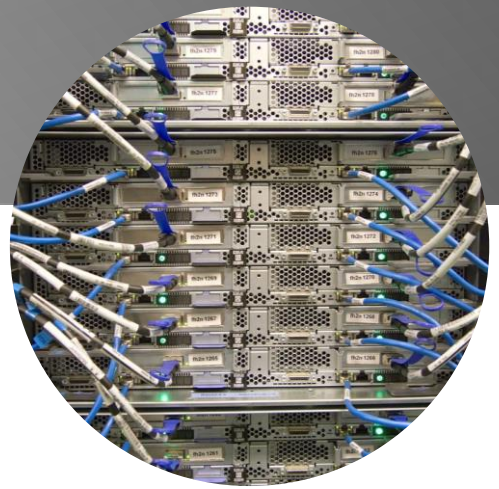
Redes de computadores 2022 -1 (11310052)

David Felipe Celeita Rodriguez

Universidad del Rosario | Escuela de Ingeniería, Ciencia y Tecnología

“My methods are really methods of working and thinking; this is why they have crept in everywhere anonymously”

Emmy Noether

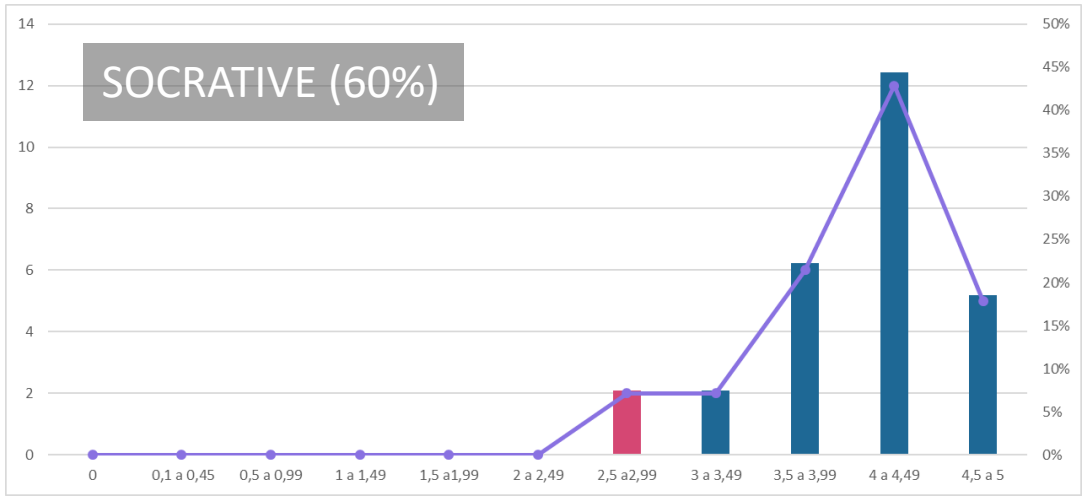
Programa

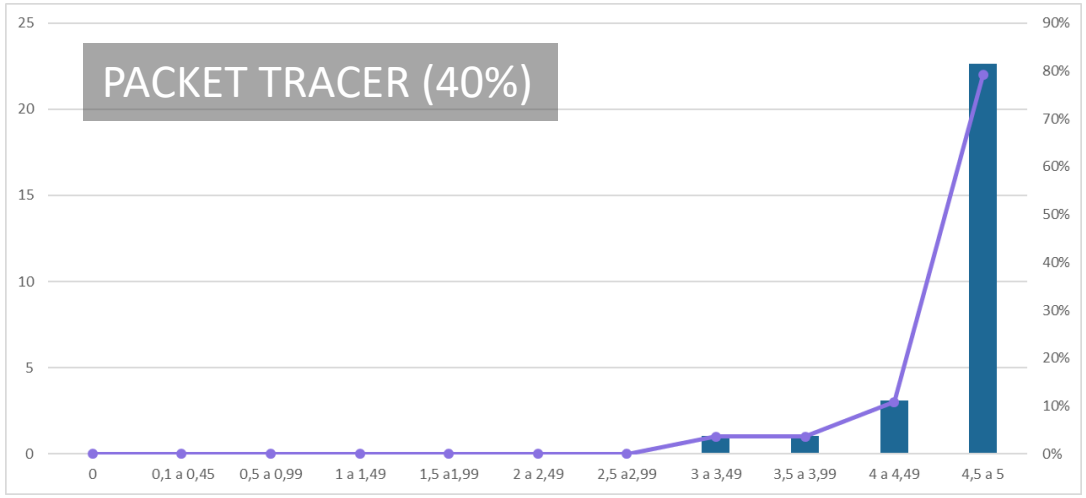
Fecha (Sesión)	Tema
Sesión 1-2 24 Ene – 28 Ene	Introducción a redes de computadores Parte 1
Sesión 3-4 31 Ene – 4 Feb	Introducción a redes de computadores Parte 2
Sesión 5-6 7 Feb – 11 Feb	Capa de aplicación Parte 1
Sesión 7-8 14 Feb – 18 Feb	Capa de aplicación Parte 2
Sesión 9-10 21 Feb – 25 Feb	Capa de transporte Parte 1
Sesión 10 21 Feb – 25 Feb	PARCIAL 1

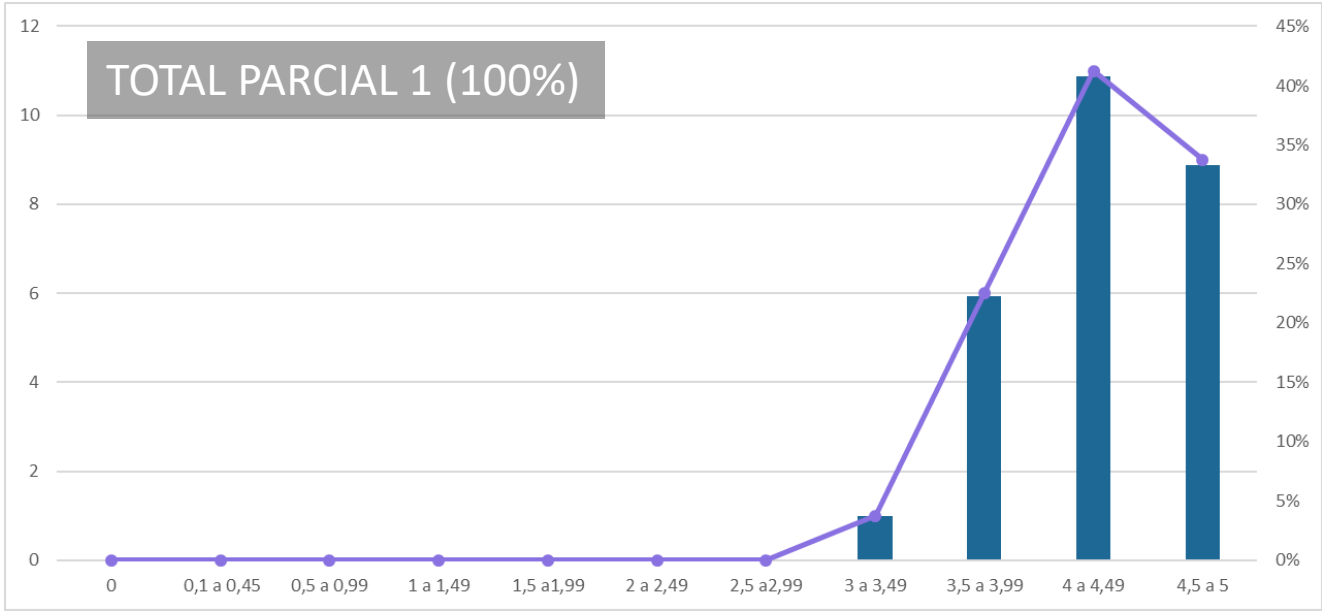
Programa

Fecha (Sesión)	Tema
Sesión 11-12 28 Feb – 4 Mar	Capa de transporte Parte 2
Sesión 13-14 7 Mar – 11 Mar	Capa de red Parte 1 (Plano de datos)
Sesión 15-16 14 Mar – 18 Mar	Capa de red Parte 2 (Plano de datos)
Sesión 17-20 21 Mar – 25 Mar	Capa de red Parte 3 (Plano de control)
Sesión 17-20 28 Mar – 1 Abr	Capa de red Parte 4 (Plano de control)
Sesión 20 28 Mar – 1 Abr	PARCIAL 2

RESULTADOS PARCIAL 1 - REDES

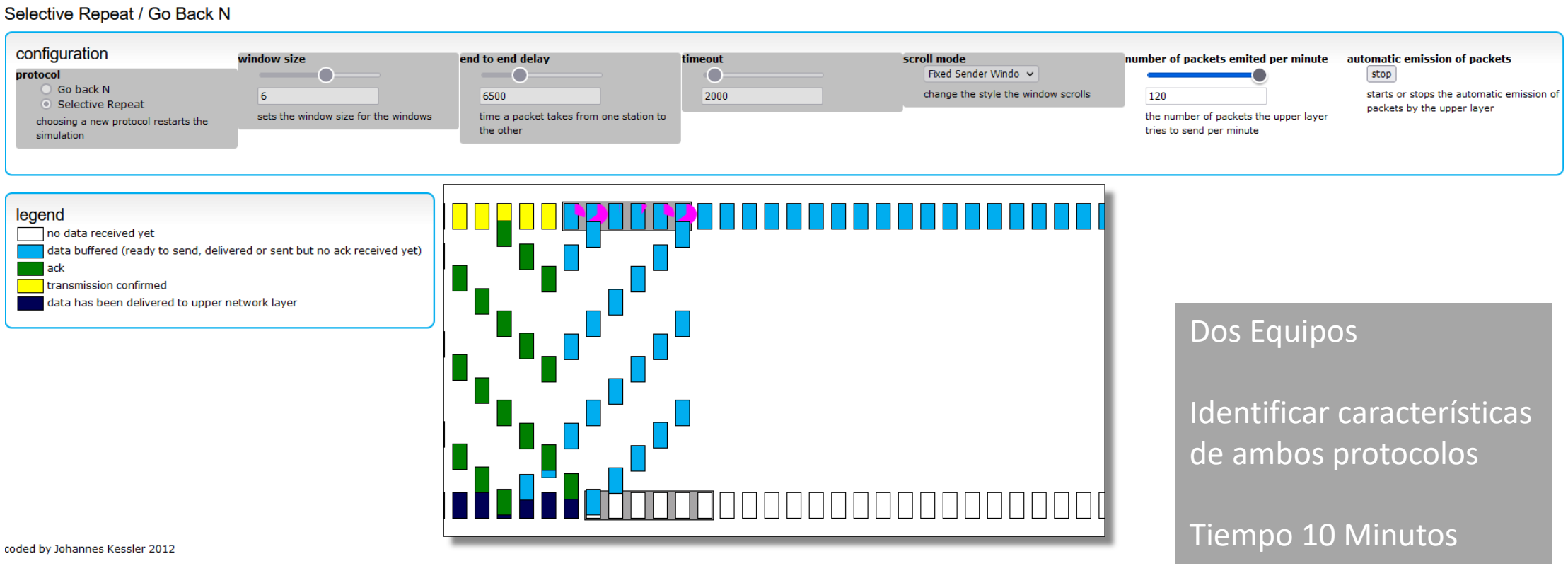






Estadísticos	
Promedio	4,27
Desviación estándar	0,37
Nota mínima	3,35
Nota Máxima	4,85
Aprobaron	27
Reprobaron	0
Aprobaron (%)	100,00%
Reprobaron (%)	0,00%

POMODORO
(Interacción SR Vs Go-Back-N)

