REDES DE COMPUTADORES 2021/2022

aula 0100 – Stop&Wait e Janela Deslizante

03/03/2022

Pedro Patinho <pp@di.uevora.pt>

(Baseado nos slides do prof. José Legatheaux Martins)

Objectivos

- A Internet não garante a entrega dos pacotes pois pode perdê-los ou entregá-los de forma desordenada
- Torna-se então necessário encontrar uma forma de transferir dados de forma fiável entre dois computadores
- Nesta aula veremos o protocolo mais simples que existe para o realizar – o protocolo *stop & wait*

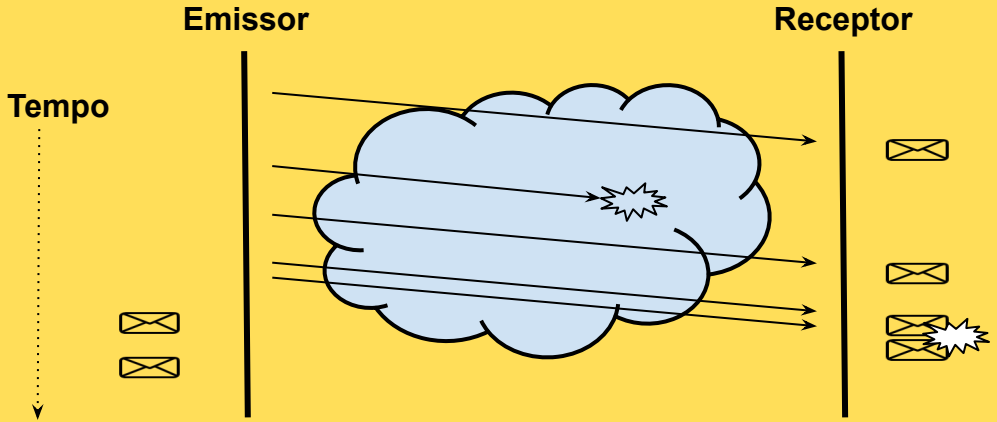
Hipóteses e Objectivo

- Nos canais, os *frames* com erros são recusados pelos mecanismos de controlo de erros e assim os erros são transformados em perdas (omissões)
- A rede também pode introduzir perda de pacotes ou alteração da ordem de entrega dos mesmos
- A rede pode perder pacotes, mas não os perde a todos, mais tarde ou mais cedo alguns pacotes vão chegar ao receptor
- Objectivo: pretende-se transferir dados de forma fiável e maximizar a taxa de transferência, ou débito, extremo a extremo

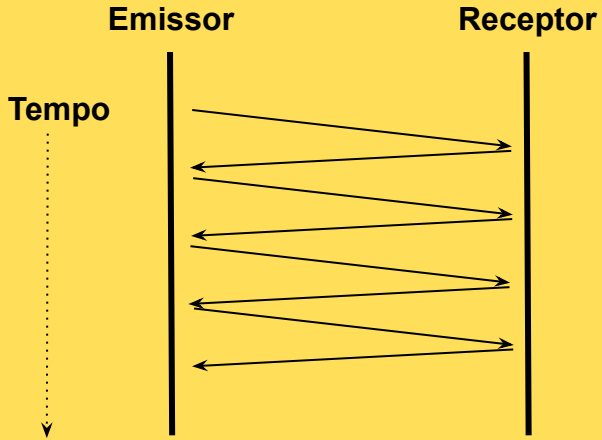
Problemas

É necessário controlo de erros e controlo de fluxo extremo a extremo.

O controlo de erros deverá mascarar a perda de pacotes e a sua chegada fora de ordem. O controlo de fluxos deverá impedir um emissor demasiado rápido de "afogar" um receptor lento.

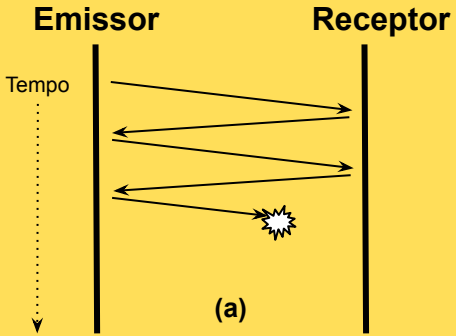


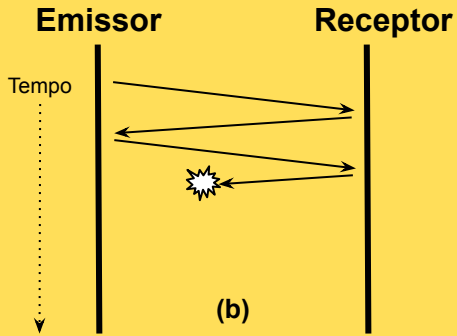
Confirmações de Recepção (ACKs)



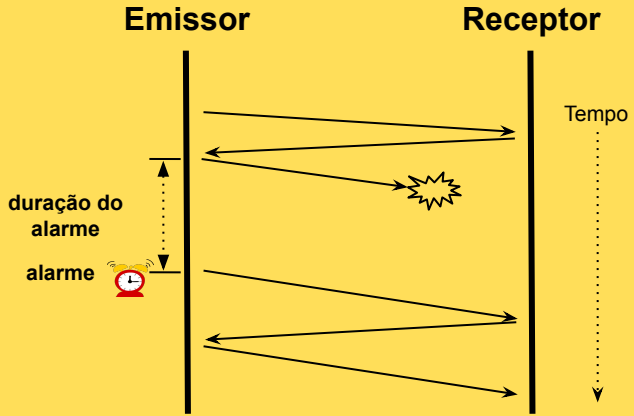
ACK = *Acknowledgement* (confirmação ou aviso de recepção = podes continuar)

Perda de Pacotes



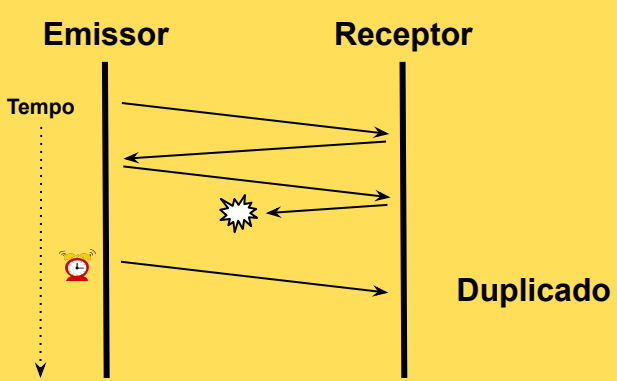


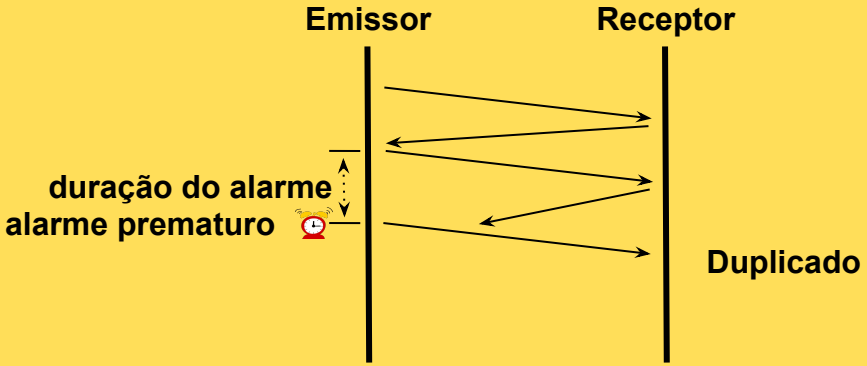
Alarmes Temporizados



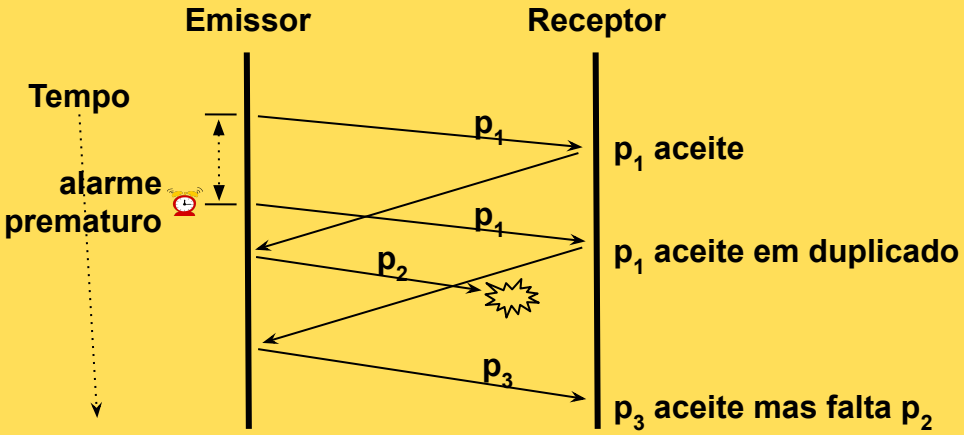
A terminologia usada em inglês nas redes de computadores e nos sistemas distribuídos corresponde ao momento de fim do temporizador do alarme: *timeout*

