

Redes de Computadoras 2021

## **TP1 Teórico:**

# **Algoritmos de control de congestión de TCP**

---

Alumnos:

Nehemias Mercau Nieves ([nehemias.mercau@mi.unc.edu.ar](mailto:nehemias.mercau@mi.unc.edu.ar))

Tomás Martín ([tomas.martin@mi.unc.edu.ar](mailto:tomas.martin@mi.unc.edu.ar))

25 de Marzo, 2021

## Objetivos

- Conocer sobre los algoritmos de control de congestión de TCP, sus ventajas y desventajas.
- Comparación entre control de congestión Las Vegas y Reno

## Consigna

Investigar los algoritmos de control de congestión de TCP y exponer las ventajas y desventajas de cada uno y cuales están implementados en los principales sistemas operativos usados hoy en día.

## Desarrollo

El protocolo TCP (Transmission Control Protocol) es el encargado de transportar la mayor parte del tráfico de internet, es por ello que el rendimiento de internet depende, en gran medida, de cómo funciona TCP. Las características de rendimiento de una versión particular de TCP se definen por el algoritmo de control de congestión que implementa.

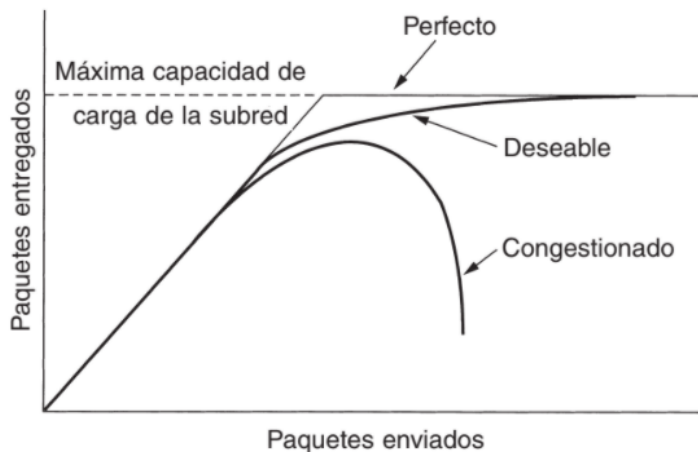
TCP ha sido proyectado para proporcionar un flujo fiable de bytes, desde la fuente hasta el destino, sobre una red no fiable. Mientras que el protocolo IP, inferior al protocolo TCP, proporciona un servicio de tipo *best-effort*, es decir, “*hace lo mejor*” para entregar los datagramas entre dos servidores en comunicación, pero **no logra garantizar** que los paquetes alcancen efectivamente un destino y que se respete el orden de entrega y la integridad de los datos.

Por tanto, el protocolo TCP beneficiándose del servicio del IP, proporciona un servicio fiable de transferencia de datos entre los procesos. Sus principales funciones son:

- Reconstruir el flujo originalmente transmitido en el caso de fenómenos de pérdida, duplicación o entrega fuera de secuencia de las unidades informativas y de congestión.
- Adoptar técnicas para el control de flujo y congestión.

Con el control de flujo, el TCP previene que el emisor de una comunicación envíe mensajes superiores a la capacidad actual del buffer del destino, mientras que con el control de congestión, limita la cantidad de datos introducidos en la red de manera que se evita sobrecargar a los encaminadores y la consiguiente pérdida de paquetes.

Una red se dice congestionada cuando hay demasiados paquetes que se disputan el mismo enlace causando la saturación de los buffer de los routers y las consiguientes pérdidas de paquetes.



Para resolver el problema del colapso de congestión, se han propuesto varias soluciones, donde todas ellas comparten la misma idea: la introducción de un mecanismo que limite la tasa de envío. Para ello se introdujo el concepto de ventana de congestión, cuyo propósito es estimar la cantidad de datos que la red puede aceptar para su entrega sin que se produzca congestión.

#### Control de congestión

Los algoritmos de control de congestión, tratan de determinar dinámicamente el ancho de banda disponible y la latencia de la red. Luego en función de su análisis se modificará la tasa de envío del emisor TCP para evitar el colapso de la subred.

El objetivo del control de congestión es que el emisor TCP debe intentar transmitir a la máxima velocidad posible sin congestionar la red para así perder el menor número de paquetes posibles y que no haya que retransmitir. Se debe encontrar la tasa justo por debajo del nivel de congestión.

