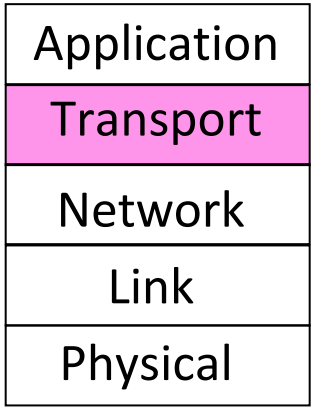
Capítulo 3

Capa de Transporte

Control de Congestión



Control de Congestión

- **Si un emisor manda a un receptor más información que la capacidad de carga de la subred:**
  - la subred se **congestionará** pues será incapaz de entregar los segmentos a la velocidad con que llegan.

Control de Congestión

- **Problema***:* Se necesita un mecanismo de control de congestión **basado en la capacidad de carga de la subred**.
  - El mismo debe aplicarse al emisor.

Control de Congestión en TCP

- **Para controlar la congestión:**
  - En TCP algunos hosts **disminuirán la tasa de datos**.
- **Para llevar la cuenta de cuántos datos un host puede enviar por la red:**
  - TCP maneja una **ventana para la congestión (VC) -** cuyo tamaño es el número de bytes que el emisor puede tener en la red en un momento dado.
- En TCP el host tiene una forma de **detectar congestión**.
- **En TCP cuando un host detecta congestión:**
  - El host **ajusta** el tamaño de la VC.

Control de Congestión en TCP

- **Propósito**: estudiar cómo hace TCP para **detectar congestión**.
- **La expiración de un temporizador causada por un paquete perdido se puede deber a:**
  1. ruido en la línea de transmisión o
  2. el descarte de paquetes en el enrutador congestionado.

Control de Congestión en TCP

- Hoy la pérdida de paquetes por errores de transmisión es rara debido a que las troncales de larga distancia son de fibra óptica.
  - Luego, la mayoría de las expiraciones de tiempo en Internet se deben a la congestión.
- **Solucíón de TCP:** Todos los algoritmos de congestión de TCP suponen que las expiraciones de tiempo son causados por congestión.