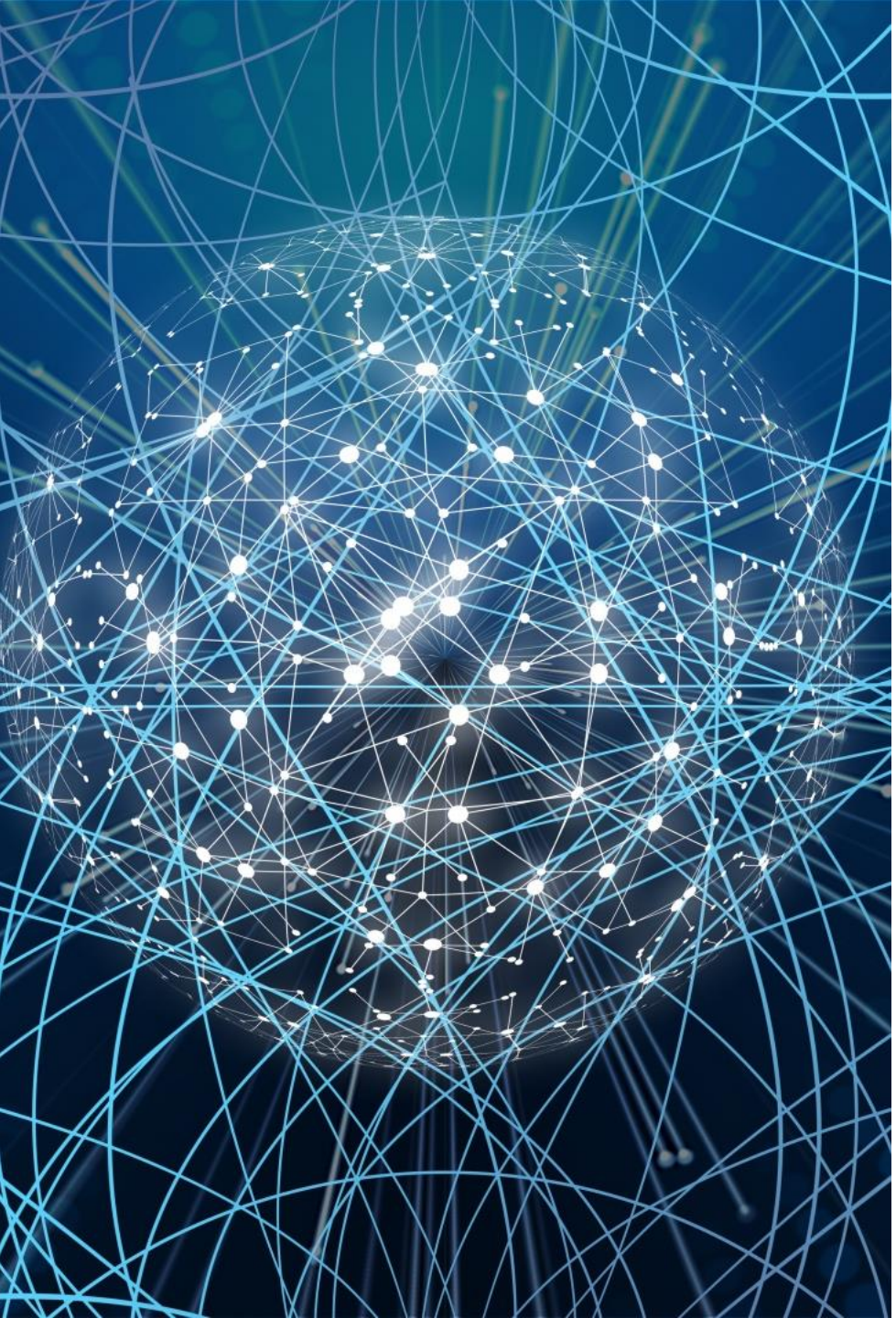


Dos Equipos

Identificar características de ambos protocolos

Tiempo 10 Minutos

https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/



Capítulo 3: Capa de transporte

Servicios de la capa de transporte
Multiplexación y demultiplexación
Transporte sin conexión: UDP
Principios de la transferencia de datos confiable
Transporte orientado a la conexión: TCP

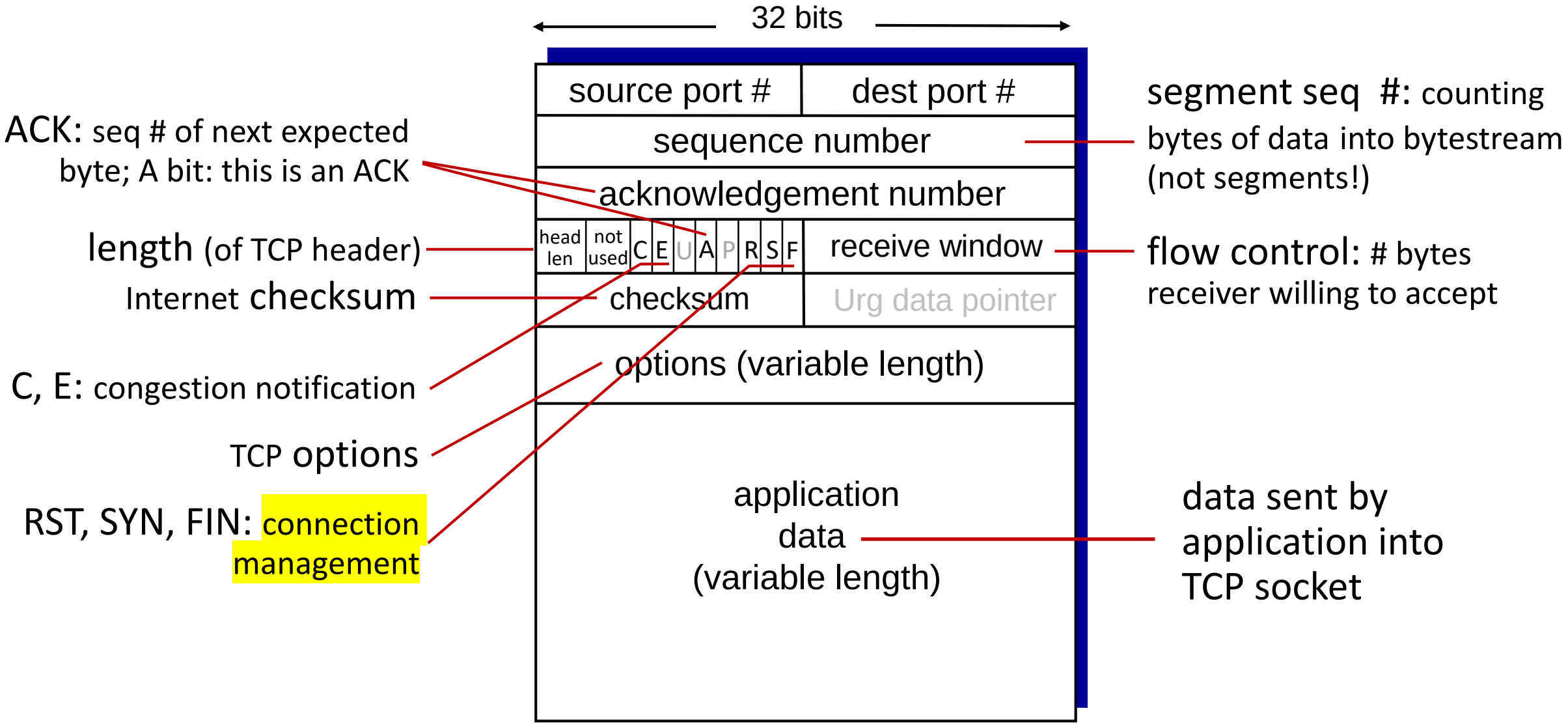
- estructura del segmento
- transferencia de datos confiable
- control de flujo
- gestión de conexión

Principios del control de la congestión
Control de congestión TCP
Evolución de la funcionalidad de la capa de transporte

TCP: Generalidades RFCs: 793,1122, 2018, 5681, 7323

- Punto a punto:
  - 1 sender/remitente,
  - 1 receiver/receptor
- Confiable con bytes en orden de envío:
  - sin “limitaciones del mensaje"
- full duplex data:
  - Flujo bidireccional de datos en la misma conexión
  - MSS: maximum segment size
- ACKs acumulativos
- pipelining:
  - Tamaño de ventana establecido de control de flujo y congestión de TCP
- Orientada a la conexión:
  - handshaking (intercambio de mensajes de control) inicializa el estado del emisor y del receptor antes del intercambio de datos
- Flujo controlado:
  - El remitente no va a abrumar a quien está recibiendo mensajes

TCP segmento (estructura)



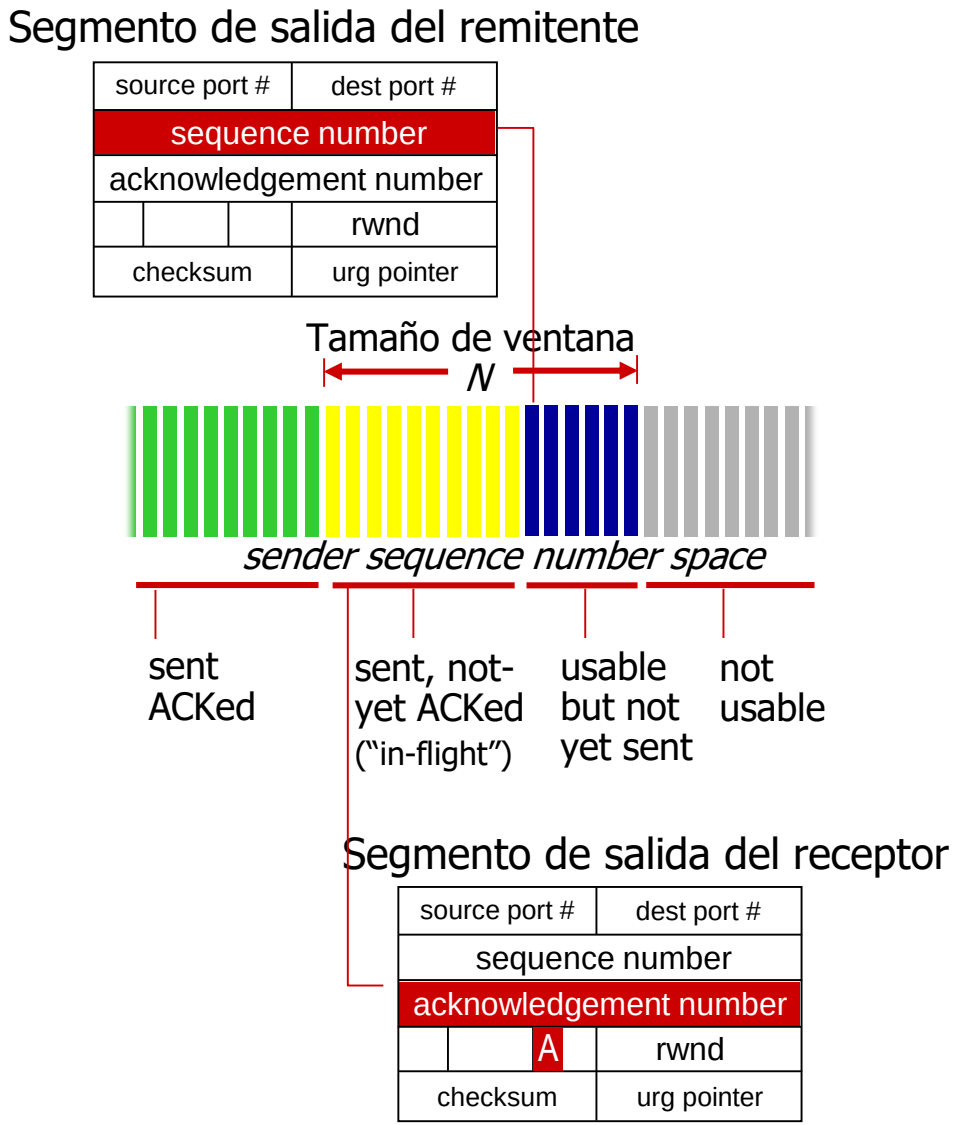
TCP Números de secuencia y ACKs

Seq #:

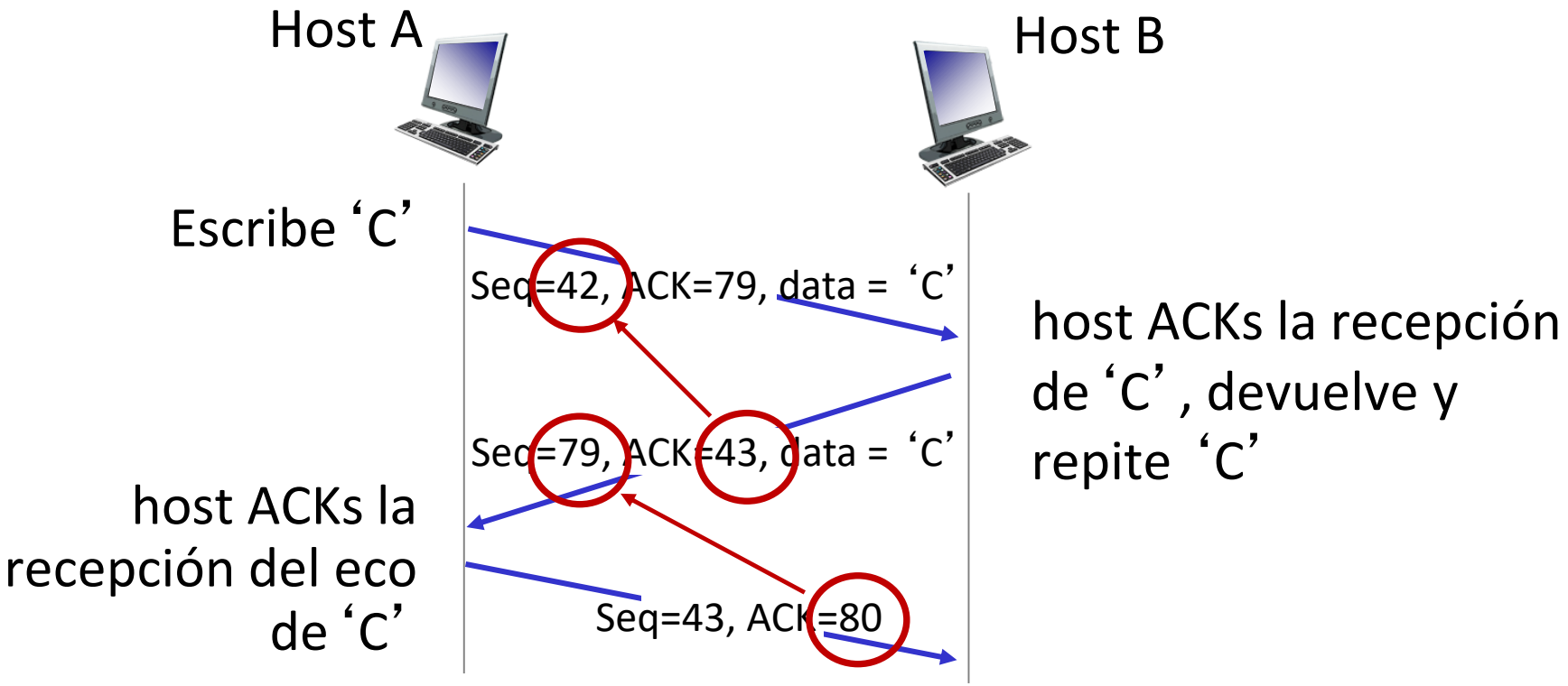
- Entrega el número del primer byte del segmento de datos

