

# Redes de Computadores

## Controlo da Saturação

# Objetivos do Capítulo

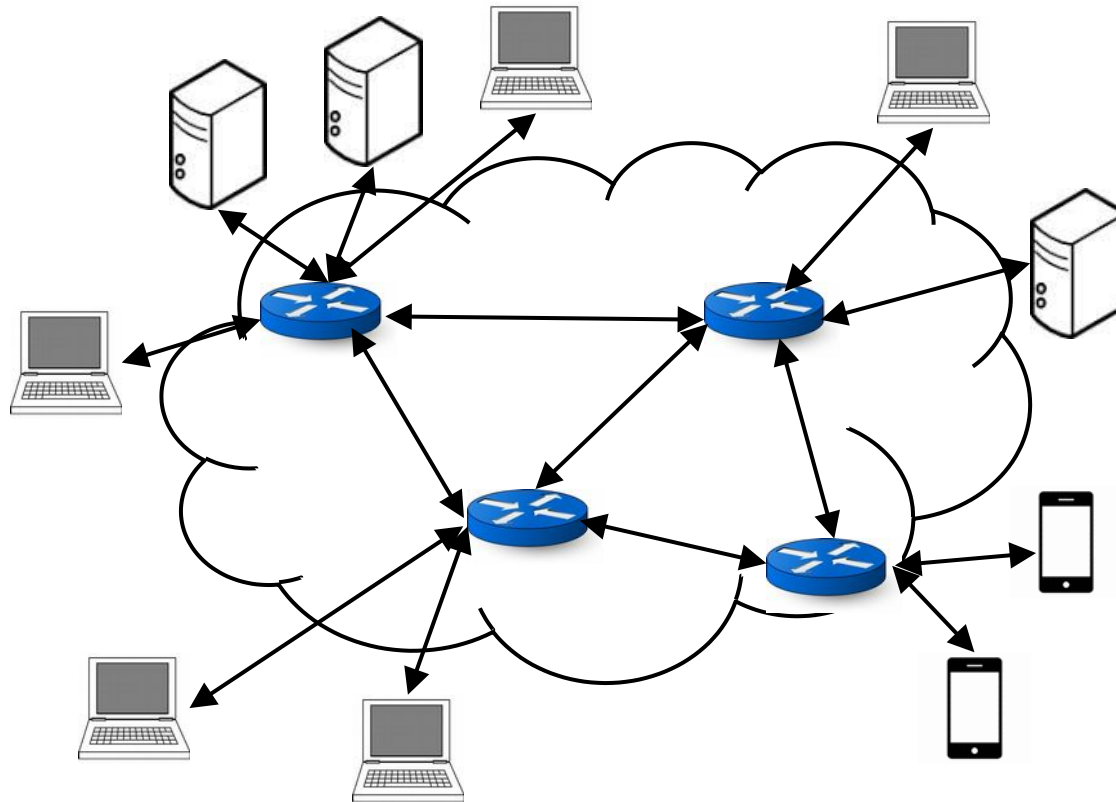
- Com comutação de pacotes e multiplexagem estatística a entrega dos pacotes e o tempo de trânsito não são garantidos
- Por detrás de ambos os comportamentos estão os erros nos canais e as filas de espera existentes nos comutadores
- Aumentar as solicitações para além do que os comutadores comportam pode ser uma estratégia errada
- Como encontrar um ponto de equilíbrio?
- Como é que o protocolo TCP lida com esta situação?

*The scientist described what is: the engineer creates what never was.*

*(O cientista descreve o que é: o engenheiro cria o que nunca existiu.)*

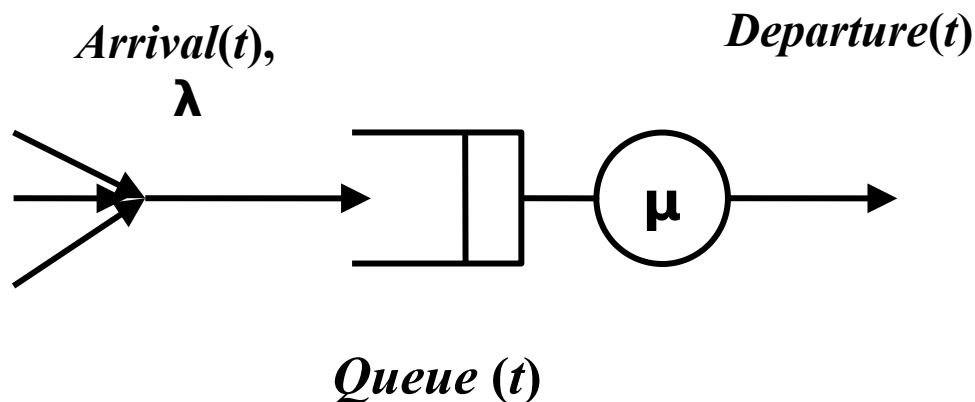
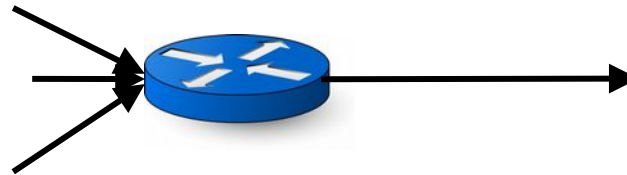
*- Autor: Theodor von Karman - the father of the supersonic flight*