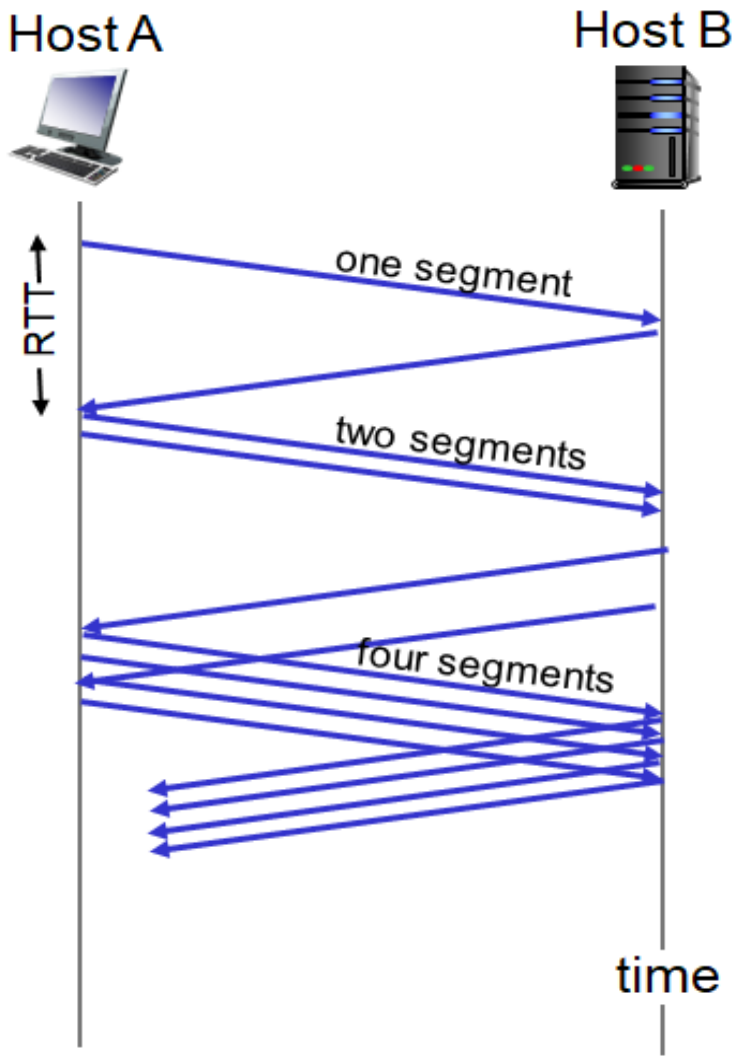
Control de Congestión en TCP

- **Problema: ¿Cómo calcular un tamaño para la ventana de congestión (VC)?**
- **Idea:** probar con un mínimo de datos e ir duplicando gradualmente hasta que no se pueda más**.**
- **Un algoritmo basado en esta idea se llama arranque lento.**

Control de Congestión en TCP

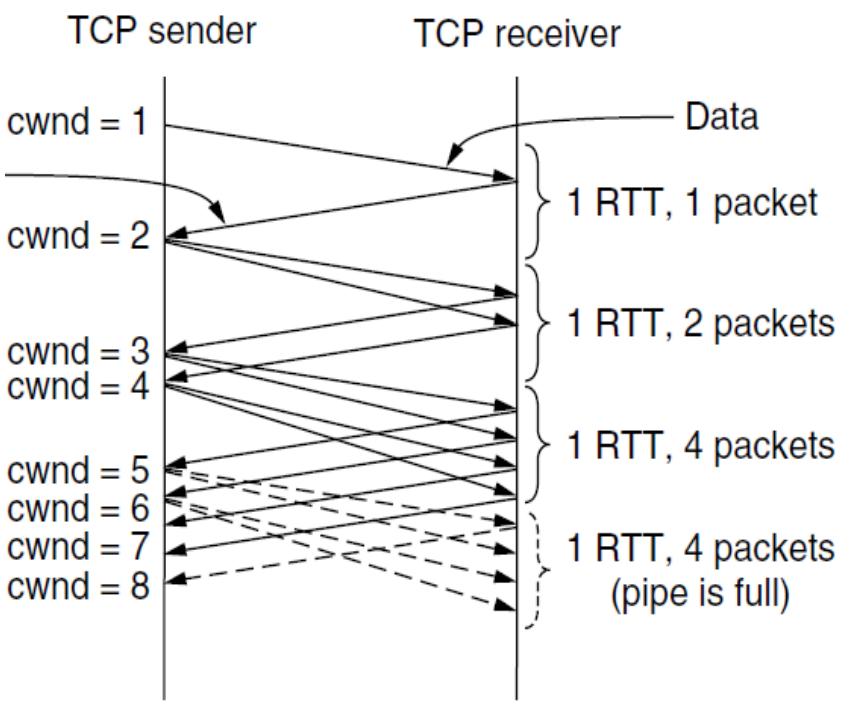
Algoritmo de arranque lento (Jacobson 1988).

- El emisor asigna a la VC el segmento de tamaño máximo (STM) usado por la conexión; entonces envía 1 STM.
  - Emisor y receptor se ponen de acuerdo en el tamaño del STM.
- Si se recibe el ack de este segmento antes que expire el temporizador, el emisor agrega el equivalente en bytes de un segmento a la VC para hacerla de 2 STM y envía dos segmentos.



Control de Congestión en TCP

- Cuando la VC es de *n* segmentos, si de todos los *n* se reciben acks a tiempo, se aumenta la VC en la cuenta de bytes correspondiente a *n* segmentos.
- La VC sigue creciendo exponencialmente hasta expiración temporizador (timeout) o **alcanzar el tamaño de la ventana receptora**.
- Si ocurre timeout se recorta la VC a tamaño VC/2, o sea no se enviarán ráfagas de segmentos mayores a VC/2.
- Esta es la solución de la edición ante penúltima del libro de Tanembaum.



Control de Congestión en TCP

- **Ejercicio**: Considerar el efecto de usar arranque lento en una línea con un RTT de 10-msec y sin congestión. La ventana del receptor es de 24 KB y el tamaño de segmento máximo es de 2 KB. ¿Cuánto tiempo lleva antes de que una ventana completa pueda ser enviada?