Anomalias e Alarmes





Anomalias Conjugadas



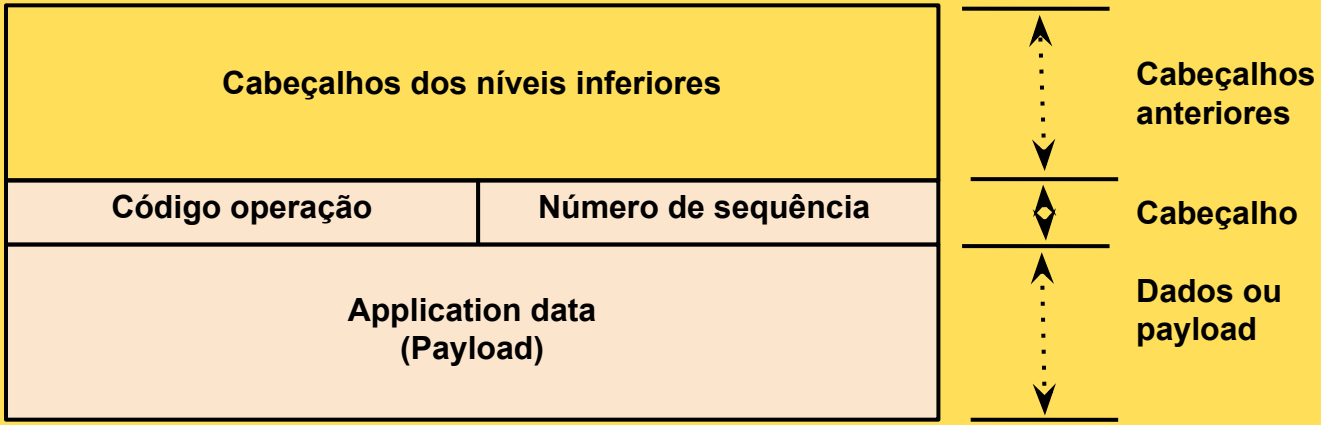
Problemas ainda mal resolvidos

- A solução anterior não resolve ainda o problema da perda de um ACK, a qual pode conduzir à aceitação de um pacote em duplicado
- O mesmo problema poderá ser introduzido por um receptor lento (face ao *timeout*) a enviar o ACK
- Um alarme (*timeout)* mal regulado, muito curto por exemplo, poderá conduzir ao mesmo problema
- Na verdade as velocidades relativas do emissor e receptor podem não ser constantes, nem conhecidas a priori
- Note-se que uma duração de alarme muito elevada, uma solução simplista, leva a uma recuperação demasiado lenta de uma perda

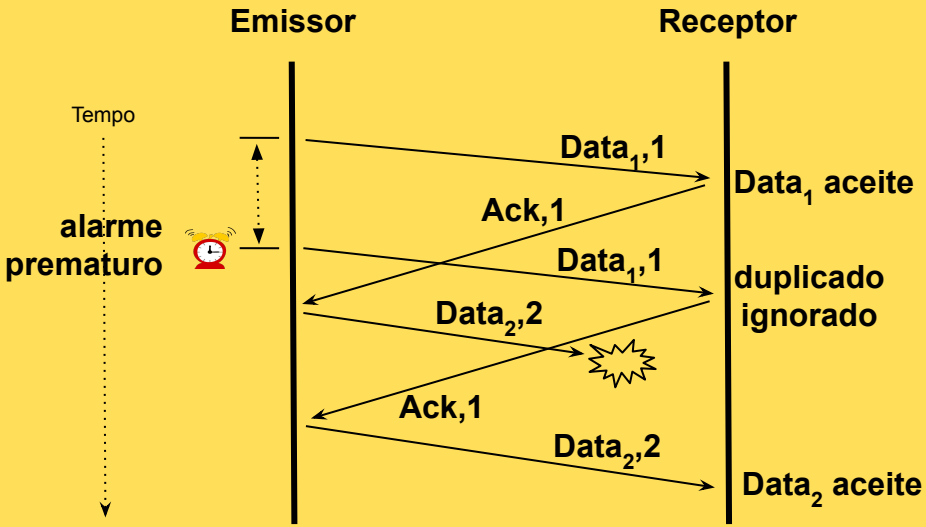
Números de sequência

- Uma solução consiste em introduzir números de sequência únicos para cada pacote. Estes números permitem ao receptor distinguir dados esperados, dados repetidos, etc.

- Tradicionalmente, para se poupar espaço nos cabeçalhos, interessa minimizar o número de bits a usar. O número de sequência pode então ser reutilizado ciclicamente desde que não se introduza confusão caso a ordem das mensagens possa ser trocada



Protocolo *stop & wait*



Funcionamento

Emissor:

- Por cada pacote a transmitir, dá-lhe um número de sequência novo, transmite-o e activa um alarme. Depois:
  - Se chega um ACK com o número de sequência esperado – passa adiante
  - Se o temporizador dispara – reenvia o pacote e activa de novo o alarme
  - Se chega um ACK com número de sequência errado – ignora-o

Receptor:

- Sempre que recebe um pacote:
  - Envia um ACK do último pacote correctamente recebido. Se o pacote é novo guarda-o, senão ignora-o

Correcção (*Safety*)

- Admitindo que a rede mais tarde ou mais cedo consegue que uma mensagem seja entregue ao receptor, isto é, que a rede permite sempre algum progresso *mesmo que pequeno*:
- O protocolo garante (tente demonstar):
  - Que cada pacote é corretamente entregue ao receptor
  - Que esse pacote não é confundido com outro
  - Que todos os pacotes chegarão ao receptor

Regulação dos Alarmes

- O valor do temporizador do alarme (*timeout)* é importante para a recuperação de erros
  - Um valor demasiado grande leva a que o emissor seja lento a recuperar dos erros
  - Um valor demasiado curto leva a duplicados inúteis
  - Em qualquer caso a correção não está em causa desde que se use sempre um alarme e o receptor envie sistematicamente um ACK perante todos os pacotes recebidos, maus ou bons
- Para evitar repetições inúteis o valor pode ser folgado
  - É possível tentar estimar um valor mais adequado
  - Pense como

Desempenho sem Erros

- Dois computadores A e B ligados por um canal directo com 10.000 Km de comprimento e o débito de 1 Mpbs
- Pacotes com 1.000 bytes
- Ignoram-se os bits dos cabeçalhos
- Ignora-se o tempo de transmissão de um ACK
- Ignora-se o tempo de processamento

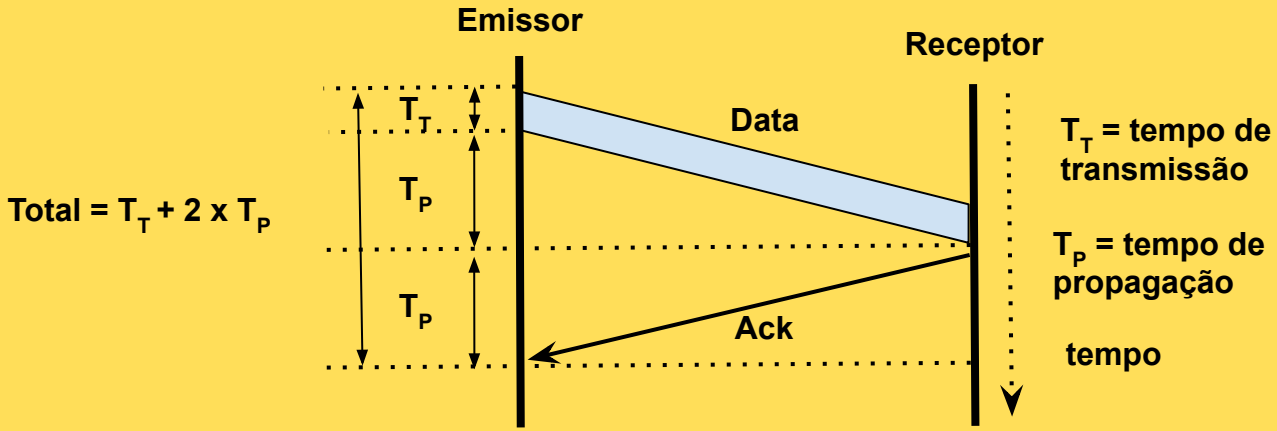
Tempo de propagação = 10.000 / 200.000 = 50 ms, RTT = 100 ms

1000 bytes = 8.000 bits = $8 . 10^3$

Tempo de transmissão = $8 . 10^3 / 10^6 = 8$ ms

- Quanto tempo leva o emissor a colocar cada pacote no receptor ?
- Qual a diferença para um protocolo em que o emissor estivesse sempre a emitir?

Desempenho do Protocolo Stop & Wait



Cada pacote leva 8 ms a transmitir, mas em cada ciclo de 108 ms apenas se transmite um pacote

1.000 pacotes levam 108 segundos a transferir ao invés de 8 segundos

O desempenho real é 8/108 ≈ 7,4% do máximo teórico

Taxa de Utilização do Canal

$$T_u = T_t / (T_t + 2 \times Tp)$$

ou

$$T_u = T_t / (T_t + RTT)$$

T_u – taxa de utilização
T_t – tempo de transmissão de uma mensagem,
T_p – tempo de propagação de extremo a extremo
RTT – tempo de ida e volta (2 x T_P)

Débito, RTT e Taxa de Utilização

Taxa de utilização de um canal pelo protocolo stop & wait numa transferência entre dois computadores ligados directamente por um canal sem erros, usando pacotes de 10.000 bits, variando o débito e o RTT.

