

TCP (Transport Control Protocol) Flow Control

2024

- 1 TCP
 - TCP Flow Control/Control de Flujo

Servicios de TCP

Control de Errores:

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

Servicios de TCP (Cont.)

Control de Flujo (Flow-Control):

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

Control de Errores y de Flujo

- Para realizar control de errores y control flujo se utilizan técnicas de ARQ (Automatic Repeat reQuest), **Transferencia de Datos Fiables**.
- ARQ **solo** no hace control de flujo, requiere de otros mecanismos como RNR (Receive-Not-Ready), o Dynamic Window (Ventana Dinámica). TCP usa Ventana Dinámica.
- La capacidad de envío será $MIN(\text{Congestion}, \text{Flujo}, \text{Errores})$.

Control de Errores TCP (Repaso)

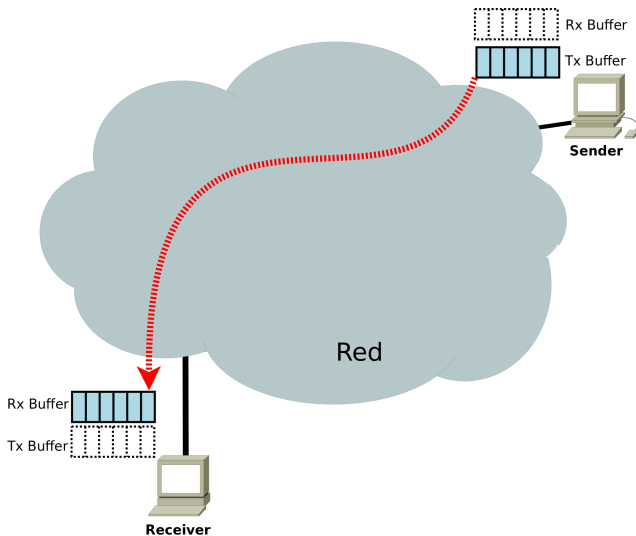
- Segmentos ACKed no indica leído por aplicación, sí recibido por TCP (RFC-793) (ubicado en el **Rx Buffer** del receptor).
- Si el receptor detecta error en el segmento simplemente descarta y espera que expire *RTO* en el emisor (podría envía un “NAK”, re-enviar ACK para el último recibido en orden,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

Control de Errores TCP (Repaso)

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

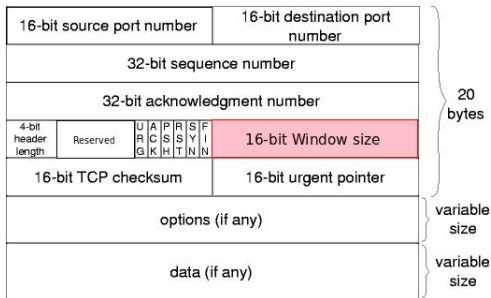
Control de Flujo TCP

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



Control de Flujo TCP

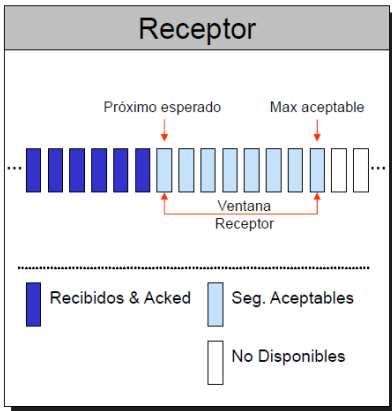
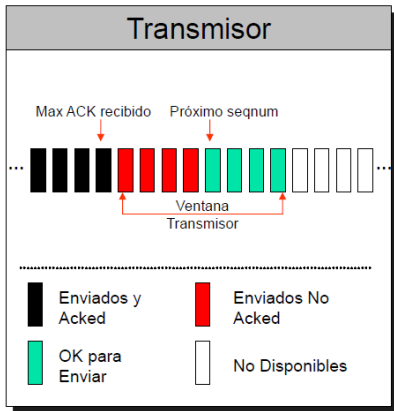
- El receptor (cada extremo puede recibir, es FDX) indica el espacio del buffer de recepción, **Rx Buffer**, en el campo del segmento: **Window** (de datos o ACK) **Advertised Window** (Ventana Anunciada).



