Copyright

Contenidos

- Servicios y protocolos de la capa de transporte
- Multiplexado y demultiplexado de segmentos
- Transporte no orientado a la conexión (**UDP**)
- Teoría de transporte confiable de datos
- Transporte orientado a la conexión (**TCP**)
- Establecimiento y cierre de conexiones
- Teoría de control de congestión
- Control de congestión en **TCP**

Control de congestión

- La congestión se produce cuando los nodos envían tal cantidad de información que el núcleo de la red no alcanza a procesar
  - No confundir con el control de flujo
- ¿Cómo se manifiesta la congestión en la red?
  - Por la pérdida de paquetes producto de la saturación del almacenamiento intermedio de los routers
  - Por el incremento en los retardos producto del aumento del tiempo de encolado en los routers

Control de congestión

- Se trata de un problema igual de desafiante que el trasporte confiable de información
- En la década del '80, antes de que sea tenido en cuenta, el núcleo de internet colapsó
  - Los administradores reseteaban el **HW** y la red volvía a funcionar sólo por un par de horas
  - ¿Se entiende la magnitud del problema?
  - ¡¡¡Reseteaban internet!!!

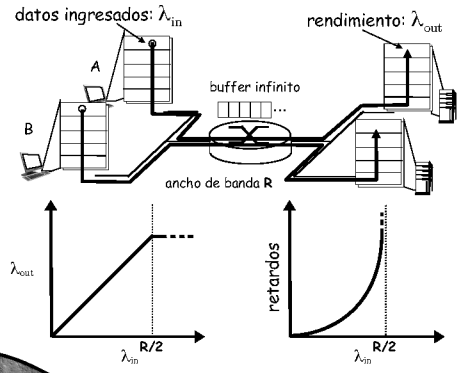
Control de congestión

- Antes de abordar cómo evitar la congestión nos concentraremos en entender cuáles son las causas de la congestión y cuáles son sus consecuencias
- La idea es partir de un escenario simplificado, más directo de analizar, para luego ir considerando situaciones más realistas
  - Esto es, seguiremos un acercamiento análogo al adoptado para presentar los desafíos asociados a la transmisión confiable de datos

Congestión con buffer infinito

- Como primer escenario consideremos la siguiente situación:
  - Dos emisores y dos receptores
  - Un único router que los comunica
  - El router cuenta con un almacenamiento intermedio infinito
- Al contar con un buffer infinito, los paquetes no se pierden, por lo que tampoco se debe retransmitir paquete alguno

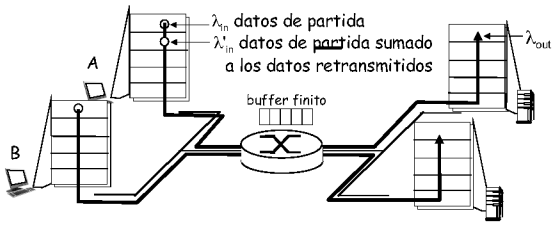
Congestión con buffer infinito



Congestión con buffer finito

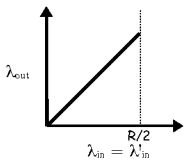
- Consideremos ahora un escenario análogo al anterior, pero con el router contando con una cantidad finita de almacenamiento intermedio
  - La eventual pérdida de paquetes ahora causa la retransmisión de los mismo
  - Idealmente queremos que $\lambda_{in} = \lambda_{out}$
  - Una retransmisión “perfecta” es producto sólo de las pérdidas de paquetes
  - Por otra parte, las retransmisiones de paquetes demorados incrementan la tasa real λ'_{in}

Congestión con buffer finito



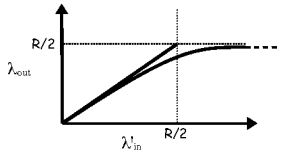
Costos de la congestión

- Primer escenario: el emisor sólo envía al tener la certeza de que hay lugar en el buffer
  - No se producirán pérdidas de paquetes
  - Por ende, $\lambda_{in} = \lambda'_{in}$



