Controlo de Congestão

FEUP/MRSC/AMSR
MPR

Bibliografia

» Aula preparada com base nos seguintes documentos

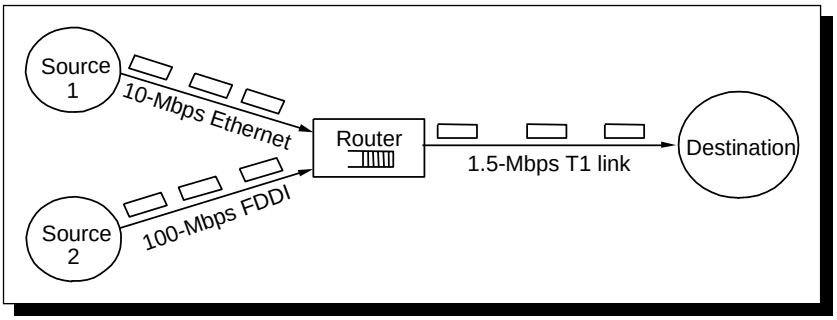
- L. Peterson, B. Davie, “Computer Networks – A Systems Approach”, Morgan Kaufamnn, 200 (Sec. 6.1, 6.2, 6.3 e 6.4)
- Acetatos do autor do livro, L. Peterson, “Congestion Control”
- V. Jacobson, M. Karels, “Congestion Avoidance and Control”, 1988
- “TCP Congestion Control”, RFC 2581, 1999

Introdução

- Disciplinas de serviço
- Reacção à congestão
- Evitar a congestão

Congestão

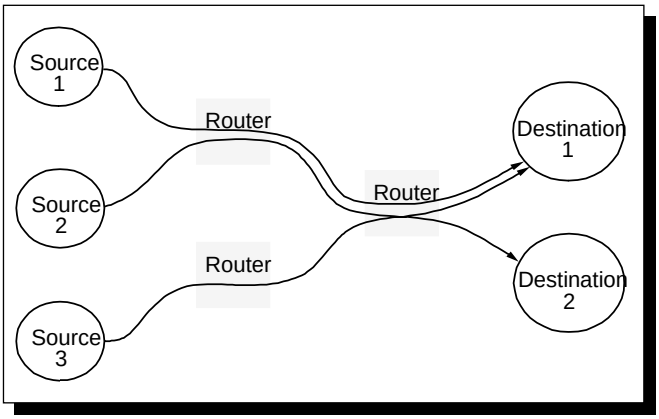
- Duas perspectivas
  - » Reserva de recursos para evitar congestão
  - » Controlo da congestão se e quando esta ocorre



- 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

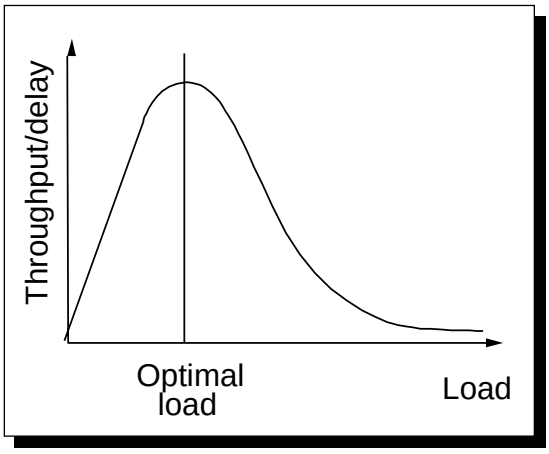
Conceitos Base

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



Avaliação de Distribuição de Recursos

- Justiça
- Potência (débito/atraso)



Disciplinas de Serviço

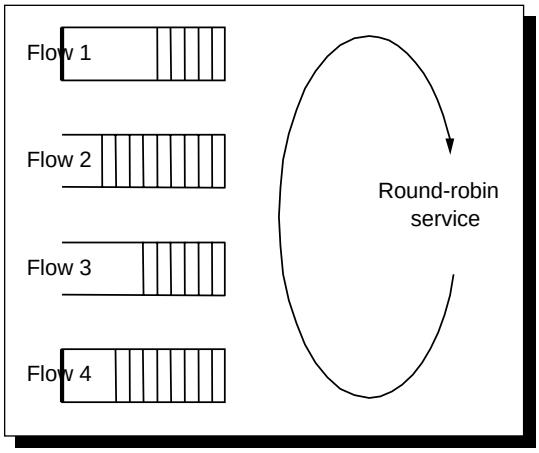
» First-In-First-Out (FIFO) → Não há distinção de fluxos

» 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})