Taxa Utilização	Débito (Mbps)	T_T (ms)	RTT (ms)	Comentários
99%	1	10	0,1	Débito baixo, RTT desprezável
50%	100	0,1	0,1	Débito razoável, RTT desprezável
33%	1	10	20	Débito baixo, RTT baixo
0,5%	100	0,1	20	Débito razoável, RTT baixo
4,8%	1	10	200	Débito baixo, RTT alto
0,05%	100	0,1	200	Débito razoável, RTT alto
0,005%	1000	0,01	200	Débito alto, RTT alto

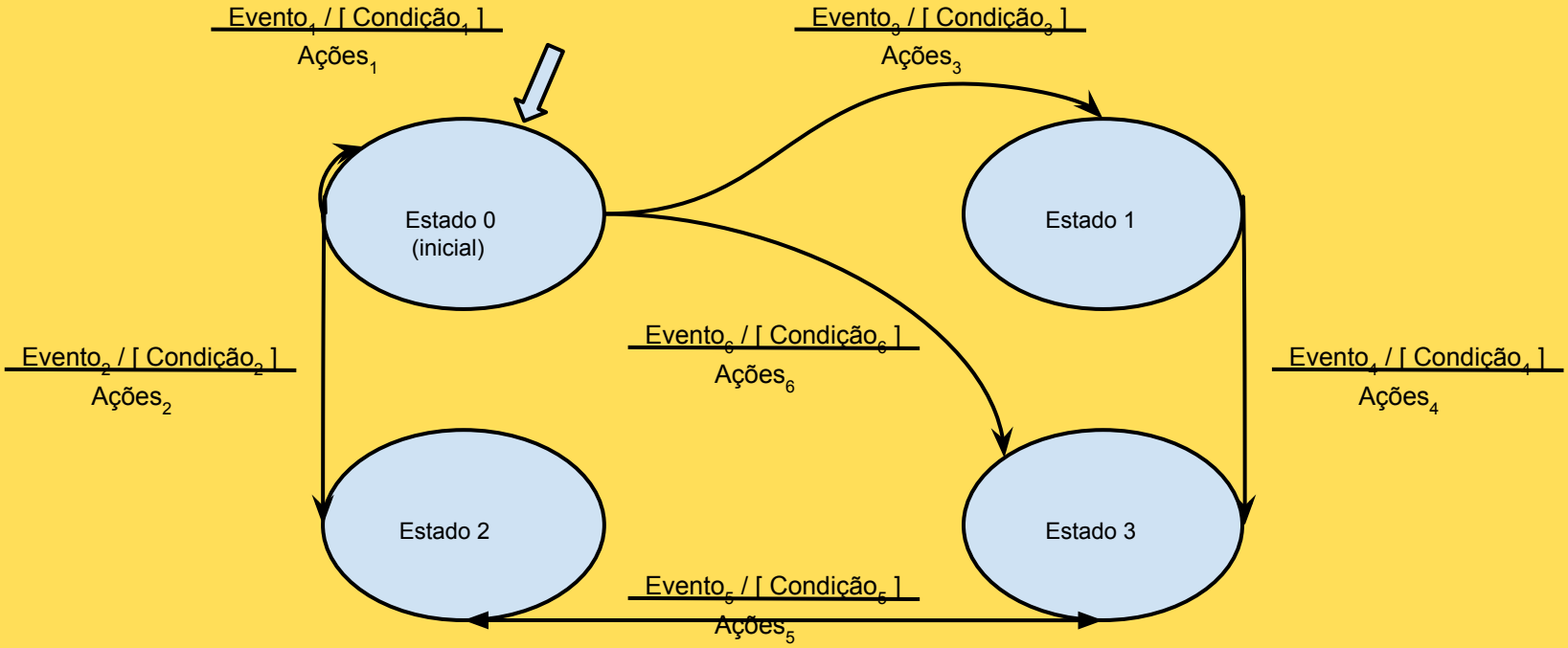
Canais de Débito Elevado

- Quando os canais têm débito muito elevado, o tempo de transmissão diminui drasticamente. Por exemplo, se um canal tem a capacidade de 1 Gbps, transmitir 10.000 bits leva 10^{-5} segundos, isto é, 10 micro segundos
- Se o RTT for de alguns milissegundos, a taxa de utilização do canal tende sempre para valores muito baixos. Nestes casos, o débito útil médio extremo a extremo permitido pelo protocolo tende para:

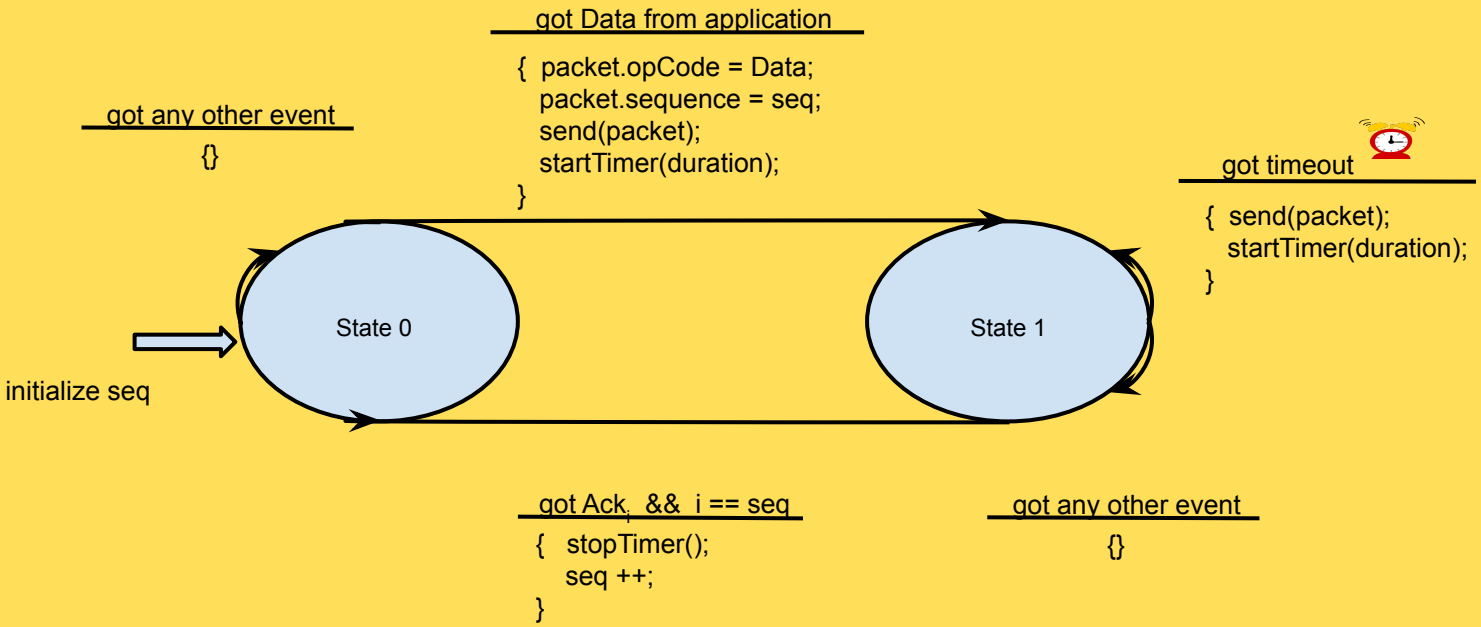
Débito útil médio extremo a extremo do protocolo S&W

= Dimensão do pacote / RTT

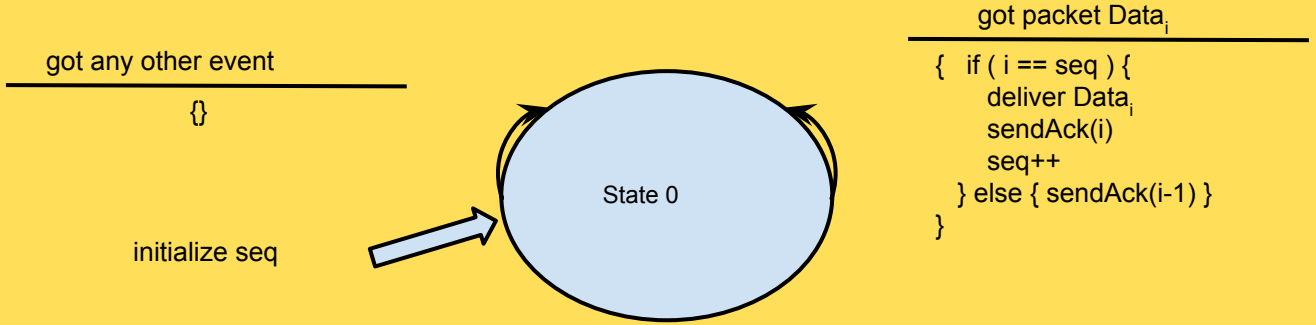
Máquinas de Estados com Acções



Máquinas de Estados com Acções
Emissor Stop & Wait



Máquinas de Estados com Acções
Receptor Stop & Wait



Conclusões

- Mesmo que um canal ou a Internet tenham erros e percam pacotes
  - É possível realizar um protocolo de transferência fiável de dados nos extremos
  - Acabámos de ver um, também conhecido por *stop & wait*