ACKs:

- seq # del siguiente byte esperado desde el otro lado
- ACK acumulativo



TCP Números de secuencia y ACKs



TCP (RTT) round trip time, timeout

¿Cómo definir el timeout en TCP?

- Timeout > RTT, pero el RTT varía!
- *Muy corto:* timeout anticipado, retransmisiones innecesarias
- *Muy largo:* reacción lenta a segmentos perdidos

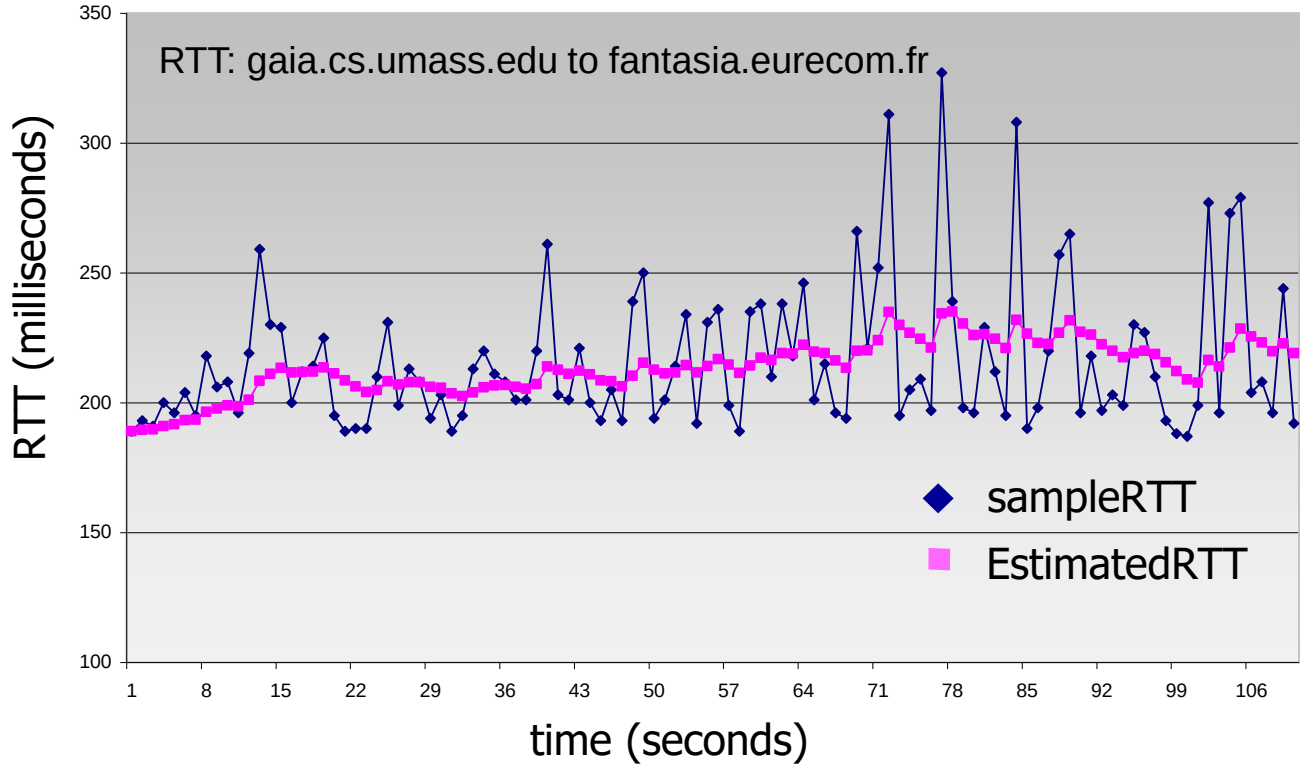
¿Cómo estimar el RTT?

- `SampleRTT:` tiempo medido desde la transmisión del segmento hasta la recepción de ACK (ignorando re transmisiones)
- `SampleRTT` va a variar, se desearía un RTT "suave"
  - promediar varias mediciones recientes, no solo las actuales `SampleRTT`

TCP (RTT) round trip time, timeout

$$\mathbf{EstimatedRTT = (1-\alpha)*EstimatedRTT + \alpha*SampleRTT}$$

- exponential weighted moving average (EWMA)
- la influencia de la muestra pasada disminuye exponencialmente rápido
- Valor típico: $\alpha = 0.125$



TCP (RTT) round trip time, timeout

- Intervalo timeout: **`EstimatedRTT`** con “un márgen de seguridad”
  - Grandes variaciones en **`EstimatedRTT:`** se desea aumentar el margen de seguridad estimando desviaciones respecto al promedio

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

“safety margin”

- **`DevRTT`**: EWMA of **`SampleRTT`** desviación de **`EstimatedRTT`**:

```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
```

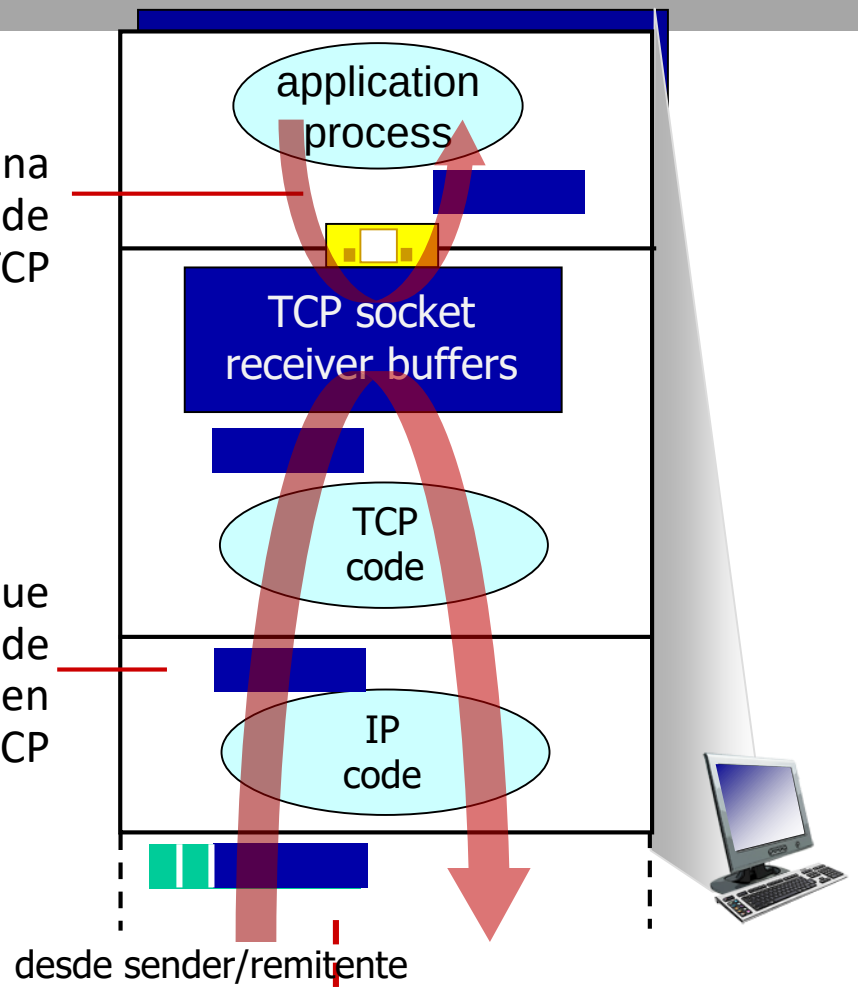
(typically, β = 0.25)

TCP – control de flujo

¿Qué sucede si la capa de red entrega datos más rápido que el tiempo que tarda la capa de aplicación en eliminar datos de los búferes del socket?

Aplicación que elimina datos de los búferes de socket TCP

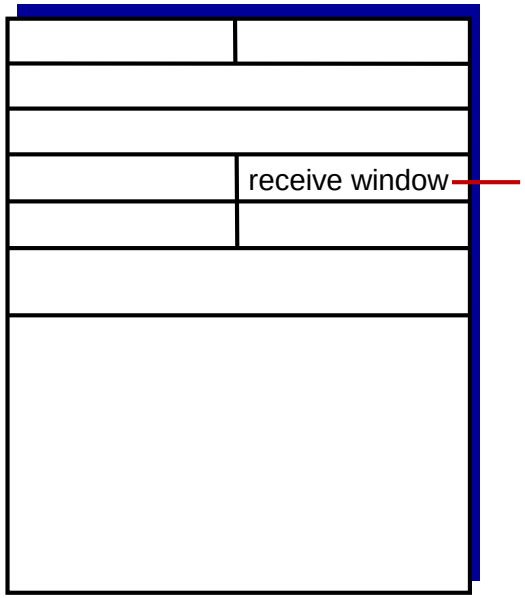
Capa de red que entrega payload de datagrama IP en búferes de socket TCP



receiver protocol stack

TCP – control de flujo

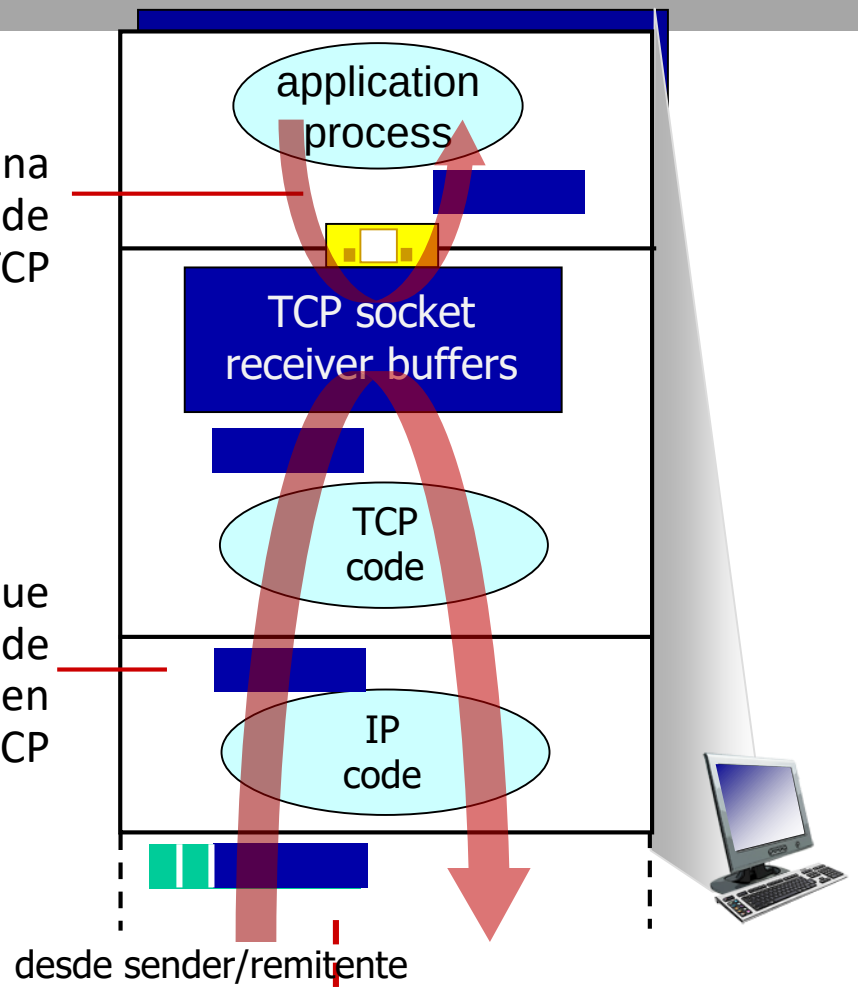
¿Qué sucede si la capa de red entrega datos más rápido que el tiempo que tarda la capa de aplicación en eliminar datos de los búferes del socket?