Costos de la congestión

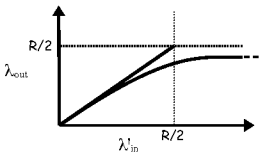
- Segundo escenario: el emisor sólo reenvía al tener la certeza de que se produjo una pérdida
  - Ocasionalmente se puede producir una pérdida al saturarse el buffer del router
  - El emisor sólo reenvía los paquetes que conoce que se han perdido (esto es algo imposible de determinar a ciencia cierta)



incluso al producirse retransmisiones logra aproximarse asintóticamente a R/2

Costos de la congestión

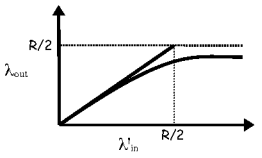
- Tercer escenario: al igual que antes, pero ahora se contempla que el emisor envíe duplicados
  - Ocasionalmente se puede producir una pérdida al saturarse el buffer del router
  - El disparo anticipado de un temporizador provoca la retransmisión innecesaria de paquetes



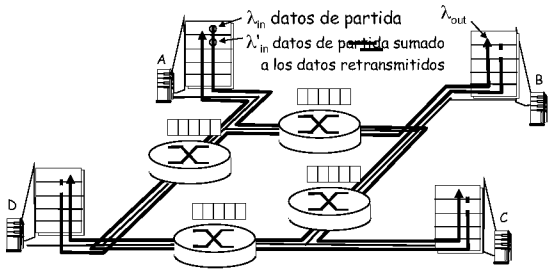
parte del ancho de banda disponible es desperdiciado en las retransmisiones

Costos de la congestión

- La congestión genera dos costos ocultos:
  - Para un mismo nivel de datos recibidos se debe enviar un mayor nivel de datos debido a las retransmisiones
  - Las retransmisiones producto de los retrasos malgastan ancho de banda en los enlaces

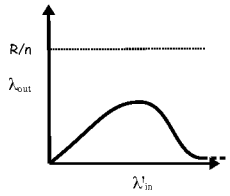


Situación más realista



Situación más realista

- Consideremos ahora un escenario con múltiples emisores y receptores y con múltiples enlaces con almacenamiento intermedio finito
  - Al producirse una pérdida todo el ancho de banda invertido en mover ese paquete a través de los routers previos termina siendo malgastado



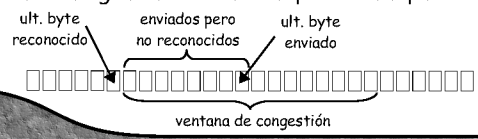
Detección de la congestión

- Control de congestión punta-a-punta (**TCP**)
  - La red no brinda información acerca de que sucede en el núcleo de la red
  - La congestión se detecta al observar pérdidas de paquetes en el origen o el destino
- Control de congestión asistida por la red (**ATM**)
  - Los routers informan a las computadoras en la frontera de la red acerca de las congestiones
  - Usando banderas (por caso, el campo **ECN** de **IP**), o imponiendo la cadencia de envío a ser usada

Control de congestión en TCP

- A diferencia de **ATM**, **TCP** adopta un esquema de control de congestión punta-a-punta
  - Por ende no recibe asistencia del núcleo de la red
  - El emisor limita su tasa de transmisión respetando la siguiente relación:

    `[ultbyteenv - ultbyterec] < ventcong`
  - El tamaño de la ventana es dinámico, depende del nivel de congestión en la red percibido por el emisor

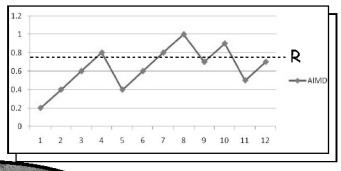


Control de congestión en TCP

- ¿Cómo hace el emisor para percibir el nivel de congestión en la red?
  - Tomando nota de los eventos de pérdida
  - Se produce un evento de pérdida toda vez se vence el tiempo de espera **TCP** o bien cuando se reciben cuatro **ACK** para el mismo segmento **TCP**
  - El emisor responde ante un evento de pérdida reduciendo su ventana de congestión
- Se han implementado diversas políticas que mejoran la respuesta de este mecanismo

