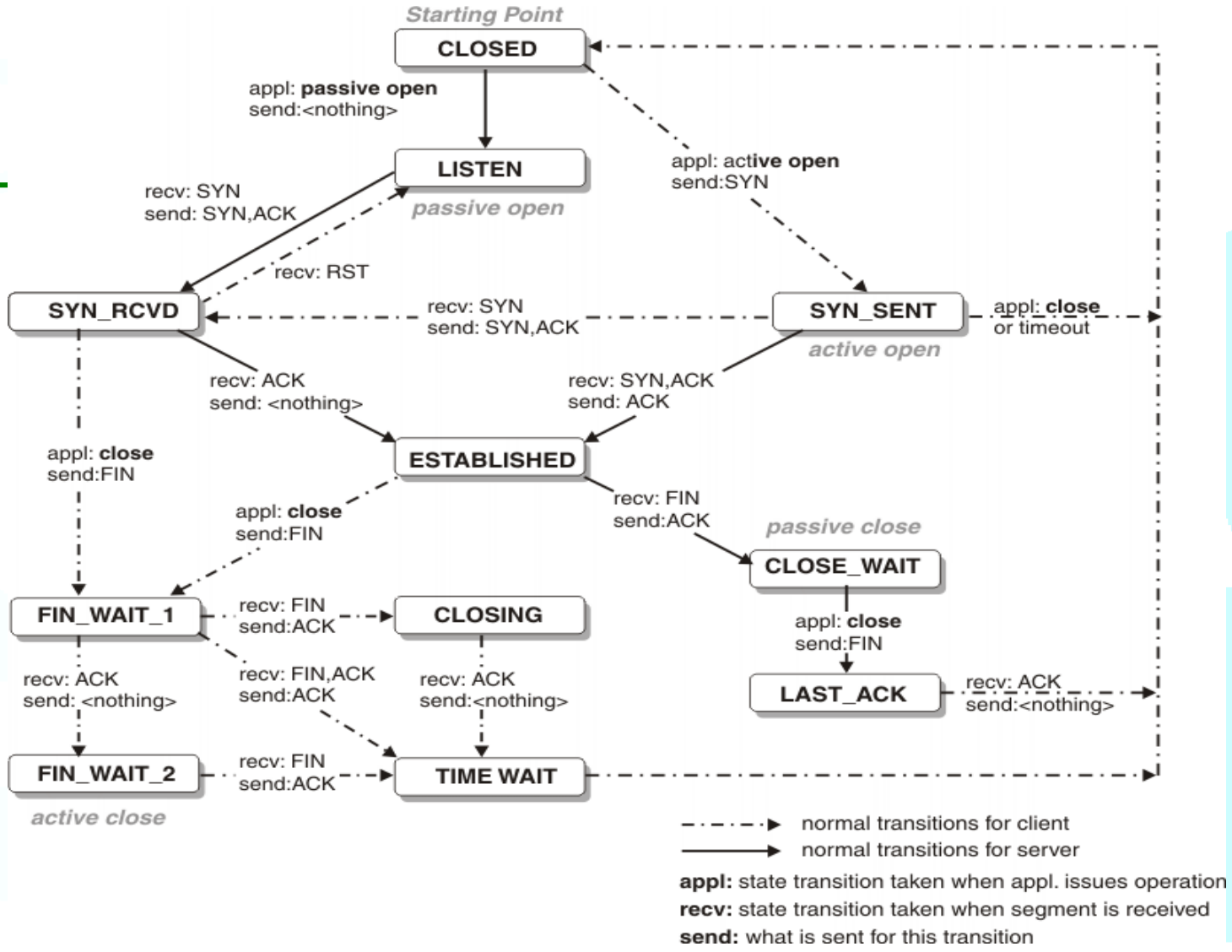
TCP
Finalización de la Conexión

Se intercambian 4 segmentos

Ambos extremos definen la finalización



TCP
States


Starting Point
CLOSED
appl: passive open
send:<nothing>
LISTEN
passive open
recv: SYN
send: SYN,ACK
recv: RST
appl: active open
send:SYN
SYN_RCVD
recv: SYN
send: SYN,ACK
SYN_SENT
appl: close
or timeout
active open
recv: ACK
send: <nothing>
recv: SYN,ACK
send: ACK
ESTABLISHED
appl: close
send:FIN
appl: close
send:FIN
recv: FIN
send:ACK
passive close
CLOSE_WAIT
FIN_WAIT_1
recv: FIN
send:ACK
CLOSING
appl: close
send:FIN
recv: ACK
send: <nothing>
recv: FIN,ACK
send:ACK
recv: ACK
send:<nothing>
LAST_ACK
recv: ACK
send:<nothing>
FIN_WAIT_2
recv: FIN
send:ACK
TIME WAIT
active close
normal transitions for client
normal transitions for server
appl: state transition taken when appl. issues operation
recv: state transition taken when segment is received
send: what is sent for this transition

TCP
Chequeo de integridad

Los Segmentos TCP, en el transito por la red, pueden
- ser **descartados** y no llegar a destino
- llegar a destino pero **alterados**
- llegar a destino pero **duplicados**
- llegar a destino pero **fuera de secuencia**

CheckSum
- sirve para el chequeo de la **integridad** del header y de los datos
- Se calcula agregando un pseudo-header que contiene la source y destination address, así como el protocol number y el TCP length, afín de poder a su vez verificar sin lugar a dudas que el segmento llego a su correcto destinatario.

TCP
Control de secuencia

Cada Byte del Data Stream
es identificado con un numero de secuencia de **32 bits**

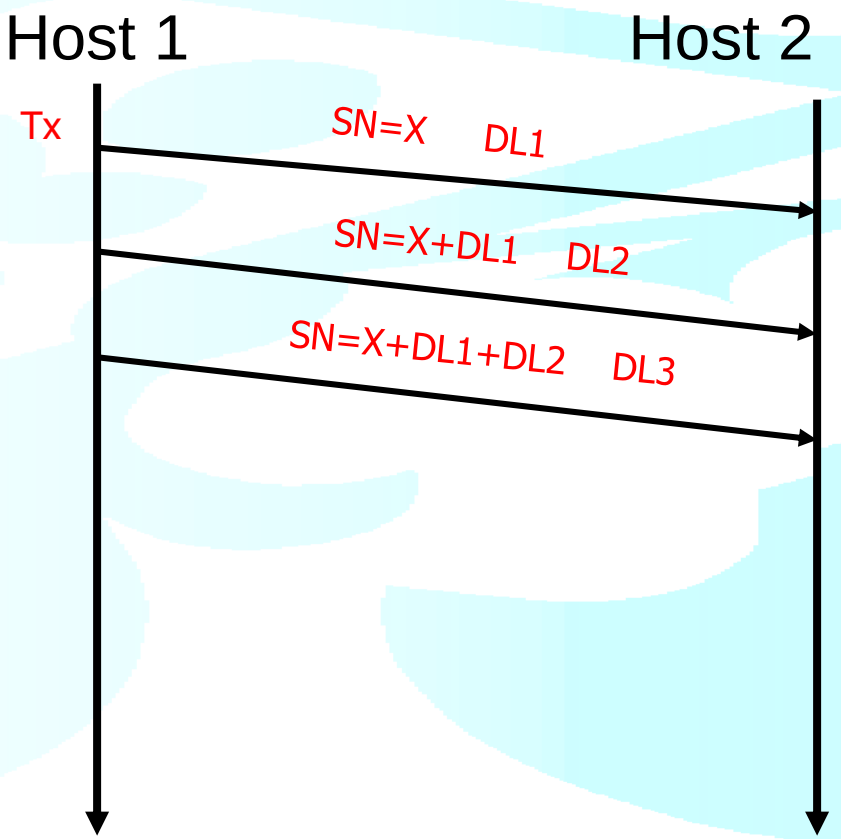
El **SN** **Sequence Number** de un Segmento TCP
indica el numero de secuencia del primer byte contenido