# Uma Rede de Paquetes

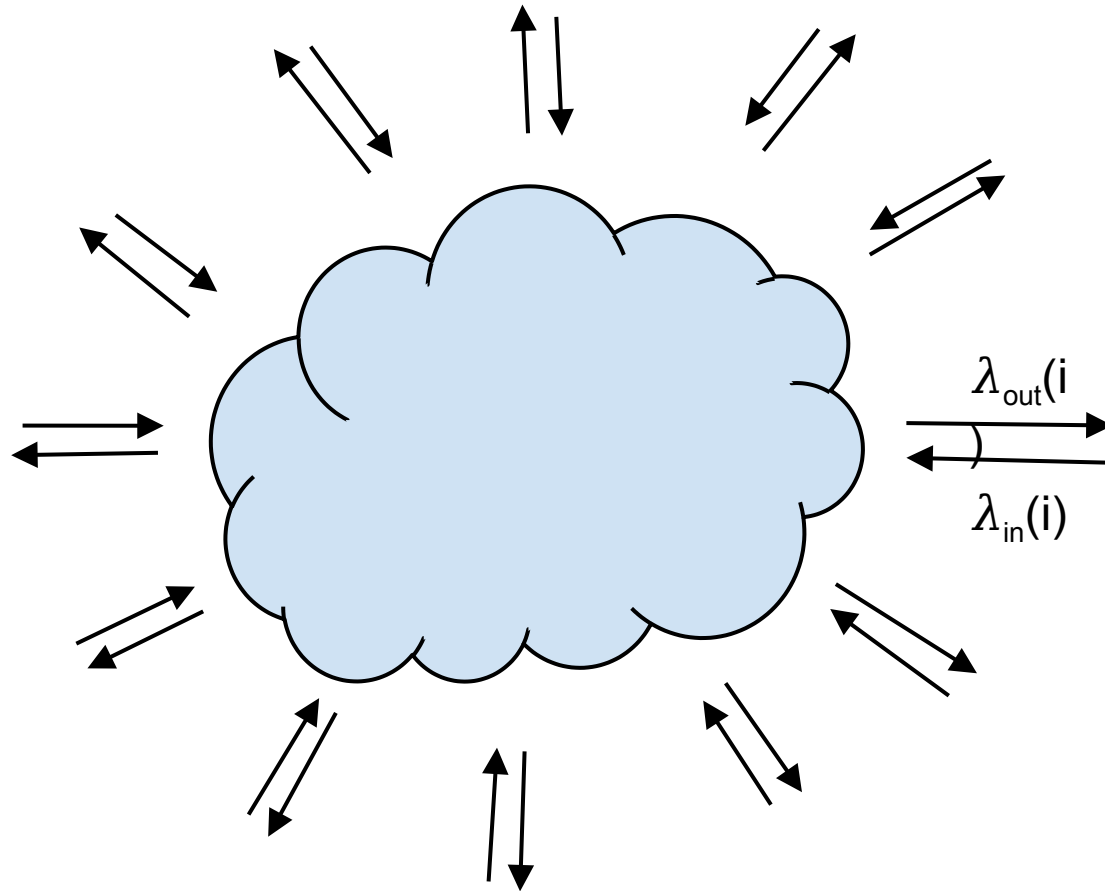


# Chegada e Transmissão de Pacotes

Modelo de uma fila de espera FIFO

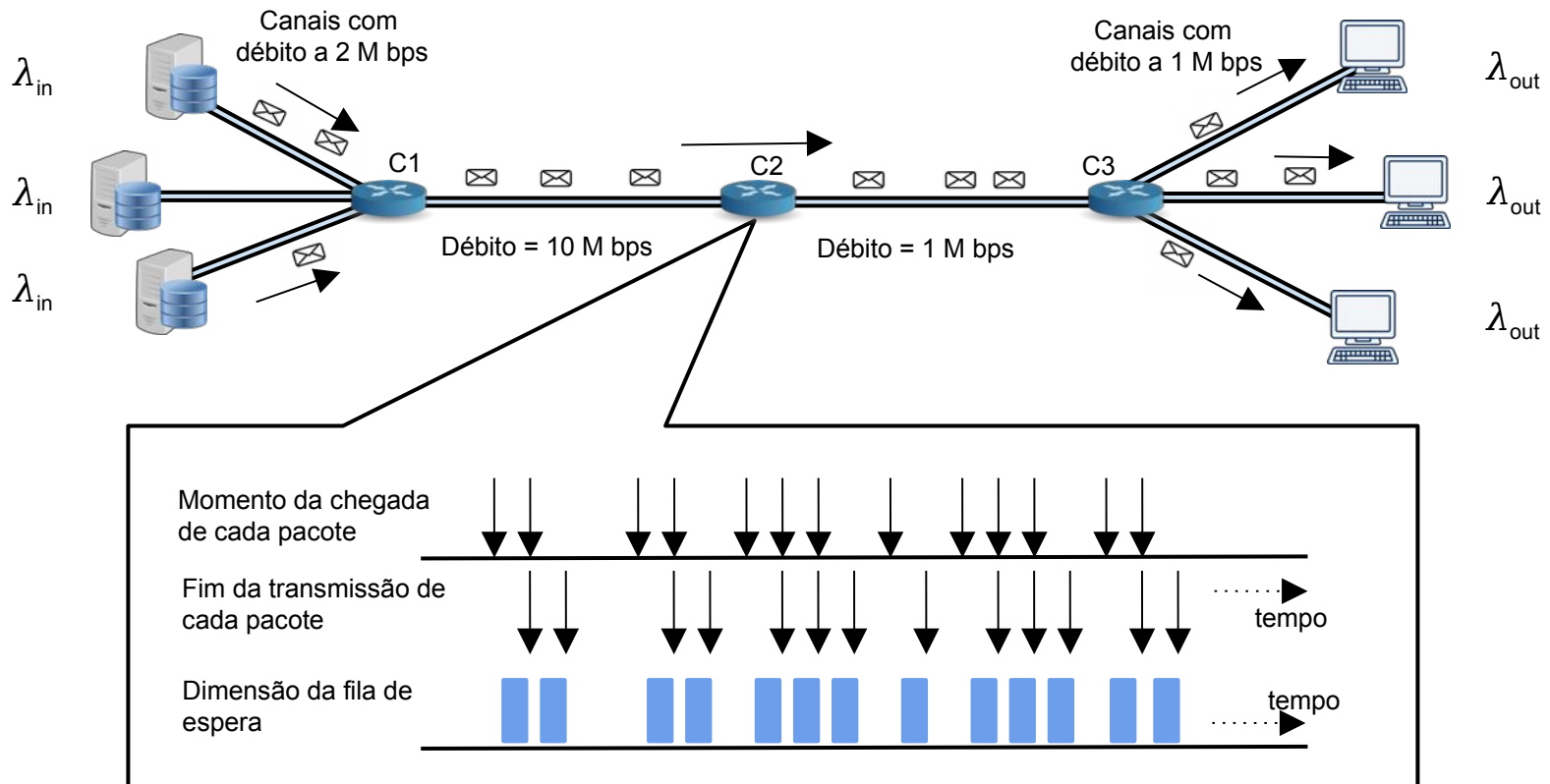


# Qual o Comportamento da Rede?

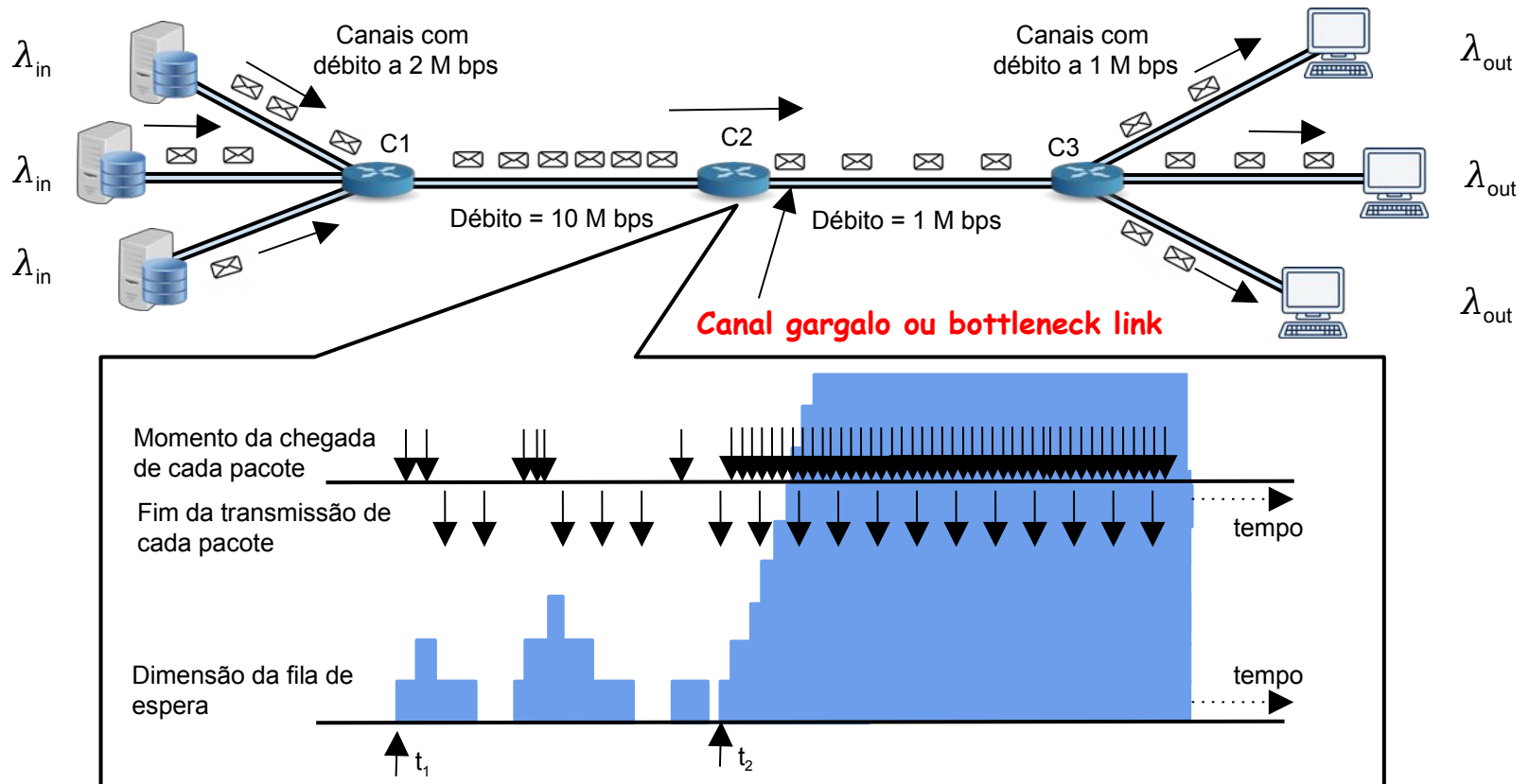


Teoricamente,  $\sum \lambda_{out}(i) \approx \sum \lambda_{in}(i)$ , mas na prática isso só é verdade numa rede pouco carregada

# Rede Pouco Carregada



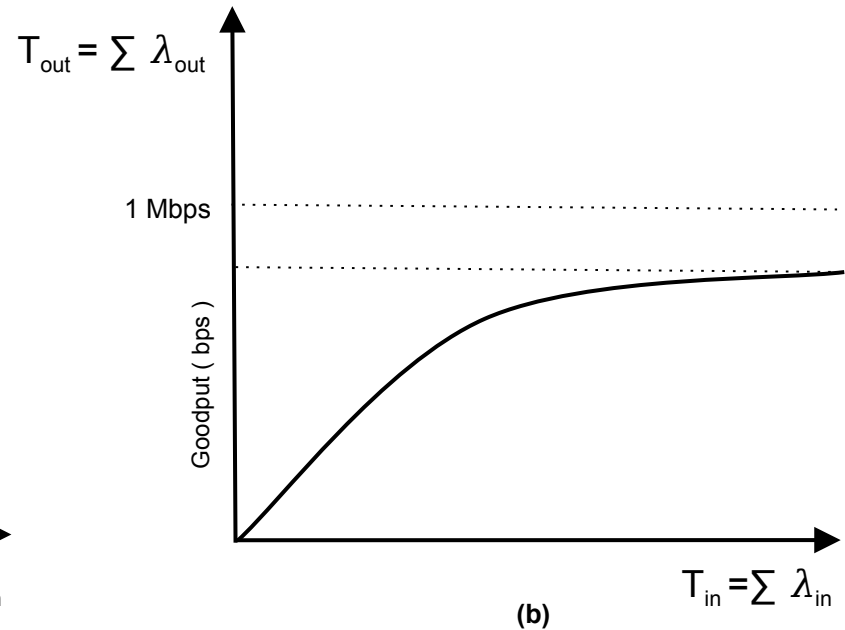
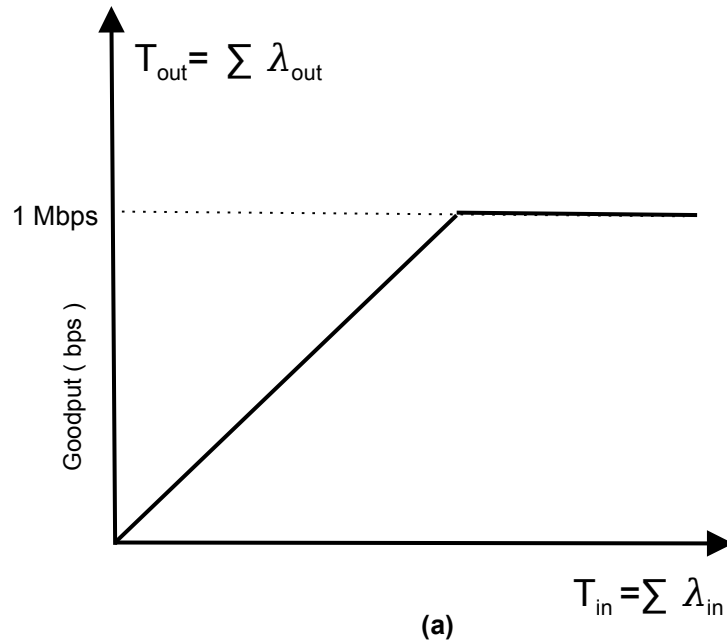
# Rede Mais Carregada