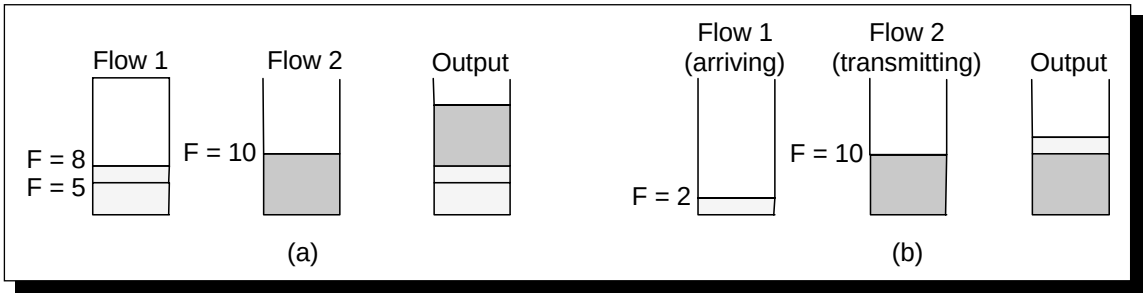
» Se não há pacote do fluxo em transmissão

– Transmite quando pacote **i** chegar (A_i)

» $F_i = MAX (F_{i-1}, A_i) + P_i$

Algoritmo FQ

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



TCP - Controlo de Congestão

- 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 congestão diminui ➔ aumenta **`CongestionWindow`**
  - » Se congestão aumenta ➔ diminui **`CongestionWindow`**

Subida Aditiva/Descida Multiplicativa

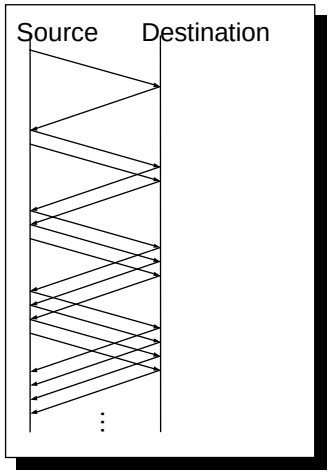
- Como sabe a fonte se/quando a rede está congestionada?

➔ 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 ➔ congestão
  - – Filas nos routers cheias

Subida Aditiva/Descida Multiplicativa

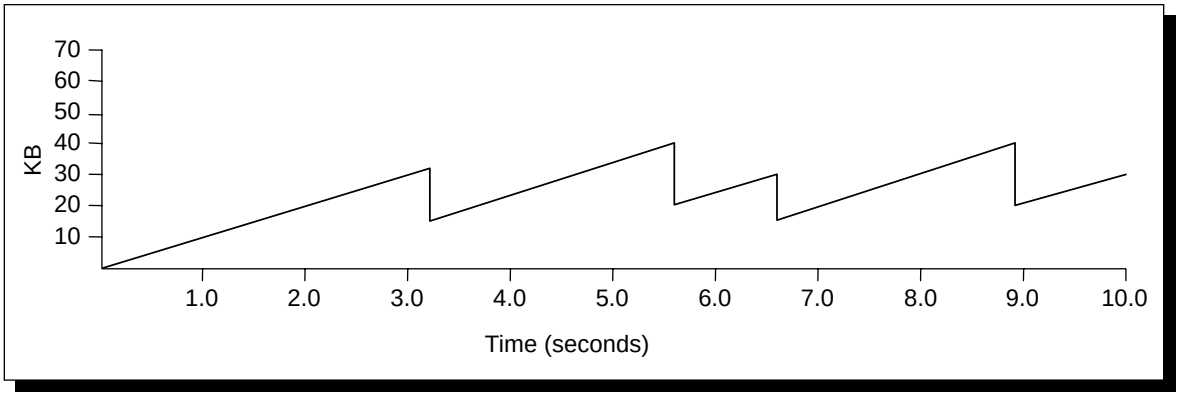
- Algoritmo
  - » Incrementa **`CongestionWindow`** de **`1`** pacote
    - – Por cada **RTT** (Round Trip Time) ➔ Subida linear
  - » Divide **`CongestionWindow`** por **2**
    - – Sempre que há perda de pacote ➔ Descida multiplicativa



- Na prática,
  - » Incrementa ligeiramente por ACK recebido
  - » **`Increment= MSS * (MSS / CongestionWindow)`**
  - » **`CongestionWindow += Increment`**
  - » **`MSS → Maximum Segment Size`**