Control de Congestión en TCP

- **Críticas al algoritmo de arranque lento:**
  - **Crítica 1:** Recortar la ventana de congestión a la mitad porque hubo una expiración de temporizador y quedarse ahí, puede ser demasiado,
    - porque puede ser que la red tenga una capacidad mayor a esa mitad y así se desaprovecharía esa capacidad de la subred.

Control de Congestión en TCP

- **Crítica 2**: con la retransmisión disparada por expiración de temporizador el tiempo de espera puede ser relativamente grande.
  - Cuando se pierde paquete el emisor se demora en reenviar el paquete perdido.
- **Problema: ¿Cómo puede reconocer rápidamente el emisor que uno de sus paquetes se perdió?**

Control de Congestión en TCP

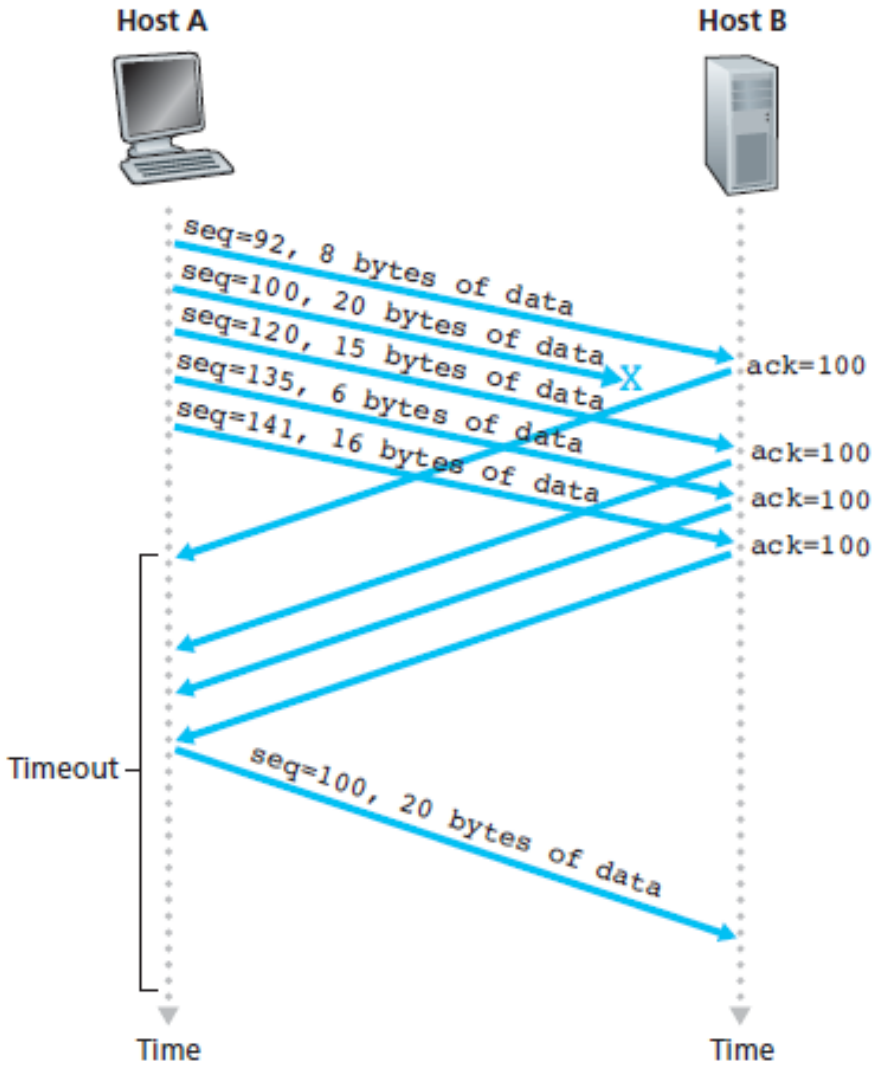
- **Asumimos:** cada paquete que llega al receptor dispara un paquete ack.
- **Cuando se pierde un segmento y otros segmentos luego del segmento perdido llegan al receptor:**
  - El receptor genera acks que confirman lo mismo (i.e. siguiente byte esperado).
  - Se llaman **acks duplicados**.
- **¿Qué significa que el emisor recibió un ack duplicado?**
  - Es probable que llegó otro segmento al receptor y el segmento perdido no dio señales de vida.