Cada emisor TCP tomará las decisiones con la información que recibe de los ACK (Acknowledgement). Un mensaje ACK es un mensaje que el destino de la comunicación envía al origen de esta para confirmar la recepción de un mensaje. Si recibe un ACK es porque la red no está congestionada por lo tanto puede aumentar la tasa de envío. Por el contrario si tiene un segmento perdido se debe a la congestión de la red por lo tanto se debe disminuir la tasa de envío.

Va probando el ancho de banda disponible, aumentando la velocidad hasta perder paquetes donde luego disminuye la velocidad. Esto nos genera una velocidad con forma de diente de sierra.

El emisor podrá disminuir o aumentar su velocidad de emisión mediante la ventana de congestión. La velocidad viene determinada por los paquetes que puede emitir (ventana de congestión) por cada RTT (Round Trip Time). El RTT es la cantidad de tiempo que le toma a una señal ser enviada sumado al tiempo en que tarda en recibir el ACK del receptor.

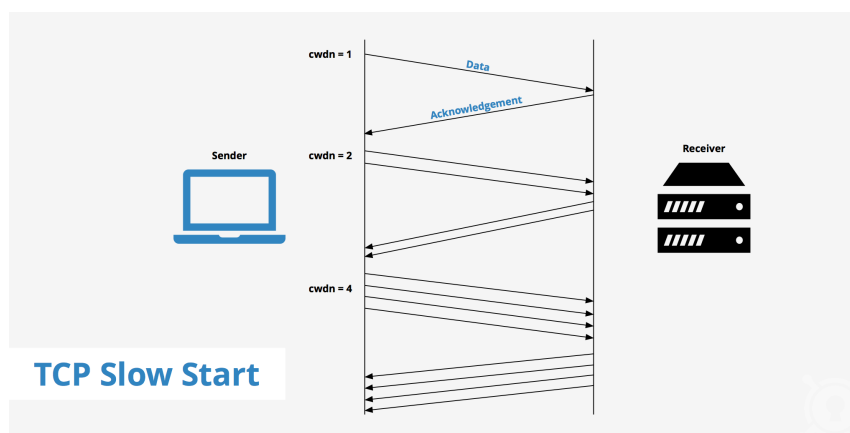
Cuando TCP no tenía control de congestión arrancaba superando la capacidad del canal lo que le llevaba a tener que retransmitir paquetes perdidos y así se la pasaba haciendo lo mismo continuamente perdiendo mucho tiempo retransmitiendo paquetes.


Detecta una pérdida de paquetes cuando se acaba el temporizador y no hay respuesta del receptor, o porque recibe tres ACK duplicados, en ese momento reduce la velocidad. Aumenta la velocidad cuando recibe ACK indicando que no hay congestión. Cuánto se reduce y cuánto se aumenta esta velocidad dependerá de las distintas versiones de TCP.

Se aumenta la ventana de congestión, y con ella la velocidad, en dos fases:

- *Slow Start* (fase de arranque lento)
- *Congestion Avoidance* (fase de evitación de la congestión)

En la fase *Slow Start* o de arranque lento, se intenta aumentar la velocidad muy rápidamente para poder encontrar el punto de saturación de la red. La velocidad se irá aumentando de **forma exponencial**, incrementando en un segmento la ventana de congestión por cada ACK recibido correctamente. **De esta forma se duplica la ventana de congestión cada RTT.**





Empieza con una ventana de un segmento, cuando recibe el ACK, aumenta en uno, enviando dos segmentos. Luego cuando recibe cada ACK de estos dos segmentos, aumenta la ventana en uno más por cada ACK recibido. Así sucesivamente hasta llegar a velocidades muy altas que saturan la red y comience a perder paquetes. Este mecanismo se puede observar en la figura de arriba.

Cuando detecte una pérdida de paquetes por fin de temporización o por triple ACK duplicado **establece un valor de umbral a la mitad** de la ventana de congestión, para evitar que al llegar a esa velocidad vuelva a saturar la red.

Luego, la segunda vez que arranque en arranque lento, se va a ir aumentando la velocidad de forma exponencial hasta alcanzar este umbral.

