

# *Controlo de Congestionamento*

*FEUP/MRSC/AMSR  
MPR*

## *Bibliografia*

---

- » Aula preparada com base nos seguintes documentos
  - L. Peterson, B. Davie, “Computer Networks – A Systems Approach”, Morgan Kaufmann, 200 (Sec. 6.1, 6.2, 6.3 e 6.4)
  - Acetatos do autor do livro, L. Peterson, “Congestion Control”
  - V. Jacobson, M. Karels, “Congestion Avoidance and Control”, 1988
  - “TCP Congestion Control”, RFC 2581, 1999

## *Introdução*

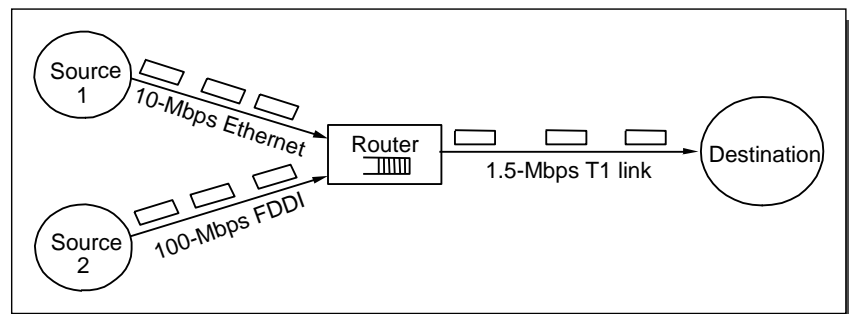
---

- ◆ Disciplinas de serviço
- ◆ Reacção ao congestionamento
- ◆ Evitar o congestionamento

## *Congestionamento*

---

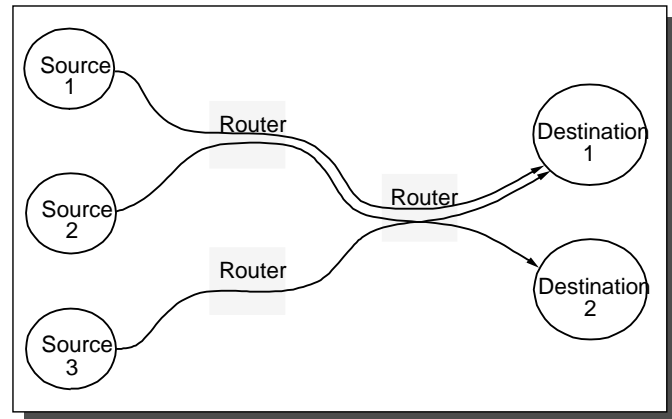
- ◆ Duas perspectivas
  - » Reserva de recursos para evitar congestionamento
  - » Controlo do congestionamento se e quando este ocorre



- ◆ Pontos de implementação
  - » Máquinas na extremidade da rede → protocolo de transporte
  - » Routers dentro da rede → disciplinas de serviço
- ◆ Modelo de transporte da rede
  - » best-effort
  - » Múltiplas classes de serviço

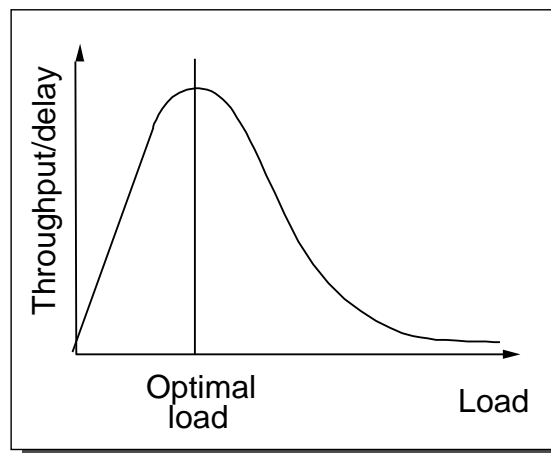
## Conceitos Base

- ◆ Fluxo (sem estabelecimento de ligação prévio)
  - » Sequência de pacotes enviados entre um par fonte-destino
  - » *Soft-state* associado, no router
- ◆ Taxonomia. Controlo de congestionamento
  - » Centrado no router / host
  - » Baseado em reserva / feedback
  - » Baseado em janela / débito



