Laboratório de Conectividade – PTC 3260

Troca Confiável de Dados

Escola Politécnica da Universidade de São Paulo
Departamentos da Engenharia Elétrica
PTC Telecomunicações e Controle

life.augmented

Abril de 2024

1.Introdução

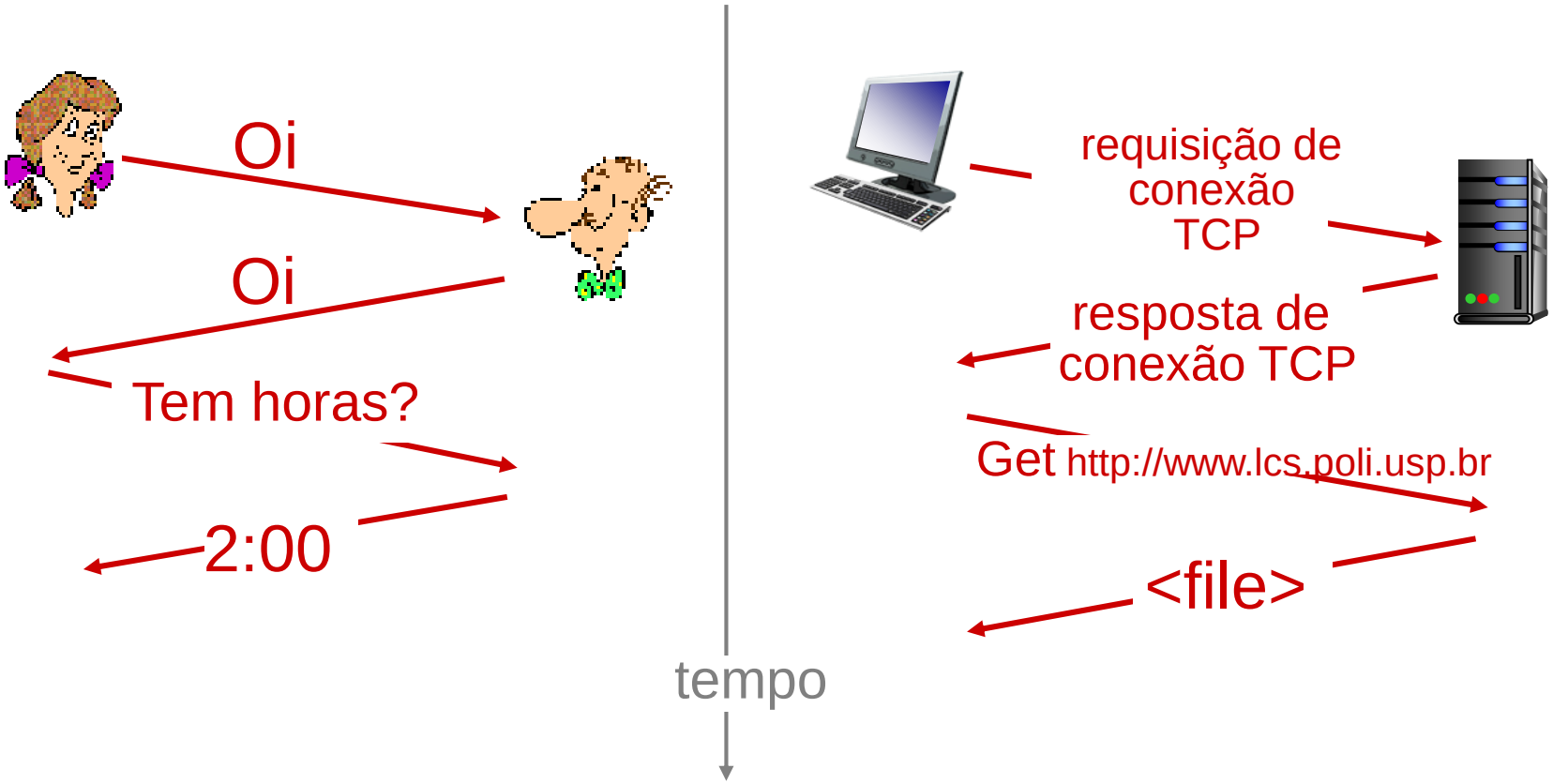
- Para a efetiva troca de dados entre terminais (*hosts*), é necessário implementar certas funcionalidades para o correto funcionamento do sistema
  - Ex.: Endereçamento dos terminais, códigos de verificação de erros, controle de perda de pacotes entre outros

- A implementação de tais funcionalidades se dá por meio de **PROTOCOLOS**

1.Introdução

- O que é um protocolo?