Cuando alcanza el umbral, **entra a la fase *Congestion Avoidance* o de evitación de la congestión**, donde aumenta la velocidad de forma lineal para acercarse al límite de forma más gradual.

Esta nueva fase de evitación de la congestión lo que intenta es acercarse a la congestión de una forma más lenta que en arranque lento. Esto se logra aumentando la ventana de congestión en un segmento por ventana ( $1/VC$ ). Es decir, si tiene una ventana de cuatro segmentos, cada ACK que reciba le aumenta la ventana en un cuarto de la misma, y cuando reciba cuatro ACK se aumenta la ventana en un segmento.

En cualquier caso, cuando detecte pérdida de paquetes vuelve a la fase de arranque lento y se disminuye la velocidad al mínimo o al umbral dependiendo del algoritmo usado.

Estos algoritmos se denominan AIMD (Additive Increase, Multiplicative Decrease):

- **Incremento Aditivo (Additive Increase)** cuando en la fase de evitación de la congestión aumenta la velocidad de forma lineal.
- **Decremento multiplicativo (Multiplicative Decrease)** cuando se detectan pérdida de paquetes, se establece el umbral y se reduce la velocidad, reduciendo la ventana de congestión.

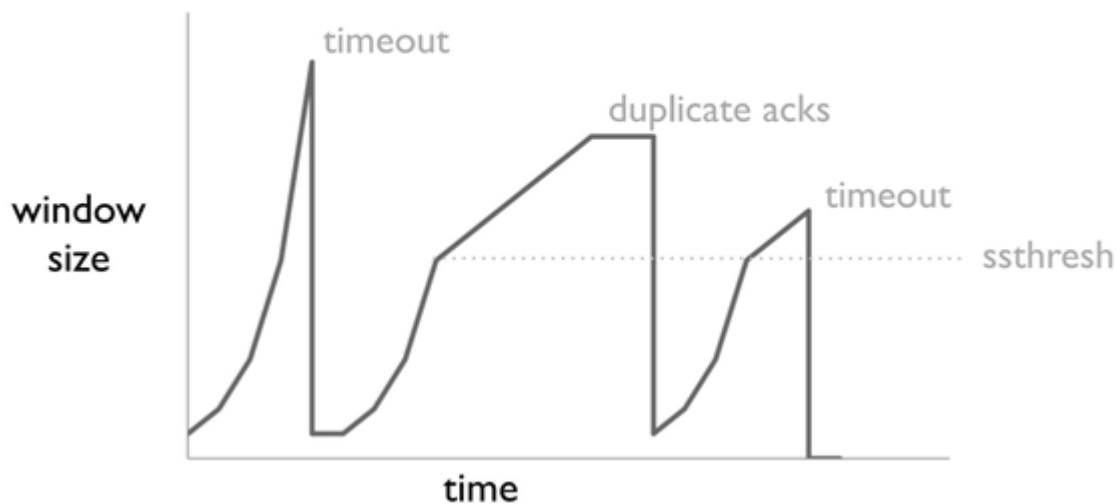
Un ACK duplicado indica el hecho de que el receptor no puede mandar un ACK relativo a un segmento llegado fuera de orden, ya que está esperando recibir de uno precedente.

El mecanismo de *Fast Retransmit* prevé, que en caso de que se reciban 3 ACK duplicados (3DUPACK) relativos al mismo segmento, se procede a la retransmisión del segmento, sin esperar a que venza el timeout remediando el problema de los largos períodos de inactividad. Este

mecanismo permite beneficios en el *throughput* (tasa de transferencia efectiva) y reduce el número de timeout.