## Avaliação de Distribuição de Recursos

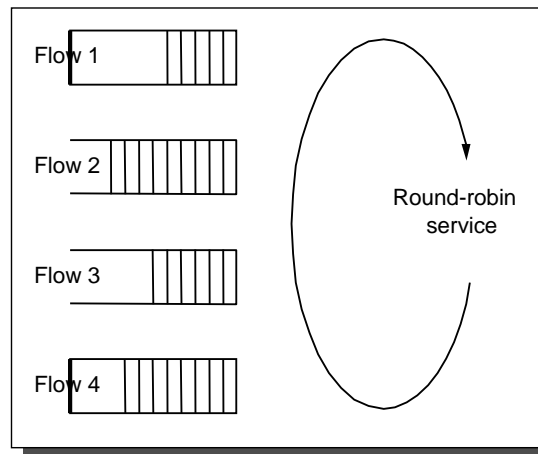
- ◆ Justiça
- ◆ Potência (débito/atraso)



## *Disciplinas de Serviço*

---

- » First-In-First-Out (FIFO) → Não há distinção de fluxos
- » Fair Queuing (FQ)
  - Tráfego separado por fluxos
  - Assegura que nenhum fluxo recebe mais do que a sua parte
  - Variação → Weighted Fair Queuing (WFQ)



## *Algoritmo FQ*

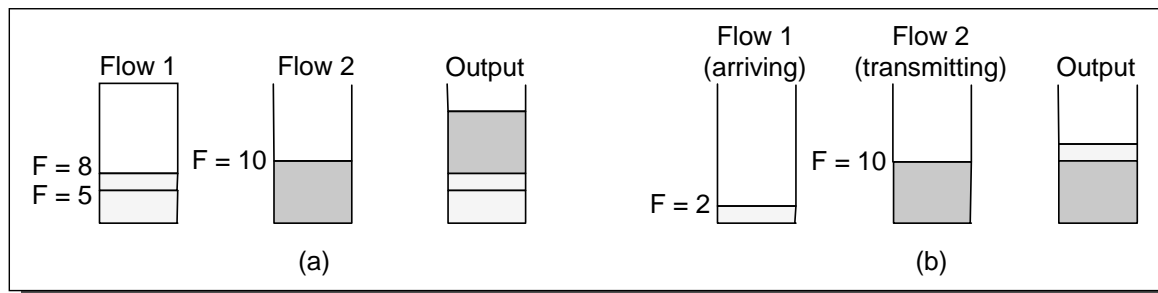
---

- ♦ Tick de relógio por cada bit transmitido
- ♦  $P_i$  → comprimento do pacote  $i$
- ♦  $S_i$  → tempo de início de transmissão do pacote  $i$
- ♦  $F_i$  → tempo de fim de transmissão do pacote  $i$
- ♦  $F_i = S_i + P_i$
- ♦ Transmissão do pacote  $i$ 
  - » Se pacote  $i-1$  já foi ou está a ser transmitido
    - Imediatamente após o último bit do pacote  $i-1$  ( $F_{i-1}$ )
  - » Se não há pacote do fluxo em transmissão
    - Transmite quando pacote  $i$  chegar ( $A_i$ )
- »  $F_i = \text{MAX} (F_{i-1}, A_i) + P_i$

## Algoritmo FQ

---

- ◆ Para múltiplos fluxos
  - » calcula  $F_i$  para pacote que chega em cada fluxo
  - » Trata todos  $F_i$  como timestamps
  - » Transmite pacote com timestamp mais baixo
- ◆ Imperfeito → pacote em transmissão não pode ser interrompido
- ◆ Exemplo



## TCP - Controlo de Congestionamento

---

- ◆ Princípio
  - » Rede best-effort (routers FIFO ou FQ)
  - » Cada fonte determina capacidade de transporte oferecida
  - » Feedback implícito
  - » **ACKs** regulam transmissão (relógio da fonte)
- ◆ Dificuldade
  - » Determinar capacidade de transmissão oferecida
  - » Ajuste a mudanças de capacidade disponível

## *Subida Aditiva/Descida Multiplicativa*

- ◆ Mudanças na capacidade de canal → ajuste na transmissão
- ◆ Nova variável de estado por ligação → **CongestionWindow**
  - » Limita a quantidade de dados em trânsito
    - **MaxWin** = **MIN(CongestionWindow, AdvertisedWindow)**
    - **EffWin** = **MaxWin - (LastByteSent - LastByteAcked)**
- ◆ Objectivo
  - » Se congestionamento diminui → aumenta **CongestionWindow**
  - » Se congestionamento aumenta → diminui **CongestionWindow**