flow control:
bytes que el receptor está dispuesto a leer

Aplicación que elimina datos de los búferes de socket TCP

Capa de red que entrega payload de datagrama IP en búferes de socket TCP



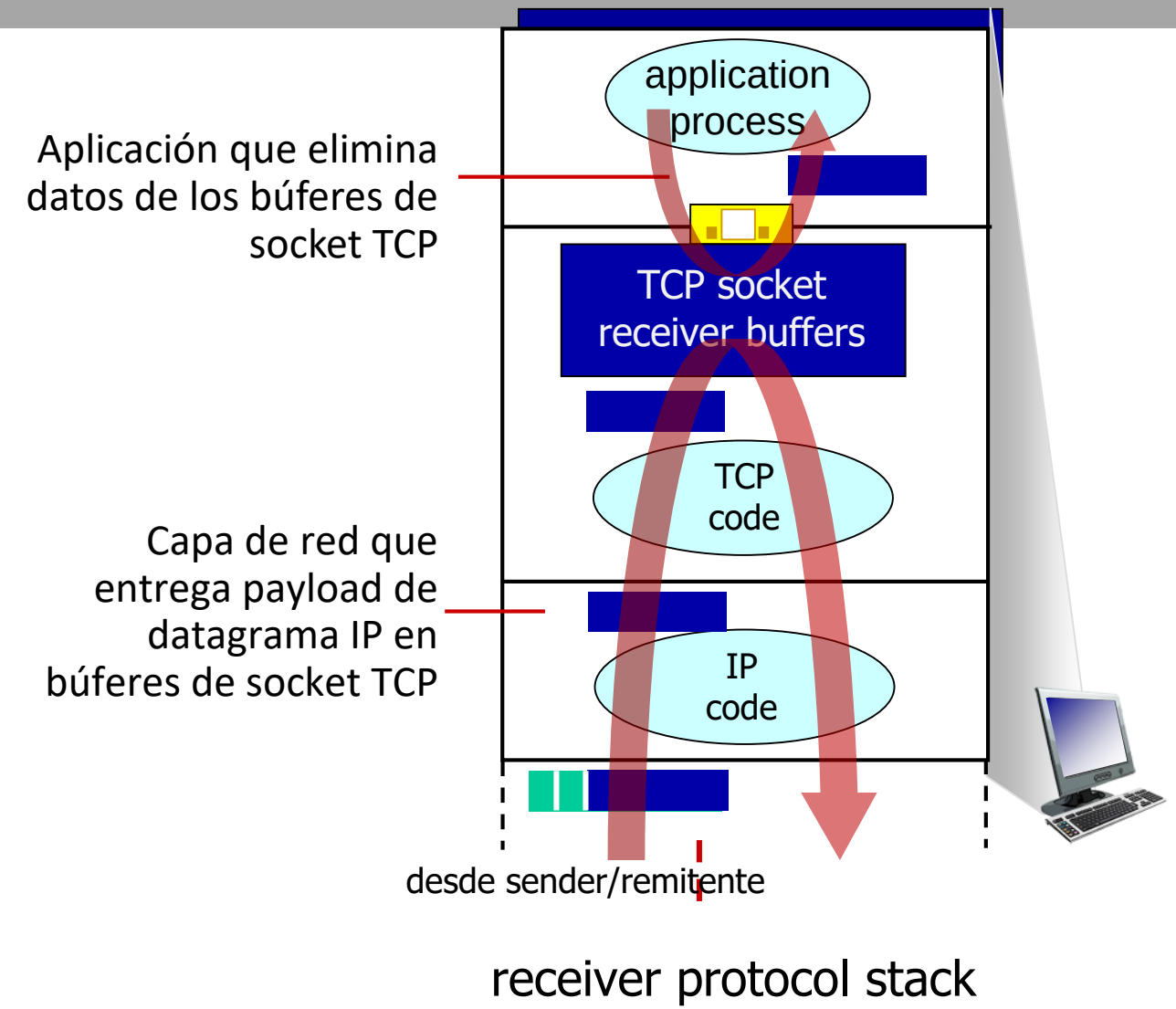
receiver protocol stack

TCP – control de flujo

¿Qué sucede si la capa de red entrega datos más rápido que el tiempo que tarda la capa de aplicación en eliminar datos de los búferes del socket?

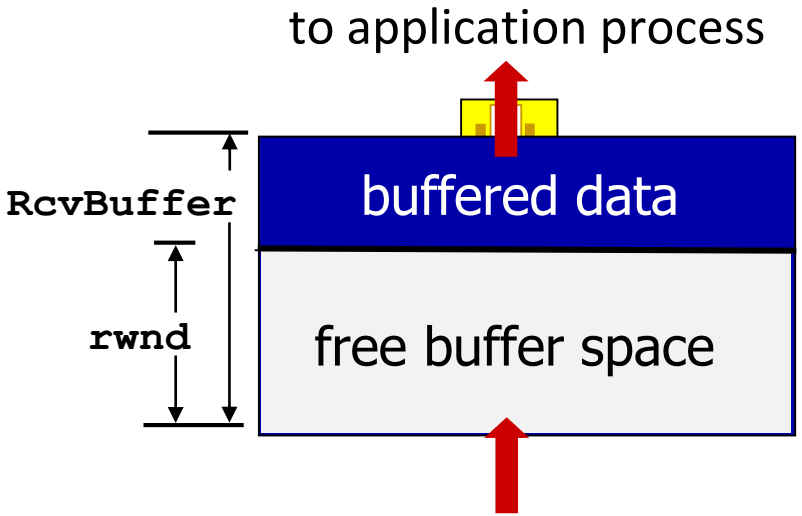
flow control

el receptor controla al remitente, por lo que el remitente no desbordará el búfer del receptor transmitiendo demasiado o muy rápido



TCP – control de flujo

- El receptor TCP "anuncia" el espacio libre de búfer en el campo **rwnd** del encabezado TCP
- El tamaño de **RcvBuffer** se establece a través de las opciones de socket (el valor predeterminado típico es 4096 bytes)
- muchos sistemas operativos autoajustan RcvBuffer
- el remitente limita la cantidad de datos no APILADOS ("en curso") al rwnd recibido
- garantiza que el búfer de recepción no se desborde

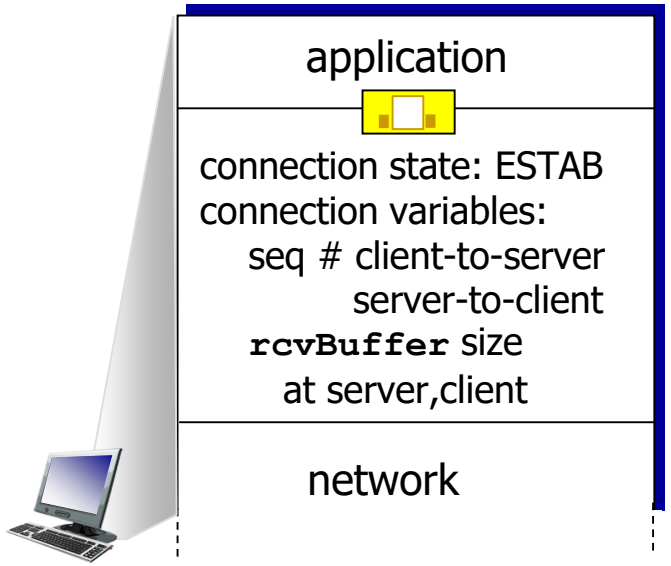


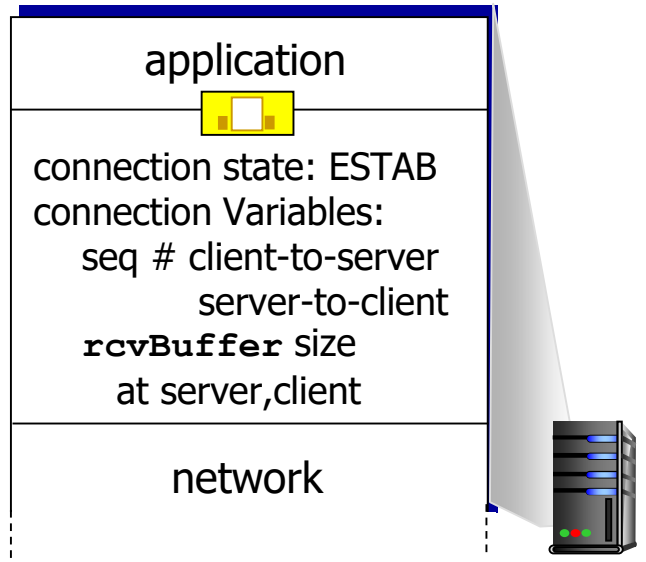
TCP buffering del receptor

TCP – gestión de la conexión

Antes de intercambiar datos → sender/receiver “handshake”:

- acordar establecer conexión (cada uno sabiendo que el otro está dispuesto a establecer conexión)
- acordar los parámetros de conexión (p. ej., número de secuencia inicial)



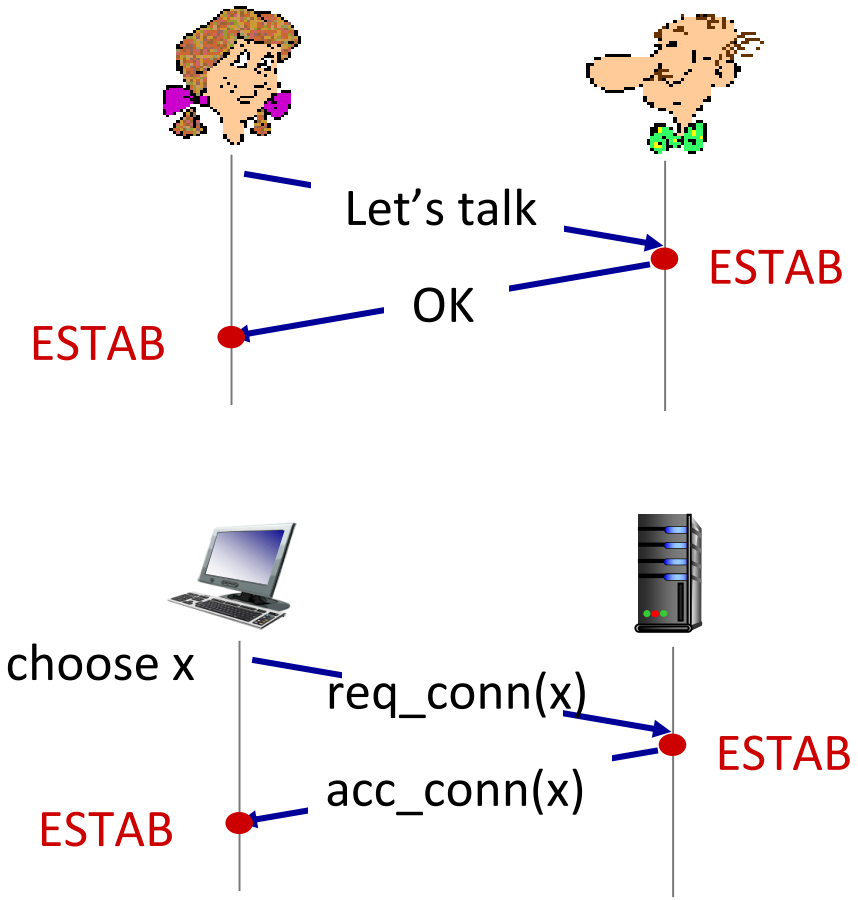



```
Socket clientSocket =
  newSocket("hostname","port number");
```

```
Socket connectionSocket =
  welcomeSocket.accept();
```