## TCP Tahoe

# TCP Tahoe Behavior

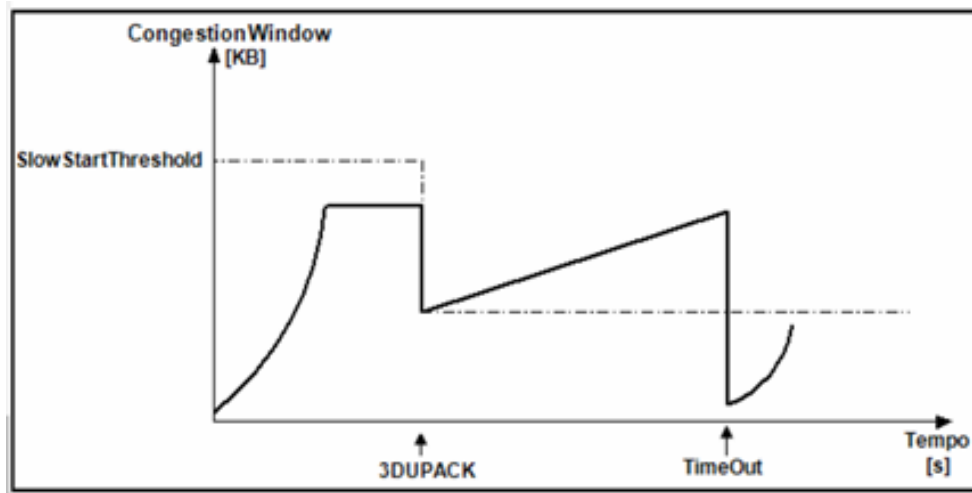


El algoritmo TCP Tahoe inicia la transmisión en arranque lento aumentando exponencialmente la velocidad. Cuando se pierden paquetes, sitúa el umbral (ssthresh) a la mitad, reduce la velocidad al mínimo y comienza de nuevo en arranque lento. Cuando llega al umbral, pasa a la fase de evitación de la congestión aumentando linealmente la velocidad. Este comportamiento se puede observar en la imagen de arriba.

Si detecta de nuevo una pérdida de paquetes, vuelve nuevamente a situar el umbral, reduce la velocidad al mínimo y comienza nuevamente en arranque lento y así sucesivamente.

## TCP Reno

TCP Reno introdujo una nueva fase a TCP Tahoe, la de recuperación rápida, para diferenciar el modo de actuar cuando la pérdida de paquetes es por fin de temporización de la pérdida de paquetes por triple ACK duplicados.



Como podemos ver en la imagen, después de haber recibido tres ACK duplicados, pone el umbral a la mitad del valor actual de la ventana de congestión, pero en vez de reducir bruscamente esta última hasta llevarla al valor de 1 segmento (velocidad mínima) la reduce a la mitad solamente, eliminando así la fase de *Slow Start*, pasando directamente a evitación de la congestión aumentando linealmente la velocidad. Esto se denomina *fast recovery*.


El motivo que lleva al control de congestión del TCP Reno a comportarse de forma distinta después de un acontecimiento de timeout y después de la recepción de tres ACK duplicados es que, en el segundo caso, aunque se haya perdido un paquete, la llegada de los 3DUPACK al remitente testimonia que la red es capaz de entregar, al menos, algún segmento, aunque otros se hayan perdido a causa de la congestión. La recepción de los tres ACK duplicados por lo tanto, **representa en todo caso un síntoma del alto nivel de congestión de la red**, pero menos grave que los timeout.

Reno se recupera antes que Tahoe de la pérdida de paquetes si ésta se ha detectado por un 3DUPACK. En Tahoe bajábamos al mínimo y en Reno bajamos la velocidad al umbral yendo a evitación de la congestión seguidamente.

Además del *Fast Recovery* y al *Fast Retransmit*, el TCP Reno también le añade al TCP Tahoe el soporte opcional por los retardados ACK. Este mecanismo incrementa la eficiencia permitiendo al receptor de no generar el segmento de ACK para todos los segmentos que le llegan, sino sólo después de una cierta cantidad de ellos (ACK acumulativo).

## TCP Vegas

Hasta ahora la ventana de congestión crecía hasta que ocurriera una pérdida de paquete. TCP Reno asume que hay congestión cuando se detectan estas pérdidas. **La idea del TCP Vegas es**



**controlar y mantener el tamaño adecuado de la ventana de manera de no llegar a la pérdida de paquetes.** El TCP Vegas administra el tamaño de la ventana observando el RTT de los paquetes que el emisor envió con anterioridad. Si el RTT crece, TCP Vegas reconoce que la red comienza a estar congestionada y reduce su ventana de congestión. Por el contrario, si los valores de RTT decrecen, el emisor determina que la red no está congestionada e incrementa el tamaño. Para evitar las oscilaciones, se define un intervalo en donde se mantiene el tamaño de la ventana.