A partir de  $t_2$  a carga aumenta



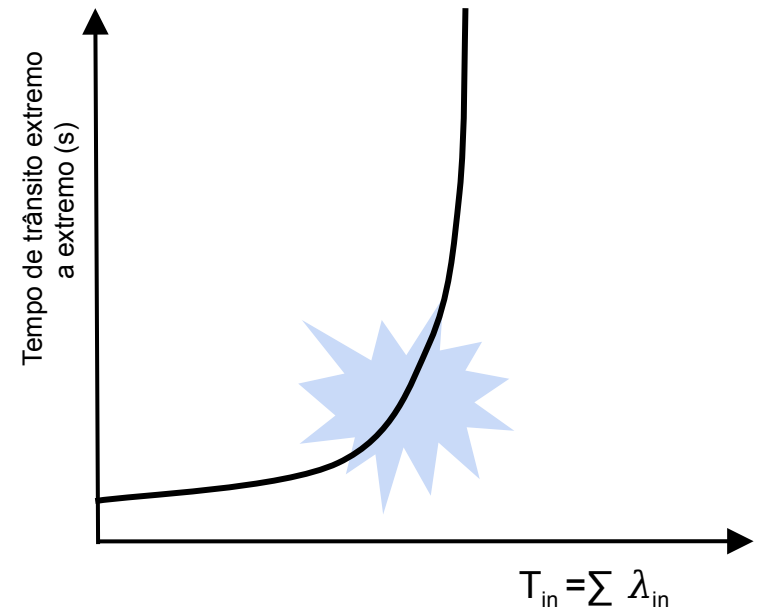
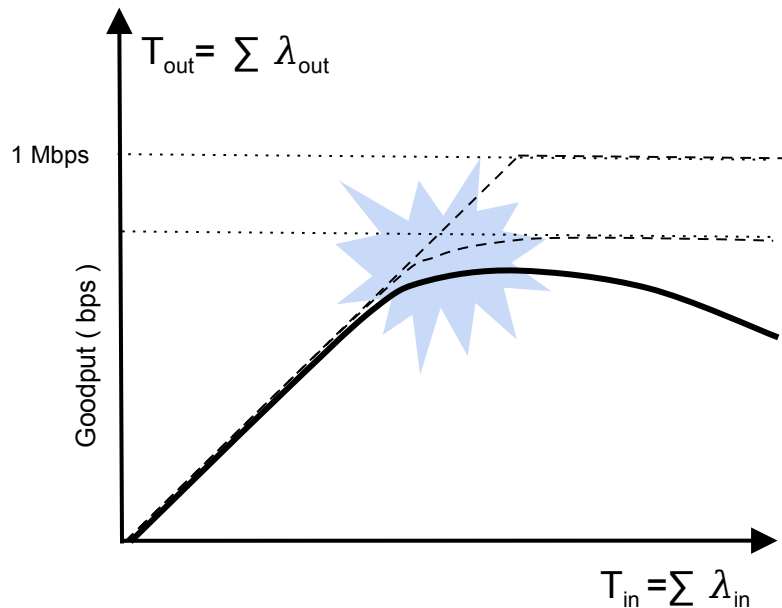
# Débito Útil (Goodput)



(a) Emissores NÃO retransmitem os pacotes atrasados

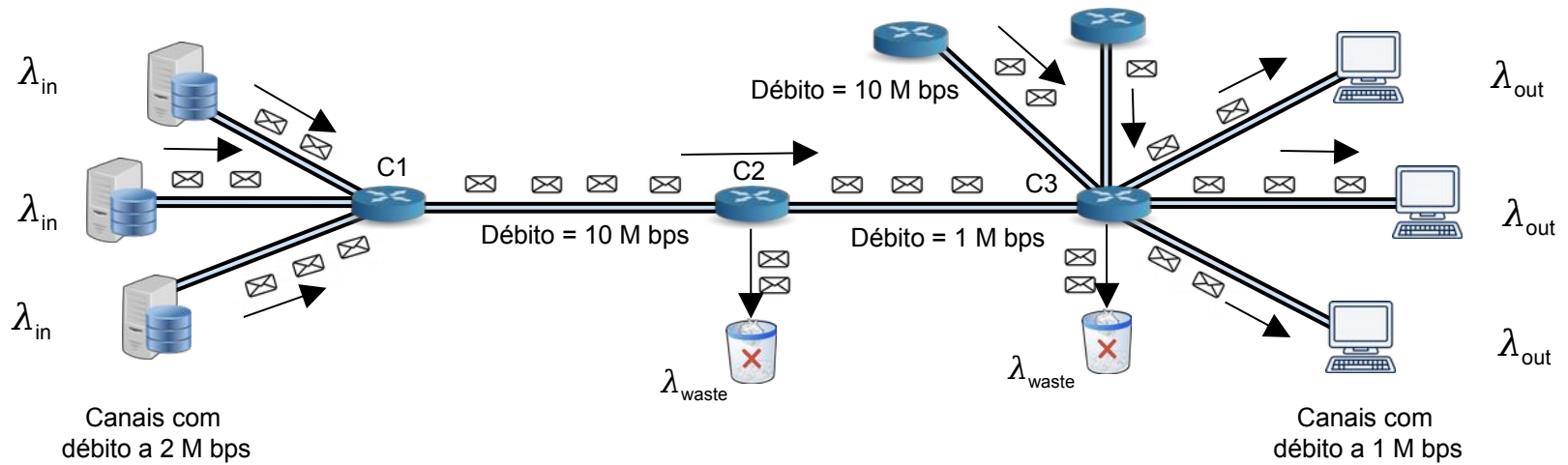
(b) Emissores retransmitem os pacotes atrasados

# Goodput e Tempo de Trânsito Finais

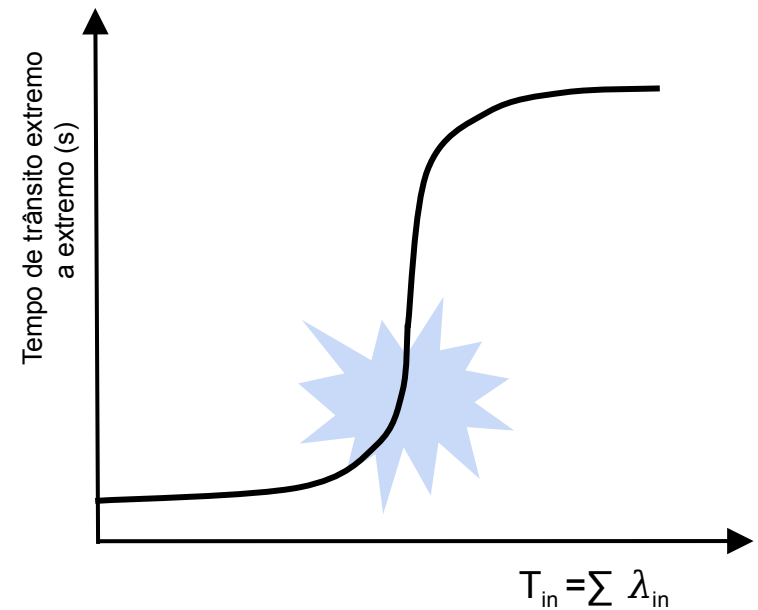
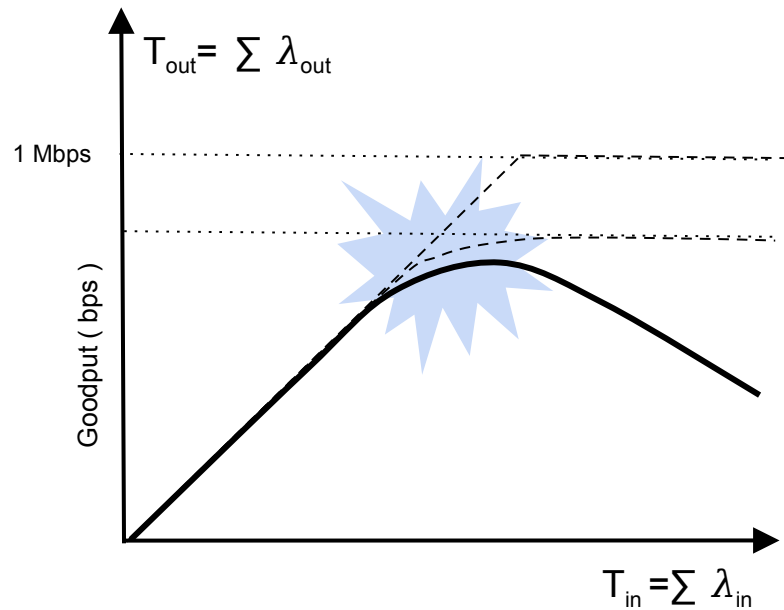


Situação mais realista pois o valor dos alarmes não consegue acompanhar a subida do tempo de trânsito

# Supressão de Pacotes



# Goodput e Tempo de Trânsito Finais



# É Possível Dimensionar a Rede?

- Se for possível caracterizar todos os fluxos de pacotes emitidos por todos os emissores
- Se for conhecida a forma de encaminhamento desses fluxos na rede
- É possível, usando teoria das filas de espera, numa pequena rede, determinar para cada canal qual a capacidade que acomoda as necessidades dos fluxos que o atravessam
- Mas tal é geralmente impossível numa rede real
- Só é mais realista por simulação usando modelos simplificados
  
- E dimensionar assim a rede não seria economicamente viável

# Caracterização da Qualidade de Serviço

- Tempo de transito extremo a extremo e sua variância (*jitter*)
- Débito de extremo a extremo e débito útil de extremo a extremo (*throughput* e *goodput*) e sua caracterização
- Taxa de perda de pacotes
- É fácil de a garantir valores (médios, mínimos, ...) ?