En función del **SN** de cada Segmento TCP recibido
el receptor guarda los datos en una posición del buffer de recepción
reconstruyendo así la secuencia correcta del byte stream
y también puede detectar y eliminar la duplicación de datos

TCP
Control de secuencia

Del lado del Transmisor

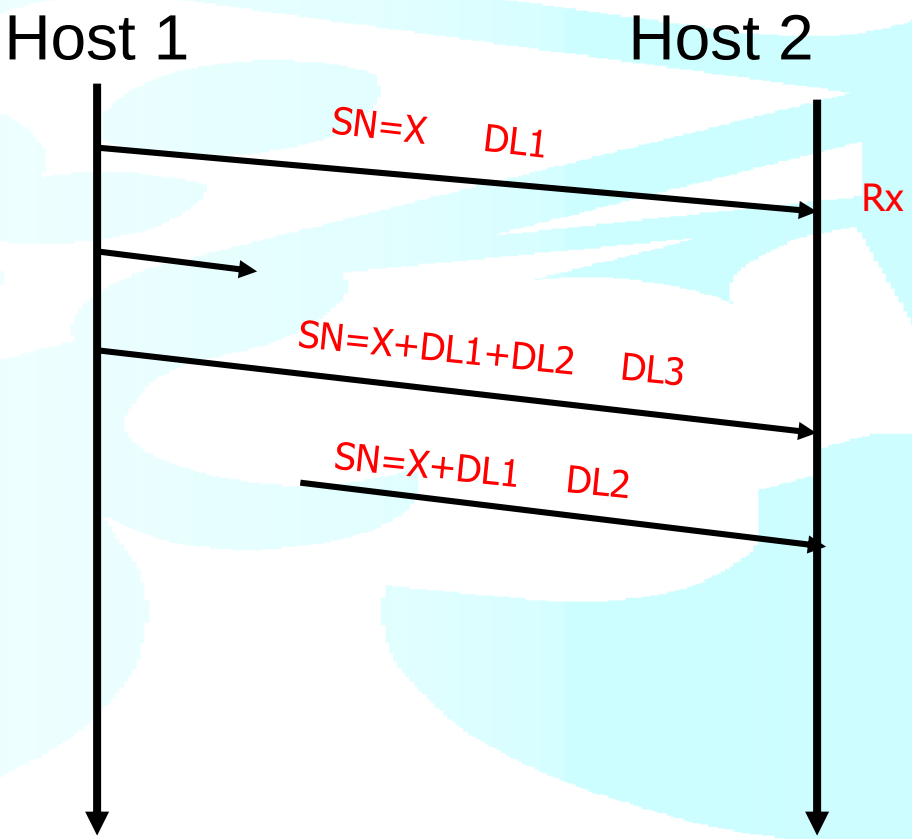
- Los SN identifican el nro de secuencia del 1er byte de datos del segmento.
- Los SN no son consecutivos.
- El SN del segmento siguiente al X, es igual a X mas el Data Length



TCP
Control de secuencia

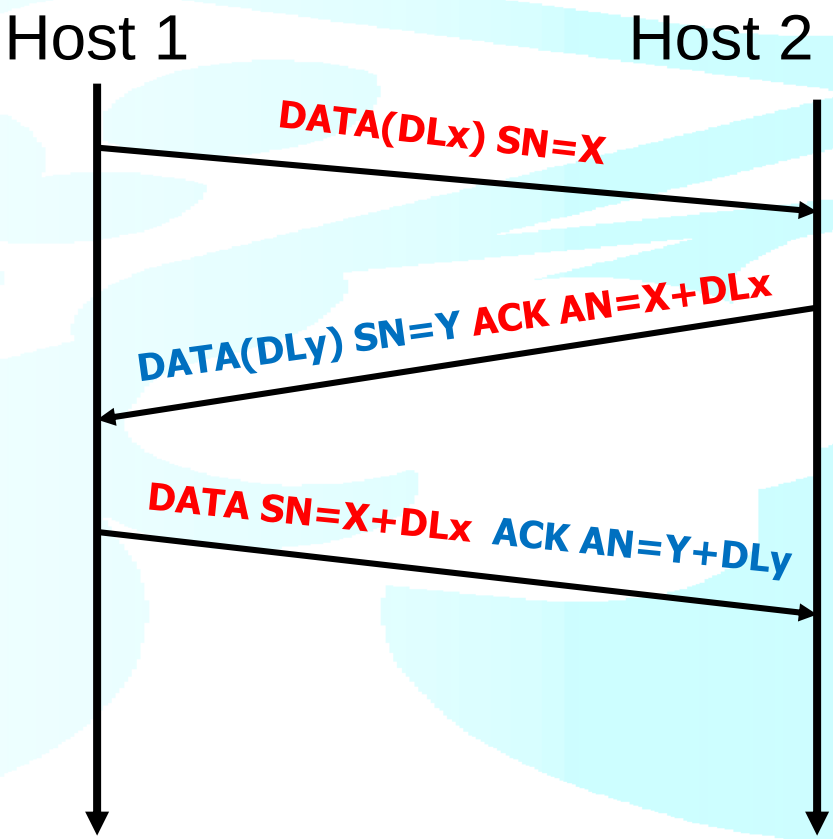
Del lado del Receptor

- Recibe el segmento X y si el checksum es correcto, guarda los DL bytes de datos en el buffer de recepción a partir de la posición X.
- Aunque los segmentos lleguen fuera de secuencia (ej: por ir por diferentes caminos), los datos quedan guardados en orden.



TCP
Control de Errores – ARQ - Acknowledgements

- Cada segmento enviado tiene un **SN** y transporta DL bytes
- Al recibir sin errores los "DL" bytes con SN="X", el receptor confirma los datos recibidos indicando el **proximo numero de secuencia que espera recibir** contestando con un **AN=X+DL**



TCP
Adaptación dinámica

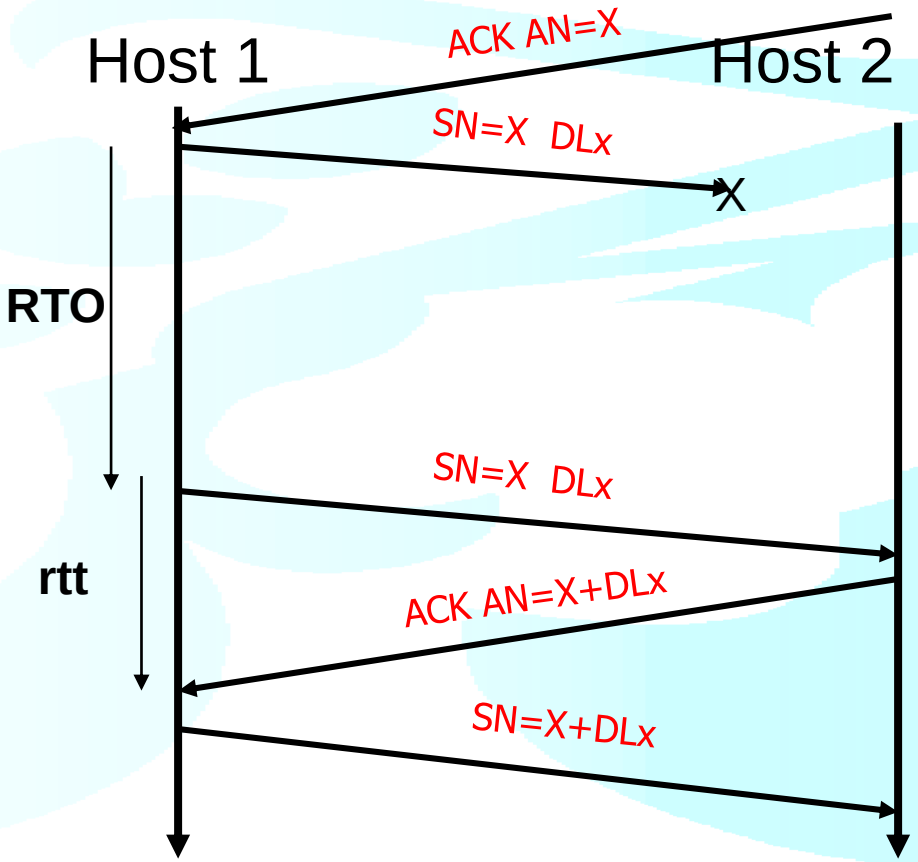
Tiempos de retransmisión RTO ajustables automáticamente

Ventana deslizante de tamaño variable

- Control de Flujo – Advertised Window
- Control de Congestion – Congestion Window

TCP
Control de errores – ARQ - Retransmisiones

- En el caso de **perdida** de segmento, después de un tiempo **RTO** el transmisor **retransmite** a partir del ultimo AN indicado por el receptor
- **RTO**
  Retransmission TimeOut



TCP
Control de errores – ARQ – RTO

Problema:

Qué **RTO** Retransmission TimeOut adoptar?