Control de Congestión en TCP

- **Significado de recibir acks duplicados:**
  - Como segmentos pueden tomar distintos caminos, pueden llegar fuera de orden y esto va a disparar acks duplicados incluso cuando no se ha perdido ningún segmento.
  - Si se pierde un segmento, habrá probablemente varios ack duplicados.

Control de Congestión en TCP

- **Solución:** TCP asume que 3 acks duplicados implican que el paquete se perdió.
  - Luego ese paquete puede retransmitirse inmediatamente y antes de que expire el temporizador.
  - Esta heurística se llama **retransmisión rápida**.



Control de Congestión en TCP

- Ahora vemos un algoritmo de control de congestión que supera las 2 críticas a arranque lento.
- **Solución 2**: **Algoritmo de control de congestión de Internet (o TCP Tahoe):**
  - Usa un **umbral** además de las ventanas de recepción y congestión.
  - Al ocurrir una expiración del temporizador o detectarse 3 acks duplicados, se fija el umbral en la mitad de la ventana de congestión actual, y la ventana de congestión se restablece a un segmento máximo.

Control de Congestión en TCP

- Luego se usa el **arranque lento** para determinar lo que puede manejar la red, excepto que el crecimiento exponencial termina al alcanzar el umbral.
- A partir del punto en el que se alcanza el umbral las transmisiones exitosas aumentan linealmente la ventana de congestión (en un segmento máximo por ráfaga).
- Recomenzar con una ventana de congestión de un paquete toma un RTT (para todos los datos previamente transmitidos que dejen la red y para ser confirmados, incluyendo el paquete retransmitido).

Control de Congestión en TCP

- Si no ocurren más expiraciones de temporizador/3 acks duplicados, la ventana de congestión continuará creciendo hasta el tamaño de la ventana del receptor.
  - En ese punto dejará de crecer y permanecerá constante mientras no ocurran más expiraciones de temporizador y la ventana del receptor no cambie de tamaño.

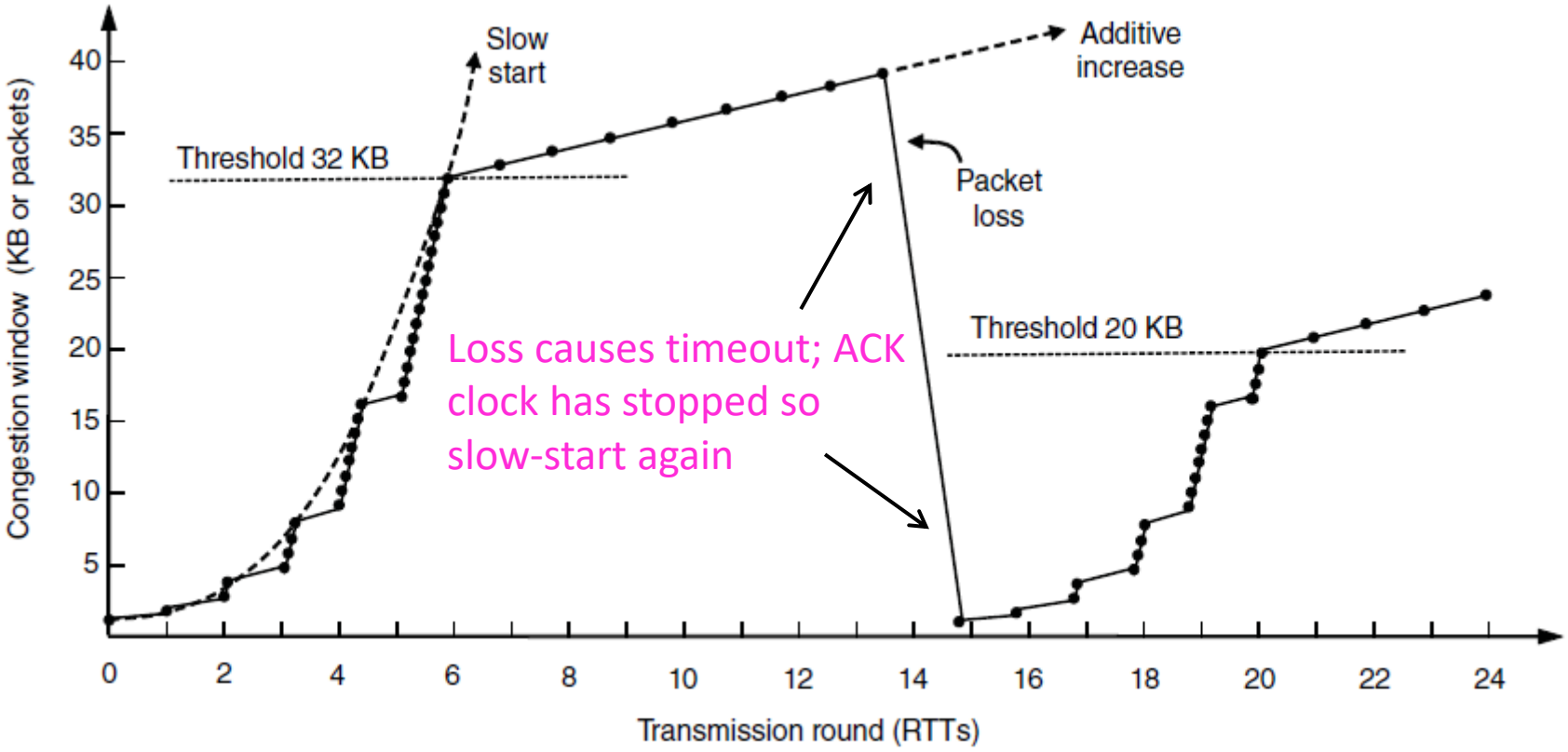
- **Crítica al algoritmo TCP Tahoe:**
- Comenzar con arranque lento cada vez que se pierde un paquete puede ser demasiado.
- **¿Qué se puede hacer para resolver este problema?**