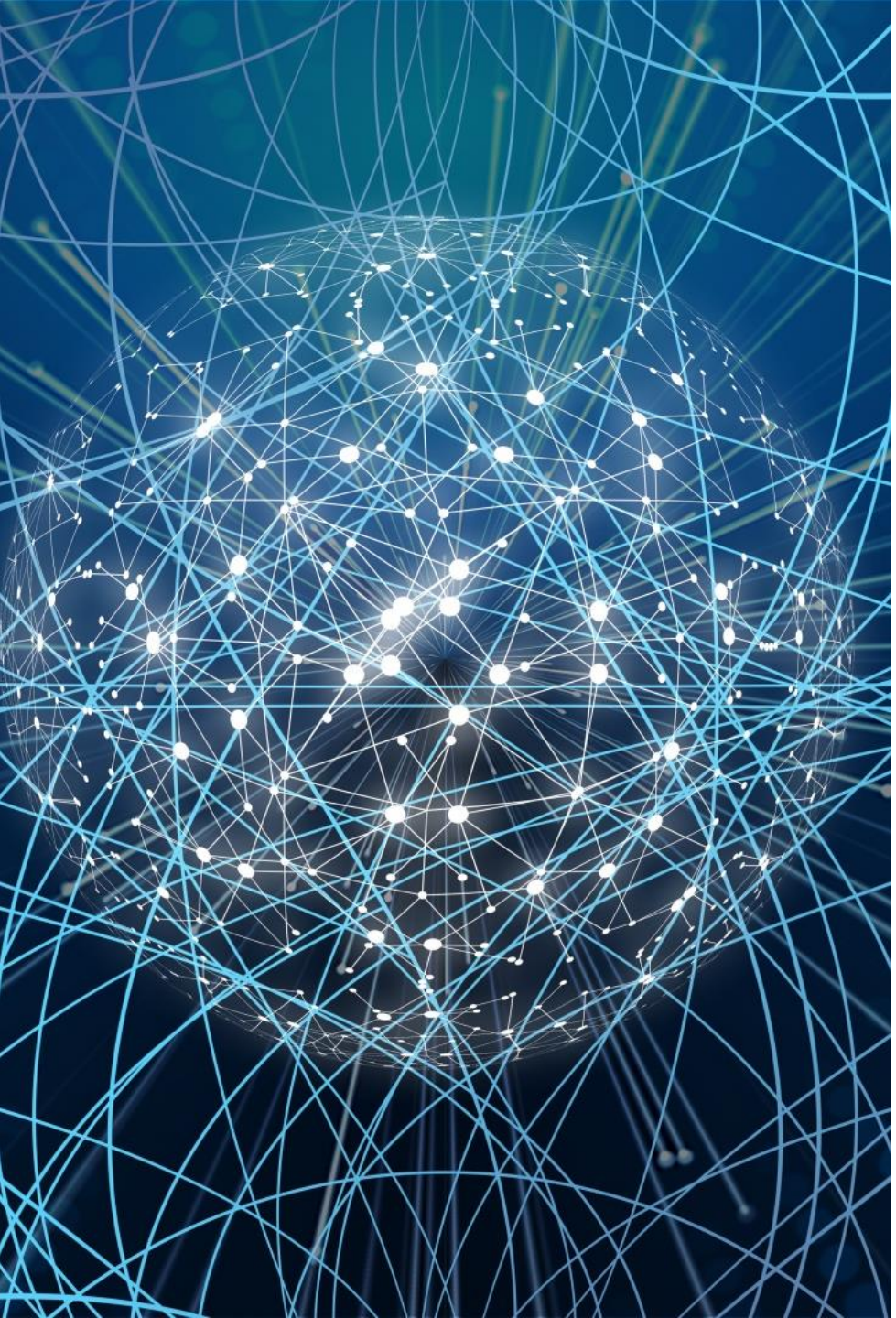
TCP - acordar establecer conexión

2-way handshake:



Q: ¿ 2-way handshake siempre funcionará en la red?

- retrasos variables
- mensajes retransmitidos (por ejemplo, req_conn(x)) debido a la pérdida de mensajes
- reordenamiento de mensajes
- no puedo "ver" el otro lado



Capítulo 3: Capa de transporte

Servicios de la capa de transporte
Multiplexación y demultiplexación
Transporte sin conexión: UDP
Principios de la transferencia de datos confiable
Transporte orientado a la conexión: TCP

- estructura del segmento
- transferencia de datos confiable
- control de flujo
- gestión de conexión

Principios del control de la congestión
Control de congestión TCP
Evolución de la funcionalidad de la capa de transporte

Principios del control de la congestión

Congestión:

- "demasiadas fuentes que envían demasiados datos demasiado rápido para que *la red* los maneje"
- Efectos:
  - retrasos prolongados (hacer cola en los búferes del enrutador)pérdida de paquetes (desbordamiento del búfer en los enrutadores)
- Diferente al control de flujo
- Problema de alta prioridad

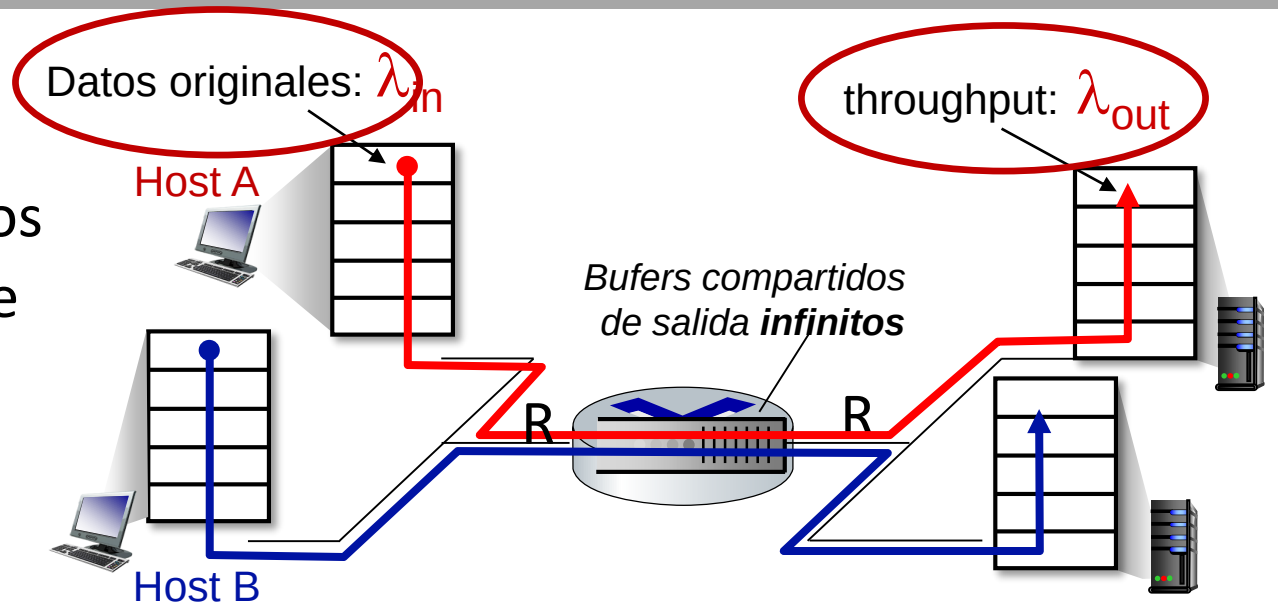
control de congestión: Demasiados remitentes enviando datos muy rápido

Control de flujo: un remitente envía muy rápido a un receptor

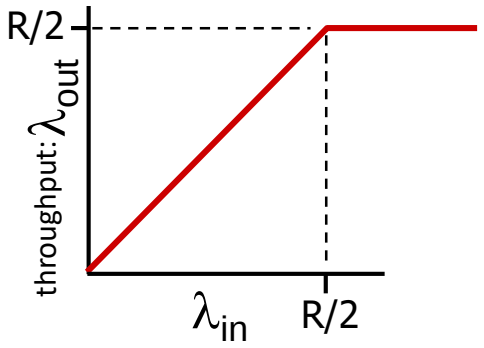
Análisis de congestión: escenario 1

Escenario básico:

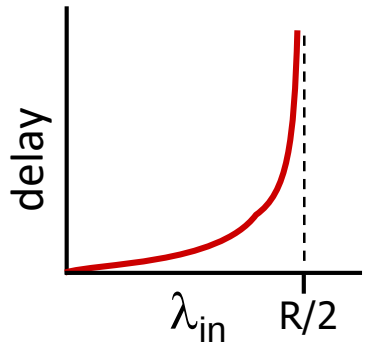
- un enrutador, búferes infinitos
- entrada, capacidad del enlace de salida: R
- Dos flujos
- Sin re-transmisión



Q: Qué pasa cuando la tasa de llegada λ_{in} se acerca a R/2?



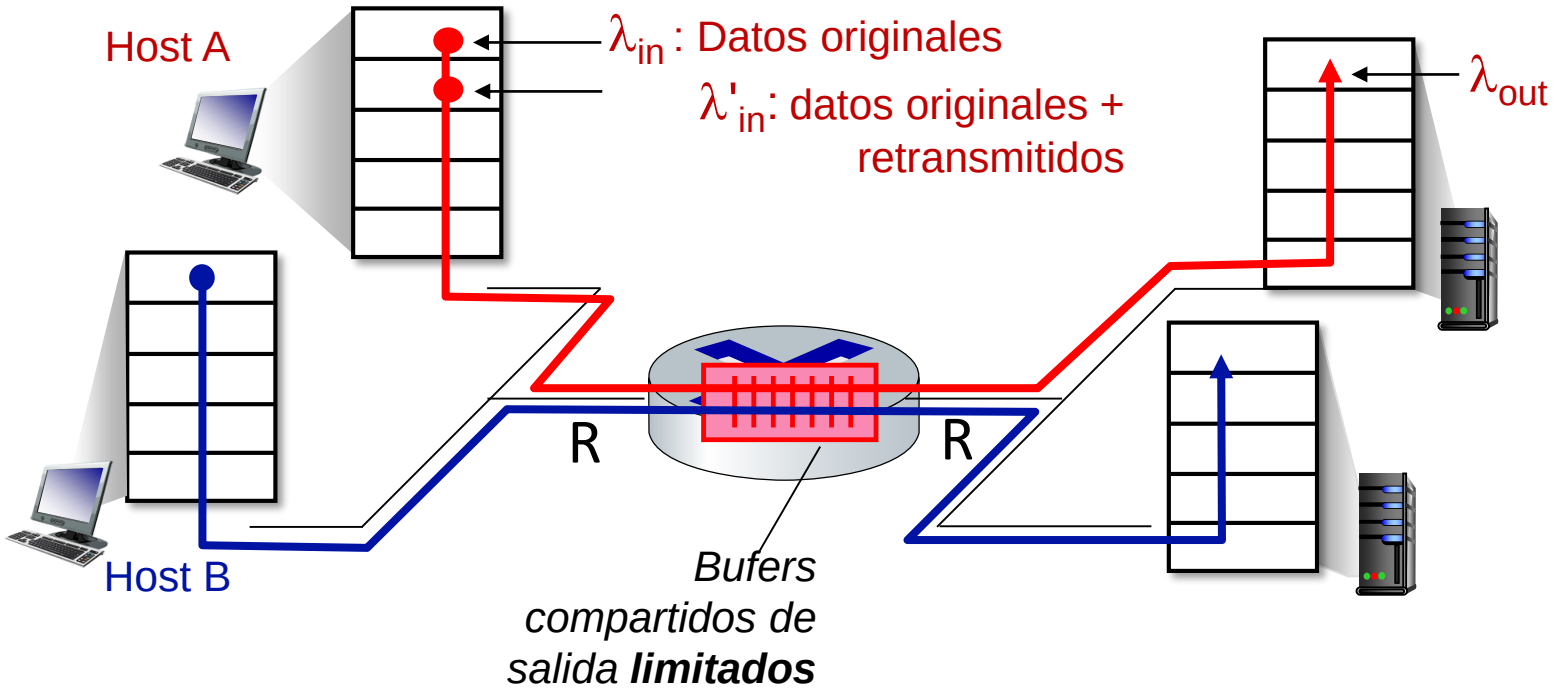
Máximo throughput por conexión: R/2



Crecimiento de retardos cuando λ_{in} se acerca a la capacidad

Análisis de congestión: escenario 2

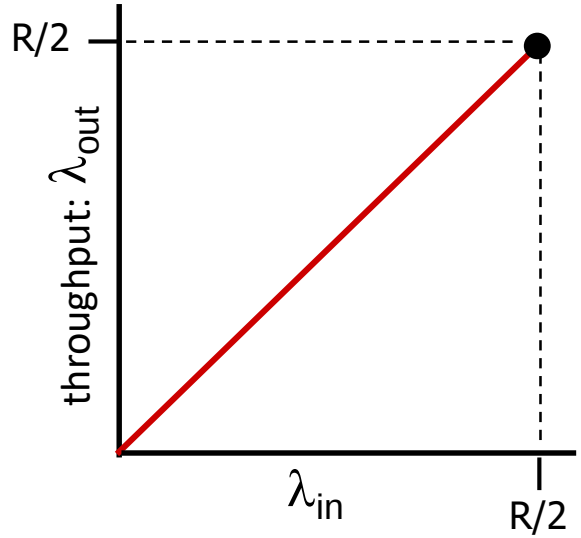
- 1 enrutador, búfers *limitados*
- el remitente retransmite el paquete perdido (timed-out packet)
  - entrada de la capa de aplicación = salida de la capa de aplicación: $\lambda_{in} = \lambda_{out}$
  - la entrada de la capa de transporte incluye retransmisiones: $\lambda'_{in} \geq \lambda_{in}$