El **RTT** Round Trip Time varía sustancialmente

Si RTO Demasiado largo => subutilización del canal
Si RTO Demasiado corto => retransmisiones inútiles.

Solución:

RTO dinámico/adaptable, estimado en base al **RTT**.

TCP

Control de errores – ARQ - RTO

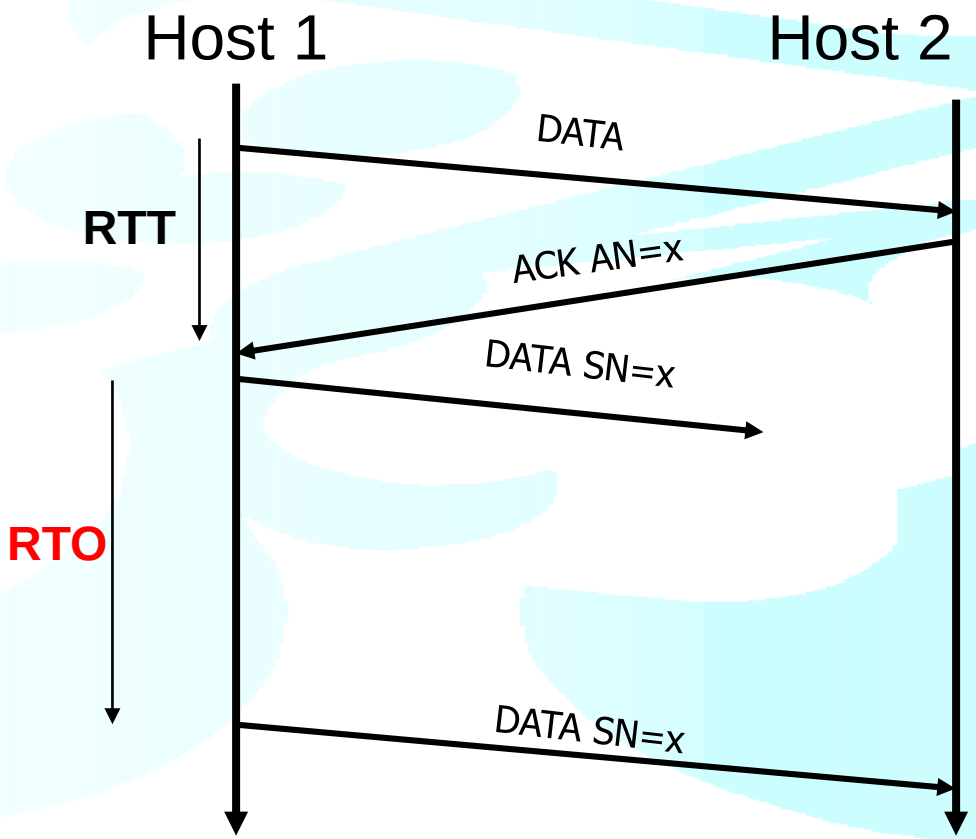
Estimación de los RTO
en base al RTT

$$R_{(n)} \leftarrow \alpha.R_{(n-1)} + (1-\alpha).RTT$$

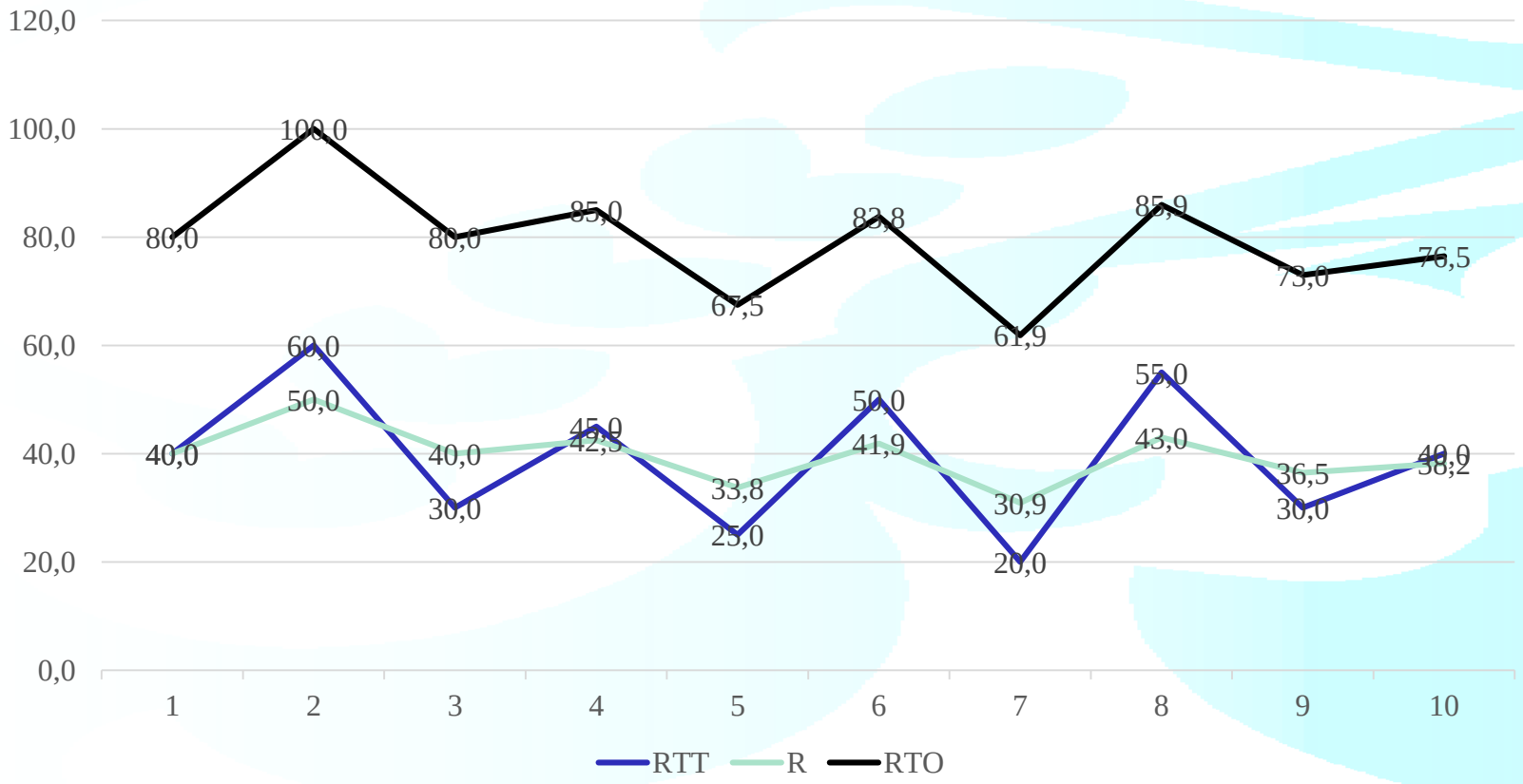
$$RTO = R.\beta$$

α: **smoothing factor**
(entre 0 y 1, tip 0,5)