Control de Congestión en TCP

TCP Tahoe:

- **Invariante:** tamaño ventana congestión ≤ tamaño ventana receptor

1. Se usa arranque lento hasta alcanzar el umbral
2. Luego vienen incrementos aditivos hasta alcanzar timeout o 3 acks duplicados
3. Luego el umbral se fija a la mitad del tamaño de la ventana de congestión
4. Goto 1



Control de Congestión en TCP

- **Solución**: **Algoritmo de TCP Reno** (1990).
  - **Idea**: Evitar arranque lento (excepto cuando la conexión es comenzada) cuando expira el temporizador de re-envíos.
  - **Funcionamiento**:
    1. Luego de iniciada la conexión se comienza con arranque lento.
    2. A continuación la ventana de congestión crece linealmente hasta que se detecta una pérdida de paquete.
       - Se cuentan acks duplicados
       - Se considera pérdida de paquete 3 acks duplicados

Control de Congestión en TCP

3. El paquete perdido es retransmitido (usando retransmisión rápida).
4. **Recuperación rápida**:
   - Se manda un paquete por cada ack duplicado recibido.
   - Un RTT luego de la retransmisión rápida el paquete perdido es confirmado.
   - La recuperación rápida termina con esa confirmación de recepción.
5. Luego de recibir el nuevo ack:
   - la ventana de congestión de una conexión se achica a la mitad de lo que era cuando se encontraron 3 duplicados (**decrecimiento multiplicativo**).
   - El conteo de ack duplicados se pone en 0.

Control de Congestión en TCP

6. Luego la ventana de congestión va incrementando de a un segmento por cada RTT (**crecimiento aditivo**).
7. Este comportamiento continua indefinidamente.