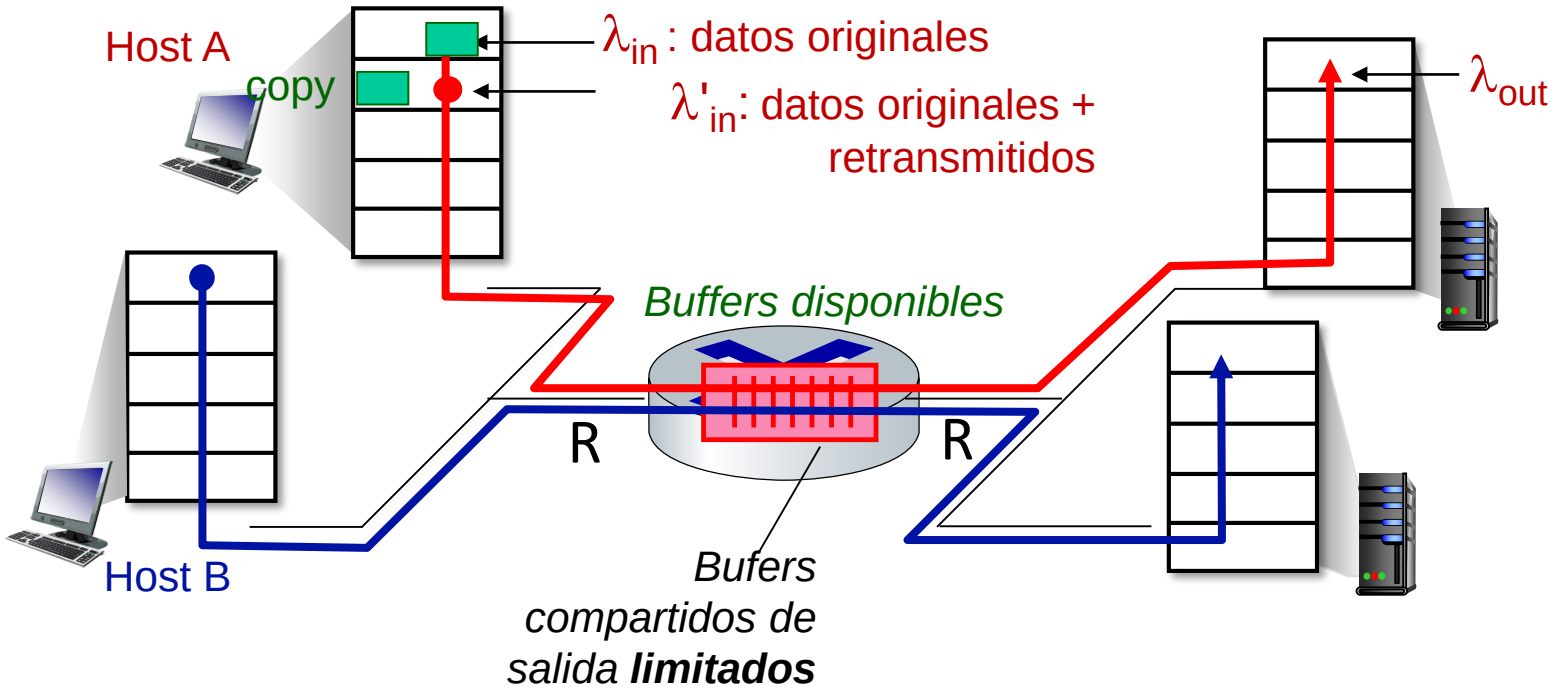


Análisis de congestión: escenario 2

Idealización: conocimiento perfecto

- El remitente solo envía cuando hay buffers disponibles

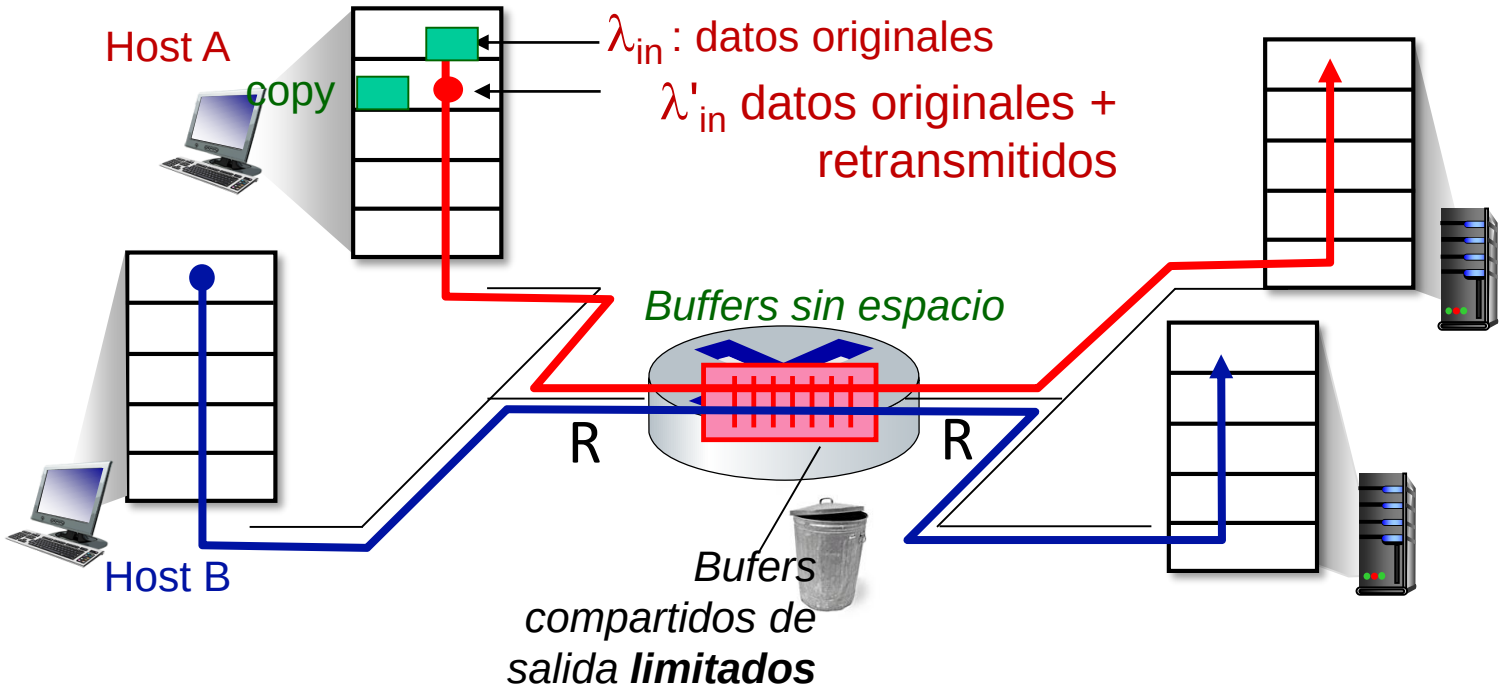




Análisis de congestión: escenario 2

Idealización: *un poco* de conocimiento

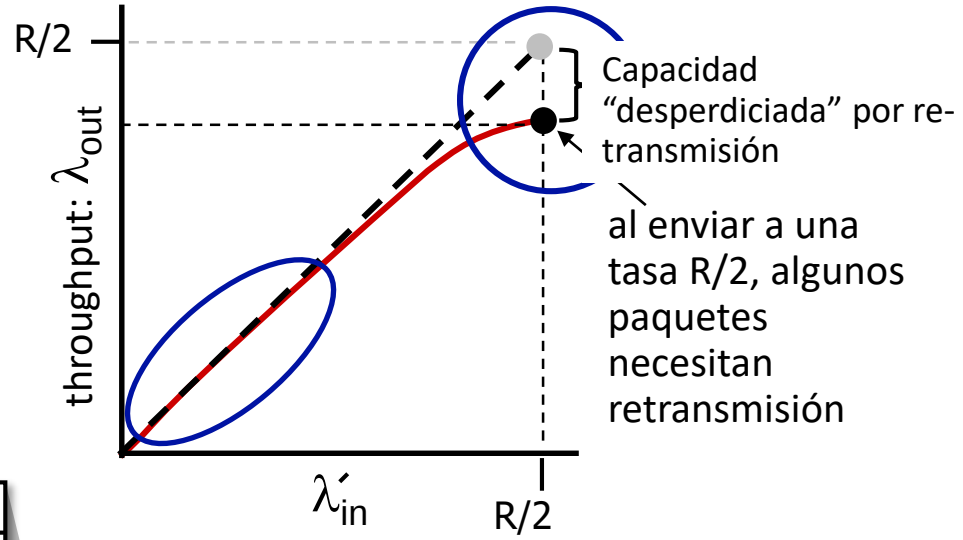
- los paquetes se pueden perder (caen en el enrutador) debido a los búferes llenos
- el remitente sabe cuándo se ha descartado el paquete: solo se reenvía si se sabe que el paquete se ha perdido

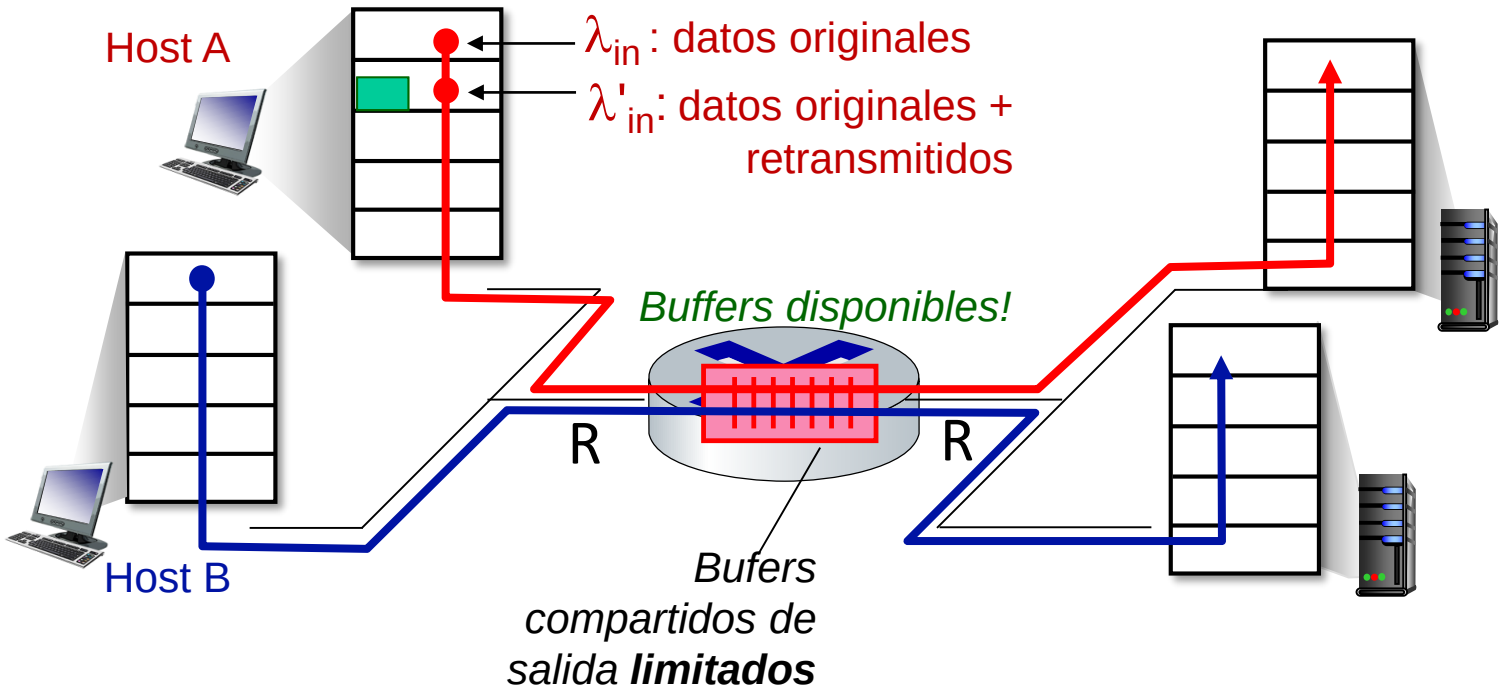


Análisis de congestión: escenario 2

Idealización: *un poco* de conocimiento

- los paquetes se pueden perder (caen en el enrutador) debido a los búferes llenos
- el remitente sabe cuándo se ha descartado el paquete: solo se reenvía si se sabe que el paquete se ha perdido