# Controlo da Saturação

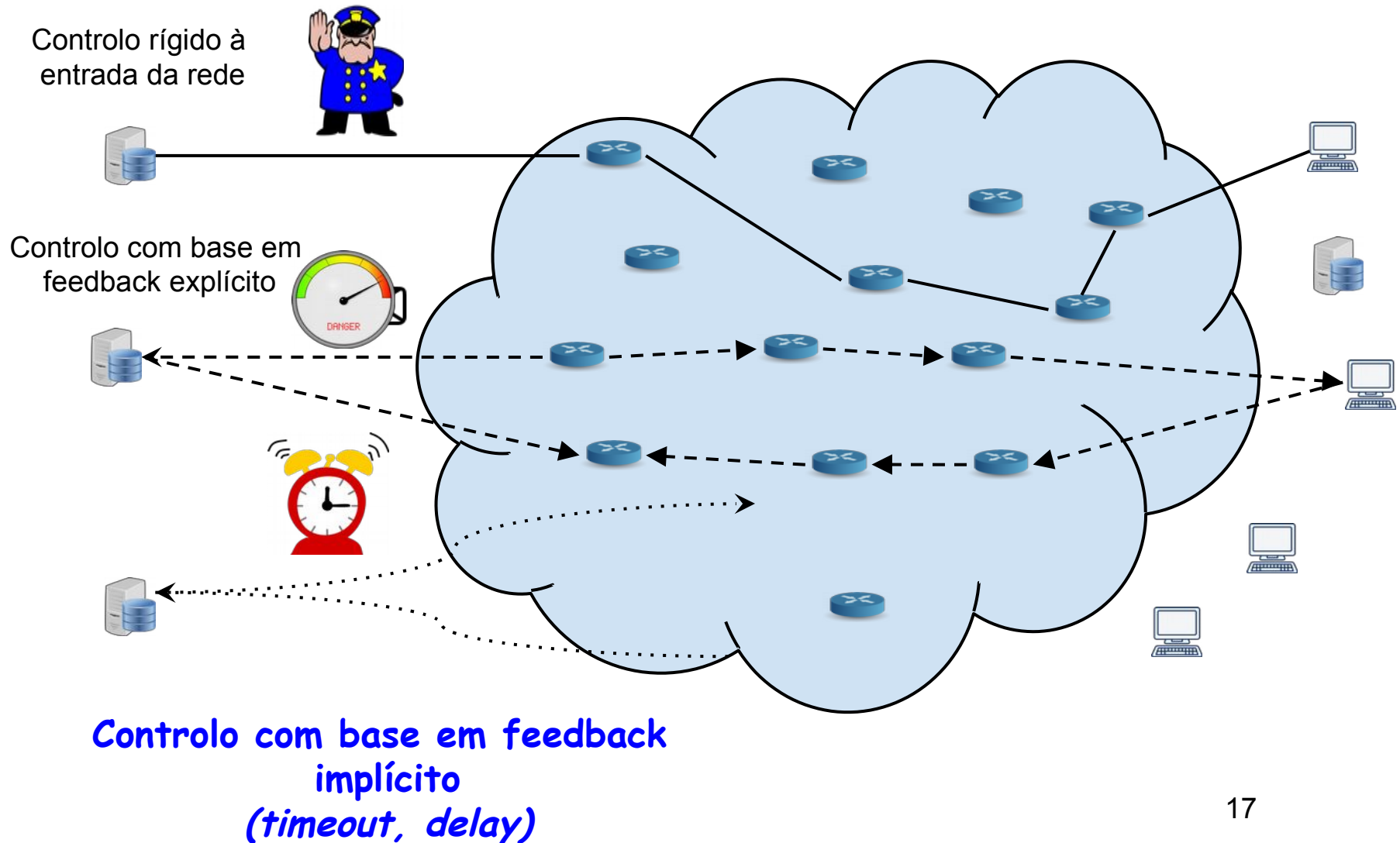
- Uma rede diz-se saturada quando o conjunto de solicitações de encaminhamento de pacotes que recebe a conduzem a uma situação de colapso, e a quantidade real de pacotes encaminhados é inferior à capacidade efetiva da rede
- Ao conjunto de arquiteturas, protocolos e algoritmos usados para controlar as solicitações à rede, de forma a evitar que esta entre em saturação, chamam-se arquiteturas, protocolos e algoritmos de controlo da saturação

# Rede Saturada





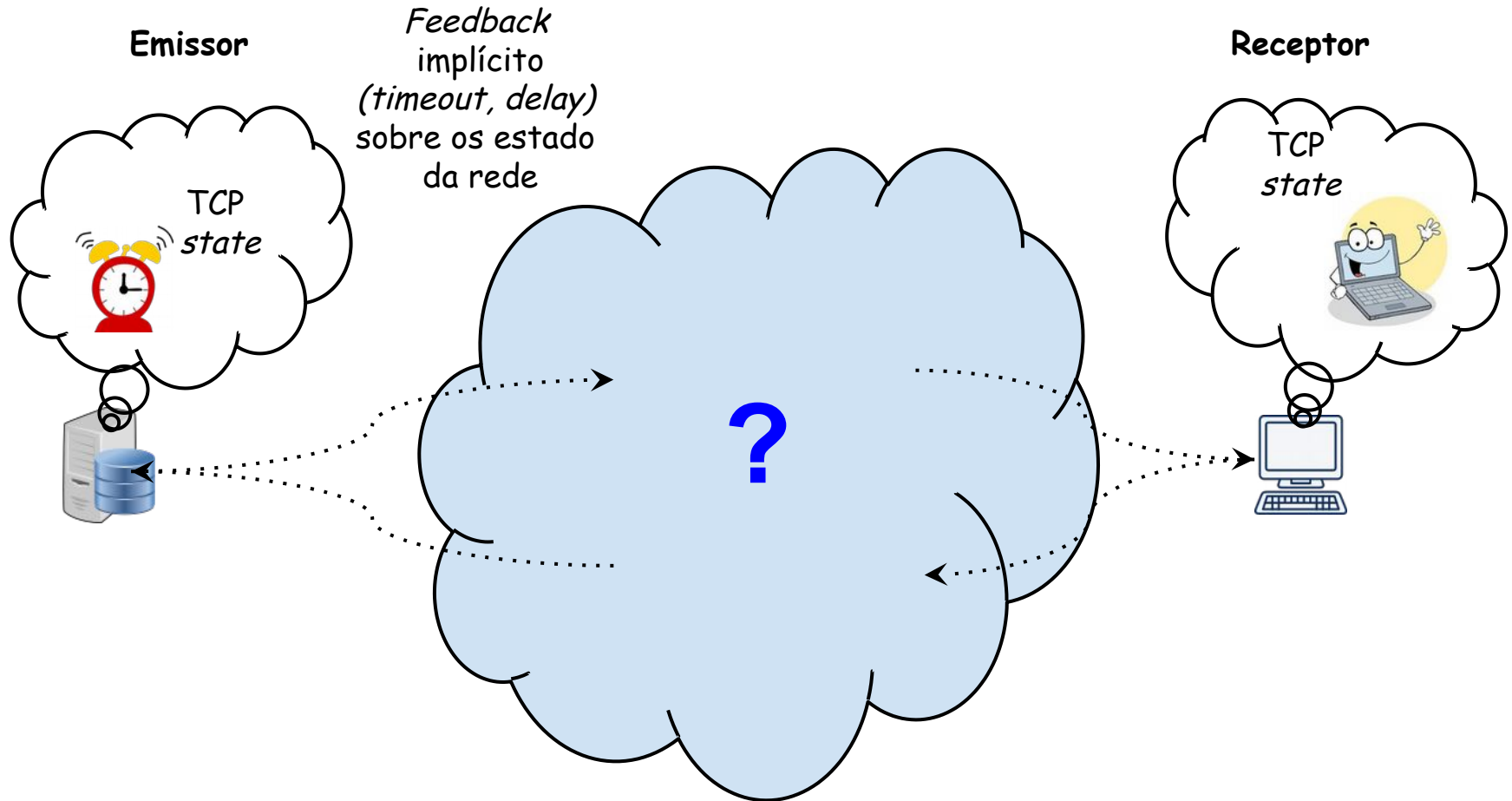
# Alternativas para Evitar a Saturação



# TCP e Controlo da Saturação

- Dado normalmente não existir nenhum mecanismo que limite o ritmo de emissão, a saturação da rede seria inevitável
- Isso levou à adoção de mecanismos de controlo ao nível do protocolo TCP cujo mecanismo de controlo da saturação, atuando "voluntariamente", é fundamental para o funcionamento da Internet
- O mecanismo fundamental usado pelo TCP para lidar com a saturação designa-se por AIMD (*additive increase, multiplicative decrease*)