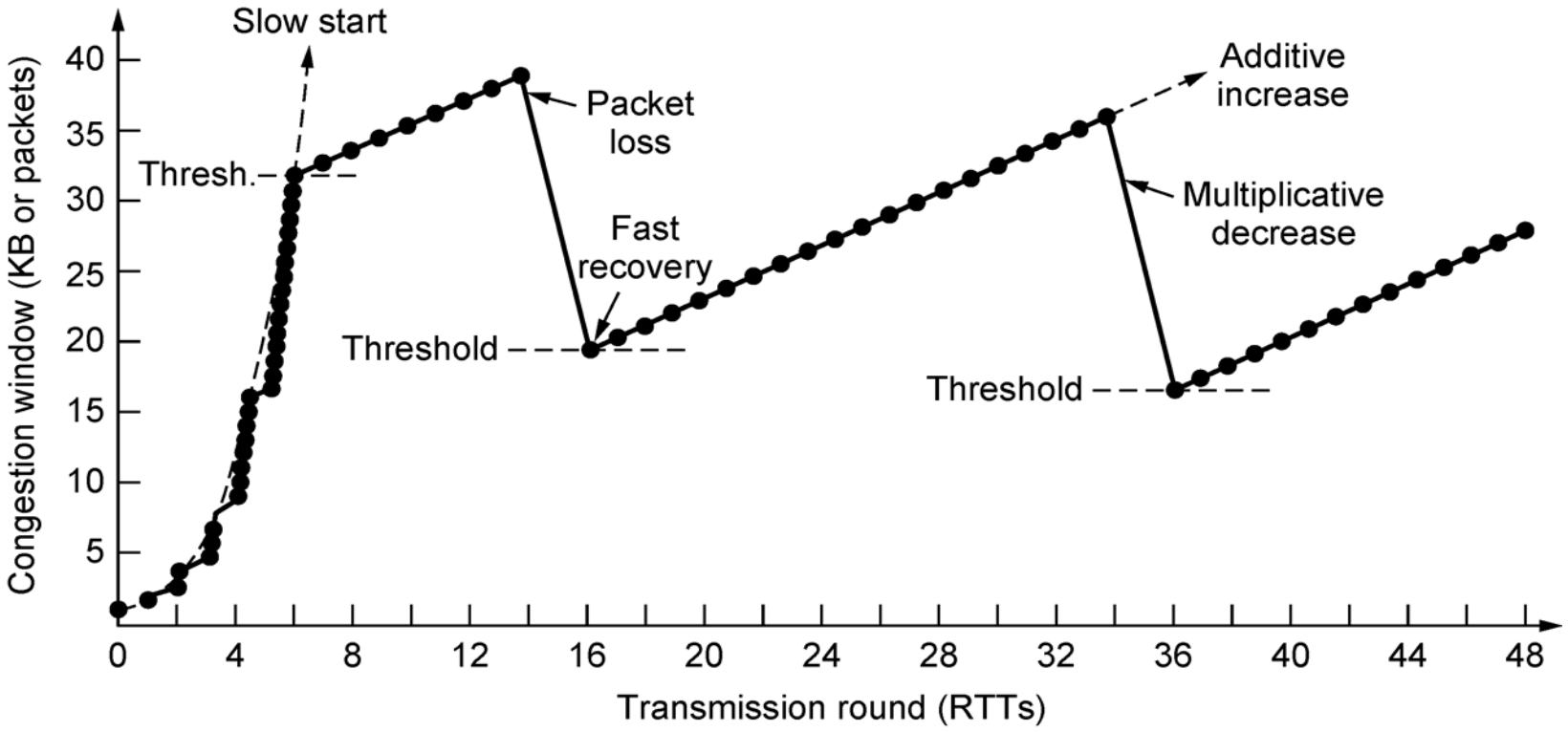
- Luego se hicieron ajustes menores a TCP Reno que no veremos.

Control de Congestión en TCP

TCP Reno:

- **Invariante:** tamaño ventana congestión ≤ tamaño ventana receptor

1. Luego de iniciada la conexión viene arranque lento hasta alcanzar umbral.
2. Luego vienen incrementos aditivos hasta 3 ack duplicados.
3. Luego viene recuperación rápida.
4. Luego se reduce ventana de congestión a la mitad
5. Goto 2



Fast recovery and the sawtooth pattern of TCP Reno.

Control de Congestión en TCP

- **Ejercicio**: Asumir que se usa algoritmo TCP Tahoe, la ventana de congestión es fijada a 36 KiB y luego ocurre un timeout; luego de esto el algoritmo hace lo que tiene que hacer y la ventana de congestión llega hasta los 24 KB con éxito sin que ocurran nuevos timeouts. Asumir que el segmento máximo usado por la conexión es de 1KB de tamaño. Responder:
  1. ¿Si tuviera que hacer un diagrama cartesiano del comportamiento del algoritmo TCP Talhoe qué representa cada uno de los ejes cartesianos?
  2. Hacer un diagrama cartesiano mostrando el comportamiento del algoritmo TCP Talhoe desde que ocurre el timeout mencionado (luego de los 36 KB) hasta que la ventana de congestión llega a 24 KB.