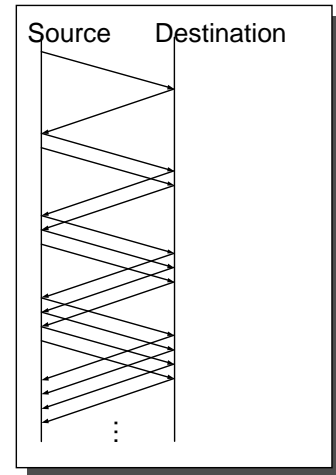
## *Subida Aditiva/Descida Multiplicativa*

- ◆ Como sabe a fonte se/quando a rede está congestionada?
- Por ocorrência de timeout!
  - » Redes fixas → pacotes raramente sofrem erros  
(Não é verdade nas redes sem fios!)
  - » Timeout assinala perda de pacote
  - » Perda de pacotes → congestionamento
    - Filas nos routers cheias

## *Subida Aditiva/Descida Multiplicativa*

### ◆ Algoritmo

- » Incrementa **CongestionWindow** de 1 pacote
  - Por cada **RTT** (Round Trip Time) → Subida linear
- » Divide **CongestionWindow** por 2
  - Sempre que há perda de pacote → Descida multiplicativa

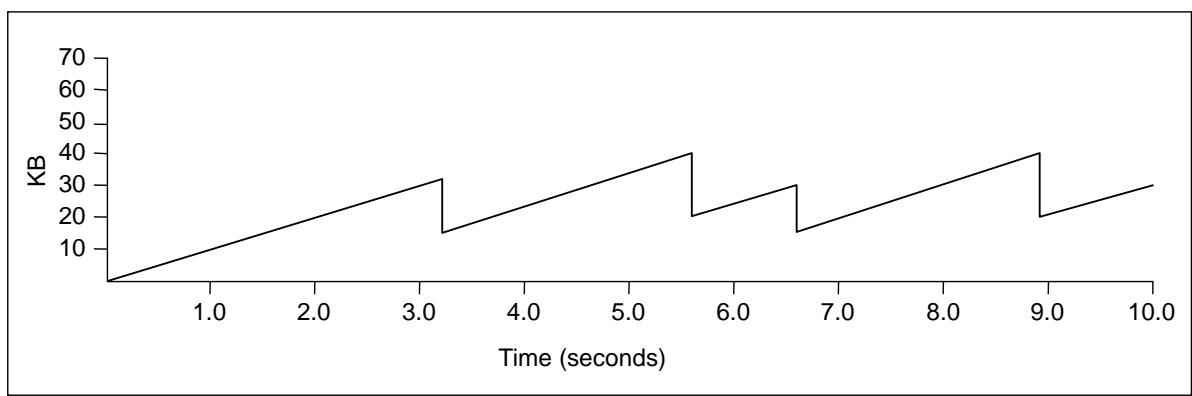


### ◆ Na prática,

- » Incrementa ligeiramente por ACK recebido
- »  $\text{Increment} = \text{MSS} * (\text{MSS} / \text{CongestionWindow})$
- »  $\text{CongestionWindow} += \text{Increment}$
- » **MSS** → **Maximum Segment Size**

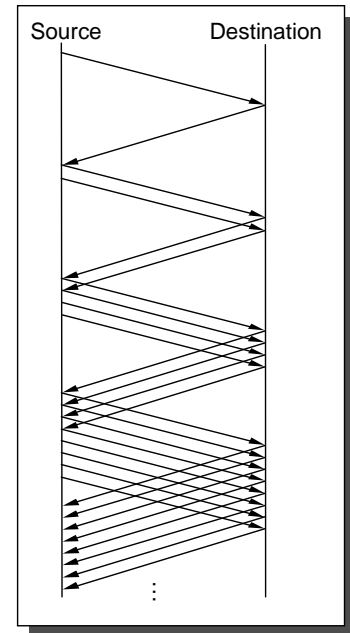
## *Subida Aditiva/Descida Multiplicativa*