β: **delay variance factor**
(mayor a 1, tip 2)

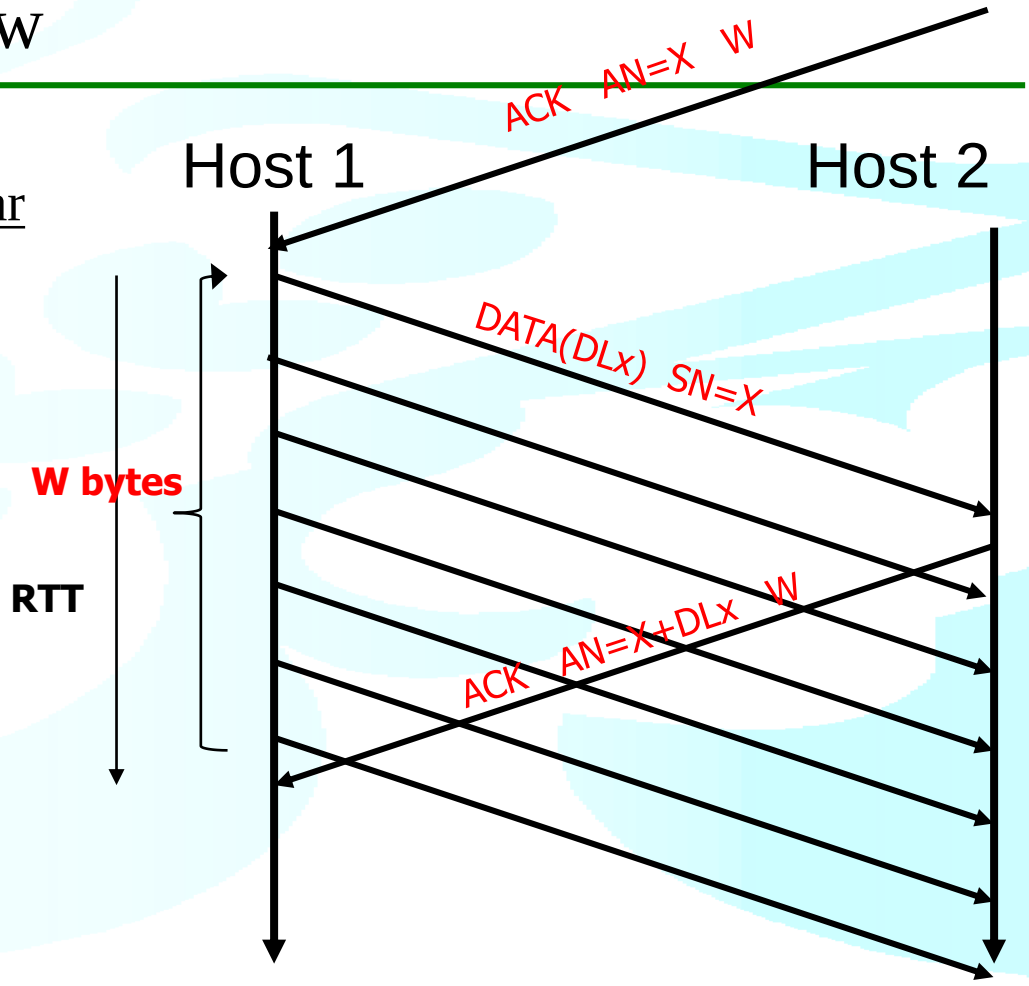


TCP
Control de errores – ARQ - RTO

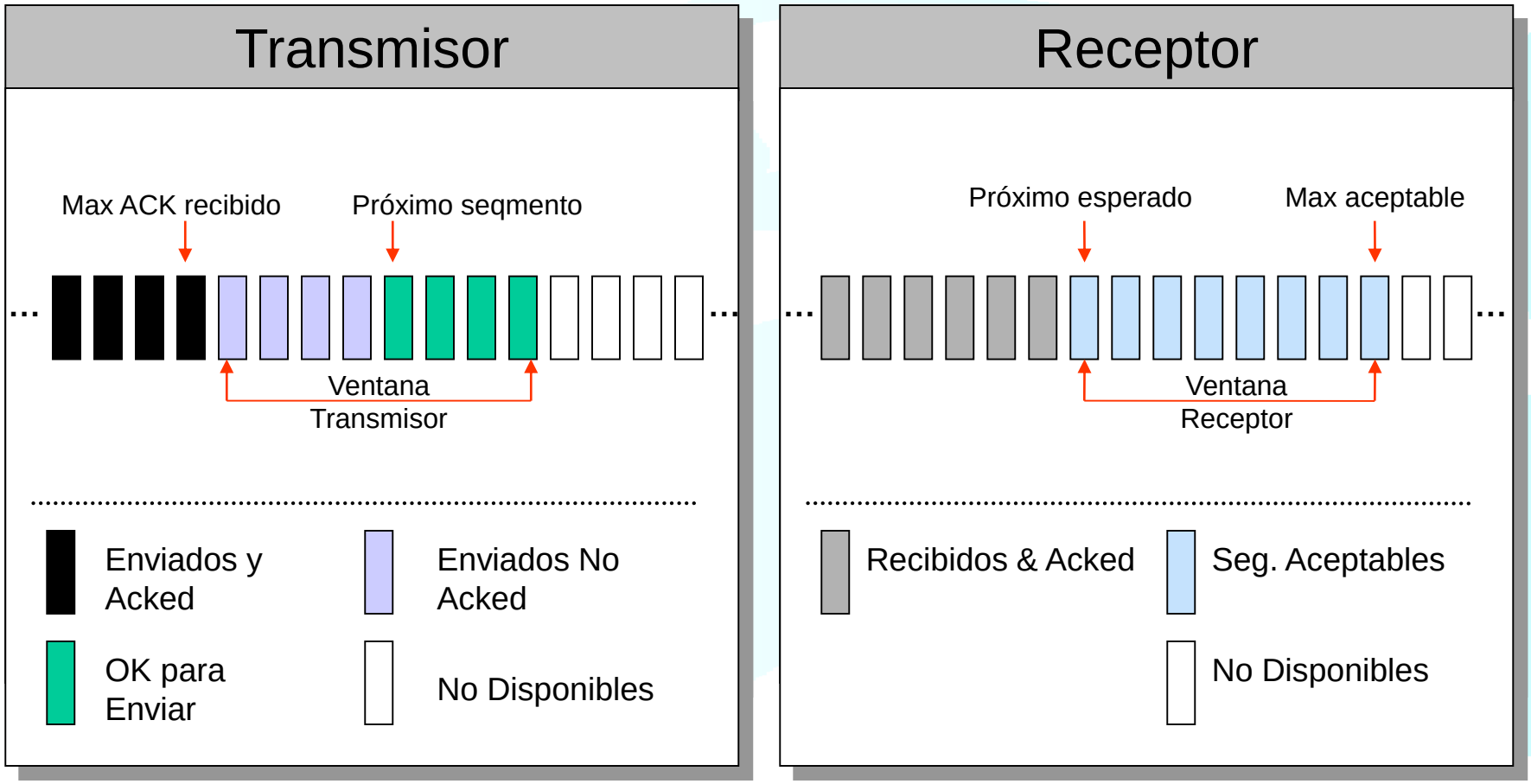


TCP
ARQ Sliding Window

- El transmisor puede enviar hasta **W** bytes a partir del ultimo **AN**
- **Throughput = W / RTT**