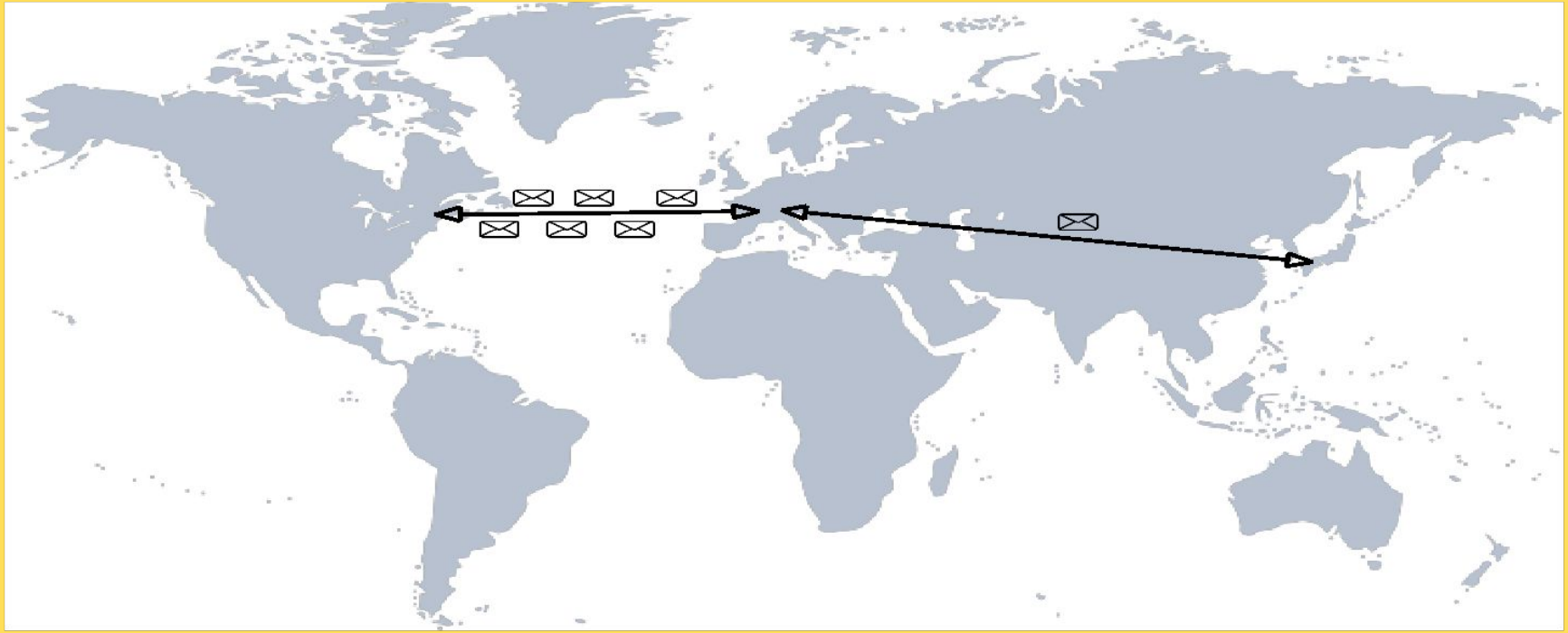
- Infelizmente tem um desempenho fraco quando o tempo de trânsito extremo a extremo é significativo face ao tempo de transmissão
  - O emissor avança ao ritmo permitido pelo RTT, pois
  - O receptor não pode emitir um novo pacote enquanto não chegar o ACK do último pacote emitido

Protocolos de Janela Deslizante

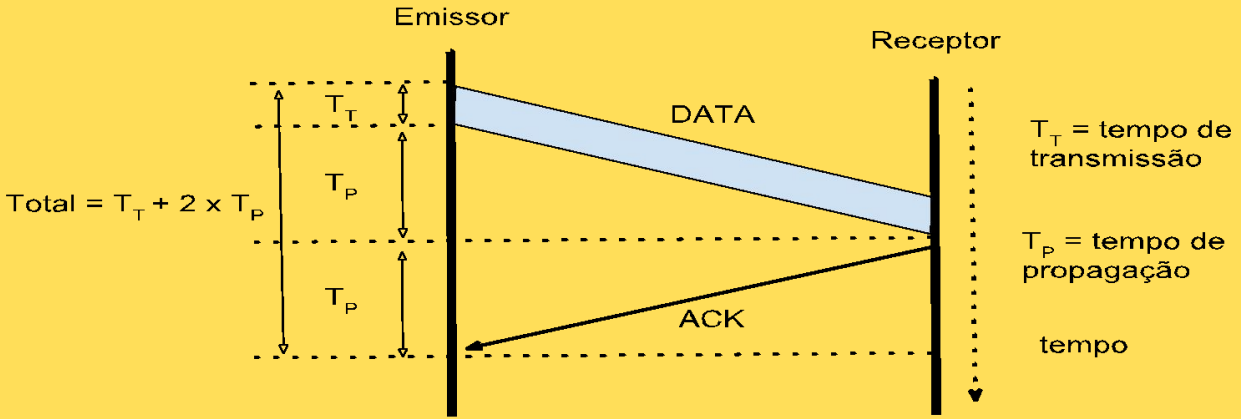
Proposta de melhoria

- É possível melhorar o protocolo stop&wait usando uma técnica chamada janela deslizante *(sliding window*) ou *pipelining*

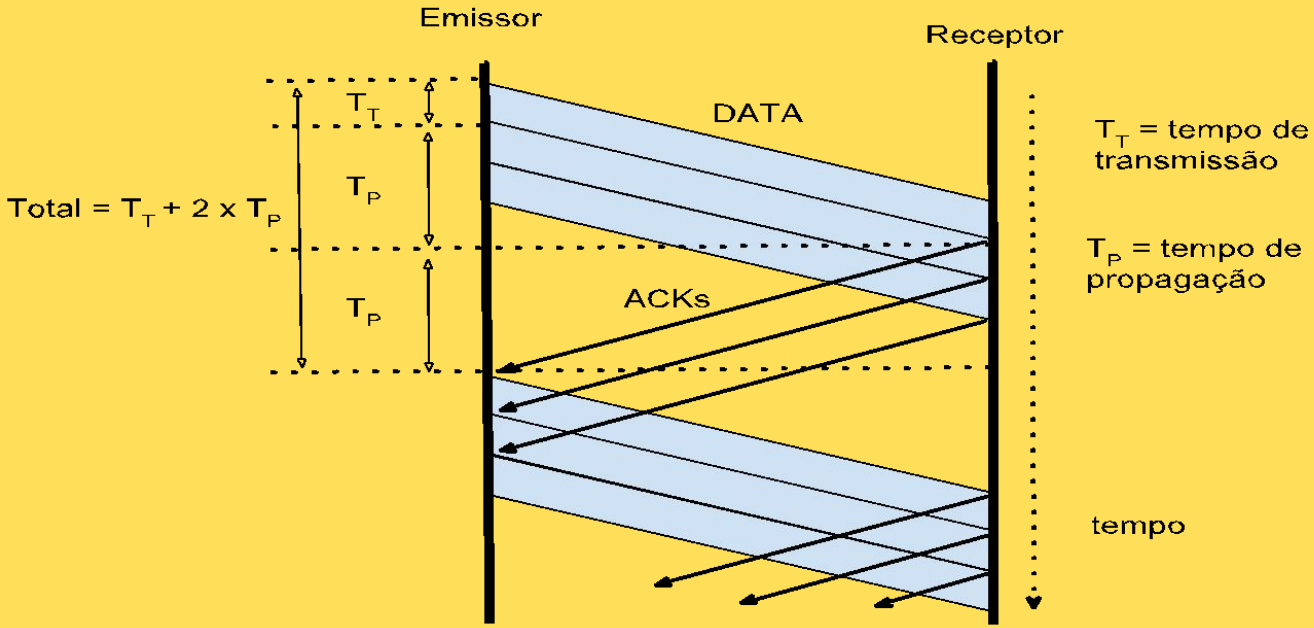
O Problema



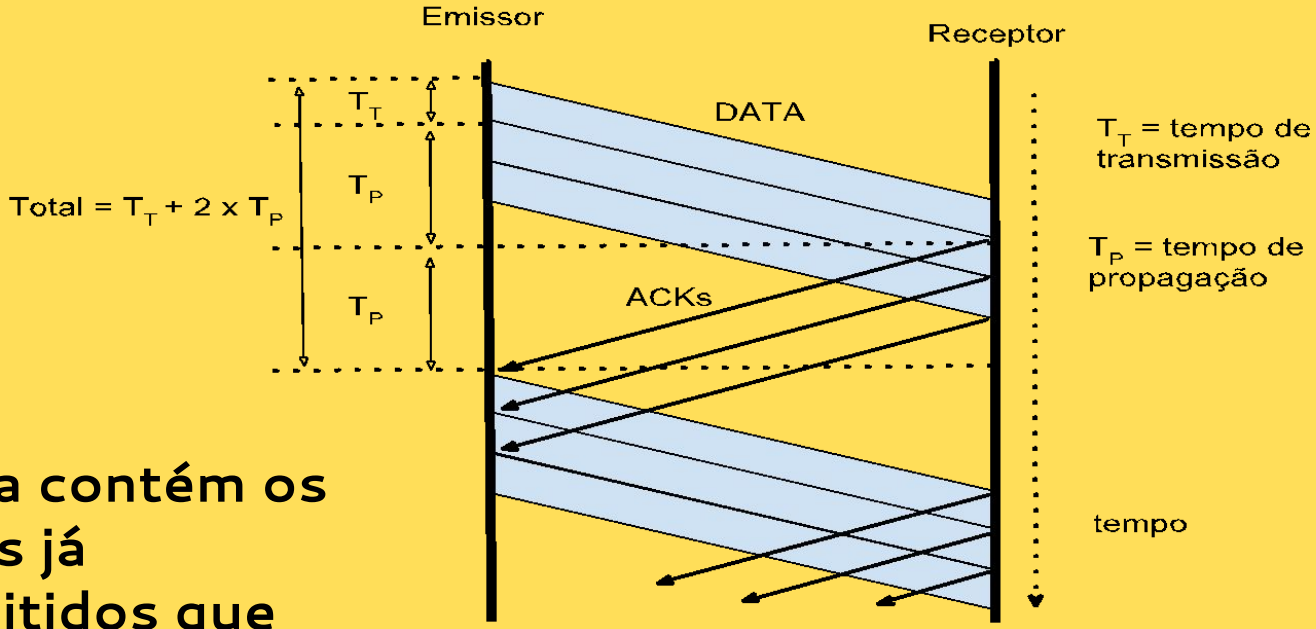
Desempenho do Protocolo S&W



Solução: Transmitir "Adiantado"