### ◆ Funcionamento Dente de Serra



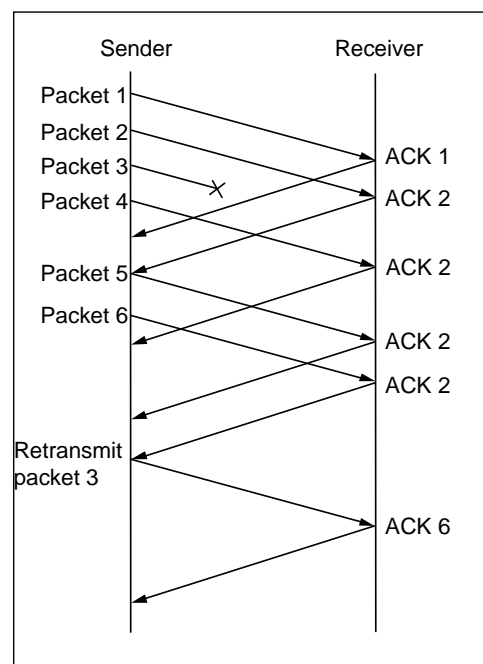
## Arranque Lento ☺

- ◆ Objectivo
  - » Determinar capacidade de transmissão disponível
  
- ◆ Aproximação
  - » Começar com **CongestionWindow = 1 pacote**
  - » Duplicar **CongestionWindow** em cada RTT



## Retransmissão e Recuperação Rápidas

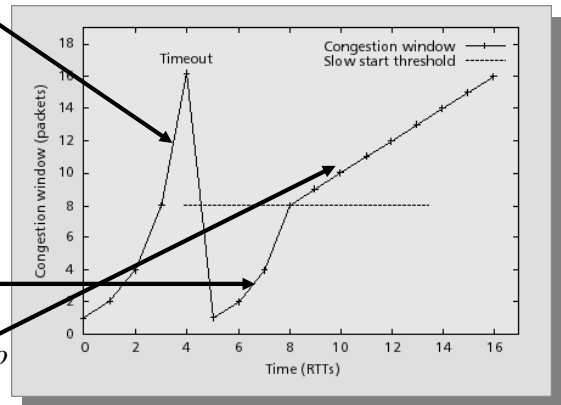
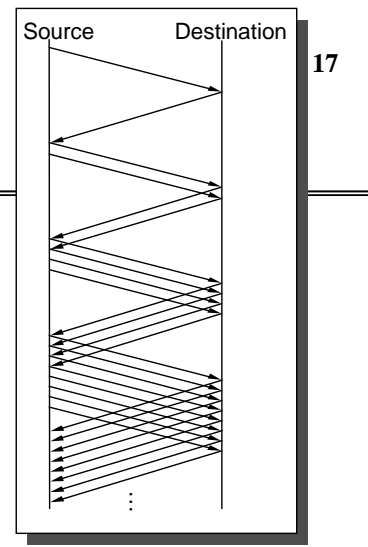
- ◆ Problema
  - » Se timeout TCP grande
    - período de inactividade grande
  
- ◆ Solução
  - » Retransmissão rápida
    - utilização de ACKs repetidos (3)





# TCP – ArranqueLento

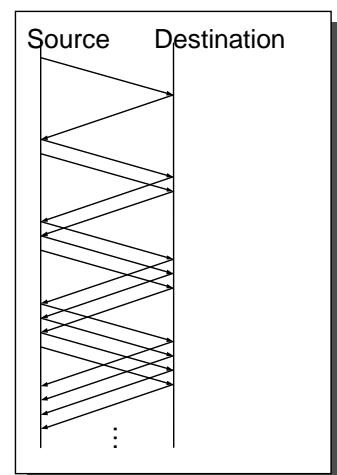
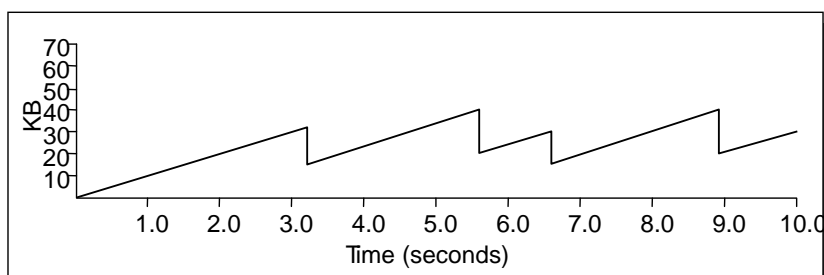
- ◆ TCP tem `congestionWindow`
  - » número de pacotes em trânsito, sem causar congestão
  - » Novos pacotes enviados se permitidos por
    - `congestionWindow`, e
    - `Advertisedwindow`, do receptor
- ◆ *ArranqueLento* ☺
  - » Emissor começa com `congestionWindow=1sgm`
  - » Duplica `congestionWindow` em cada `RTT`
- ◆ Quando detecta perda de pacote, por timeout
  - » `threshold = ½ congestionWindow`
  - » `congestionWindow=1sgm`  
(router esvazia filas)
  - » Pacote perdido retransmitido
  - » *ArranqueLento* enquanto  
`congWindow < threshold`
  - » Depois → fase de *PrevençãoDeCongestionamento*