TCP
ARQ Sliding Window



TCP
ARQ Sliding Window - Control de flujo

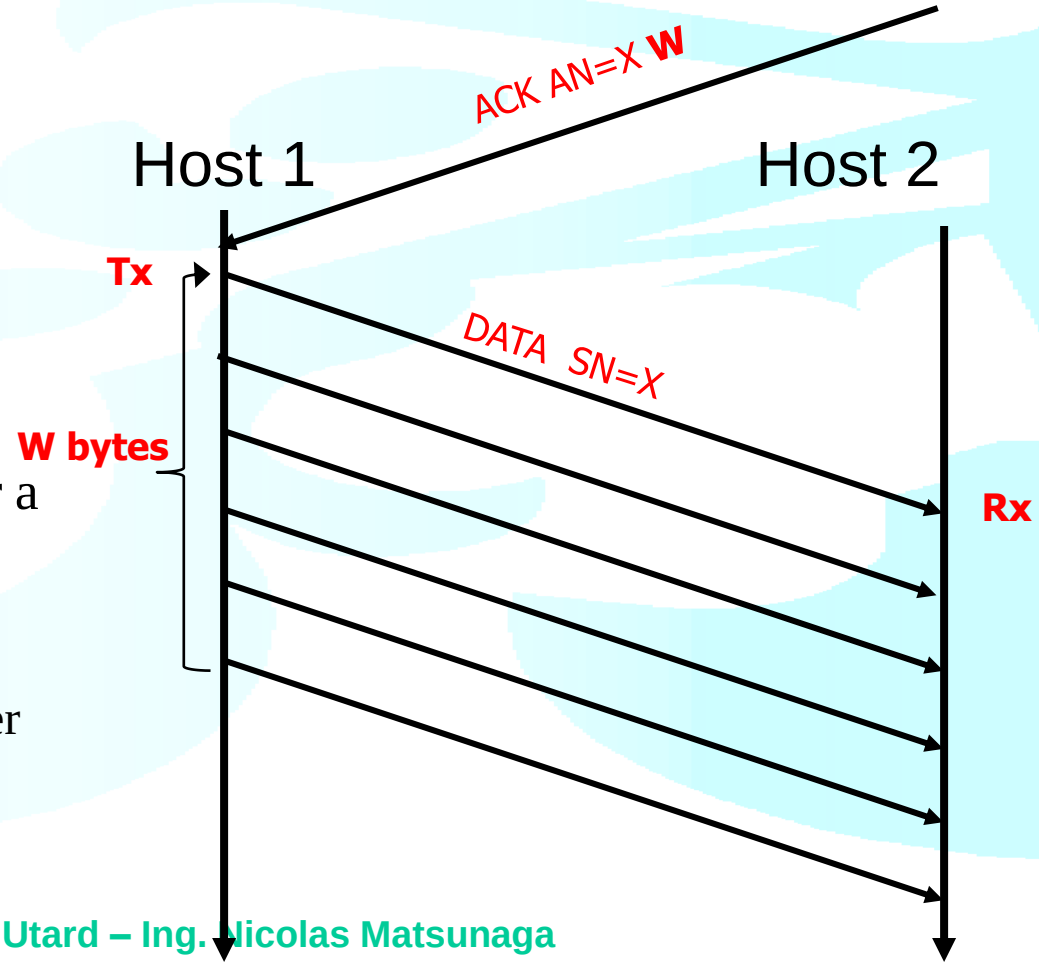
Problema:

El **buffer de recepción** puede desbordar.

Solución:

El Receptor Rx envia la **W** "**advertised window**" **(awnd)**, indicando la cantidad de bytes que el Transmisor Tx puede enviar a partir del AN Acknowledge Number.

Cuando W=0 significa que el buffer del Rx esta lleno y el Tx no puede seguir transmitiendo.



TCP
ARQ Sliding W - Control de Congestión

Problema: **CONGESTION**

Las **COLAS** en la red de conmutación de paquetes pueden crecer, introduciendo delay (Congestión suave), e incluso pueden desbordar, descartando paquetes (Congestión severa).

Solución: **Control de Congestión**

El Transmisor Tx calcula una **cwnd** "**congestion window**", en función de la **detección implícita** del estado de congestión de la red.
Cuando el Tx detecta la perdida de un segmento, asume que fue descartado por congestión severa, por lo cual reduce drásticamente el throughput reduciendo la cwnd. A medida que va recibiendo ACK, vuelve a incrementar la cwnd.

El Tx implementa el ARQ Sliding Window eligiendo entre **awnd** (advertised W) y **cwnd** (congestion W) **la menor** .

TCP
Algoritmos de Control de Congestión

- **TAHOE** (Jacobson 1988)
  Slow Start
  Congestion Avoidance
  Fast Retransmit
- - **RENO** (Jacobson 1990)
  Fast Recovery
- **New RENO** (Jacobson 1990)
  Fast Recovery mejoras
- **VEGAS** (Brakmo & Peterson 1994)
  Evitar congestion a partir del RTT
- **SACK, D-SACK** (M. Mathis 1996)
  Deteccion de retransmisiones innecesarias
- **BIC-TCP** (L. Xu 2004)
  Binary Increase Congestion Control
- **CUBIC** (I. Rhee 2007)
  Cubic function instead of a linear window increase

TCP
ARQ Sliding W - Control de Congestión (Tahoe)

Problema: **Congestion inicial**

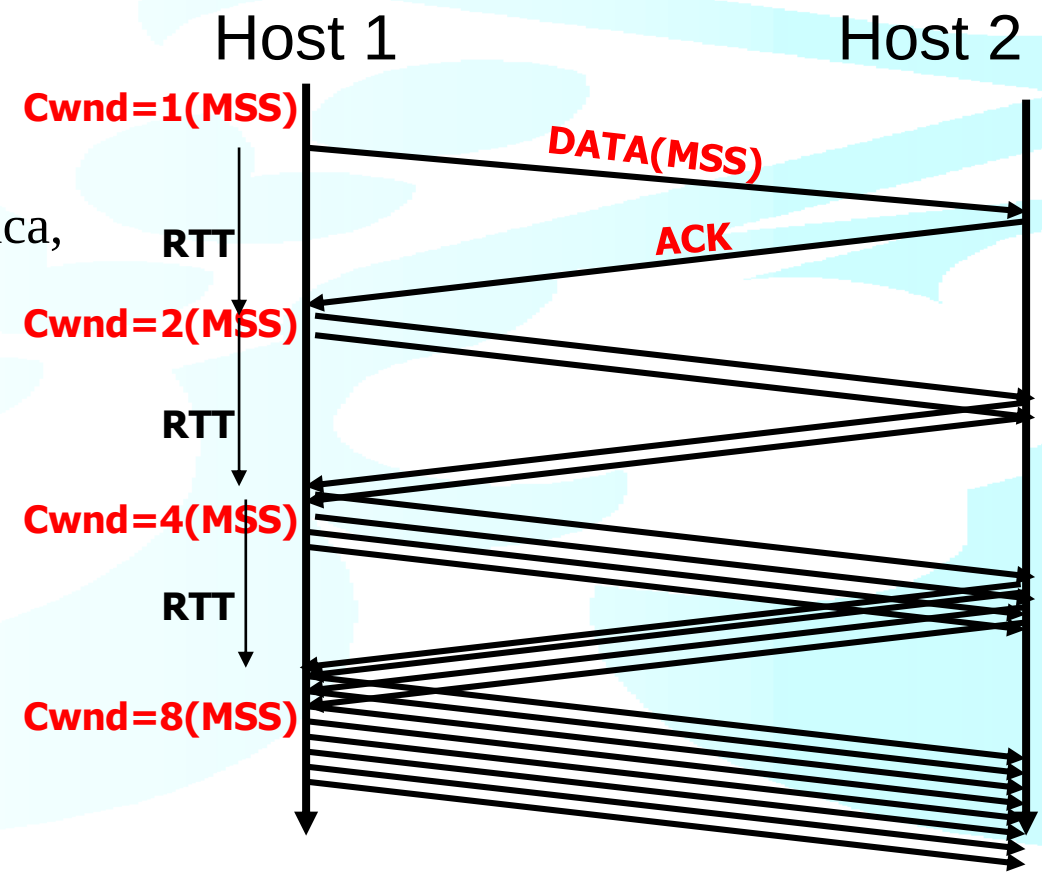
Si la entidad TCP, una vez establecida la conexión, envía los datos con un throughput dado por la awnd y el rtt, lo mas probable es que genere una congestion en la cola del primer enlace de menor capacidad que **awnd/rtt**.