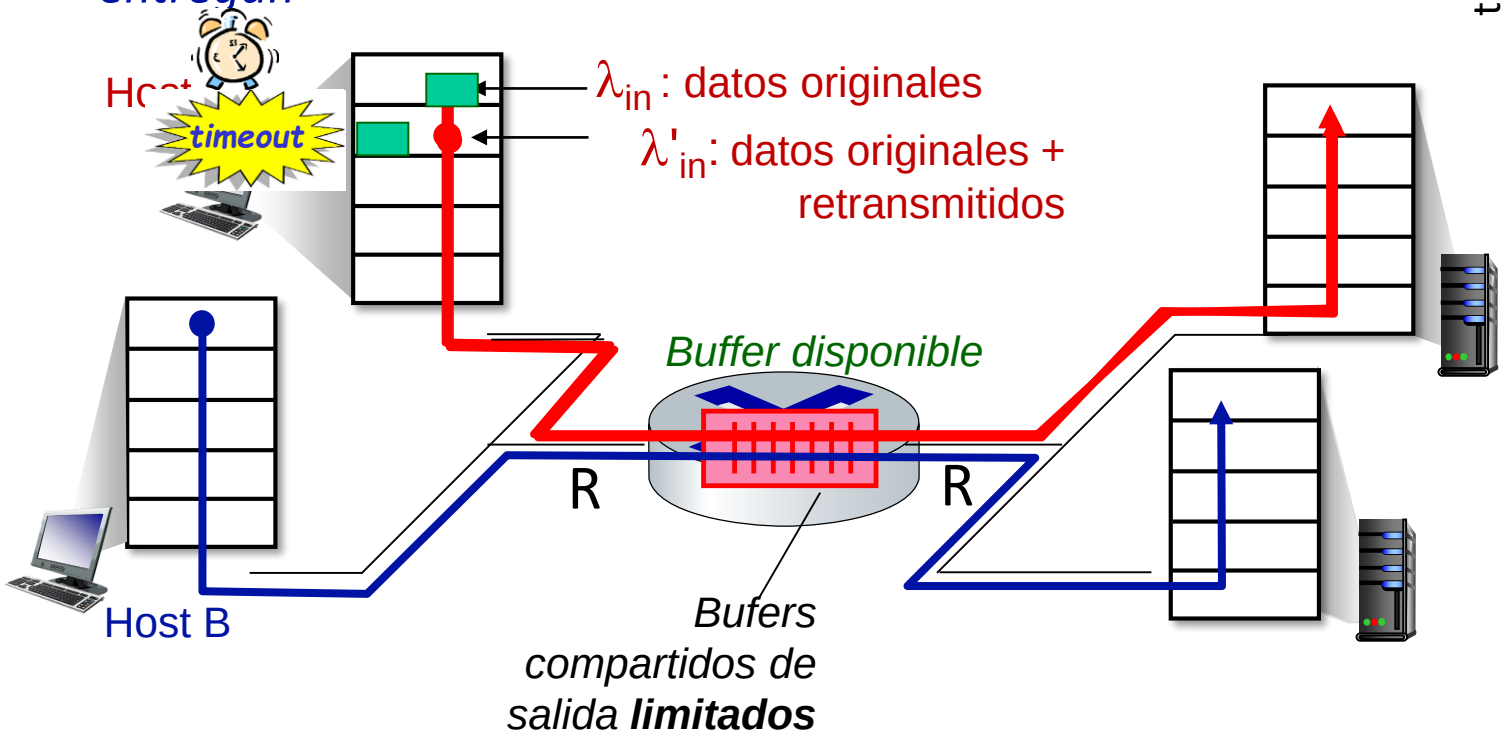


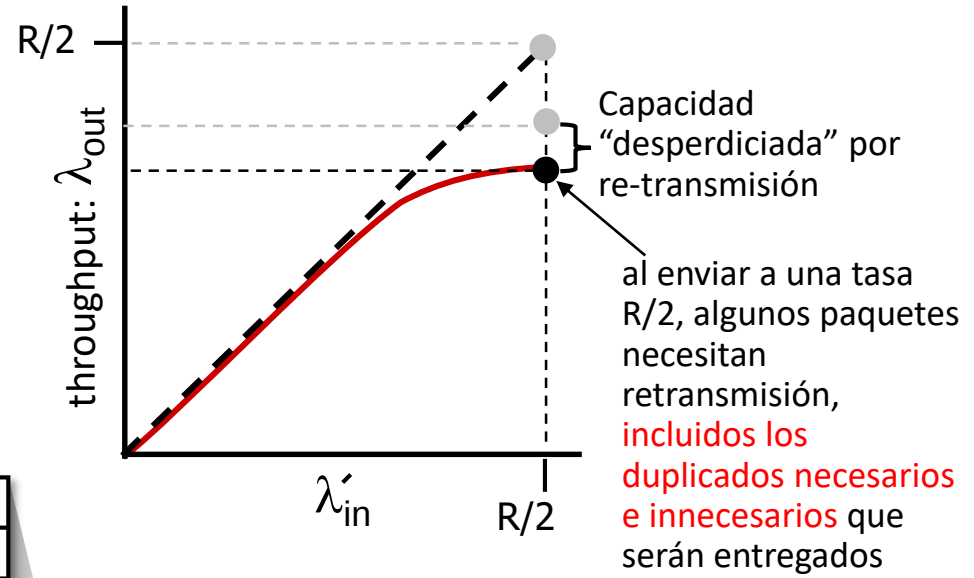


Análisis de congestión: escenario 2

Escenario realista: *duplicados innecesarios*

- los paquetes se pueden perder, descartar en el enrutador debido a los búferes llenos, lo que requiere retransmisiones
- pero los tiempos del remitente pueden expirar prematuramente, *enviando dos copias, las cuales se entregan*

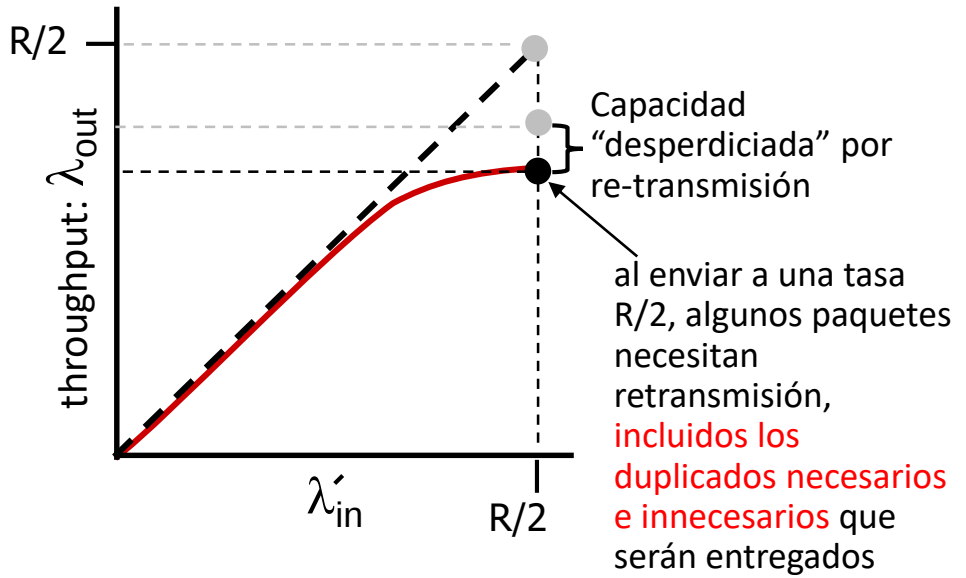




Análisis de congestión: escenario 2

Escenario realista: *duplicados innecesarios*

- los paquetes se pueden perder, descartar en el enrutador debido a los búferes llenos, lo que requiere retransmisiones
- pero los tiempos del remitente pueden expirar prematuramente, *enviando dos copias, las cuales se entregan*



Costos y consecuencias de la congestión:

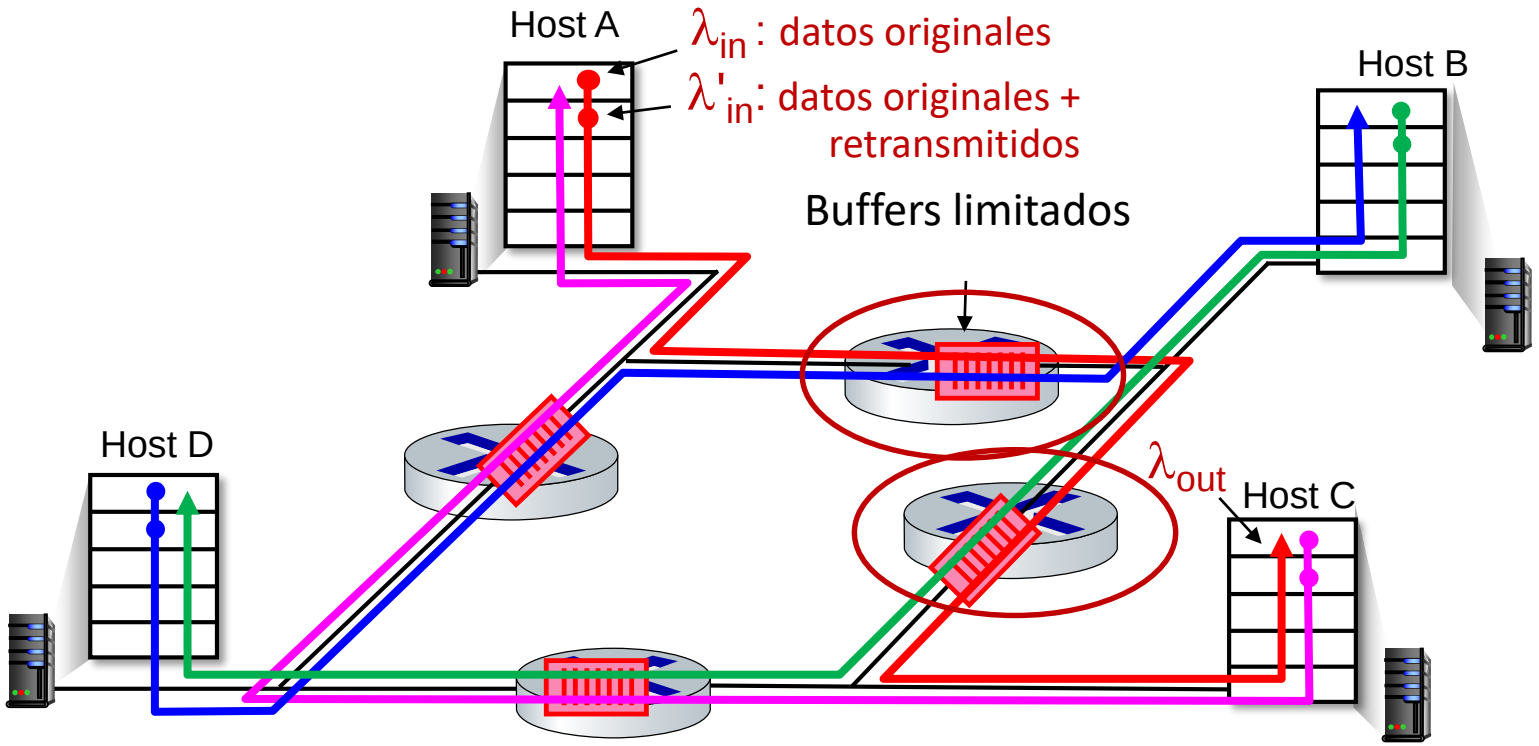
- más trabajo (retransmisión) para el rendimiento (throughput)del receptor dado
- retransmisiones innecesarias: el enlace lleva varias copias de un paquete
  - disminución del rendimiento (throughput) máximo alcanzable

Análisis de congestión: escenario 3

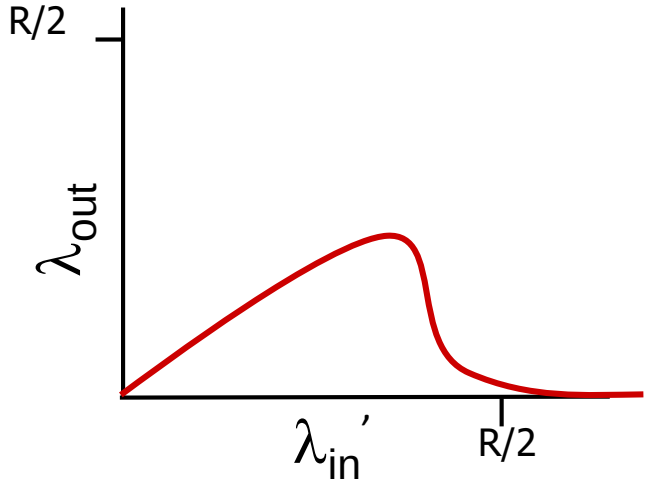
- *4* senders/remitentes
- *multi-hop* paths (Caminos de muchos saltos)
- timeout/retransmit

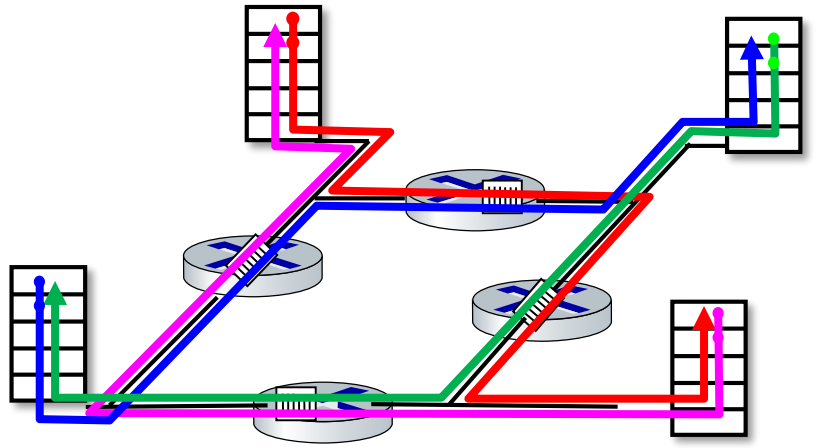
Q: ¿Qué pasa cuando λ_{in} y λ_{in}' se incrementan?