# PrevençãoDeCongestionamento (Congestion Avoidance)

CNG 18

- ◆ *PrevençãoDeCongestionamento* (subida aditiva)
  - » Incrementa `congestionWindow` de 1 `sgm`, por cada `RTT`
- ◆ Detecção de perda de pacote, por recepção de 3 ACKs duplicados
  - » Deduz que pacote se perdeu,
    - não por congestionamento severo, porque segm seguintes chegaram destino
  - » Retransmite pacote perdido
  - » `congestionWindow = congestionWindow / 2`
  - » Fase de *PrevençãoDeCongestionamento*



## *TCP – Controlo de Congestionamento*

---

- ◆ Na realidade, um pouco mais complexo
- ◆ RFC 2001, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”

## *Evitar o Congestionamento*

---

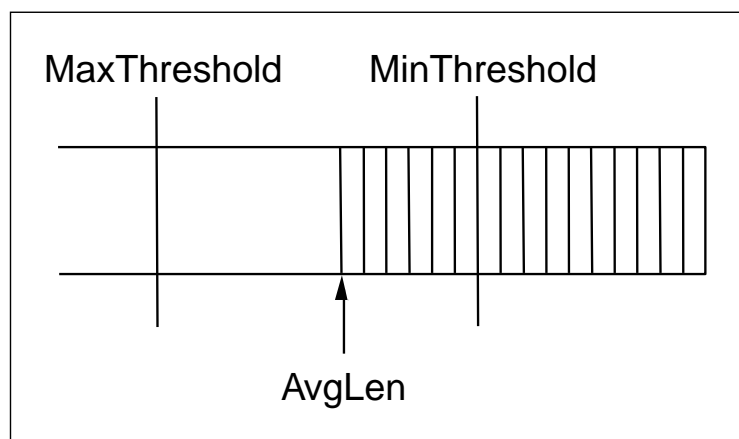
- ◆ Estratégia do TCP
  - » Controlo imediato do congestionamento
  - » Incrementa sistematicamente a carga
    - Detecta ponto de congestionamento → diminui carga
- ◆ Método alternativo
  - » Previsão do ponto de congestionamento
  - » Reduz débito antes de pacotes começaram a ser perdidos
  - Evitar o congestionamento (em vez de a controlar)
- ◆ Duas soluções
  - » Baseado em routers → RED
  - » Baseado em computadores → TCP Vegas

## Random Early Detection (RED)

- ◆ Notificação implícita
  - » Eliminação de pacote → timeout no TCP
  
- ◆ Prematuramente
  - » Em vez de eliminar quando a fila fica cheia,
  - » Elimina quando comprimento da fila superior a valor prédefinido

## RED – Cálculo do Comprimento Médio da Fila

- »  $AvgLen = (1 - Weight) * AvgLen + Weight * SampleLen$
- »  $0 < Weight < 1$  (normalmente 0,002)
- »  $SampleLen$  → comprimento da fila quando o pacote chega



## *RED – Limiares na Fila*

---

```

if AvgLen <= MinThreshold then enqueue the packet
if MinThreshold < AvgLen < MaxThreshold then
    calculate probability P
    drop arriving packet with probability P
if MaxThreshold <= AvgLen then drop arriving packet
  