Solución: **Control de Congestión** con **Slow Start**

Al inicio de la conexión TCP,
el Transmisor Tx calcula la **cwnd** = 1 MSS.
Cada vez que recibe un ACK aumenta la **cwnd** = cwnd + 1 MSS

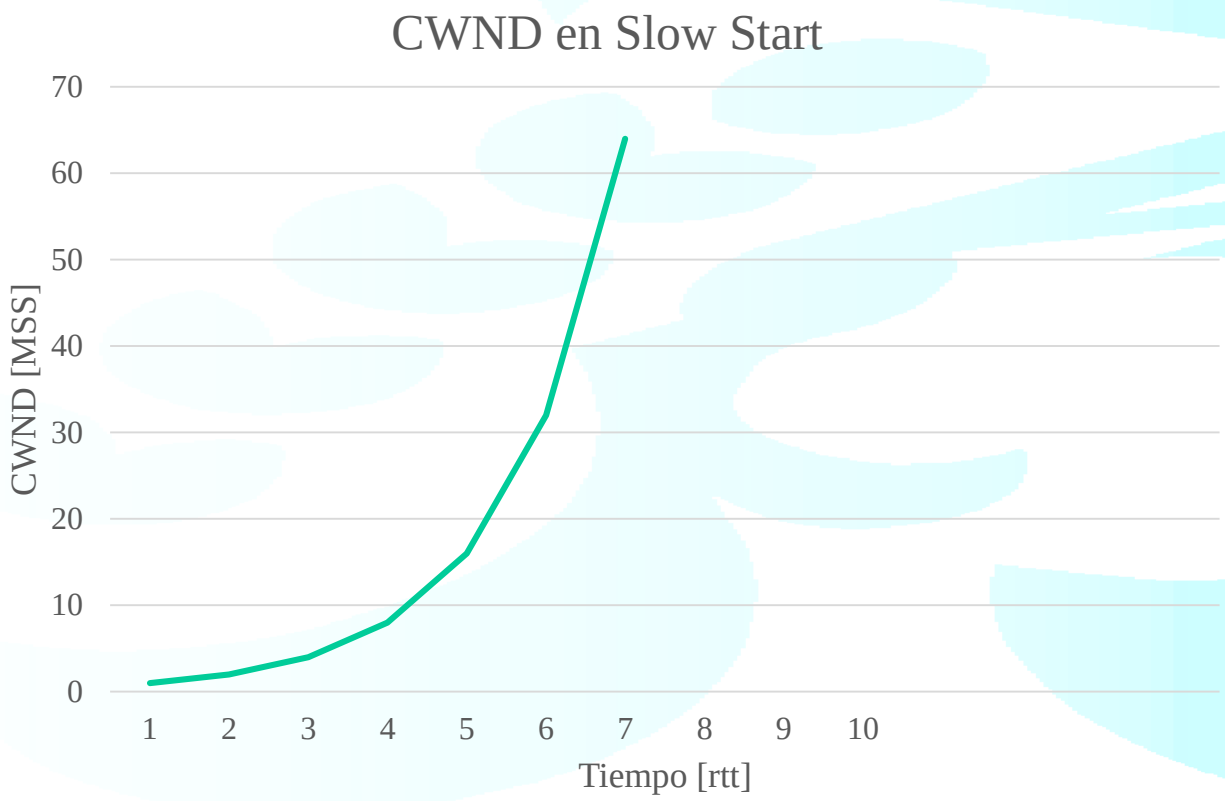
TCP
ARQ Sliding W - Control de Congestión (Tahoe)

Slow Start
A cada RTT,
la cantidad de ACKs se duplica,
por lo cual la cwnd
aumenta exponencialmente



TCP

ARQ Sliding W - Control de Congestión (Tahoe)



TCP
ARQ Sliding W - Control de Congestión (Tahoe)

Problema: Como lograr el mayor throughput evitando la congestion?

Si la entidad TCP, sigue aumentando el troughput, aumentando la cwnd con cada ACK, en algún momento se puede producir una perdida de segmento por congestion severa.

Solución: **Control de Congestión** con **Congestion Avoidance**

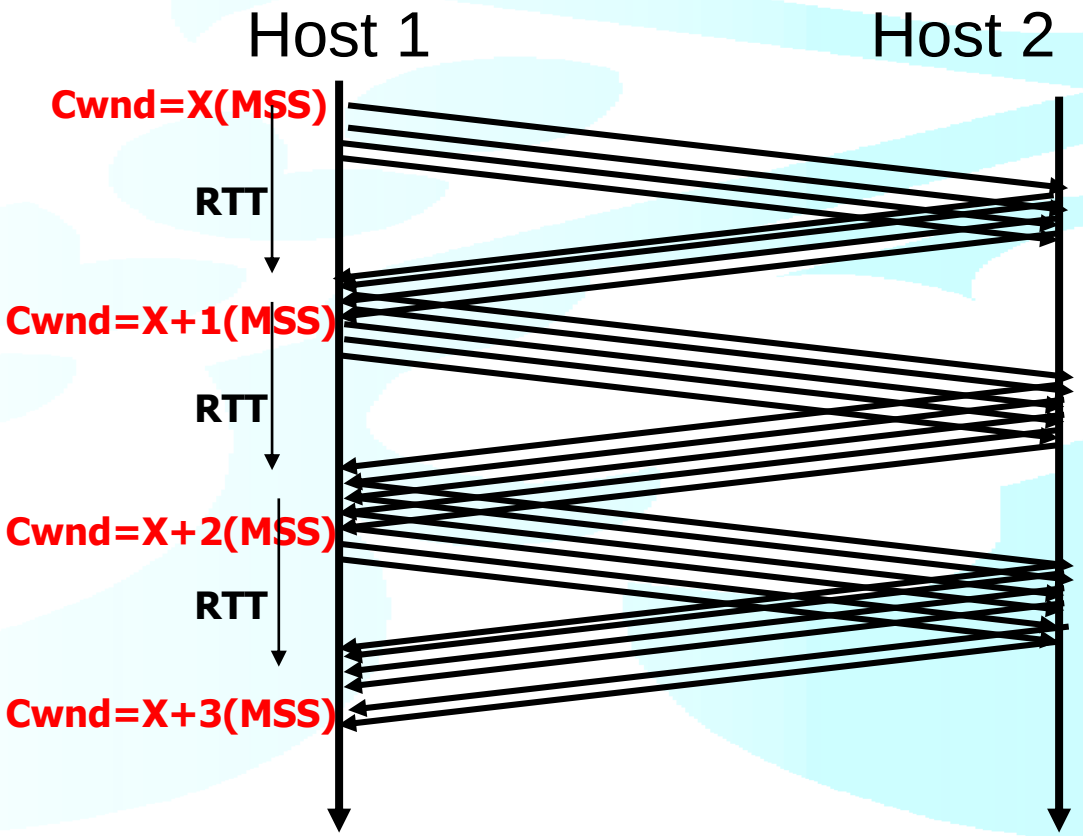
Cuando el Transmisor Tx detecta una perdida de segmento TCP, asume que se debió a congestion severa, y reduce drásticamente la **cwnd** = 1 MSS, por lo cual reinicia el proceso de Slow Start.

El Tx setea la variable **ssthresh** (slow start threshold) a la mitad del valor que tenia la cwnd cuando se detecto la perdida del segmento TCP.