# Só é Realista Atuar do Lado do Emissor

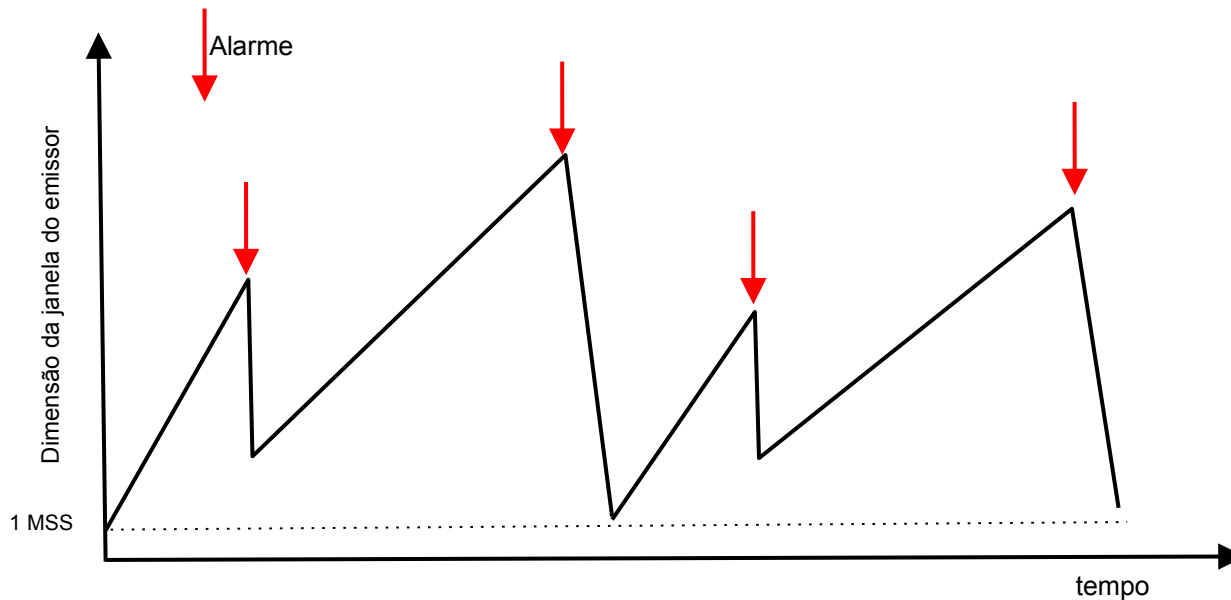


# Respostas Possíveis dos Emissores

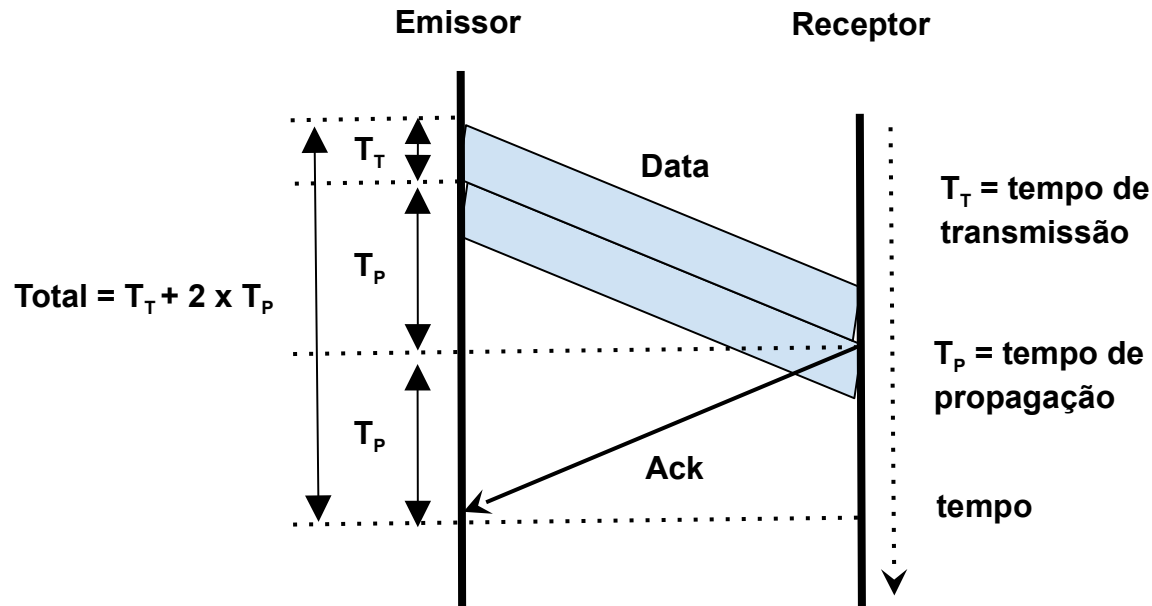
- Ignorar — só piora a situação
- O principal sintoma da saturação é a perda de pacotes (*timeout ou múltiplos ACKs repetidos*) — quando a mesma tem lugar pode-se diminuir o ritmo de emissão — mas quanto?
- Como as filas de espera estão cheias, deve-se adotar um comportamento agressivo — diminuir o ritmo para metade ou mesmo para próximo de zero
- Mas que fazer se os pacotes começam a passar? Deve-se aumentar o ritmo a pouco e pouco

# Solução TCP (AIMD)

- Incremento aditivo, recuo multiplicativo (AIMD)
  - A quando da detecção de uma perda de pacote, dividir a janela por dois (recuo multiplicativo)
  - Quando uma janela foi transmitida com sucesso, incrementar a janela a pouco e pouco (aditivamente)



# Como Controlar o Débito de Emissão ?



A dimensão da janela de emissão determina o ritmo de emissão. Sempre que o tempo de transmissão é negligenciável face ao RTT, o débito extremo a extremo é dado por:

Débito extremo a extremo = dimensão em bits da janela / RTT

# Receiver Window e Congestion window

- Controlo de fluxo — evita a saturação do recetor
  - O emissor TCP sabe qual a dimensão da janela do recetor e evita enviar mais de cada vez que esta dimensão ( $sending\ window \leq receiverAdvertisedWindow$ )
- Controlo de saturação — evita a saturação da rede
  - O emissor tenta estimar o ritmo máximo que não sature a rede em função da deteção de perda de segmentos TCP emitidos ( $sending\ window \leq congestionWindow$ )
- Janela máxima de emissão:

mínimo ( $congestionWindow, receiverAdvertisedWindow$ )

# E no Início?

- Arrancar com um valor arbitrário? Mas que valor?
- Se for com todo o espaço livre na janela do recetor (que se fica a conhecer durante a abertura da conexão) poderá conduzir a um valor muito alto?
- Por exemplo, se a janela inicial for igual a 128 K byte, ou seja quase 80 pacotes de  $\approx 1500$  bytes.
- Essa foi a solução adotada inicialmente e levava muitas vezes a saturação.
- Arrancar com  $\text{congWnd} = 1 \text{ MSS}$ , não leva a saturação mas não levará demasiado tempo a subir a janela com AIMD ?