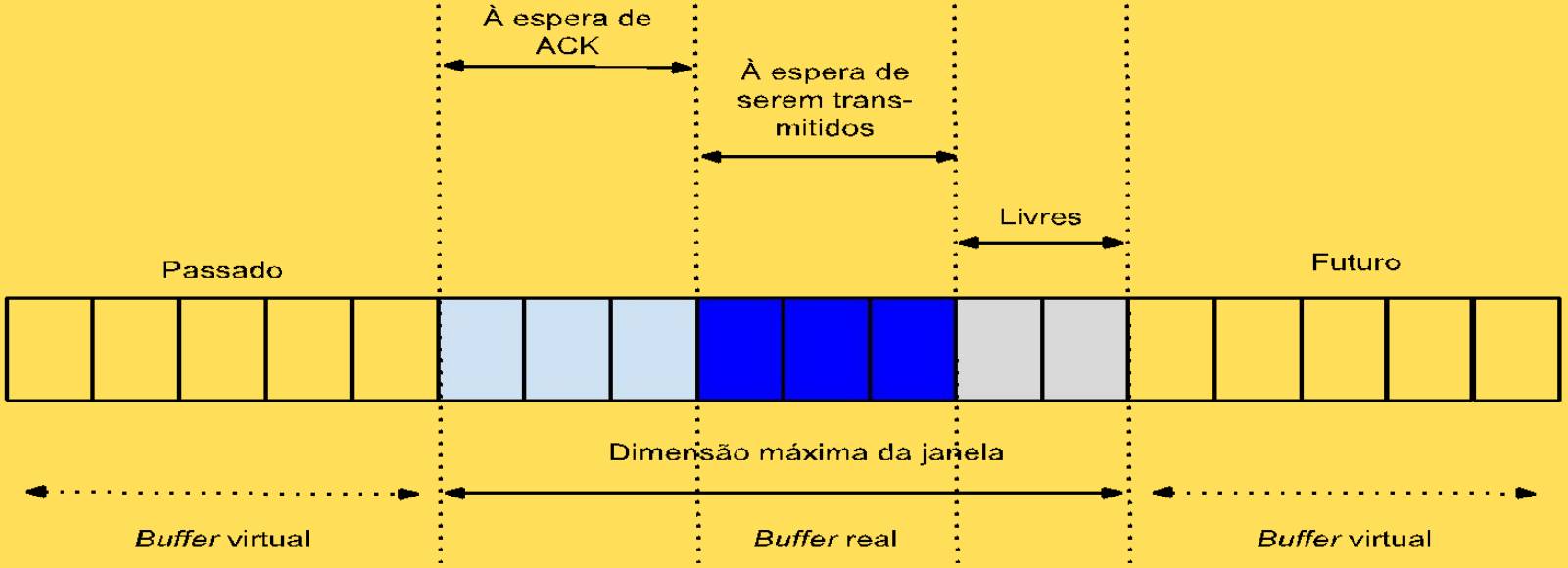


Janela Deslizante



A janela contém os pacotes já transmitidos que ainda não foram ACK'd. No protocolo stop & wait a janela é

Funcionamento da Janela



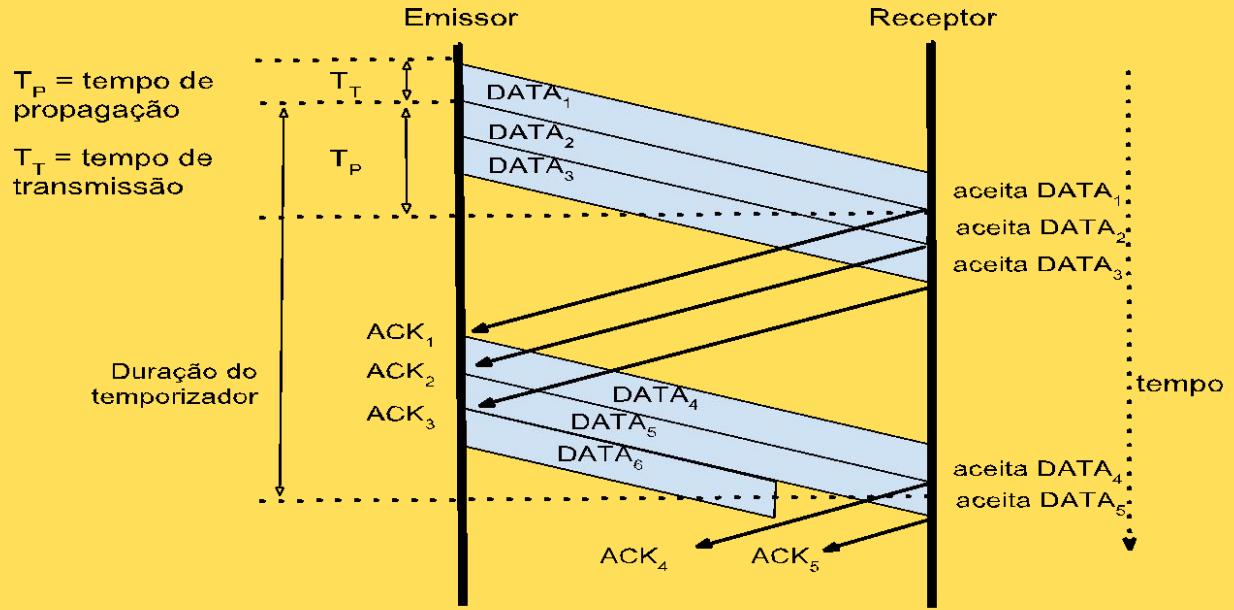
Funcionamento da Janela

- O emissor tem um *buffer* (a janela) onde estão os pacotes já transmitidas e que ainda não foram *ACK'ed,* assim como os pacotes que estão à espera de serem transmitidas mas ainda não houve tempo

- Quando chegam ACKs, a janela desliza para a direita e os pacotes já *ACK'd* podem ser esquecidos pois já se tem a certeza que foram bem recebidos

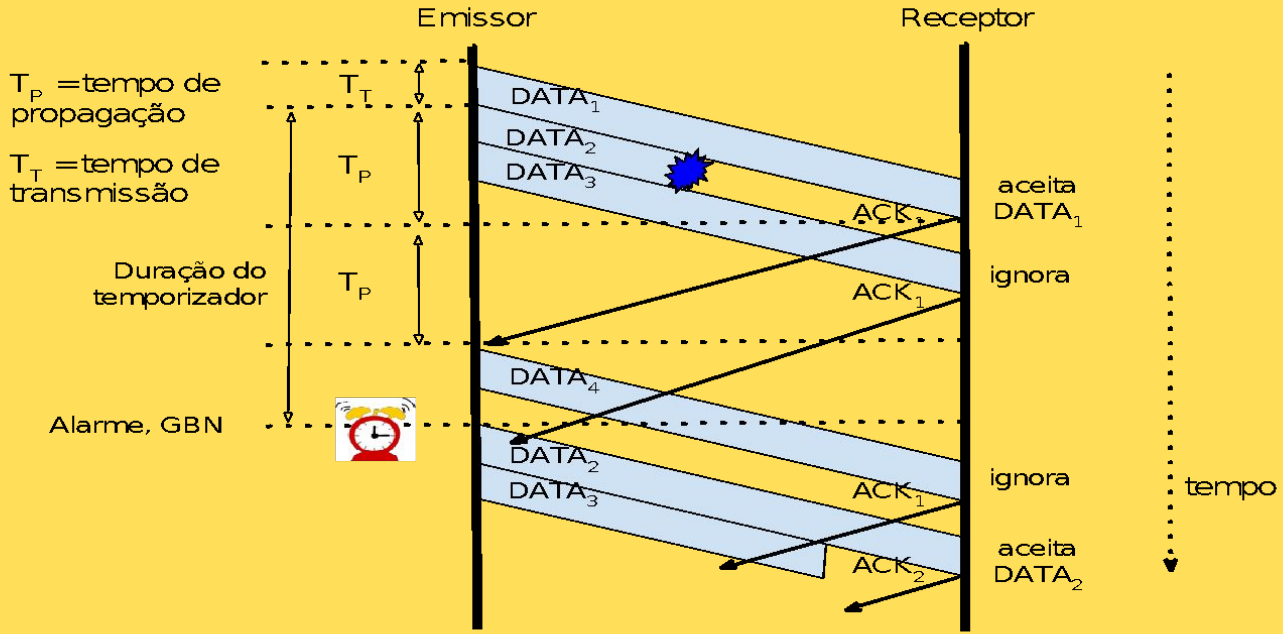
- A dimensão da janela é limitada pelas seguintes razões: controlo de fluxo, eventual desperdício em caso de perda e controlo da saturação da rede (ver a seguir)

Receptor com Janela para um Pacote



Uma janela para um pacote no receptor é suficiente caso a aplicação consuma muito rapidamente os pacotes chegados

Mas se Há Erros



não há espaço no receptor para os pacotes fora de ordem, pelo que o emissor tem de voltar

Go-Back-N (GBN)

- Como o receptor não guarda os pacotes fora de ordem, em caso de anomalia o emissor não recebe o ACK adequado, e só lhe resta recomeçar a emissão por ordem de todos os pacotes a partir do pacote cujo ACK faltou, ou seja, o mais à esquerda da janela.