TCP AIMD

- **TCP** adopta la política **AIMD** (Additive Increase Multiplicative Decrease) para controlar el tamaño de la ventana de congestión
  - El emisor incrementa de manera lineal su ventana de congestión a cada **RTT**, pero ante un evento de pérdida se reduce exponencialmente a la mitad

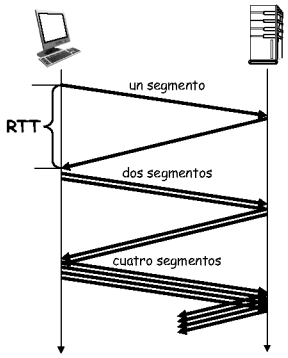


por prueba y error se intenta determinar el ancho de banda a disposición

TCP slow start

- Al comenzar, la política **AIMD** puede tomar un tiempo excesivo en alcanzar el régimen óptimo
- Consideremos el siguiente ejemplo:
  - **RTT** = 200ms y **MSS** = 500 bytes
  - Inicialmente **ventcong** = 1, pero **MSS**/**RTT** = 20 Kbps
  - Si se dispone de un mayor ancho de banda, **TCP** va a tardar bastante tiempo hasta hacer uso del mismo
  - La política slow start indica que al comenzar se debe agrandar exponencialmente la ventana de congestión hasta que se detecte el primer evento de pérdida

TCP slow start



ventcong se duplica al pasar un **RTT** (es decir, cada vez que recibe un **ACK**)

Ajuste fino

- **TCP** implementa un ajuste sutil en función de cómo es que se detectó la congestión
- Si se vence el tiempo de espera **TCP**:
  - **ventcong** se reduce a 1
  - de ahí crece exponencialmente hasta alcanzar un determinado umbral para luego crecer linealmente
- Ante tres **ACK** duplicados para un segmento:
  - **ventcong** se reduce a la mitad
  - de ahí crece linealmente como siempre

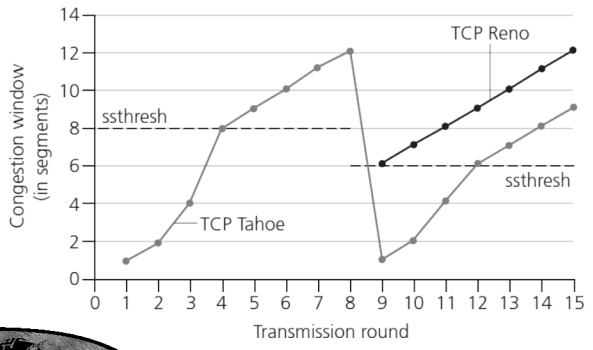
Ajuste del umbral

- ¿Cómo determinar el valor más apropiado para el umbral que altera el ritmo de crecimiento de la ventana de congestión?
  - El umbral tiene que depender del nivel de congestión actual en el núcleo de la red
  - Un posibilidad es usar el valor de **ventcong** antes del evento de pérdida como guía
  - A tal efecto, cuando la nueva ventana de congestión alcance la mitad del valor que tenía antes se deberá cambiar de un crecimiento exponencial a uno lineal

TCP Reno vs. TCP Tahoe

- Los distintos ajustes finos al mecanismo de gestión de la congestión dieron a lugar a diferentes implementaciones
  - **TCP** Reno implementa el ajuste fino antes visto
  - En contraste, **TCP** Tahoe dictamina que en ambos casos la ventana de congestión debe reducirse a 1
- Estos llamativos nombres de las versiones de **TCP** se corresponden con las versiones **BSD** en las que se implementaron por primera vez