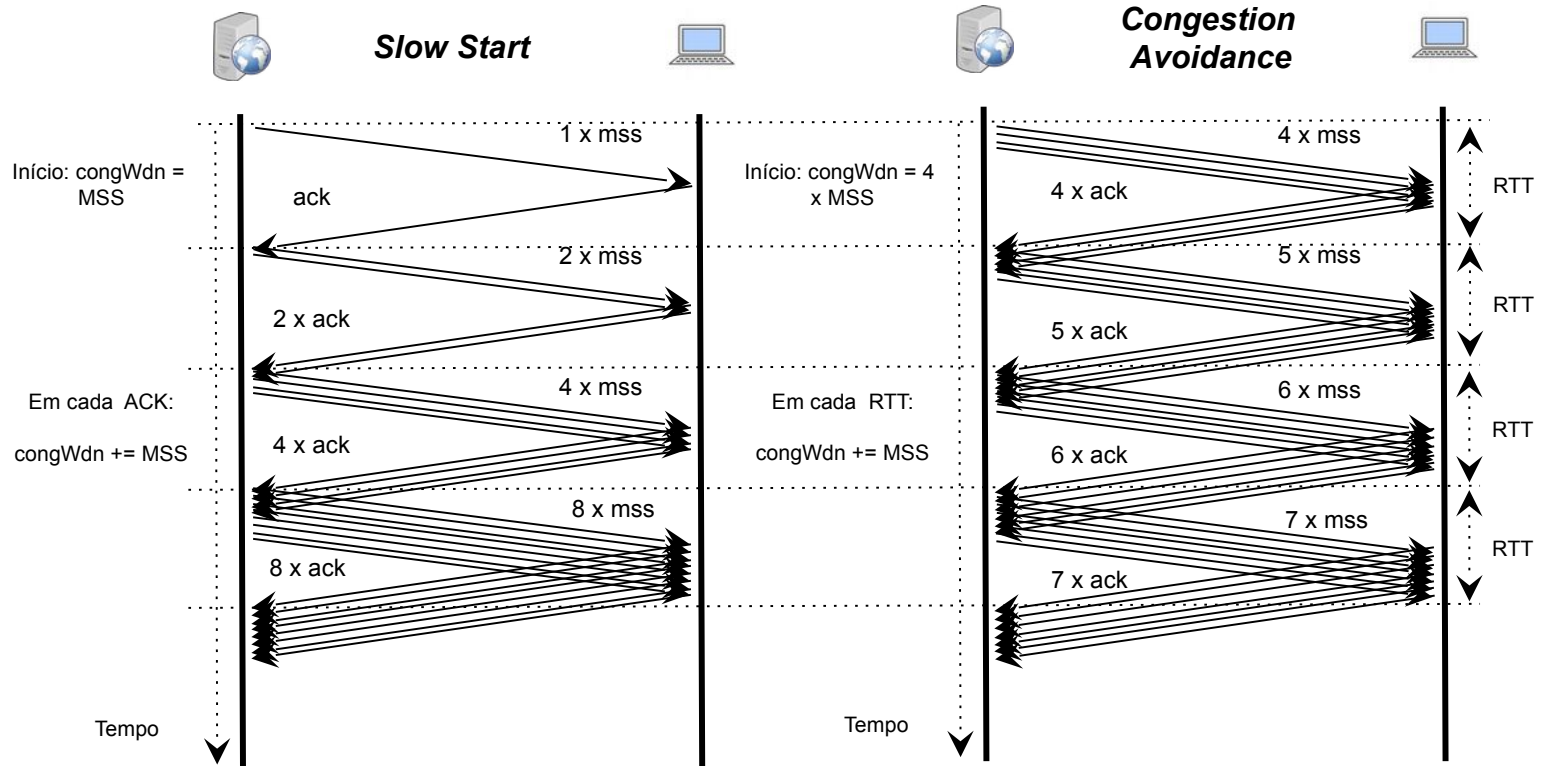
# Exemplo

- Supondo por hipótese que se incrementa a janela de 1 MSS no fim de receber todos os ACKs correspondentes aos segmentos da janela anterior:
  - Segmentos de 1 K Byte (MSS) =  $1.024 \times 8 = 8.192 \approx 8.10^3 \approx 10^4$  bits
  - RTT = 50 ms
  - Capacidade disponível no *bottleneck link* = 100 M bps =  $10^8$  bps
  - Quantos bits se deveriam transmitir em 50 ms para que o débito resultante seja 100 M bps?
  - Resposta:  $50 \times 10^{-3} \times 10^8 = 5 \times 10^6$
  - Quantos segmentos de  $10^4$  bits (10.000 bits) correspondem a  $5 \times 10^6$  bits?
  - Resposta:  $5 \times 10^6 / 10^4 = 500$  segmentos
- Com *Additive Increase* leva  $500 * 50 \text{ ms} = 25 \text{ segundos}$  a atingir a dimensão ideal da janela!

# Solução TCP - “*Slow Start*”

- Arrancar devagar (*slow start*)
  - Inicialmente  $\text{congWnd} = 1 \text{ MSS}$
  - Logo, inicialmente o ritmo é  $1 \text{ MSS} / \text{RTT}$
  - por exemplo:  $8000 \text{ bits} / 20 \text{ ms} = 400 \text{ Kbps}$
- Por cada ACK recebido aumentar  $\text{congWnd}$  de 1 MSS
  - A subida do ritmo de emissão passa a ser exponencial
- Isto é, *slow start is in fact fast start*
- Ao invés de *slow-start*, esta solução deveria chamar-se antes *Multiplicative Increase Start*

# Comparação

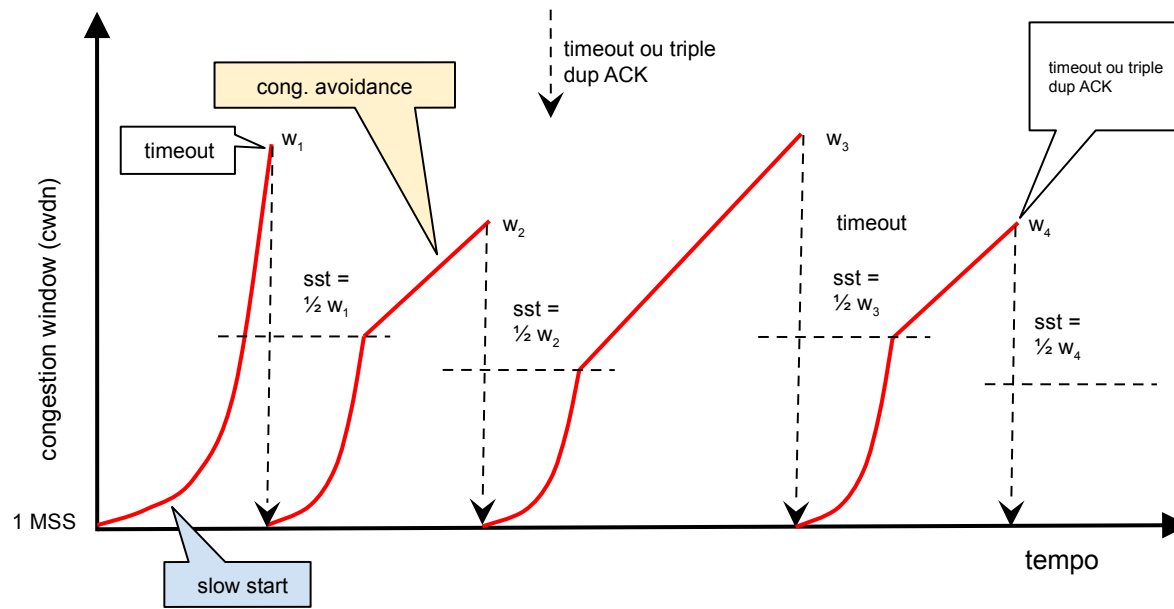


Por cada ACK:  $\text{congWdn} += \text{MSS}$

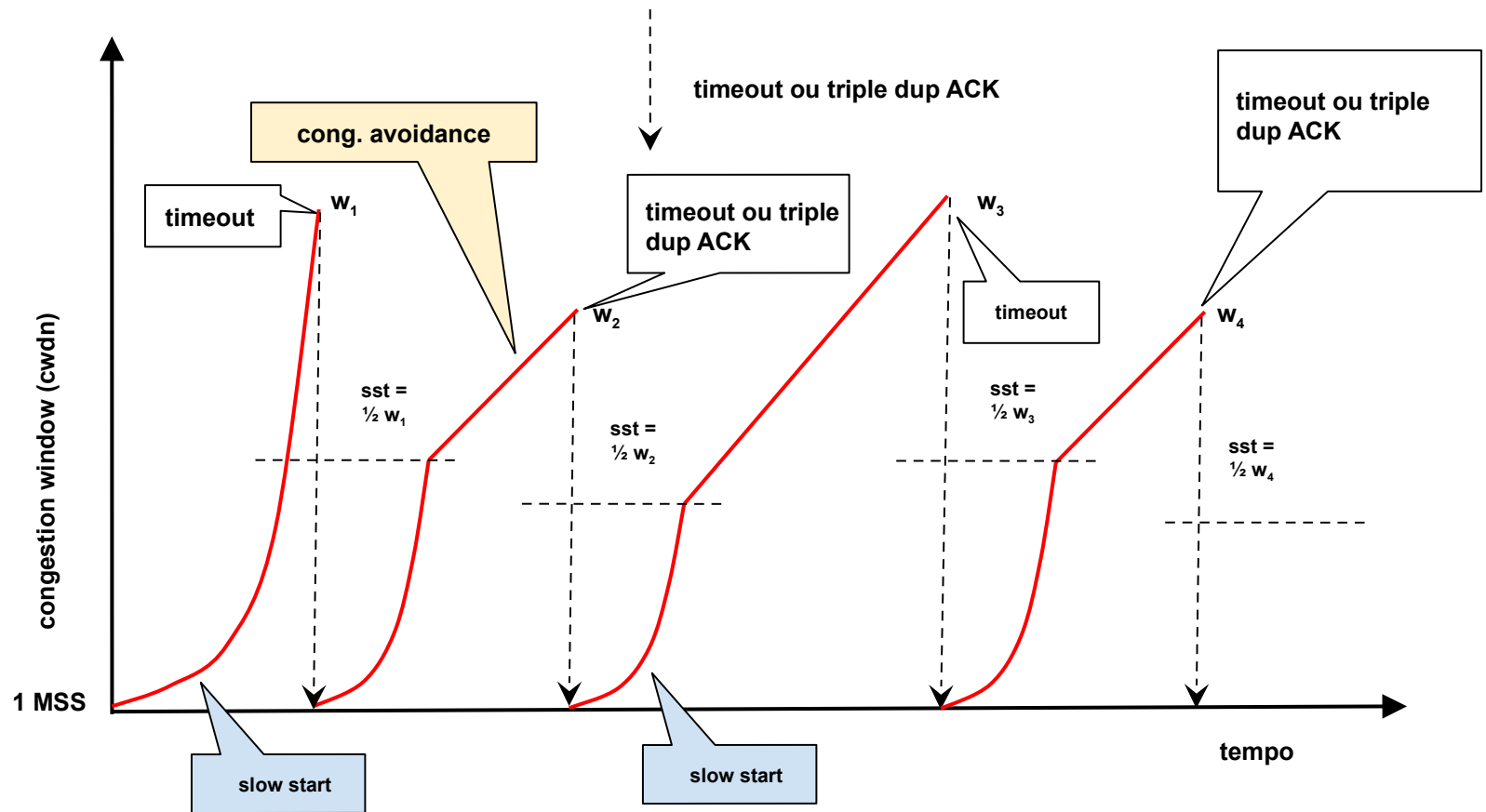
Por cada ACK:  $\text{congWdn} += \text{MSS} / \text{congWdn}$