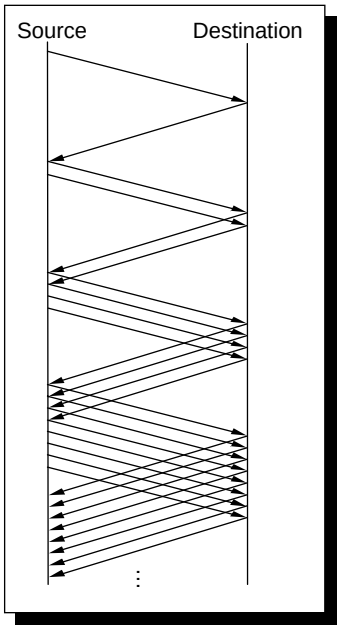
Subida Aditiva/Descida Multiplicativa

- Funcionamento **`Dente de Serra`**



Arranque Lento ☺

- Objectivo
  - » Determinar capacidade de transmissão disponível



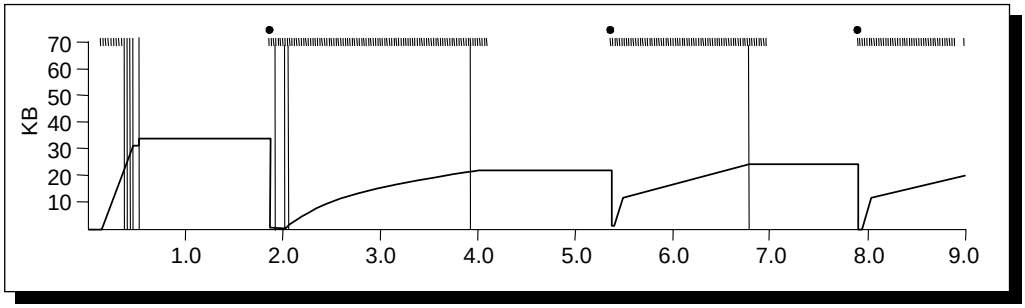
- Aproximação
  - » Começar com **`CongestionWindow = 1 pacote`**
  - » Duplicar **`CongestionWindow em cada RTT`**

Arranque Lento

- Crescimento exponencial mas, no início, menor que máximo

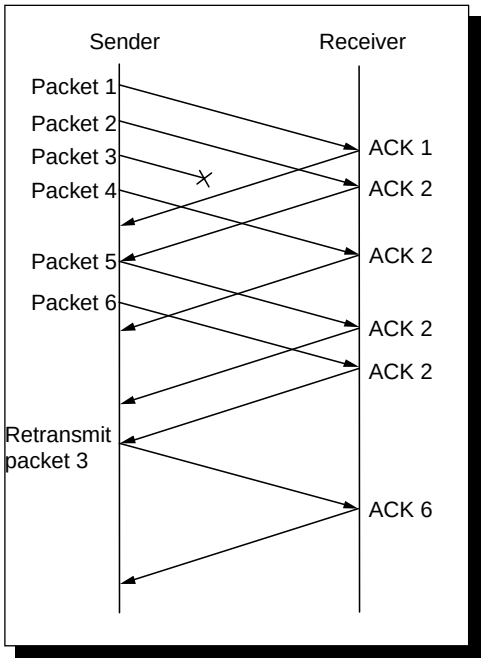
- Usado
  - » No início da ligação
  - » Quando a ligação fica morta **`(AdvertisedWindow=0)`** em espera de timeout

- Problema → perda de informação sobre a nova (1/2) **`CongestionWindow`**



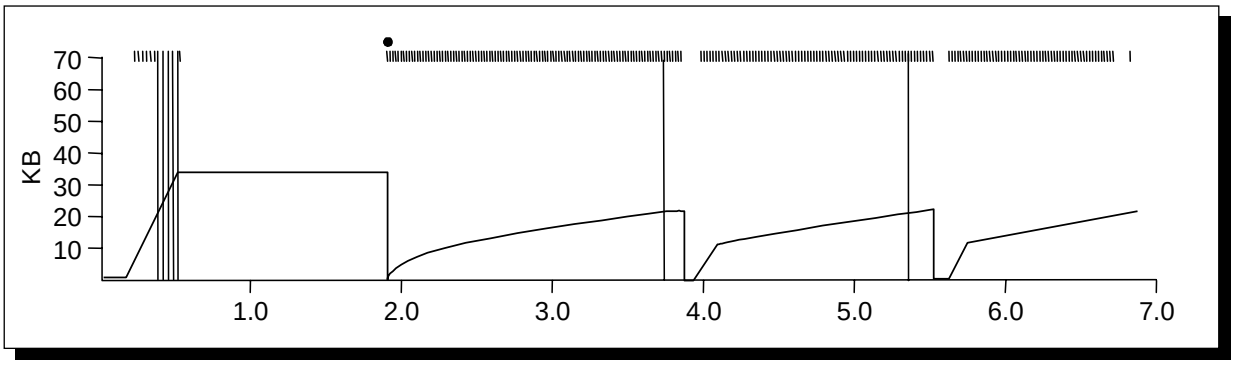
Retransmissão e Recuperação Rápidas

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



Resultado

- Recuperação rápida
  - » Evita fase de arranque lento
  - » Salto para **½ * CongestionWindow (ssthresh)**