```

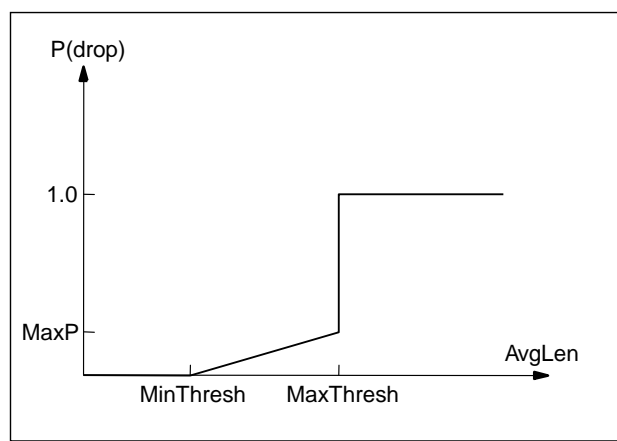
## *RED – Cálculo da Probabilidade P*

---


$$\text{TempP} = \text{MaxP} * (\text{AvgLen} - \text{MinThreshold}) / (\text{MaxThreshold} - \text{MinThreshold})$$

$$P = \text{TempP} / (1 - \text{count} * \text{TempP})$$

» Curva de probabilidade de eliminação de pacote



## *RED*

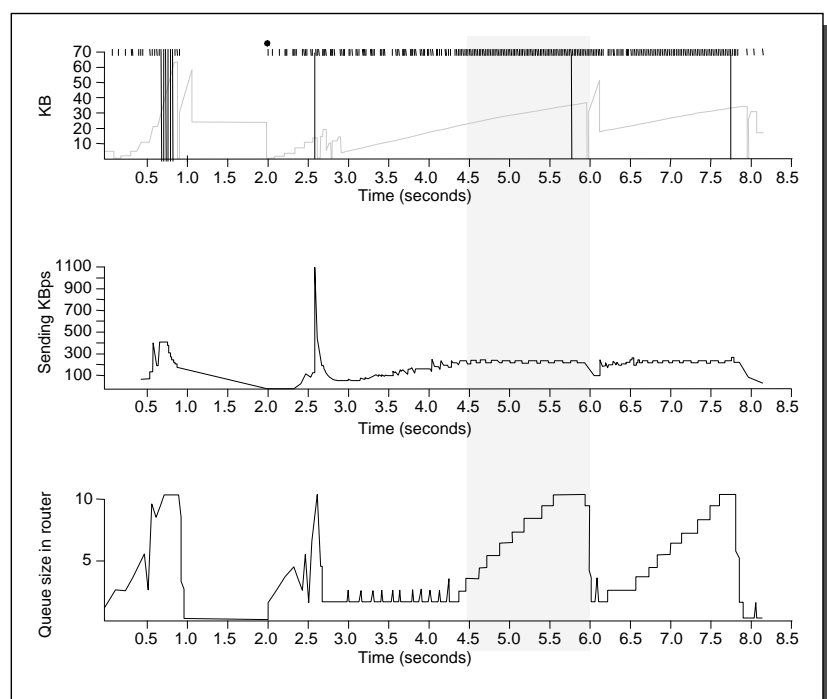
---

- ◆ Probabilidade de perda de pacotes num fluxo →
  - » Proporcional à capacidade de transmissão associada ao fluxo, na ligação
- ◆ Tipicamente, **MaxP=0.02**
  - » se comprimento médio da fila =  $\frac{1}{2}$  do intervalo entre os limiares  
→ eliminado 1 em cada 100 pacotes
- ◆ Com tráfego bursty, **MinThreshold** → elevado
- ◆ Tipicamente, **MaxThreshold = 2x MinThreshold**

## *TCP Vegas*

---

- ◆ Fonte procura sinais de congestão
  - » Aumento de RTT
  - » Taxa de transmissão  
→ curva horizontal



## Algoritmo

---

- ◆ **BaseRTT** → menor dos **RTT** medidos (tipicamente o do 1º pacote)
- ◆ Em condições normais, **ExpectRate** = **CongestionWindow**/**BaseRTT**
- ◆ Fonte
  - » calcula débito de emissão em cada medida de **RTT**
  - » compara **ActualRate** com **ExpectRate**

```

Diff = ExpectedRate - ActualRate
  if Diff < a
      aumenta CongestionWindow linearmente
  else if Diff > b
      diminui CongestionWindow linearmente
  else
      mantém CongestionWindow

```

## Algoritmo

---

- ◆ Parâmetros
  - » a = 1 pacote
  - » b = 3 pacotes