El algoritmo depende en gran medida del cálculo preciso del valor de RTT base. Si es demasiado pequeño, el rendimiento de la conexión será menor que el ancho de banda disponible, mientras que si el valor es demasiado grande, invadirá la conexión.

Cuando Vegas interactúa con otras versiones como Reno, el rendimiento de Vegas se degrada porque Vegas reduce su velocidad de envío antes que Reno, ya que detecta la congestión de forma más rápida y, por lo tanto, proporciona un mayor ancho de banda a los flujos TCP Reno coexistentes.

TCP Vegas se ha implementado en el kernel de Linux, en FreeBSD y posiblemente también en otros sistemas operativos.

## **TCP BIC**

Denominado Binary Increase Congestion. Es una implementación de TCP con una optimización para redes de alta velocidad con una alta latencia (*long fat networks*).

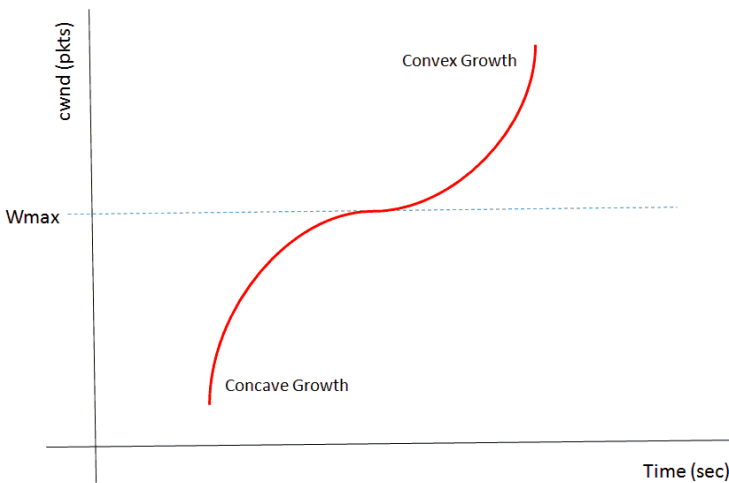
Implementa un algoritmo que intenta encontrar la ventana máxima, buscando en tres partes:

- *Binary Search Increase*
- *Additive Increase*
- *Slow Start*

## **TCP CUBIC**

Como los enlaces de comunicaciones suelen disponer de niveles de ancho de banda cada vez mayores, se puede sacar ventaja del tamaño. Si se utiliza un algoritmo que incremente el ratio de transmisión lentamente, se puede desperdiciar la gran capacidad de los enlaces. Para evitar esto, se propone que el esquema de incremento y disminución del ratio de transmisión se establezca de acuerdo a una función cúbica:





El procedimiento que sigue el algoritmo es,:

- En el momento de experimentar un evento de congestión se registrará el tamaño de la ventana para ese instante como  $W_{max}$  o el tamaño máximo de ventana.
- Se fijará el valor  $W_{max}$  como el punto de inflexión de la función cúbica que regirá el crecimiento de la ventana de congestión.
- Luego se recomenzará la transmisión con un valor de ventana menor y, de no experimentarse congestión, este valor se incrementará según la porción cóncava de la función cúbica.
- A medida que la ventana se aproxime al  $W_{max}$  los incrementos se irán ralentizando.
- Una vez alcanzado el punto de inflexión -es decir,  $W_{max}$ - se seguirá aumentando discretamente el valor de la ventana.
- Finalmente, si la red sigue sin experimentar congestión alguna, se seguirá aumentando el tamaño de la ventana según la porción convexa de la función.

El algoritmo implementa un esquema de incrementos grandes al principio, los cuales disminuyen alrededor del tamaño de ventana que causó la última congestión, para luego seguir aumentando con incrementos grandes.

### TCP Proportional Rate Reduction

Es un algoritmo que asegura que el tamaño de la ventana después de una recuperación es lo más parecido posible al umbral de *slow start*.



Los algoritmos más usados hoy en día son:

TCP CUBIC: adoptado por Microsoft por defecto en Windows 10.1709 y en Windows Server 2016 1709 update. También fue adoptado por defecto en Linux Kernel en la versión 2.6.19 (Nov 2006)

TCP BBR: Incorporado por Linux Kernels a desde la versión 4.9. También fue incorporado por Google, DropBox y Spotify ya sea para uso en producción o para experimentos.