TCP (Transport Control Protocol)
Flow Control

2019

UNIVERSIDAD NACIONAL DE LA PLATA

Contenidos

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, o Dynamic Window (Ventana Dinámica).
- La capacidad de envío será *MIN*(*Congestion*, *Flujo*, *Errores*).

Control de Errores TCP (Repaso)

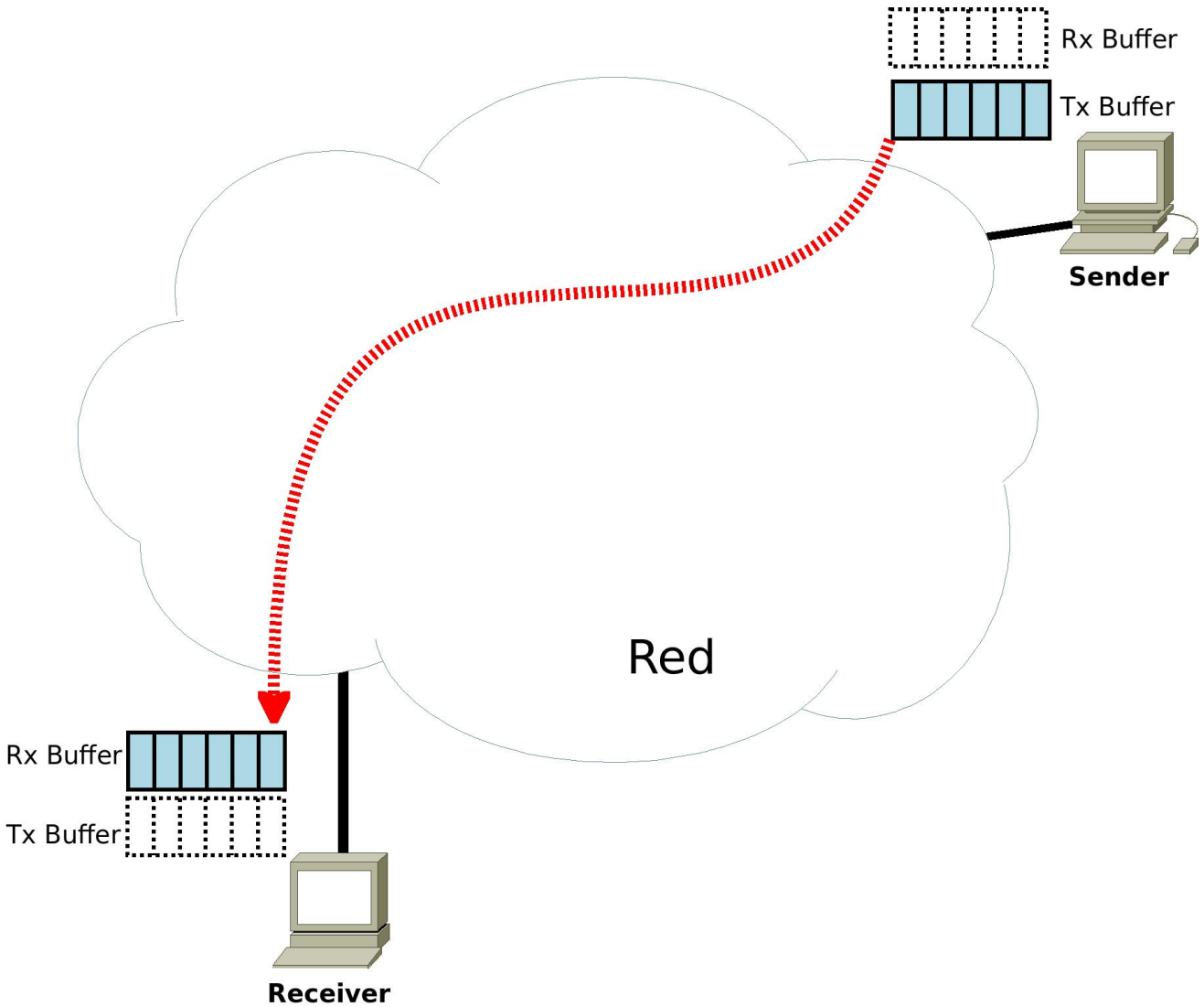
- Segmentos ACked no indica leído por aplicación, sí recibido por TCP (RFC-793) (ubicado en el **Rx Buffer** del receptor).
- Si el receptor detecta error en el segmento simplemente descarta y espera que expire *RTO* en el emisor (podría envía un NAK, re-enviar ACK para el último recibido en orden, forma de solicitar lo que falta).
- 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.
- Si se confirman (ACked) datos, se inicia un nuevo *RTO* (RFC-6298) recomendado.
- El nuevo *RTO* le esta dando más tiempo al segmento más viejo aún no confirmado.
- Si vence un RTO se debe retransmitir el segmento más viejo no ACKed y se debe doblar: Back-off timer $RTO = RTO * 2$ $RTO_{MAX} = 60s$ (RFC-6298) recomendado.
- TCP calcula el RTO de forma dinámica. RFC-6298(2011), ayudado por Timestamp Option.