Control de Flujo TCP

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

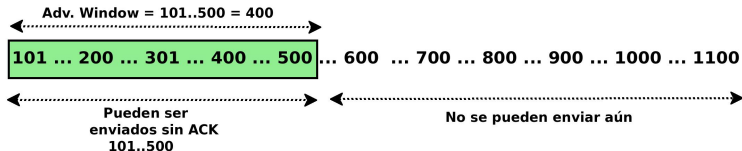
Ventana Deslizante TCP



Ventana Deslizante TCP (Inicial)

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

Stream de datos a enviar 101..1100

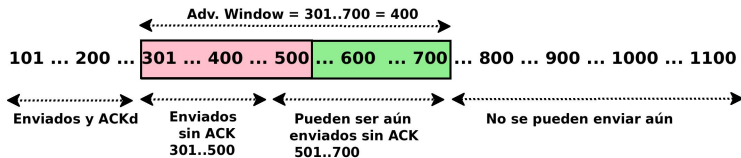


- 2 Luego, la aplicación que envía, escribe, `write()`, y se envían 400 bytes usando toda la ventana (los 400 bytes se pudieron enviar en múltiples segmentos).
- 3 Se recibe un segmento con $ACK = 301$ y $WIN = 400$.
- 4 Se desliza ventana.

Ventana Deslizante TCP I

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

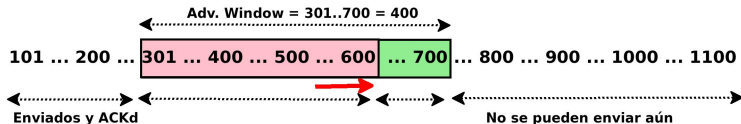
Stream de datos a enviar 101..1100



Ventana Deslizante TCP II

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

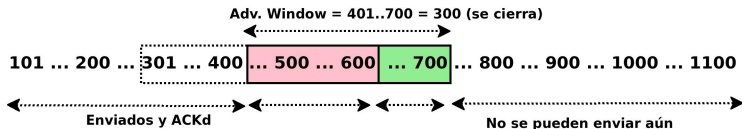
Stream de datos a enviar 101..1100



Ventana Deslizante TCP III

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

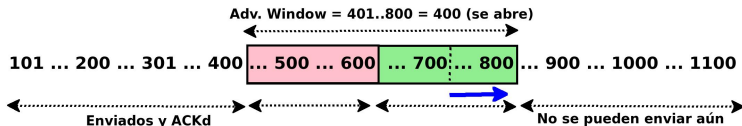
Stream de datos a enviar 101..1100



Ventana Deslizante TCP IV

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

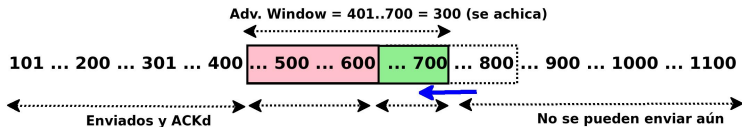
Stream de datos a enviar 101..1100



Ventana Deslizante TCP V

- 11 Se recibe un segmento $ACK = 401$, $WIN = 300$.
- 12 Ventana se achica.
 - 401..600 en Tx Buffer, “en vuelo”.
 - 601..700 en Tx Buffer, aún no han sido enviados.
 - Raro que suceda, TCP achica el Rx Buffer <https://www.rfc-editor.org/rfc/rfc7323#section-2.4>.

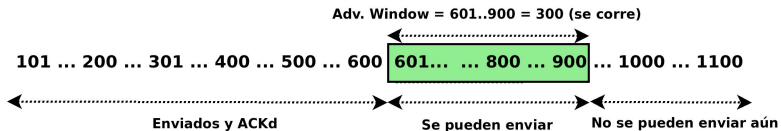
Stream de datos a enviar 101..1100



Ventana Deslizante TCP VI

- 13 Se leyeron los datos del Rx Buffer y se confirma.
- 14 Se recibe un segmento $ACK = 601$, $WIN = 300$.
- 15 Ventana se corre.
 - Rx Buffer vacío.
 - 601..900 en Tx Buffer, aún no han sido enviados, se pueden enviar.