# Alarme no Meio da Conexão

- *Slow start restart* — como se conhece a *congWnd* antes do *timeout*, pode-se tomar nota do seu valor na variável *sst* e da próxima vez abandonar a fase de *slow start* quando se atinge  $\frac{1}{2}$  da *congWnd* anterior

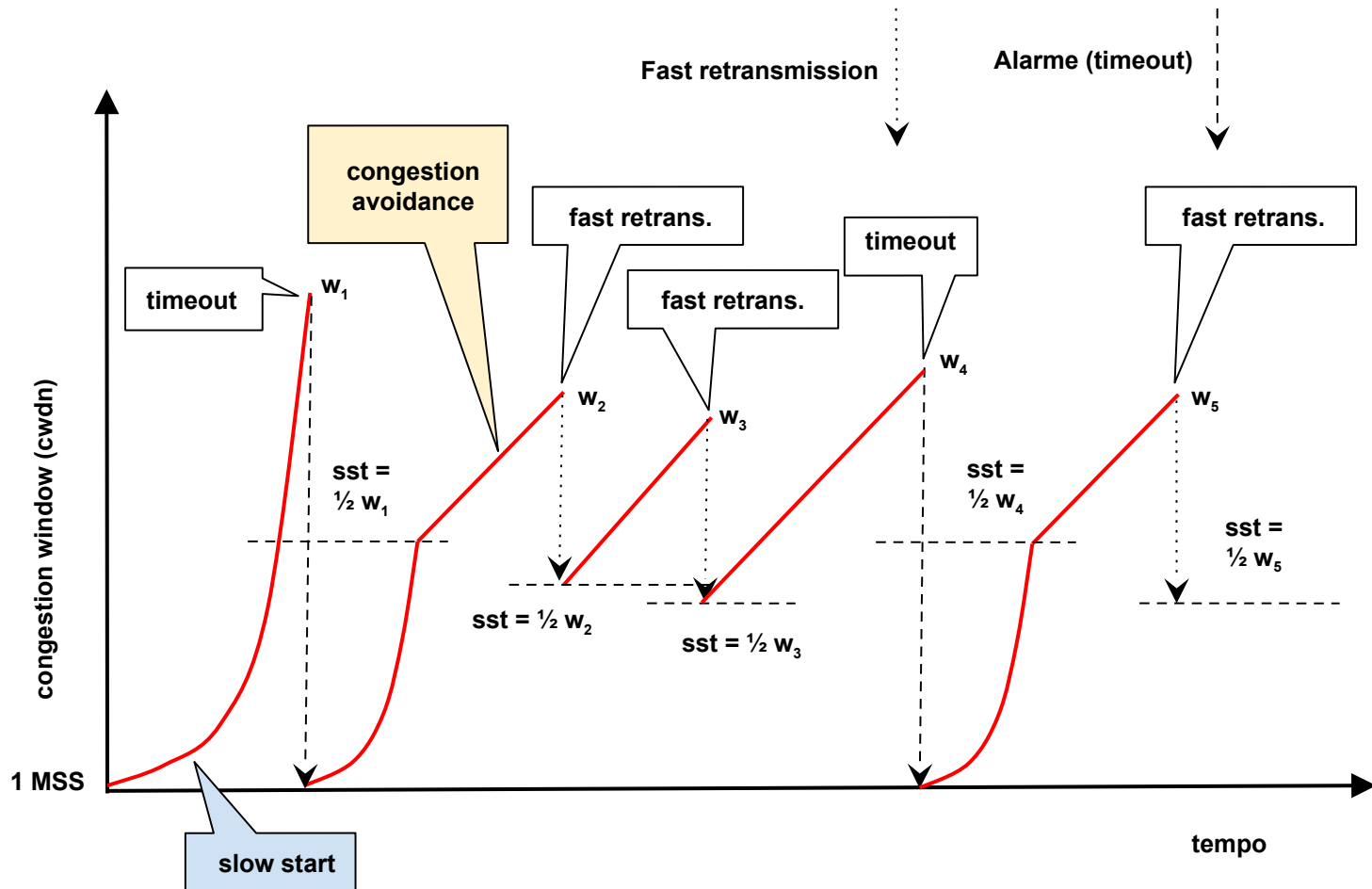


# TCP Tahoe



No início, e após cada alarme ou *fast retransmit*, começa-se com 1 MSS mas sobe-se a janela de forma multiplicativa, na esperança de recuperar rapidamente um débito aceitável, mas a partir daí vai-se "sondando" a capacidade aditivamente, isto é, com prudência

# TCP Reno



É inútil recuar tão agressivamente perante um evento "fast retransmit"

# *Slow Start* depois de inatividade

- Depois de um período de inatividade do emissor qual a situação corrente ?
  - Talvez mais fluxos estejam agora a atravessar a rede
- Se for o caso, seria perigoso arrancar de novo com a `congWnd` anterior
- Muitas implementações TCP voltam nesta situação a entrar de novo na fase *slow start*

# Resumo

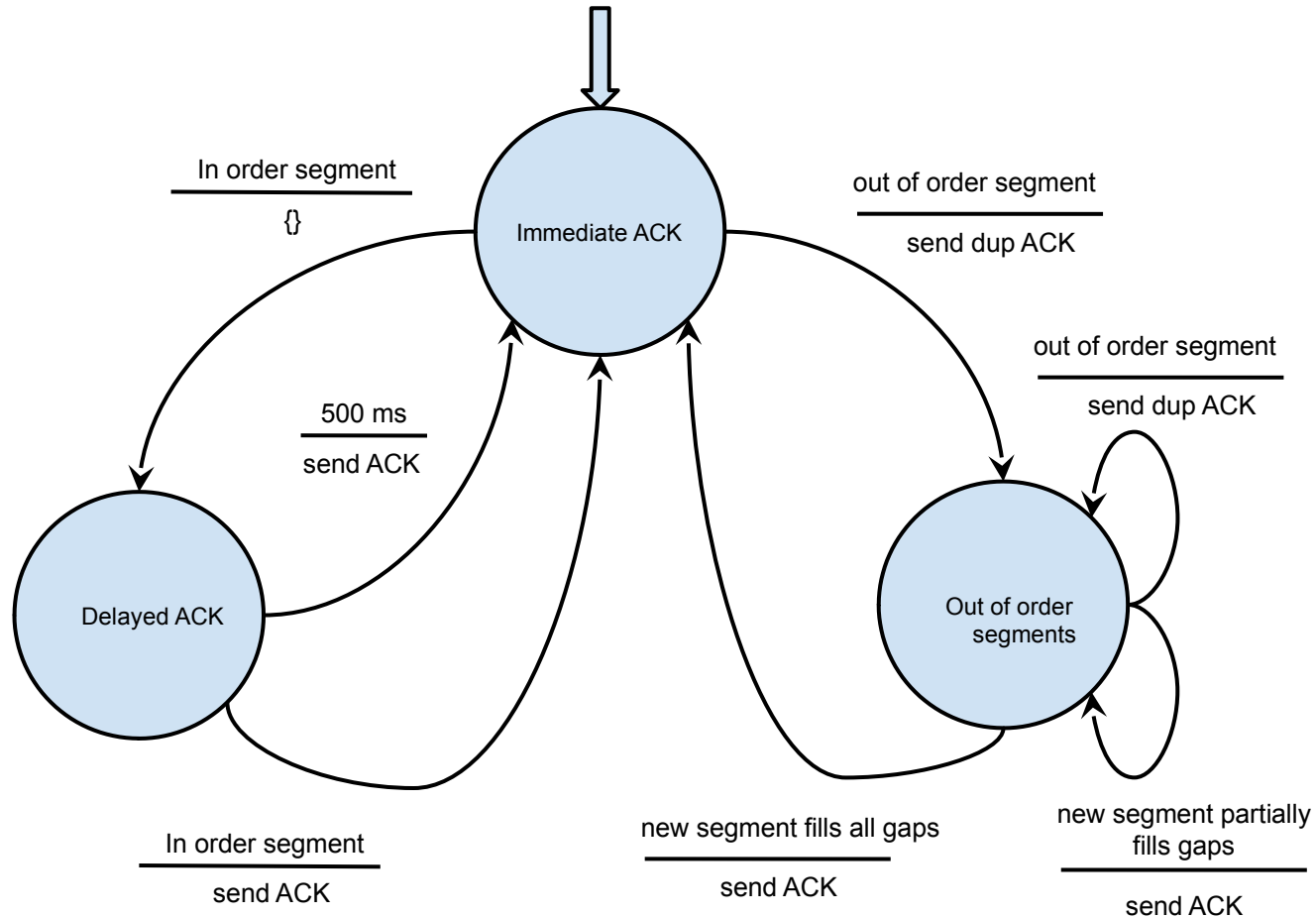
- Para a gestão da **cwnd** considerar dois estados
  - Slow start (multiplicative increase / multiplicative decrease)
  - Congestion avoidance (aditive increase / multiplicative decrease)
- Inicialmente
  - **sst = 64K; congWnd = 1 MSS; state = slow-start;**
- Incrementar a janela de emissão
  - Slow start — de mais um MSS por cada ACK
  - Consgestion avoidance — de mais um MSS por cada RTT
  - **if ( congWnd > sst ) state = congestion avoidance;**
- Timeout
  - **sst =  $\frac{1}{2}$  congWnd; congWnd = 1 MSS; state = slow start;**
- Triple ACK
  - **sst =  $\frac{1}{2}$  congWnd; congWnd = sst;**