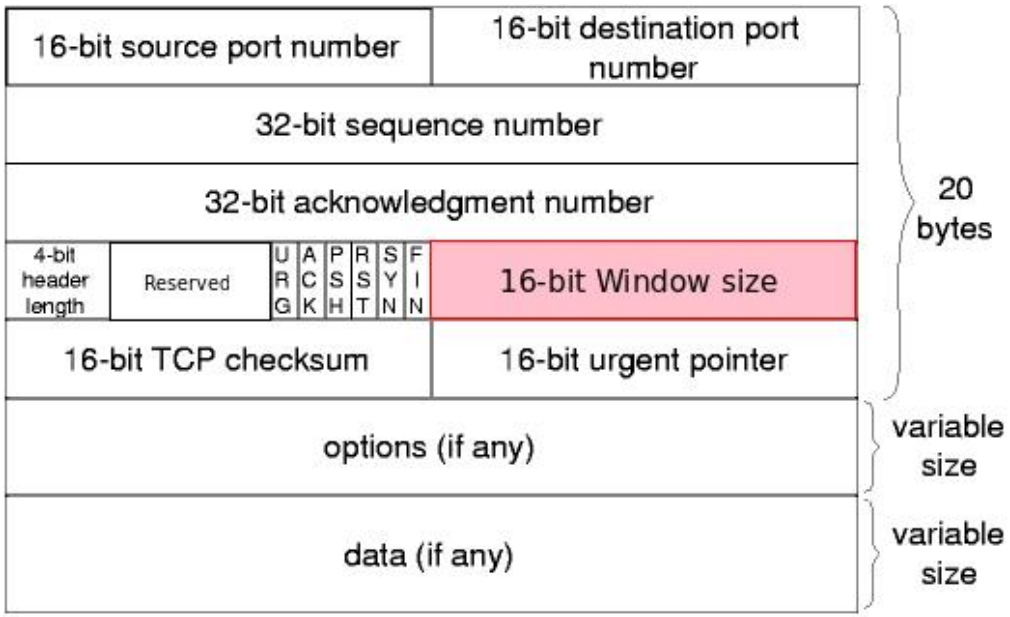
Control de Flujo TCP

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



Control de Flujo TCP

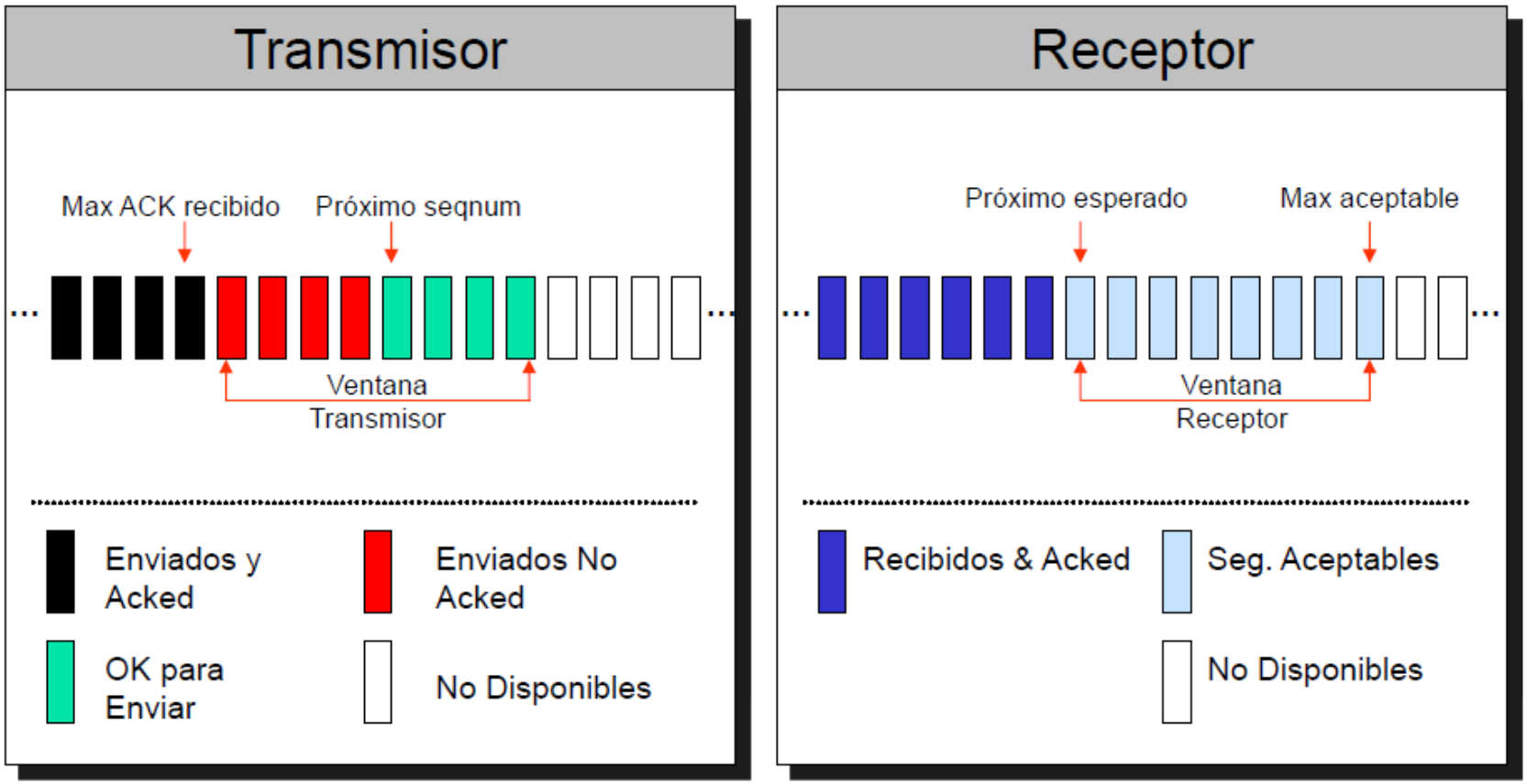
- El receptor (cada extremo puede recibir, es FDX) indica el espacio del buffer de recepción, **Rx Buffer**, en el campo del segmento: **Window** (de datos o ACK) **Advertised Window** (Ventana Anunciada).