Stream de datos a enviar 101..1100



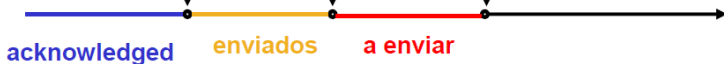
Ventana Deslizante TCP

Segmento enviado

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

Segmento recibido

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



Control de Flujo TCP

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

TCP Bulk y TCP Interactivo

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

TCP Escalado de Ventana

- El campo Ventana del segmento TCP es de 16bits dando $MAX=64KB-1$.
- Para obtener mejor rendimiento ante BDP ($BW*DELAY$) grandes se requiere aumentar, para LFNs (Long Fat Networks).
- RFC-7323: Opción TCP de escalado de ventana en SYN.
- Multiplicar el valor del campo por 2,4,8,...128...16KB (shift 1,2,3...7...14).
- Valor máximo con escalado
$$W = 65535 \times 2^{14} \cong 2^{16} \times 2^{14} = 2^{30} = 1GB.$$

Opción Escalado SYN TCP

tcp.option_kind == 3

No.	Time	Source	Destination	Protocol	Length	Info
2	0.000101	10.0.3.10	10.0.1.100	TCP	74	5001 → 51373 [SYN, ACK] Seq=0 Ack=1 Win=0

```

Internet Protocol Version 4, Src: 10.0.3.10, Dst: 10.0.1.10
Transmission Control Protocol, Src Port: 5001, Dst Port: 51373, Seq: 0, Ack: 1, Len: 0
  Source Port: 5001
  Destination Port: 51373
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 2219352712
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 3436242235
  1010 .... = Header Length: 40 bytes (10)
  Flags: 0x012 (SYN, ACK)
  Window: 14480
  [Calculated window size: 14480]
  Checksum: 0x1842 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), W
    TCP Option - Maximum segment size: 1460 bytes
    TCP Option - SACK permitted
    TCP Option - Timestamps: TSval 214042, TSecr 214023
    TCP Option - No-Operation (NOP)
    TCP Option - Window scale: 4 (multiply by 16)
      Kind: Window Scale (3)
      Length: 3
      Shift count: 4
  
```

TCP Option - Window scale (tcp.options.wscale), 3 bytes

Packets: 607 · Displayed: 2 (0.3%) Profile: Classic

Valor de Ventana SYN TCP

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
596	30.981905	10.0.3.10	10.0.1.10	TCP	66	5001 → 51373 [ACK] Seq=1 Ack=518409 Win=15023

```

> Frame 596: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
> Ethernet II, Src: 00:00:00:aa:00:07 (00:00:00:aa:00:07), Dst: 00:00:00:aa:00:06 (00:00:00:aa:00:06)
> Internet Protocol Version 4, Src: 10.0.3.10, Dst: 10.0.1.10
> Transmission Control Protocol, Src Port: 5001, Dst Port: 51373, Seq: 1, Ack: 518409, Len: 0
  Source Port: 5001
  Destination Port: 51373
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 1 (relative sequence number)
  Sequence Number (raw): 2219352713
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 518409 (relative ack number)
  Acknowledgment number (raw): 3436760643
  1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)
  Window: 15023
  [Calculated window size: 240368]
  [Window size scaling factor: 16]
  Checksum: 0x183a [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [Timestamps]
  > [SEQ/ACK analysis]
  