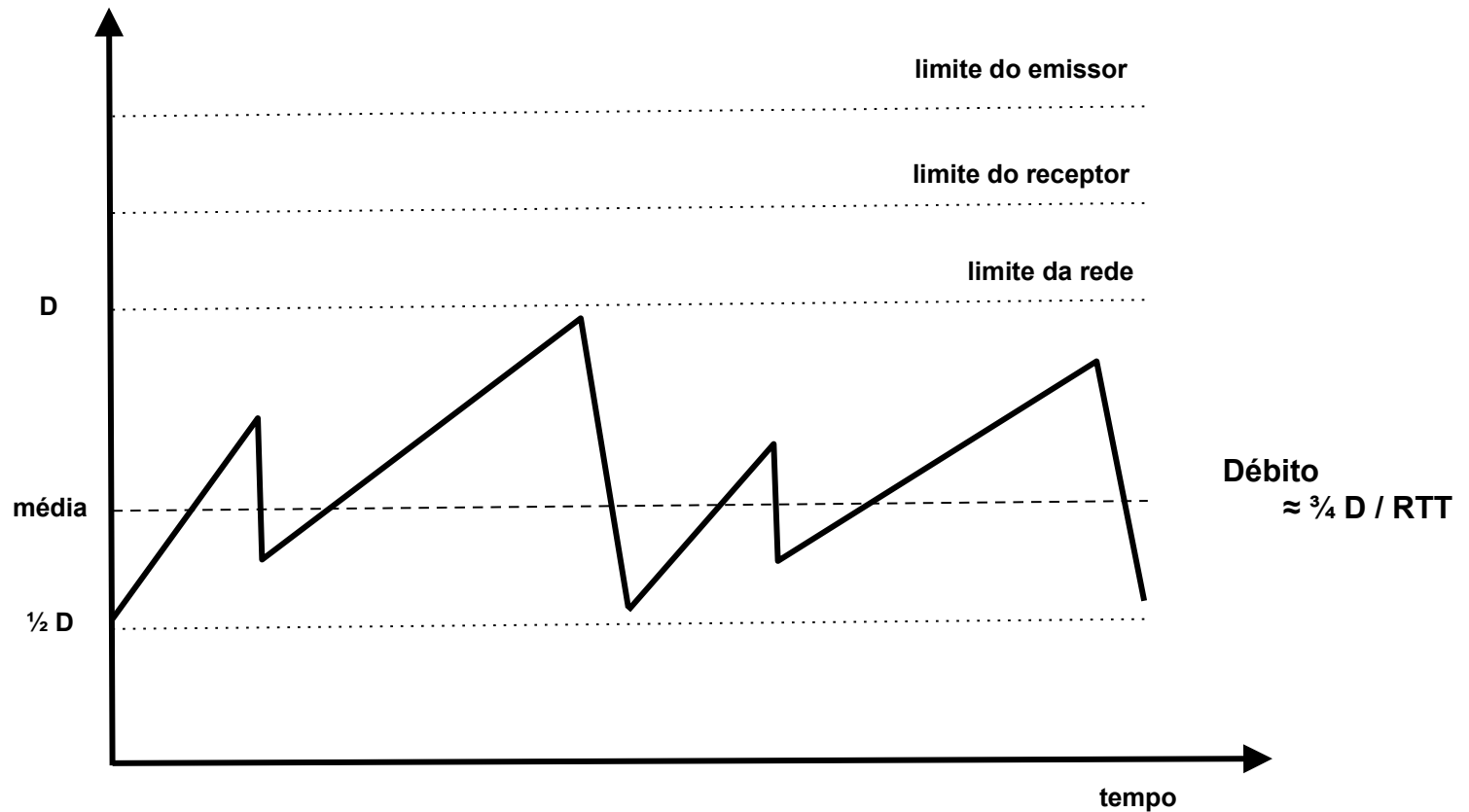
# Versão Simplificada de Reno

Event	State	TCP Sender Action	Comment
ACK receipt for previously unacked data	Slow Start (SS)	$\text{CongWnd} = \text{CongWnd} + \text{MSS}$ , If ( $\text{CongWin} > \text{sst}$ ) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
ACK receipt for previously unacked data	Congestion Avoidance (CA)	$\text{CongWin} = \text{CongWnd} + \text{MSS} * (\text{MSS} / \text{CongWnd})$	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
Loss event detected by triple duplicate ACK	SS or CA	$\text{sst} = \text{CongWnd} / 2$ , $\text{CongWnd} = \text{sst}$ , Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
Timeout	SS or CA	$\text{sst} = \text{CongWnd} / 2$ , $\text{CongWnd} = 1 \text{ MSS}$ , Set state to "Slow Start"	Enter slow start
Duplicate ACK	SS or CA	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

# TCP Congestion Control State Machine



# Débito aproximado do TCP



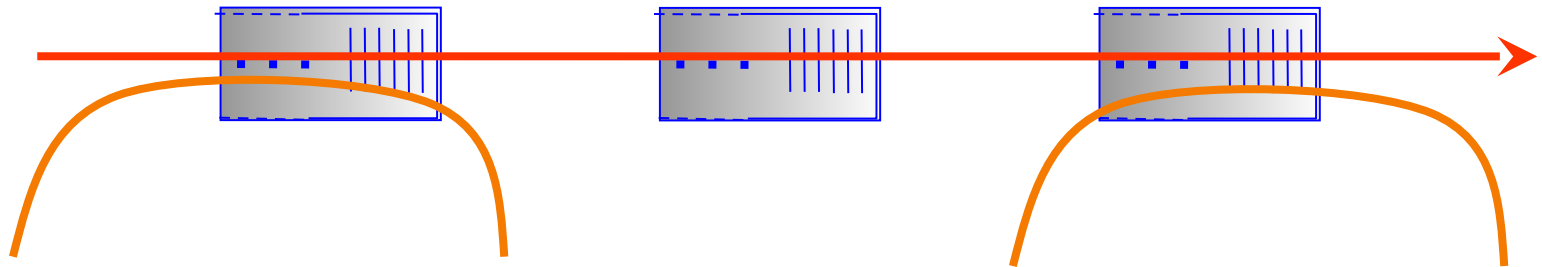
Em média, e se for o controlo de saturação a condicionar a janela do emissor, esta varia entre  $D$  e  $\frac{1}{2} D$ , logo uma estimativa será Débito médio de extremo a extremo =  $\frac{3}{4} D / \text{RTT}$

# Equidade do protocolo

- Maximizar a utilização da rede sem a saturar é um dos objetivos. O outro é ser equitativo para outros fluxos
- Verifica-se na prática e pode ser "demonstrado" que se  $N$  fluxos TCP partilham um canal gargalo, todos com o mesmo RTT, cada um obtém em média  $1/N$  do débito disponível (ver a explicação intuitiva no livro)

# Caso geral

- Quando existem diferentes fluxos TCP a usarem diferentes caminhos, a capacidade é partilhada em função do RTT pois os fluxos com o RTT mais baixo aumentam o tamanho da janela mais rapidamente



- Por outro lado, o TCP não consegue competir com fluxos (e.g. UDP) que não usam controlo de saturação

# Outros Problemas

- Pacotes perdidos por erros (canais sem fios) — `congWnd` tende para pequeno MSSs
- Tamanho inicial da janela com  $MSS = 1$  com conexões curtas (rede com capacidade mas RTT elevado e pequenos objetos)
- Perdas consecutivas (`congWnd` vai tendendo para 1 MSS) — TCP SACK é melhor nestas situações
- É possível fazer melhor num centro de dados? — tudo indica que sim
- Canais de grande RTT e elevado débito extremo a extremo

# TCP Sobre “Long Fat Pipes”

- Exemplo: segmentos de 1500 bytes, RTT de 100ms e o débito poderia ser 10 Gbps
- Precisa de uma janela  $W$  que acomode 83,333 (*in-flight*) segmentos
- *Throughput* em termos da probabilidade de perda de pacotes, [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

→ para se chegar a um *throughput* de 10 Gbps, é preciso uma taxa de erros (*loss rate*)  $L \leq 2 \cdot 10^{-10}$  - *a very small loss rate!*

- Conclusão: são necessárias alternativas para este tipo de situações

# É possível Fazer Batota?

- Sim, usando diferentes alternativas
  - Abrir várias conexões TCP em paralelo
  - Arrancar com a  $\text{congWnd} \gg 1$
  - Usar UDP mesmo quando se pretende fiabilidade
  - Modificar o mecanismo de controlo da saturação do emissor
- É possível combater isso?
  - De forma completa só se os *routers* controlassem a capacidade usada por cada fluxo



# Situação Actual

- Como é o TCP hoje em dia?

- Imensas afinações e variantes — só no Linux é possível seleccionar 10 variantes de algoritmo de controlo da saturação

- A Interoperação funciona?

- Sim, porque as variações implicam apenas com o comportamento do emissor e eventual uso de opções, sem colocar em causa os campos essenciais do cabeçalho nem modificar a sua semântica

- Versões que violassem a equidade não são aceites

# Conclusões

- A saturação é inevitável em redes *best-effort* onde não se controla o ritmo de emissão dos computadores
- O TCP usa uma solução AIMD que é fundamental para melhorar o funcionamento da Internet
- Esta solução não pode ser considerada definitiva
  - Existe o problema dos batoteiros
  - É necessita de ser afinada à medida que a Internet evolui