Control de Flujo TCP

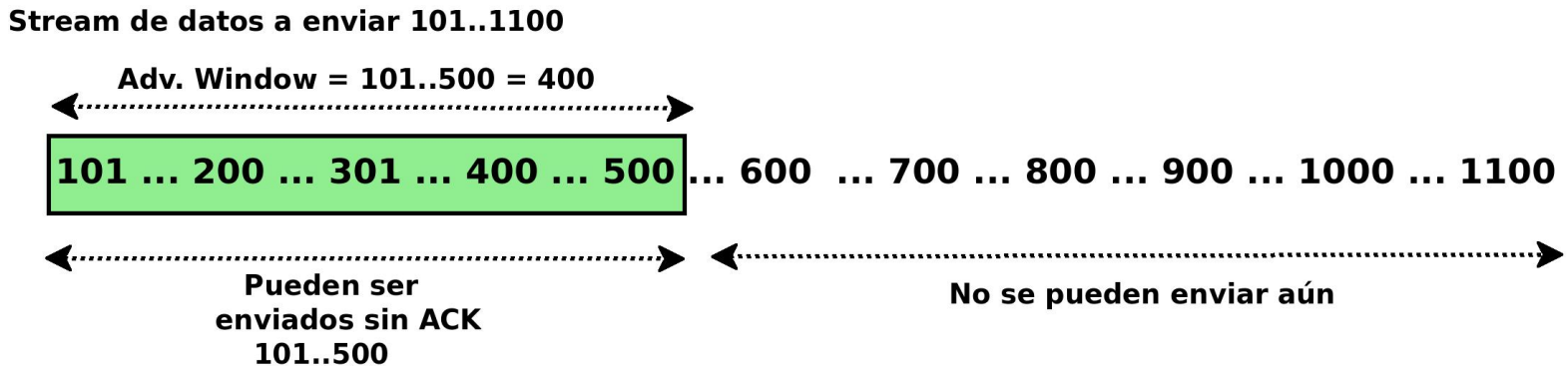
- Por cada segmento que envía indica el tamaño del buffer de recepción **Rx Buffer** (mbufs). (Cada conexión mantiene su propio buffer) en espacio del kernel (TCP).
- Window (Ventana) indica la cantidad de datos en bytes que el emisor le puede enviar sin esperar confirmación (mejora notablemente contra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

Ventana Deslizante TCP



Ventana Deslizante TCP (Inicial)

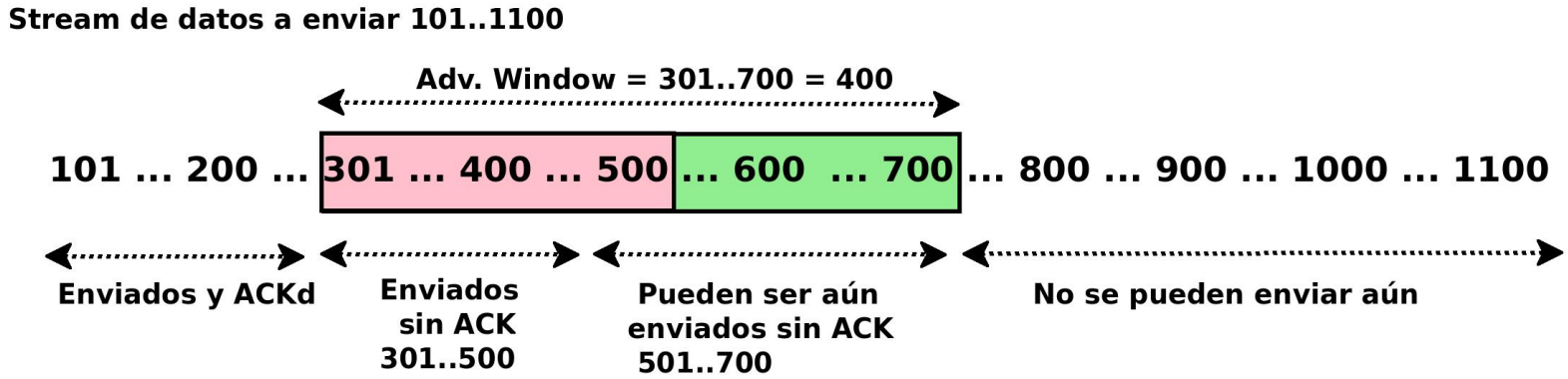
- Se establece la conexión, se indica $WIN = 400$.