- Existe sempre um alarme associado ao pacote emitido mais antigo, que terá de ser atualizado sempre que este pacote é ACK'ed

- Existem outras formas de detetar antes do disparo do alarme que um pacote se perdeu (ver a seguir)

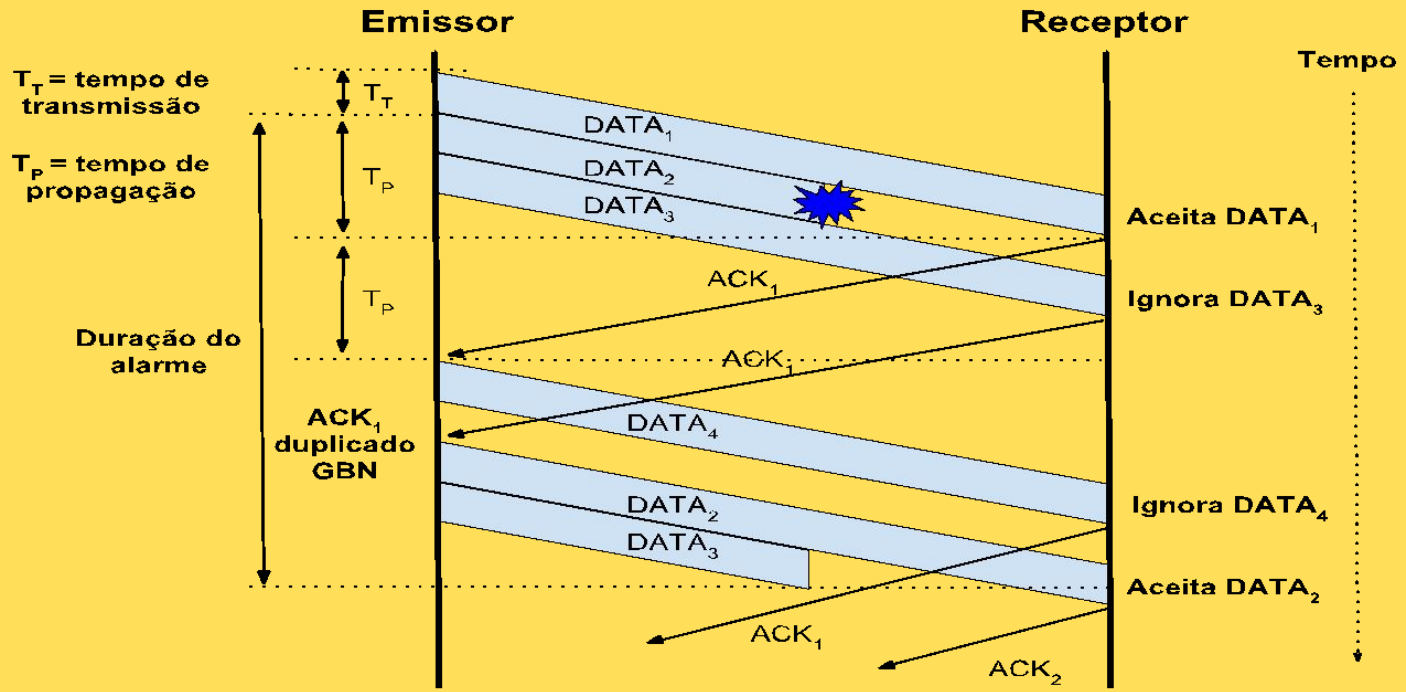
GBN e Recuperação

- Quanto mais cedo o emissor detetar que se perdeu um pacote melhor, pois isso evita continuar a emitir pacotes que vão ser deitados fora

- Existem pelo menos duas formas de o emissor detetar que um pacote se perdeu mesmo que o alarme ainda não tenha disparado

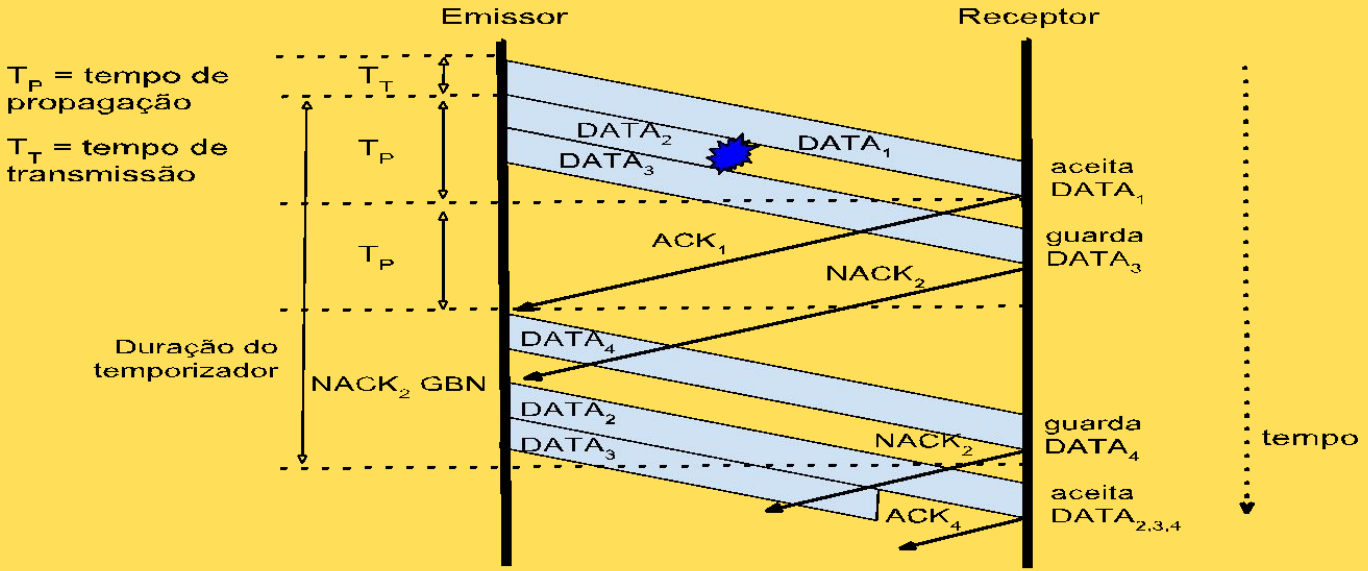
  - Interpretar os ACKs duplicados como sinal de que um pacote se perdeu
  - O receptor emitir uma indicação explícita (um NACK) de que um pacote se perdeu

GBN Com Recuperação Mais Rápida



Um ACK cumulativo repetido pode ser interpretado como um sinal de que um pacote se perdeu e serve para entrar em GBN mais cedo