```

The scaled window size (if scaling has been used) (tcp.window_size, 2 bytes) Packets: 607 · Displayed: 607 (100.0%) Profile: Classic

$$15023 \times 16 = 240368 \text{ (bytes)}$$

Control Flujo - wireshark

tcp.analysis.zero_window

No.	Time	Source	Destination	Protocol	Length	Info
279	8.953425	10.0.3.10	10.0.1.10	TCP	66	5001 → 51373 [ACK] Seq=1 Ack=254873 Win=0
280	12.168621	10.0.1.10	10.0.3.10	TCP	1090	[TCP Window Full] 51373 → 5001 [PSH, ACK] Seq=1
281	12.168664	10.0.3.10	10.0.1.10	TCP	66	[TCP ZeroWindow] 5001 → 51373 [ACK] Seq=1
282	15.336591	10.0.1.10	10.0.3.10	TCP	66	[TCP Keep-Alive] 51373 → 5001 [ACK] Seq=1
283	15.336634	10.0.3.10	10.0.1.10	TCP	66	[TCP ZeroWindow] 5001 → 51373 [ACK] Seq=1
284	21.521222	10.0.1.10	10.0.3.10	TCP	66	[TCP Keep-Alive] 51373 → 5001 [ACK] Seq=1

▶ Frame 281: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
 ▶ Ethernet II, Src: 00:00:00_aa:00:07 (00:00:00:aa:00:07), Dst: 00:00:00_aa:00:06 (00:00:00:aa:00:06)
 ▶ Internet Protocol Version 4, Src: 10.0.3.10, Dst: 10.0.1.10
 ▶ **Transmission Control Protocol, Src Port: 5001, Dst Port: 51373, Seq: 1, Ack: 255897, Len: 0**
 Source Port: 5001
 Destination Port: 51373
 [Stream index: 0]
 [Conversation completeness: Complete, WITH_DATA (31)]
 [TCP Segment Len: 0]
 Sequence Number: 1 (relative sequence number)
 Sequence Number (raw): 2219352713
 [Next Sequence Number: 1 (relative sequence number)]
 Acknowledgment Number: 255897 (relative ack number)
 Acknowledgment number (raw): 3436498131
 1000 = Header Length: 32 bytes (8)
 Flags: 0x010 (ACK)
 Window: 0
 [Calculated window size: 0]
 [Window size scaling factor: 16]
 Checksum: 0x183a [unverified]
 [Checksum Status: Unverified]
 Urgent Pointer: 0
 Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

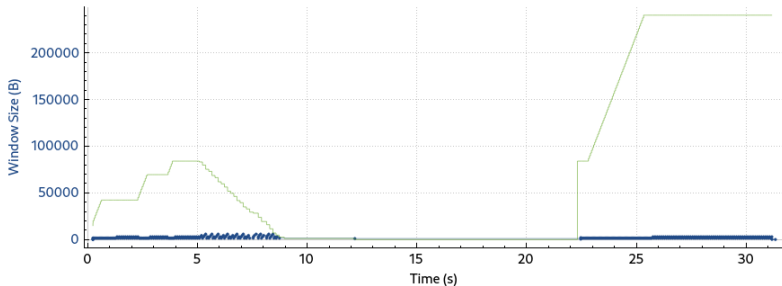
"tcp.analysis.zero" is neither a field nor a protocol name.

Packets: 607 · Displayed: 607 (100.0%) Profile: Classic

Control Flujo - wireshark (win. scale)

Window Scaling for 10.0.1.10:51373 → 10.0.3.10:5001

tcp-winscale-zerowin.pcap.gz



Referencias

- [KR] Kurose/Ross: Computer Networking (5th Edition).
- [Stev] TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens.
- [StevII] TCP/IP Illustrated, Volume 1: The Protocols, 2nd Ed. W. Richard Stevens, Kevin R. Fall.
- [RFCs] RFCs: <http://www.faqs.org/rfcs/> RFC-793, ... RFC-791, RFC-1323, RFC-2001, RFC-2018, RFC-2581, RFC-5681, RFC-2582, RFC-6582, RFC-3168, RFC-3649, RFC-2988, RFC-6298, RFC-7323.
- [TCPIPg] TCP/IP Guide: <http://www.tcpipguide.com/>.
- [Transport Layer] <http://people.westminstercollege.edu/faculty/ggagne/spring2007/352/notes/unit4/index.html>