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

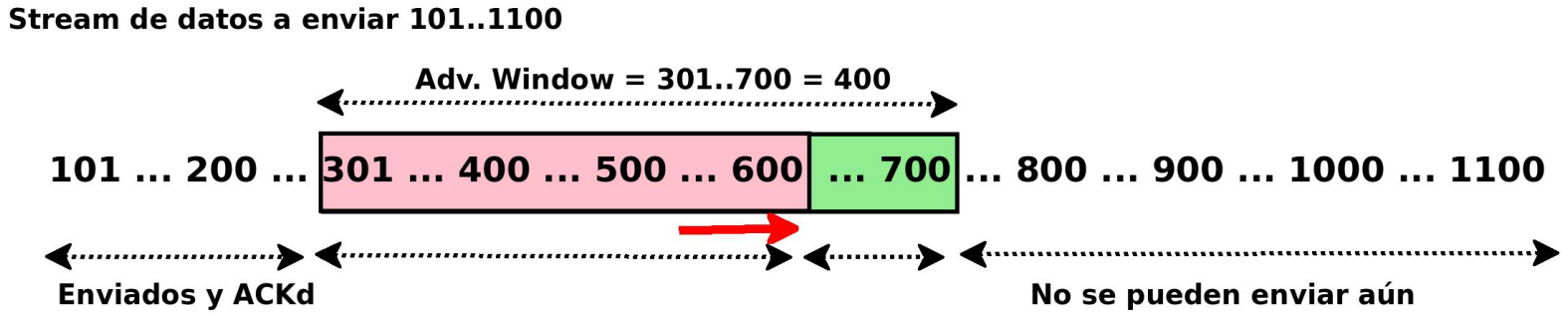
Ventana Deslizante TCP I

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



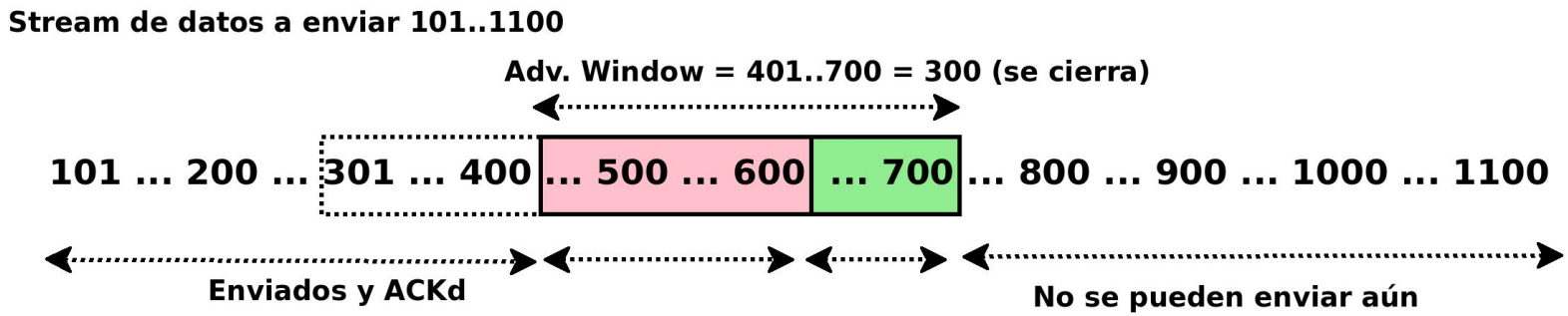
Ventana Deslizante TCP II

- Se envía un segmento con los bytes 501..600.
- 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.