Janela do Receptor Maior Que Um Pacote

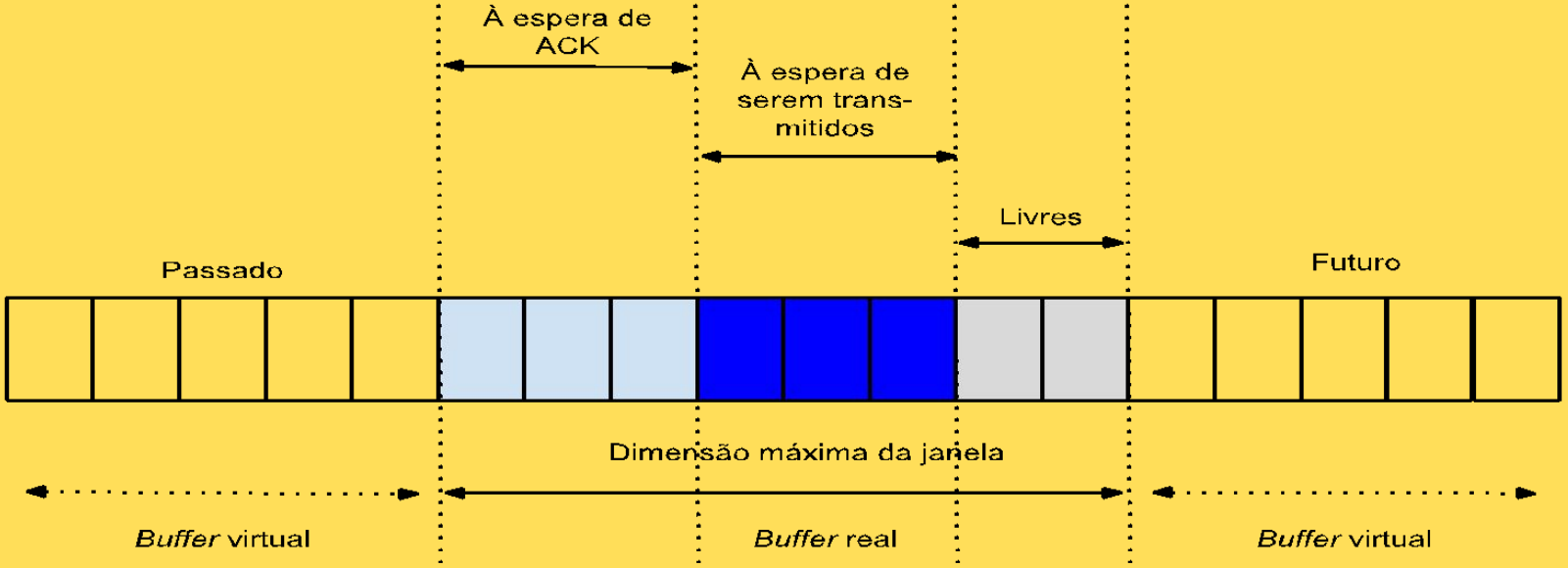


Uma janela do receptor maior que 1 também ajuda

Acções Executadas pelo GBN

- Por cada mensagem n recebida, o receptor
  - Guarda o pacote se este estiver na ordem
  - Se estiver fora de ordem, mas tiver a janela de recepção > 1 tenta guardá-lo também
  - Em qualquer caso envia um ACK cumulativo de tudo o que recebeu até aí na ordem, e opcionalmente pode enviar (também) um NACK
  -
- O emissor arma um alarme (*timeout)* associado ao pacote mais antigo enviado
- Sempre que o *timeout* dispara no emissor, ou este recebe um NACK, volta atrás e recomeça a enviar os pacotes na janela
- Quando o emissor recebe um ACK "útil", isto é, à esquerda da janela, faz avançar a janela e reajusta o valor do alarme

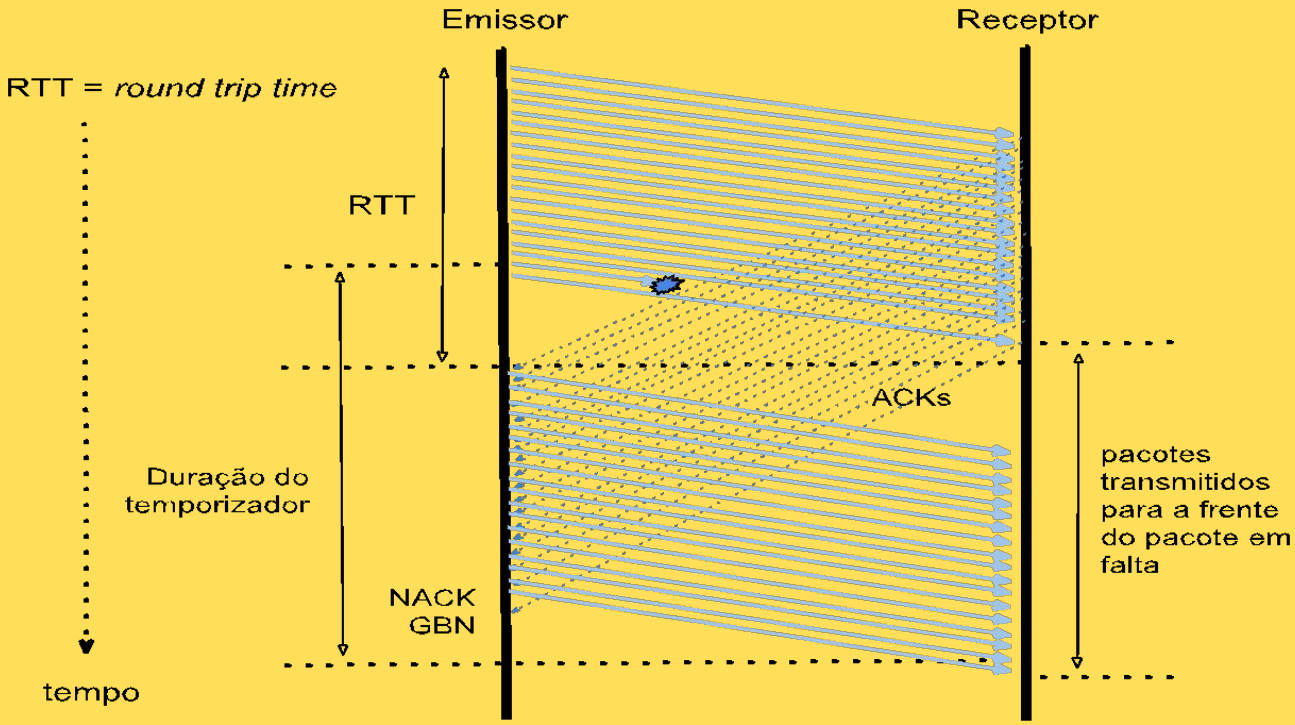
Janela do Emissor



Podemos Melhorar?

- Quando o RTT é muito baixo, ou o valor do Tt / RTT se aproxima de 1, uma janela de emissão relativamente pequena é suficiente para manter o emissor quase sempre a emitir
- Se houverem erros de transmissão (admitamos que não são frequentes) o funcionamento GBN introduz atrasos e eventualmente emissões em duplicado. A penalização, se existir, é proporcional ao tamanho da janela
- Se o valor de Tt é muito pequeno (canal de alta capacidade) e o RTT é muito grande (canal muito extenso), a janela do emissor tem de ser muito grande e o funcionamento GBN é muito penalizante se houverem erros, mesmo que espaçados

Custo da Recuperação com GBN



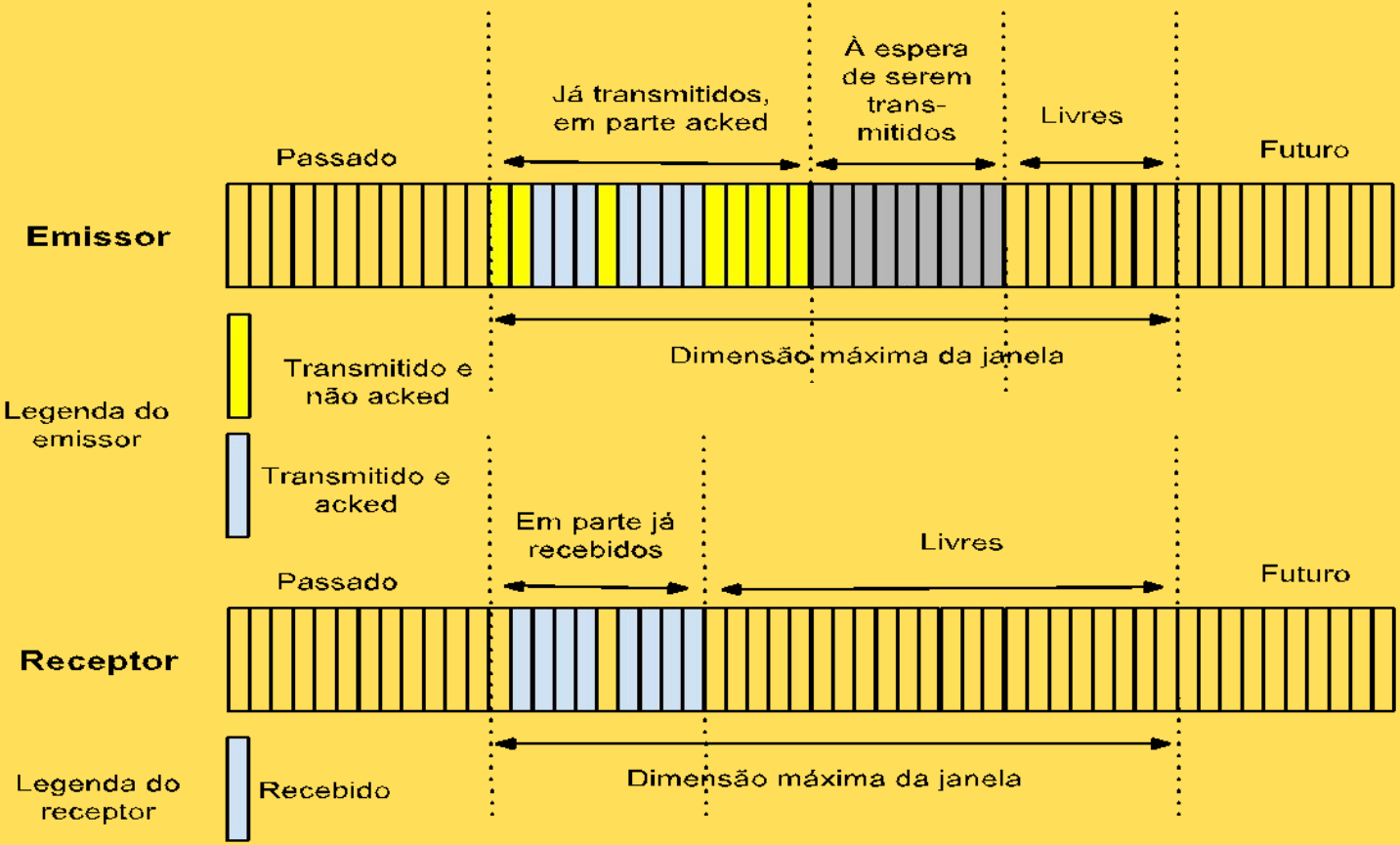
Repetição Selectiva (RS)