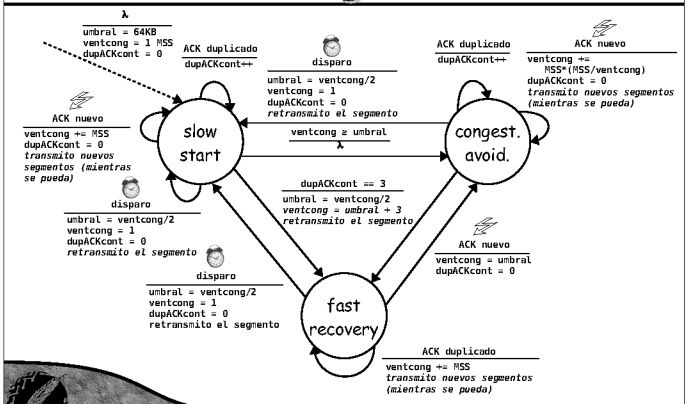
TCP Reno vs. TCP Tahoe



En síntesis

- Mientras **ventcong <= umbral**, el emisor está en la fase “slow start”, por lo que la ventana debe crecer exponencialmente
- Cuando **ventcong > umbral**, el emisor entra en la fase de “evasión de congestión”, por lo que ahora la ventana crece linealmente.
- Si se reciben tres **ACK** repetidos, tanto **umbral** como **ventcong** se ajustan a **ventcong/2**
- Al vencerse el tiempo de espera **TCP**, se hace **umbral = ventcong/2** y luego **ventcong = 1**

Control de congestión en TCP



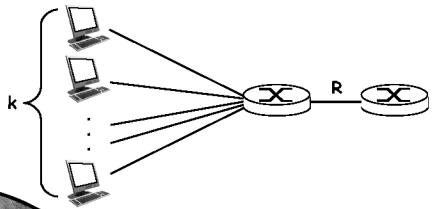
Desempeño de TCP

- ¿Cuál será el desempeño de **TCP** en función de **ventcong** y del **RTT** actual?
- Sea **W** el tamaño de la ventana en el momento que se produce una pérdida
  - Cuando **ventcong** es **W** el desempeño es **W / RTT**
  - Acto seguido, el tamaño de ventana se restringe a **W / 2**, por lo que el desempeño cae **W / 2RTT**
  - Por lo tanto, el desempeño promedio es $\frac{3}{4}$**W / RTT**

¡El **RTT** está fuertemente vinculado al desempeño!

Equidad TCP

- Idealmente sería interesante contar con un protocolo que se comporte de manera justa
  - Esto es, si se dispone de un enlace de capacidad **R** el cual es compartido entre **k** conexiones, cada una debería recibir un ancho de banda de **R / k**



Equidad TCP

- Imaginemos que dos sesiones **TCP** compiten por hacer uso de la totalidad del ancho de banda de un determinado enlace
  - El incremento lineal de la ventana de congestión hace que ambas sesiones vayan consumiendo partes equitativas de ese ancho de banda
  - El retroceso exponencial que se produce ante los eventos de pérdida también se dispara en ambas sesiones
  - Por ende, se converge hacia un uso equitativo del enlace entre las sesiones

Equidad TCP vs. UDP

- Las aplicaciones multimediales suelen no hacer uso de **TCP**
  - Precisamente, el control de congestión lejos de ser una virtud sería un inconveniente
  - No es deseable que la ventana de transmisión ni la de congestión detengan el flujo de datos
- Es por esta razón que las aplicaciones multimediales suelen optar por **UDP**
  - Envían datos a un ritmo constante, tolerando la eventual pérdida de algún paquete

Conexiones TCP paraleralas

- ¿Cómo interactúa la equidad **TCP** con el establecimiento de conexiones en paralelo?
  - Nada impide que una misma aplicación abra una cantidad de conexiones en paralelo
- Por caso, supongamos que un enlace de capacidad **R** está soportando 9 conexiones
  - Si una cierta aplicación abre una nueva conexión, recibirá apenas **R / 10** del canal
  - Si más tarde otra aplicación abre 10 nuevas conexiones, recibirá en cambio **R / 2** del canal

¿Preguntas?