Cuando la **cwnd >= ssthresh**, el Tx pasara a "congestion avoidance" cada vez que recibe un ACK aumenta la **cwnd** = cwnd + 1/cwnd

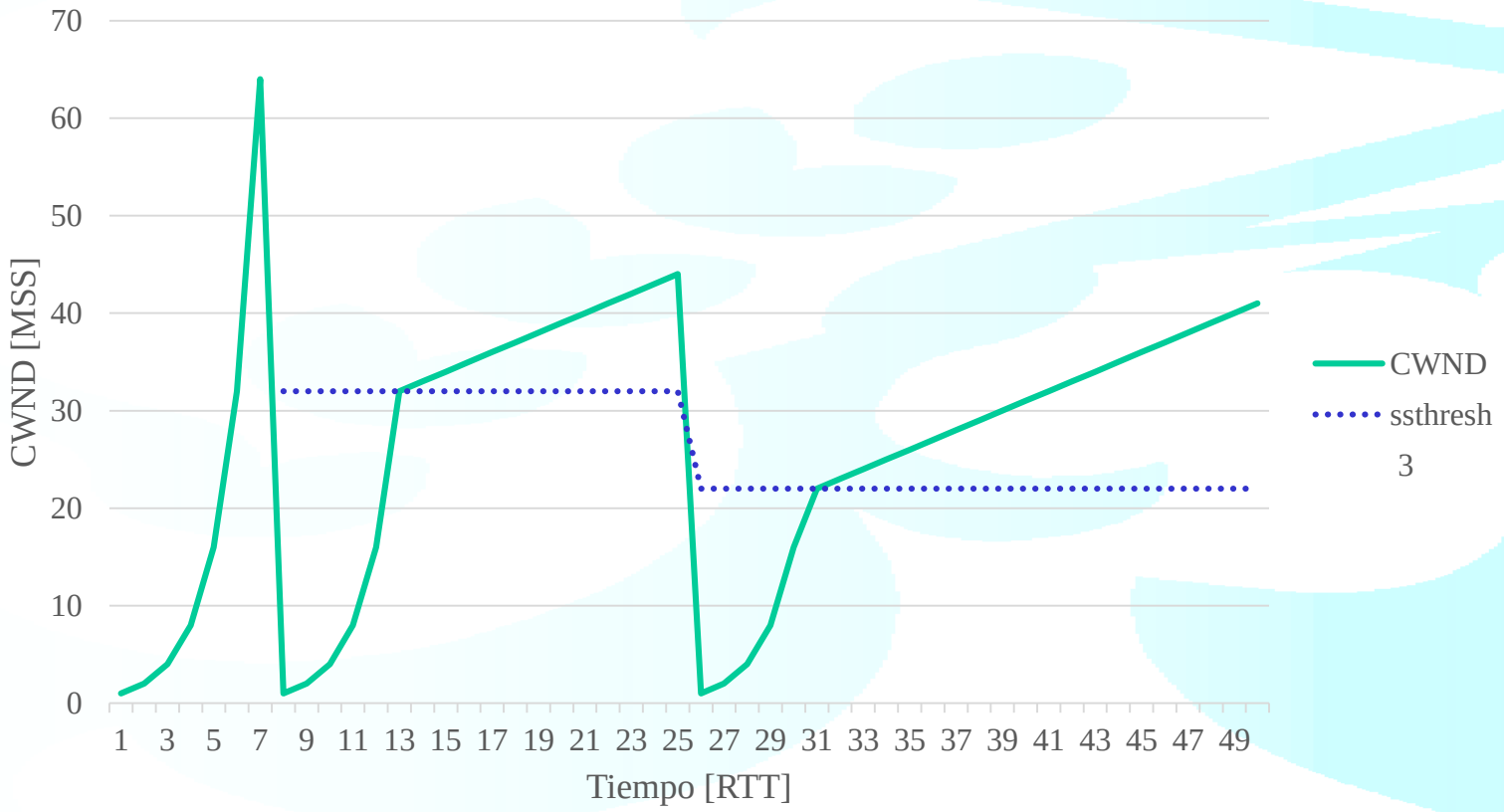
TCP
ARQ Sliding W - Control de Congestión (Tahoe)

Congestion Avoidance
A cada RTT,
la cantidad de ACKs
aumenta en 1,
por lo cual la cwnd
aumenta linealmente



TCP

ARQ Sliding W - Control de Congestión (Tahoe)



TCP

Control de errores - Fast Retransmit

Problema: Como lograr retransmitir mas rápido?

Si la perdida de un segmento se debe a errores (ej: por ruido impulsivo),no por congestion, seguramente varios segmentos posteriores al descartado/perdido logran llegar al Rx (**perdida menor al 1%**) el cual comienza a enviar ACKs indicando en todos los ACKs el mismo AN (se dice que son **ACKs duplicados**)

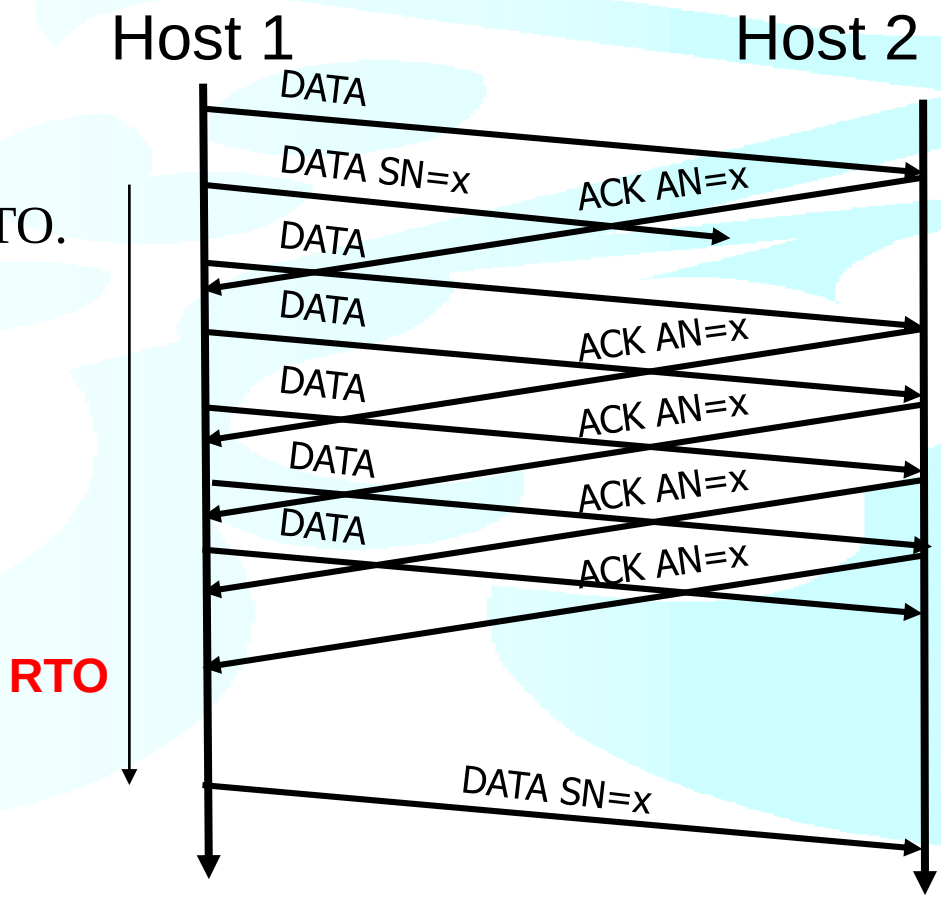
Solución: **Control de Errores** con **Fast Retransmit**

Cuando el Transmisor Tx recibe mas de **3 ACKs duplicados**, asume que el segmento indicado en el AN fue descartado/perdido y **retransmite** los segmentos a partir del AN, sin esperar que se venza el RTO.