Evitar a Congestão

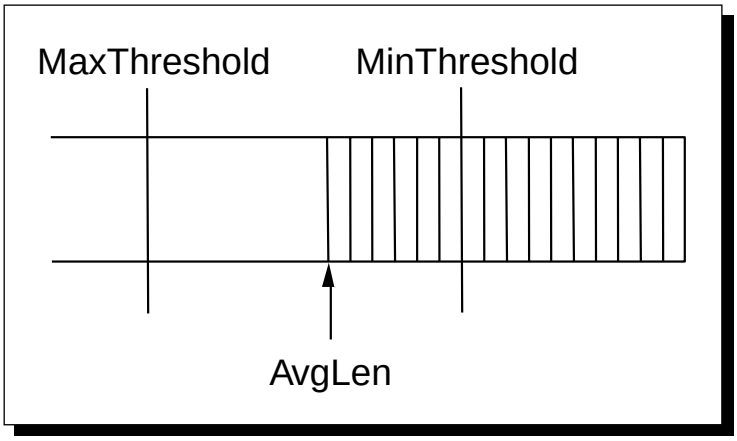
- Estratégia do TCP
  - » Controlo imediato da congestão
  - » Incrementa sistemáticamente a carga
    - – Detecta ponto de congestão → diminui carga
- Método alternativo
  - » Previsão do ponto de congestão
  - » Reduz débito antes de pacotes começaram a ser perdidos
  - ➔ Evitar a congestão (em vez 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



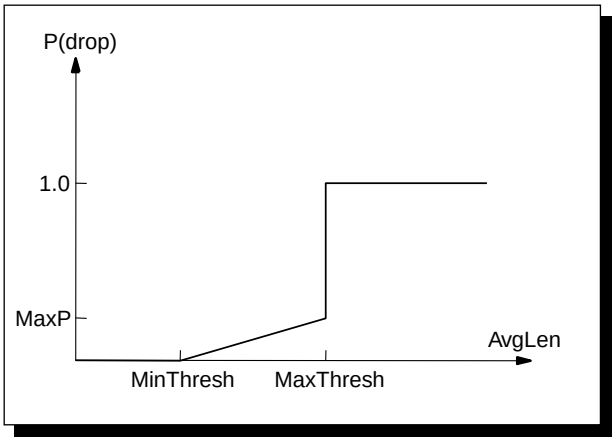
RED – Limiares na Fila

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

RED – Cálculo da Probabilidade P

```
TempP=MaxP*(AvgLen-MinThreshold)/(MaxThreshold-MinThreshold)
P = TempP/(1-count*TempP)
```

» Curva de probabilidade de eliminação de pacote

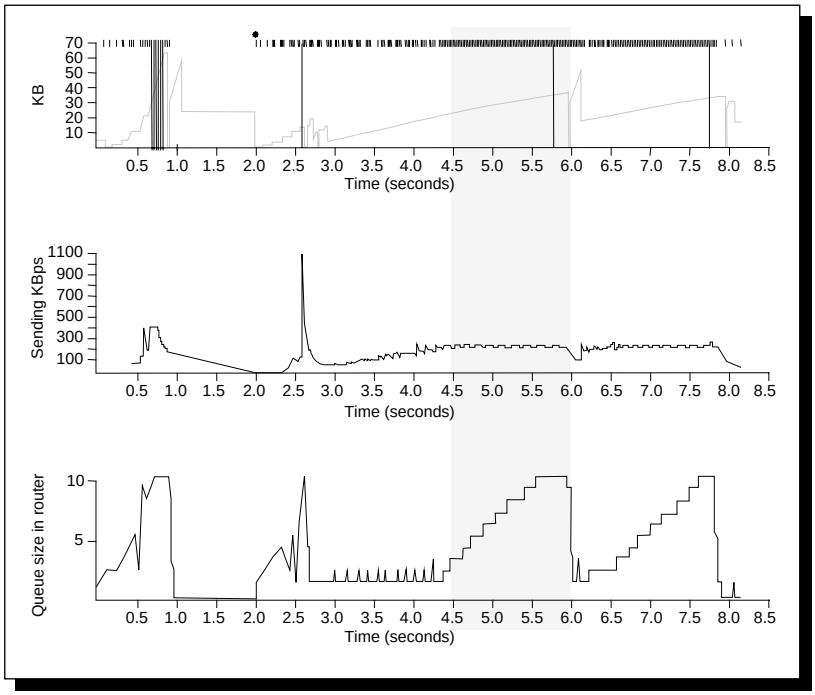


RED

- 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 = ½ do intervalo entre os limiares
    - ➔ eliminado 1 em cada 50 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