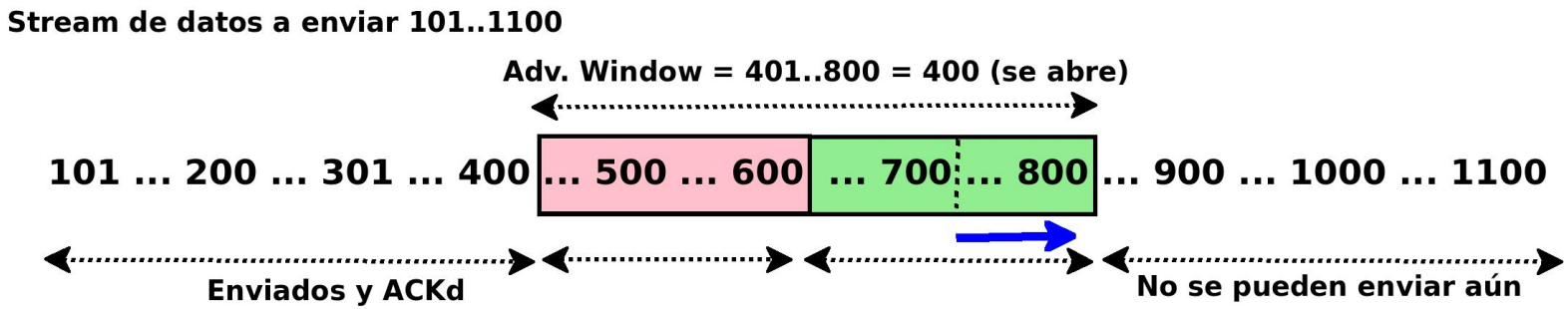
Ventana Deslizante TCP III

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



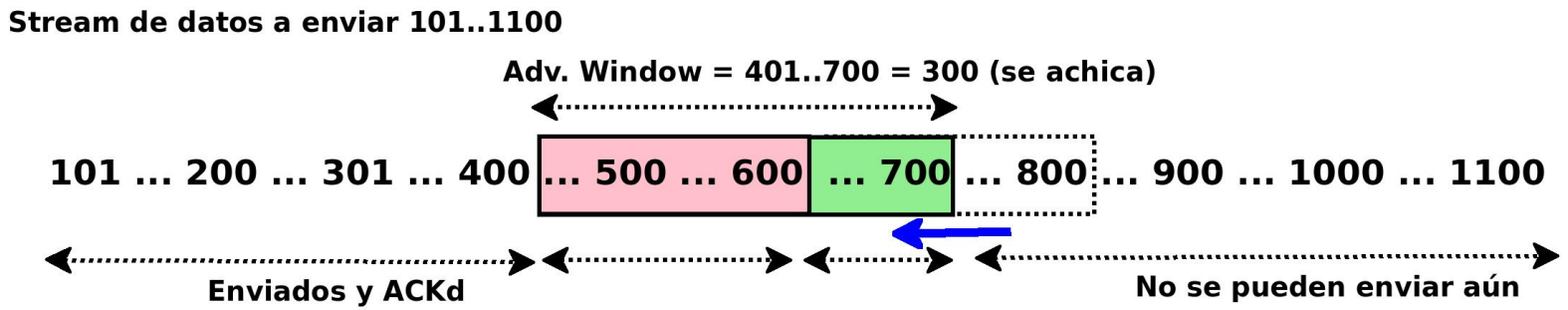
Ventana Deslizante TCP IV

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

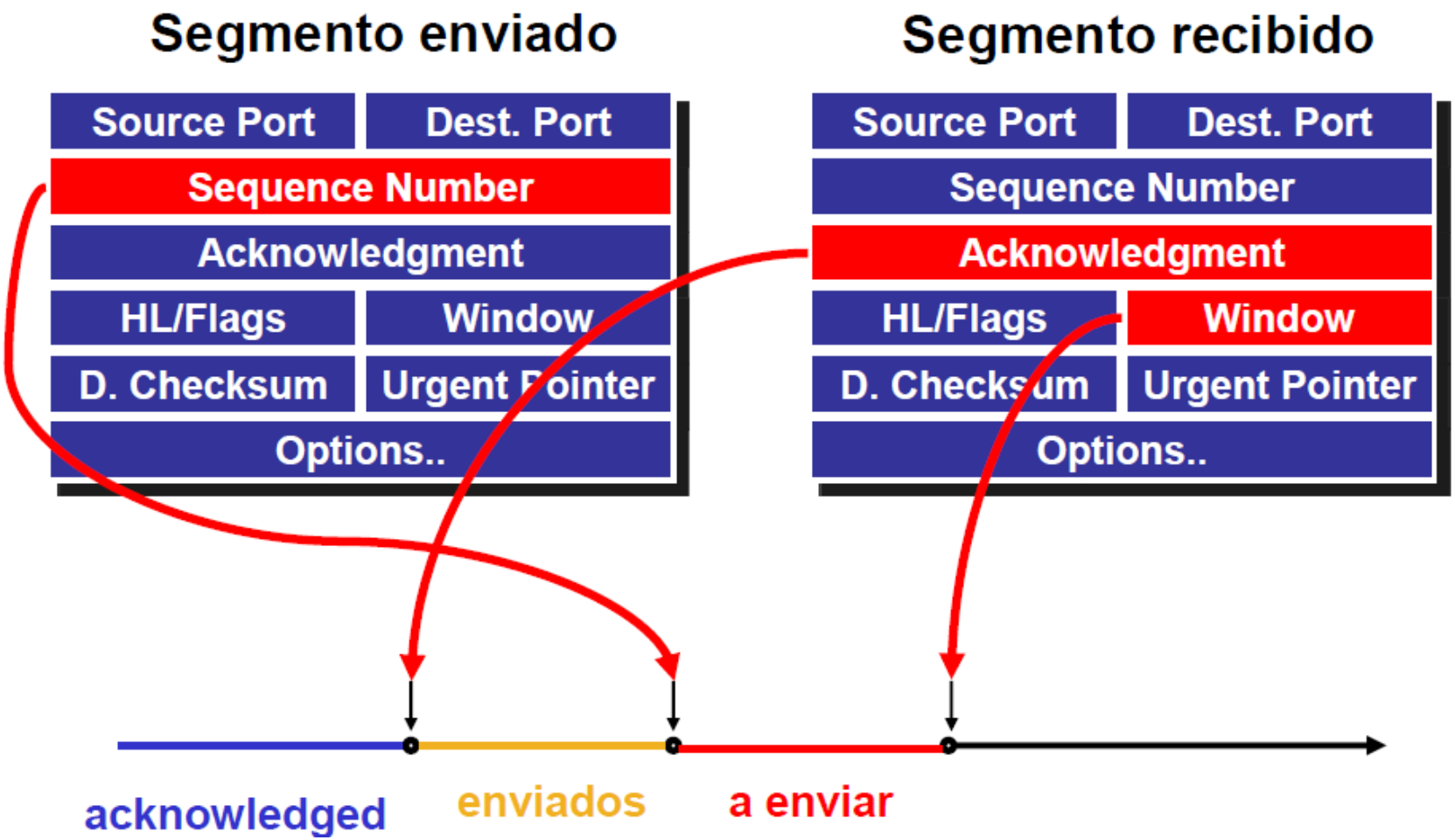


Ventana Deslizante TCP V

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



Ventana Deslizante TCP



Control de Flujo TCP

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

TCP Bulk y TCP Interactivo

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

Referencias

[KR] Kurose/Ross: Computer Networking (5th Edition).

[Stev] TCP/IP Illustrated, Volume 1: The Protocols, W. Richard Stevens.

[StevII] TCP/IP Illustrated, Volume 1: The Protocols, 2nd Ed. W. Richard Stevens, Kevin R. Fall.

[RFCs] RFCs: http://www.faqs.org/rfcs/ RFC-793, ... RFC-791, RFC-1323, RFC-2001, RFC-2018, RFC-2581, RFC-5681, RFC-2582, RFC-6582, RFC-3168, RFC-3649, RFC-2988, RFC-6298.

[TCPIPg] TCP/IP Guide: http://www.tcpipguide.com/.

[Transport Layer] http://people.westminstercollege.edu/faculty/ggagne/spring2007/352/notes/unit4/index.html