- É possível melhorar o algoritmo introduzindo ACKs independentes para cada pacote enviado e recebido
- Significa isso que só são retransmitidos os pacotes para os quais o emissor recebeu um NACK ou um alarme disparou
- Nesses casos, o pacote perdido é retransmitido, mas não se executa o procedimento GBN. O emissor continua sempre a enviar novas mensagens enquanto o tamanho da janela o permitir
- Esta versão do protocolo designa-se por ***selective repeat*** **(SR) ou repetição selectiva (RS)**

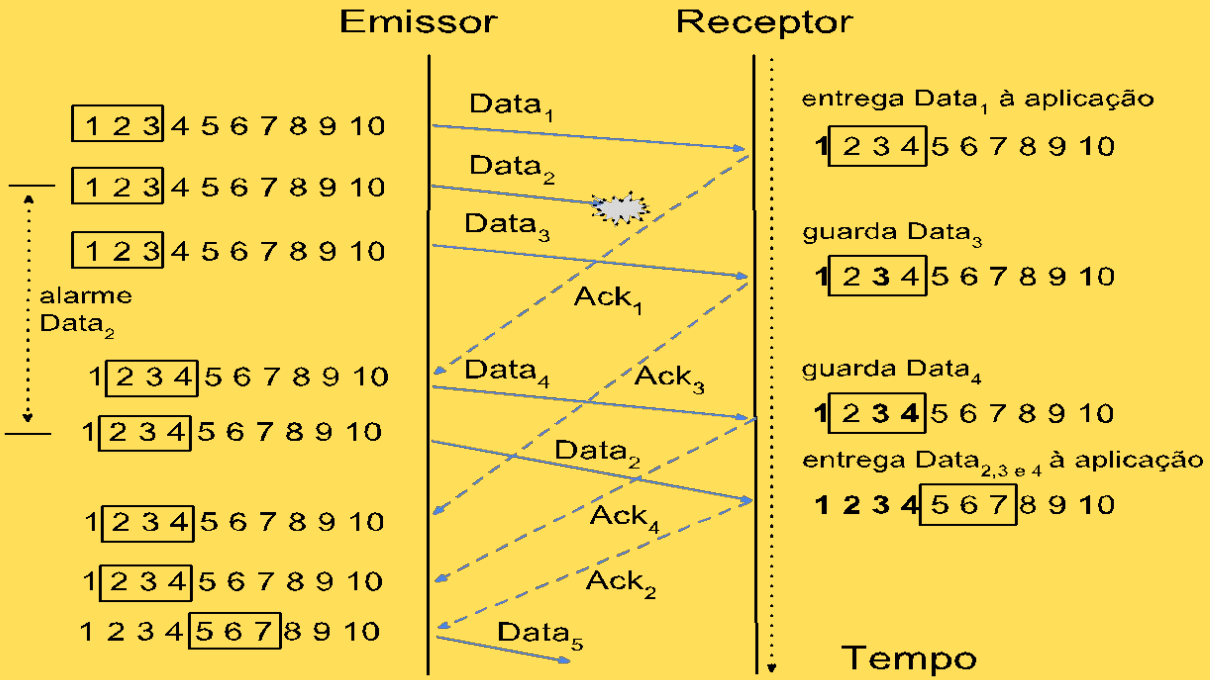
Funcionamento do Protocolo RS

- Por cada pcote n recebido, o receptor envia o respectivo ACK(n) (pode também enviar um NACK do mais antigo pacote que lhe falta se tem um buraco)
- Por cada pacote n enviado, o emissor arma um *timeout T(n)*
- Sempre que um *timeout* T(n) dispara no emissor (ou este recebe um NACK(n)) o pacote n é reenviado
- O emissor continua limitado pela dimensão máxima da sua janela

Janelas no SR



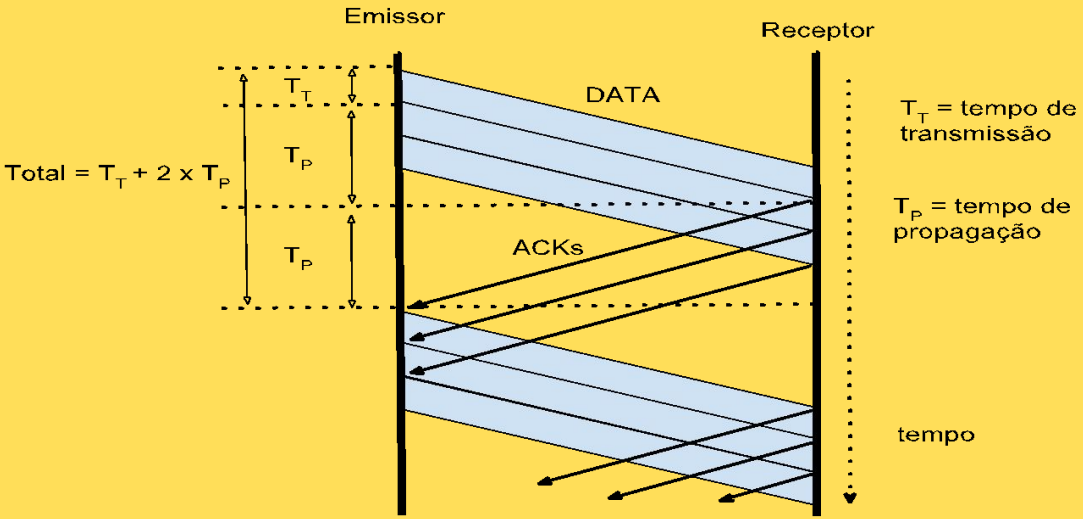
Exemplo


Emissor
Receptor
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
alarme
Data2
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
Data1
Data2
Data3
Ack1
Data4
Ack3
Data2
Ack4
Ack2
Data5
entrega Data1 à aplicação
1 2 3 4 5 6 7 8 9 10
guarda Data3
1 2 3 4 5 6 7 8 9 10
guarda Data4
1 2 3 4 5 6 7 8 9 10
entrega Data2,3 e 4 à aplicação
1 2 3 4 5 6 7 8 9 10
Tempo

Nomenclatura

- Janelas (emissor, receptor)
- (1,1) – ***stop & wait***
- (N,*) – janela deslizante ou ***pipelining***
- (N,1) – janela deslizante com ***Go-Back-N (GBN)***
- (N,M) – com ACKs apenas do que foi bem recebido de forma contígua (ACKs acumulativos) continua a ser GBN mas com recuperação mais rápida
- (N,M) – com ACKs do que foi bem recebido mesmo que de forma não contígua (ACKs selectivos) – ***selective repeat***

Desempenho Sem Erros



Débito útil médio extremo a extremo do protocolo

≈ Dimensão "útil" da janela em bits / (T_T + RTT)

≈ N x Débito do protocolo Stop & Wait

Canais com Débito Muito Elevado

- Quando os canais têm débito muito elevado, o tempo de transmissão diminui drasticamente. Por exemplo, se um canal tem a capacidade de 1 Gbps, transmitir 10.000 bits leva 10^{-5} segundos, isto é, 10 micro segundos
- Se o RTT for de alguns milissegundos e o T_T desprezável, o débito útil médio extremo a extremo permitido pelo protocolo tende para:

Débito útil médio extremo a extremo do protocolo

= Dimensão da janela "útil" em bits / RTT

- O débito útil extremo a extremo diz-se *goodput* em inglês

Dimensão da Janela do Emissor

- Não vale a pena ser muito maior do que o que se consegue emitir continuamente durante um RTT
- Depende também da versão do protocolo pois janelas enormes com GBN incrementam o desperdício potencial a quando da recuperação dos erros (são estes frequentes?)
- Uma janela muito grande também pode potencialmente afogar um receptor lento (controlo de fluxo) ou saturar a rede (controlo da saturação)
- Por isso TCP usa uma janela de dimensão variável

Dimensão da Janela do Receptor

- Teoricamente não vale a pena ser maior do que a do emissor
- No caso geral, implica gerir bocados em falta antes de os entregar à aplicação, o que é mais complicado
- Por isso a solução GBN e janela do receptor = 1 não é assim tão má desde que a relação T_T / RTT não seja demasiado pequena e a taxa de perda de pacotes seja baixa
- O TCP usa uma janela de recepção de valor constante e pode ou não usar a versão SR

Valor dos Alarmes

- Necessariamente superiores ao do RTT
- Se não existirem *buffers* na rede entre o emissor e o receptor (e.g. ambos estão ligados por um canal ponto a ponto direto), o RTT é constante e o valor do *timeout* é mais fácil de estimar
- Se houverem *buffers* pelo meio (e.g. comutadores de pacotes a funcionarem em modo *store & forward* e com filas de espera significativas), o RTT é variável e o valor do *timeout* é mais difícil de estimar
- O protocolo TCP usa um valor de *timeout* ajustado dinamicamente

Conclusões

- A relação entre o tempo de transmissão e o RTT é determinante para o rendimento de um protocolo stop & wait
- Quando esse rendimento é baixo (e.g. Tt << RTT) é fundamental usar protocolos de janela deslizante para melhorar o rendimento
  - É o caso dominante na Internet quando os parceiros em comunicação estão "longe"
- Trata-se de um protocolo complexo com imensos parâmetros ajustáveis a cada cenário
  - O protocolo TCP é um protocolo de janela deslizante que adapta dinamicamente esses parâmetros à cada situação concreta em que é usado