TCP

Control de errores - Fast Retransmit

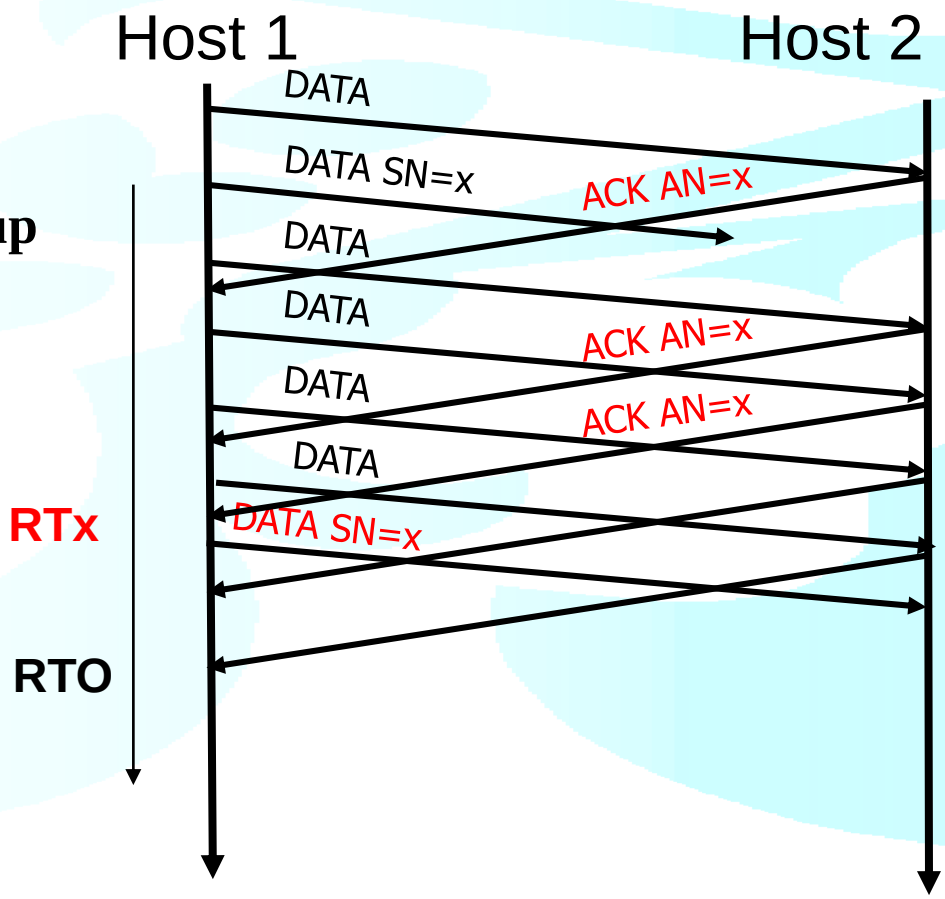
Sin Fast Retransmit,
ante la perdida de 1 segmento,
hay que esperar a que expire el RTO.



TCP

Control de errores - Fast Retransmit

Con **Fast Retransmit**,
ante la perdida de 1 segmento,
en cuanto se detectan **3 ACKs Dup**
se **retransmite** el segmento
sin esperar a que expire el RTO.



TCP

Control de Congestion (Reno) - Fast Recovery

Problema: Como evitar que el canal se vacie luego de un Fast Retransmit?

Cuando el Transmisor Tx detecta la perdida de 1 segmento baja la cwnd a 1, o sea entra en Slow Start, a pesar que no hay congestion, y tarda varios RTT en volver a ocupar el canal.

Solución: **Control de Congestion** con **Fast Rcovery**

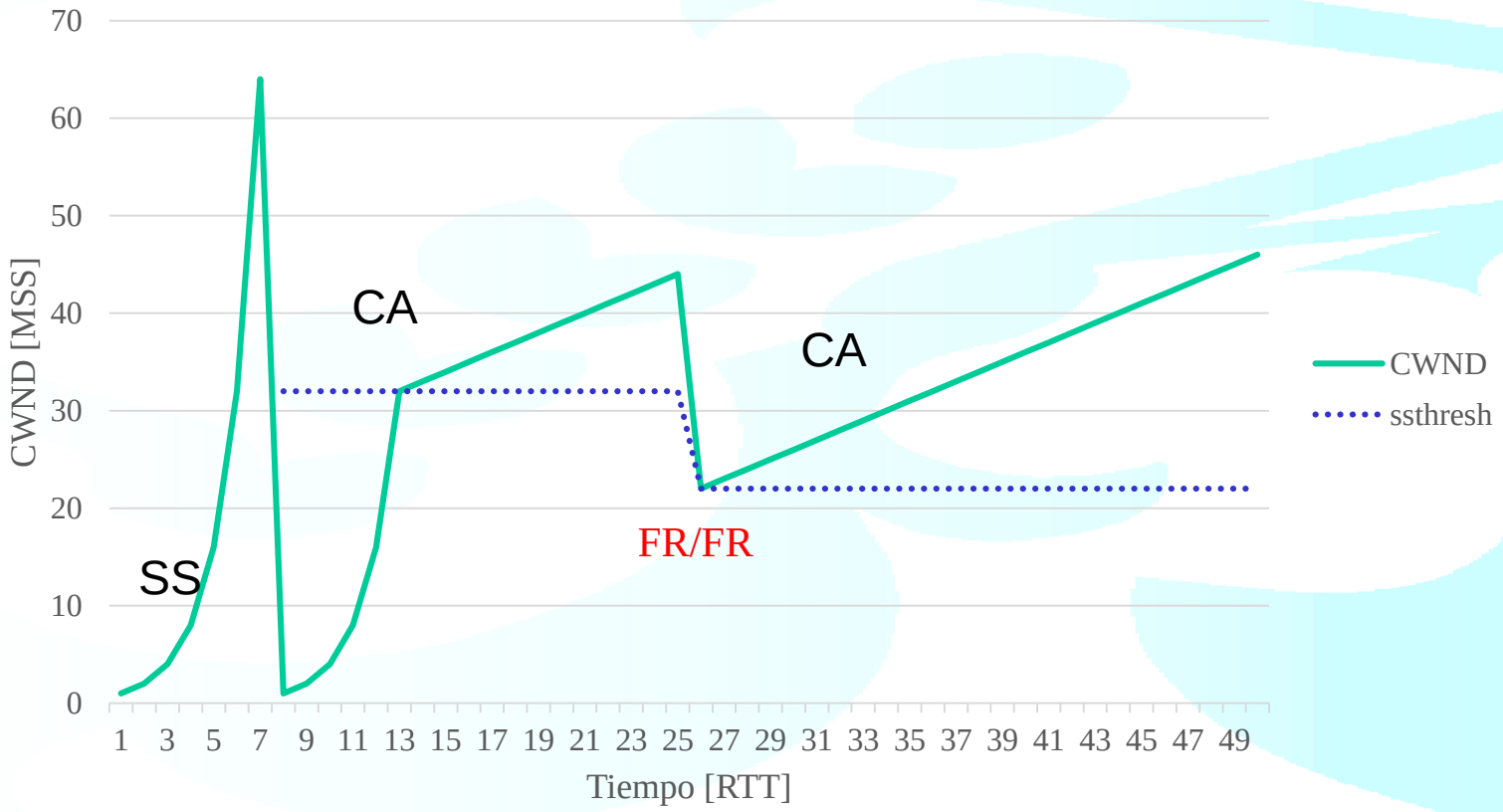
Cuando el Transmisor Tx recibe mas de **3 ACKs duplicados**, detecta la perdida de1 segmento, hace Fast Retransmit, pero baja la cwnd al valor del ssthresh (o sea, a la mitad de la cwnd que tenia al momento de la detección de los 3 ACKs Dup)

ssthresh = cwnd / 2

cwnd = ssthresh

TCP

ARQ Sliding W - Control de Congestión (Reno)



TCP
Funcionalidades

- **Multiplexado** de trafico IPC con **Port Numbers**
- **Transmisión Byte Oriented Full duplex**
- **Segmentación**
- **Control de secuencia**
- **ARQ Sliding Window**
- **Control de errores**
- **Control de Flujo explícito**
- **Control de Congestión implícito**