A: de rojo λ_{in}' aumenta, todos los pkts de llegada en azul en la parte superior de la cola son eliminados, throughput (azul) → 0



Análisis de congestión: escenario 3



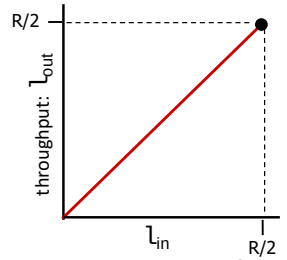


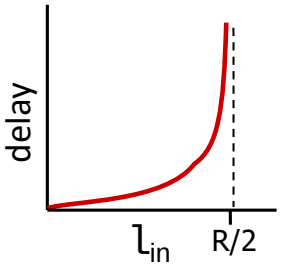
Costos adicionales de congestión:

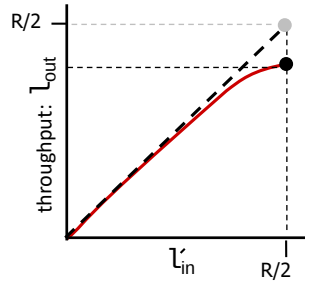
- cuando el paquete se cae, se desperdicia la capacidad de transmisión ascendente y el almacenamiento en búfer utilizado para ese paquete.

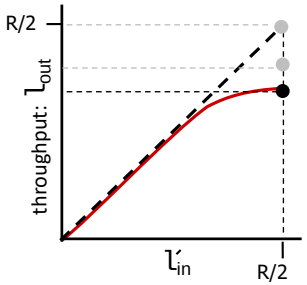
Perspectivas de análisis de congestión

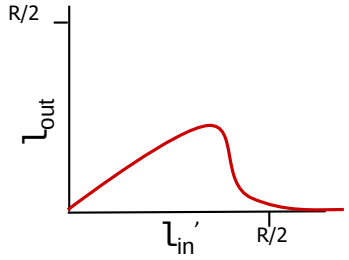
- El desempeño (throughput) nunca puede exceder la capacidad
- El retardo incrementa cuando se acerca a la capacidad
- Perdidas/retransmisiones disminuyen la efectividad de desempeño (throughput)
- Los duplicados innecesarios lo disminuye aun más
- Capacidad de transmisión ascendente Vs almacenamiento en búfer desperdiciado por paquetes perdidos en sentido descendente







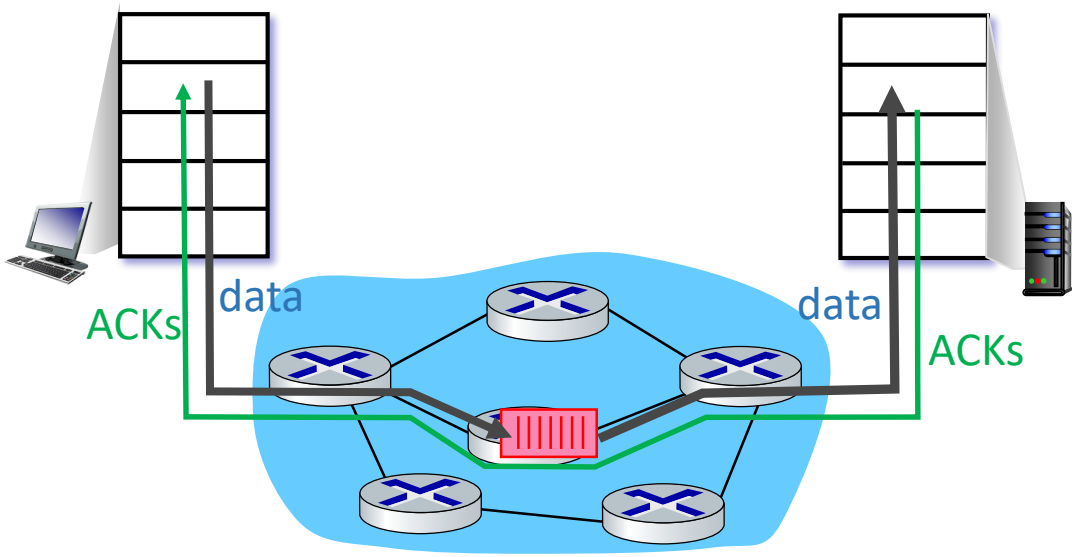




Posibles metodologías ante congestión

End-end congestion control:

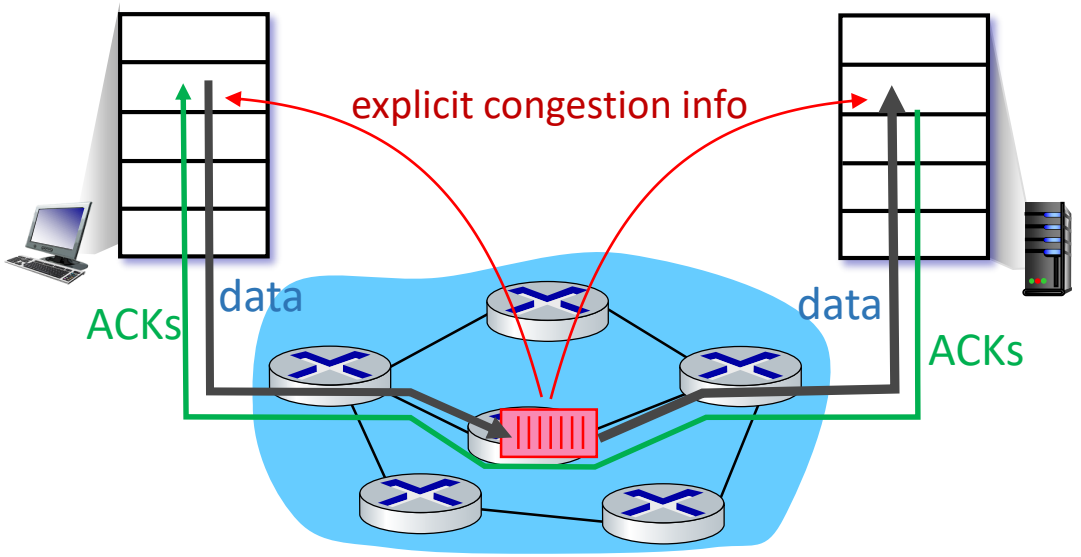
- Sin realimentación explícita de la red
- congestión *inferida* desde las pérdidas observadas, o retardos leídos
  - (TCP original)



Posibles metodologías ante congestión

Network-assisted congestion control:

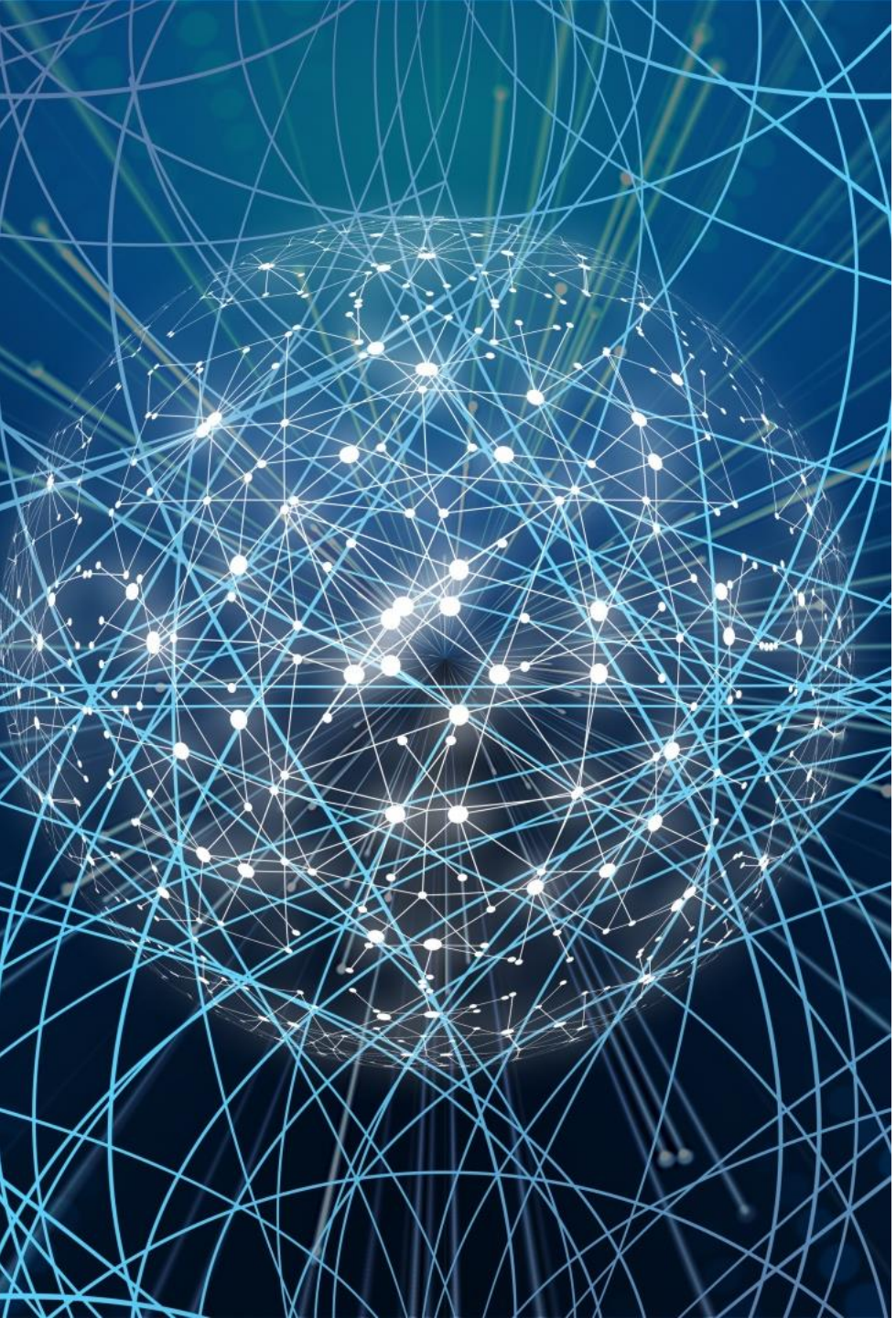
- Los enrutadores brindan *retroalimentación directa* a los hosts de envío / recepción con flujos que pasan a través del enrutador congestionado
- puede indicar el nivel de congestión o establecer explícitamente la tasa de envío



- Protocolos TCP ECN, ATM, DECbit

HANDS-ON

- Python EWMA_EstimatedRTT
- Incluir la opción de leer un archivo CSV con 200 muestras
- Integrar el cálculo y plot de:
  - DevRTT
  - Timeout



Capítulo 3: Capa de transporte

Contexto: (Próxima sesión)

Servicios de la capa de transporte

Multiplexación y demultiplexación

Transporte sin conexión: UDP

Principios de la transferencia de datos confiable

Transporte orientado a la conexión: TCP

- estructura del segmento
- transferencia de datos confiable
- control de flujo
- gestión de conexión

Principios del control de la congestión

Spoiler alert: Proyecto de curso Parte 1

Control de congestión TCP

Evolución de la funcionalidad de la capa de transporte