  Ex.: Um protocolo humano e um protocolo de rede de computadores



1.Introdução

- No contexto desta disciplina que funcionalidades são minimamente necessárias?
  1. Endereçamento dos terminais
  2. Tratamento de perda de pacotes/ Implementação de comunicação confiável
  3. Envio e recebimento de pacotes pela internet (protocolos TCP/IP)

- O item 1 já foi tratado anteriormente
- O item 2 será tratado nesta experiência
- O item 3 será tratado na experiência seguinte

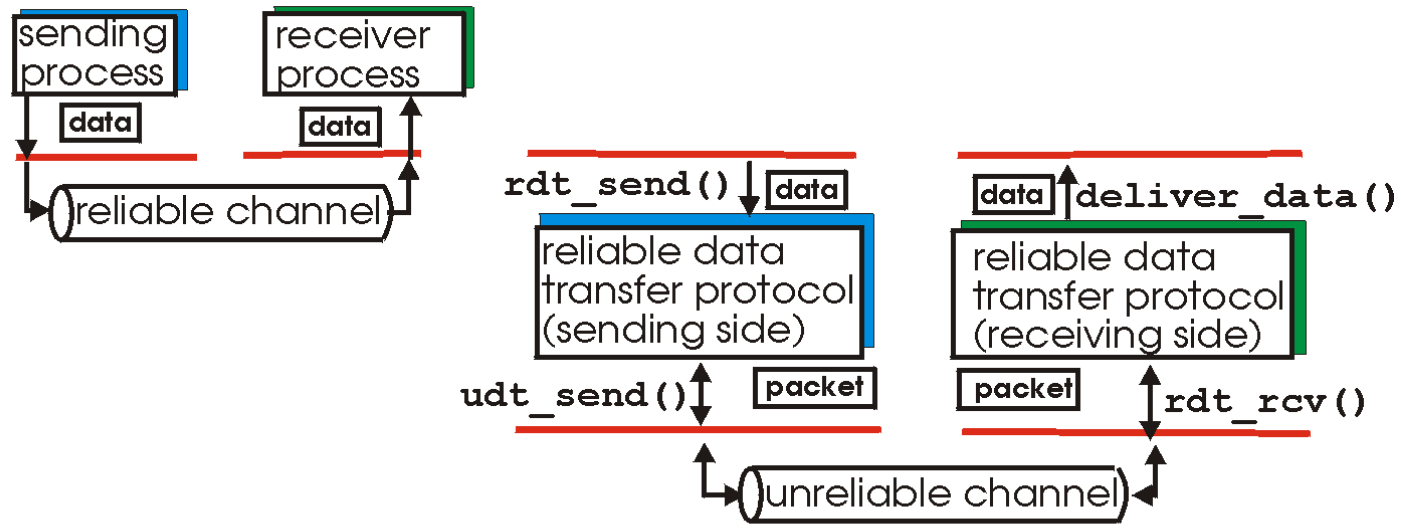
2. Princípios de comunicação confiável

- O canal de comunicação pode fazer com que se perca/corrompa dados
  - Colisão de pacotes – sinais de diferentes terminais chegam ao mesmo tempo no receptor, impossibilitando a correta recepção
  - Ruído corrompe um ou mais bits do pacote
- Erros são detectados pelo CRC e o pacote como um todo é descartado ou o pacote sequer chega a ser recebido (perda de sincronismo, por exemplo)
- Algumas aplicações não toleram que os dados transmitidos sejam perdidos

Como resolver esta questão?

2. Princípios de comunicação confiável

- Ideia: Implementar um protocolo que em caso de perda do pacote, o transmissor refaça a transmissão de forma transparente para a aplicação:



(a) provided service

(b) service implementation

2. Princípios de comunicação confiável

- Existem vários protocolos que realizam comunicação confiável
- A escolha depende de limitações do sistema, de características do canal e do desempenho que se deseja alcançar
- Em geral, protocolos que geram resultados melhores exigem sistemas mais complexos (e.g., mais recursos computacionais, hardware específico)
- Implementaremos um protocolo do tipo *stop-and-wait* que é simples, adaptado a algumas limitações e particularidades do nosso sistema

2. Princípios de comunicação confiável

- O princípio do *stop-and-wait* (condições ideais):
  - O transmissor envia um pacote
  - O receptor recebe corretamente o pacote e envia uma confirmação na forma de um pacote especial de acknowledgement (ACK)
  - O receptor recebe o ACK do pacote e pode transmitir novos dados em outros pacotes

- E se o pacote do transmissor chega com erros detectados pelo CRC?
  - Enviar um negative acknowledgement (NACK): receptor explicitamente conta ao transmissor que pacote tem erros
  - Transmissor retransmite pacote quando recebe NACK - Protocolos ARQ (Automatic Repeat reQuest)

2. Princípios de comunicação confiável

- **Problema:** E se o ACK ou o NACK não é recebido pelo transmissor devido a uma perda ou estes são corrompidos?
  - Se não houver nenhum tratamento, o transmissor não enviará novos pacotes entrando em uma condição de *deadlock* e perderá definitivamente a informação inicial enviada

- **Solução**: Considera-se um prazo máximo para dar *timeout* e retransmite-se o pacote, tal como se um NACK fosse recebido
  - Dará certo se de fato fosse um NACK perdido, mas e o caso de um ACK perdido?

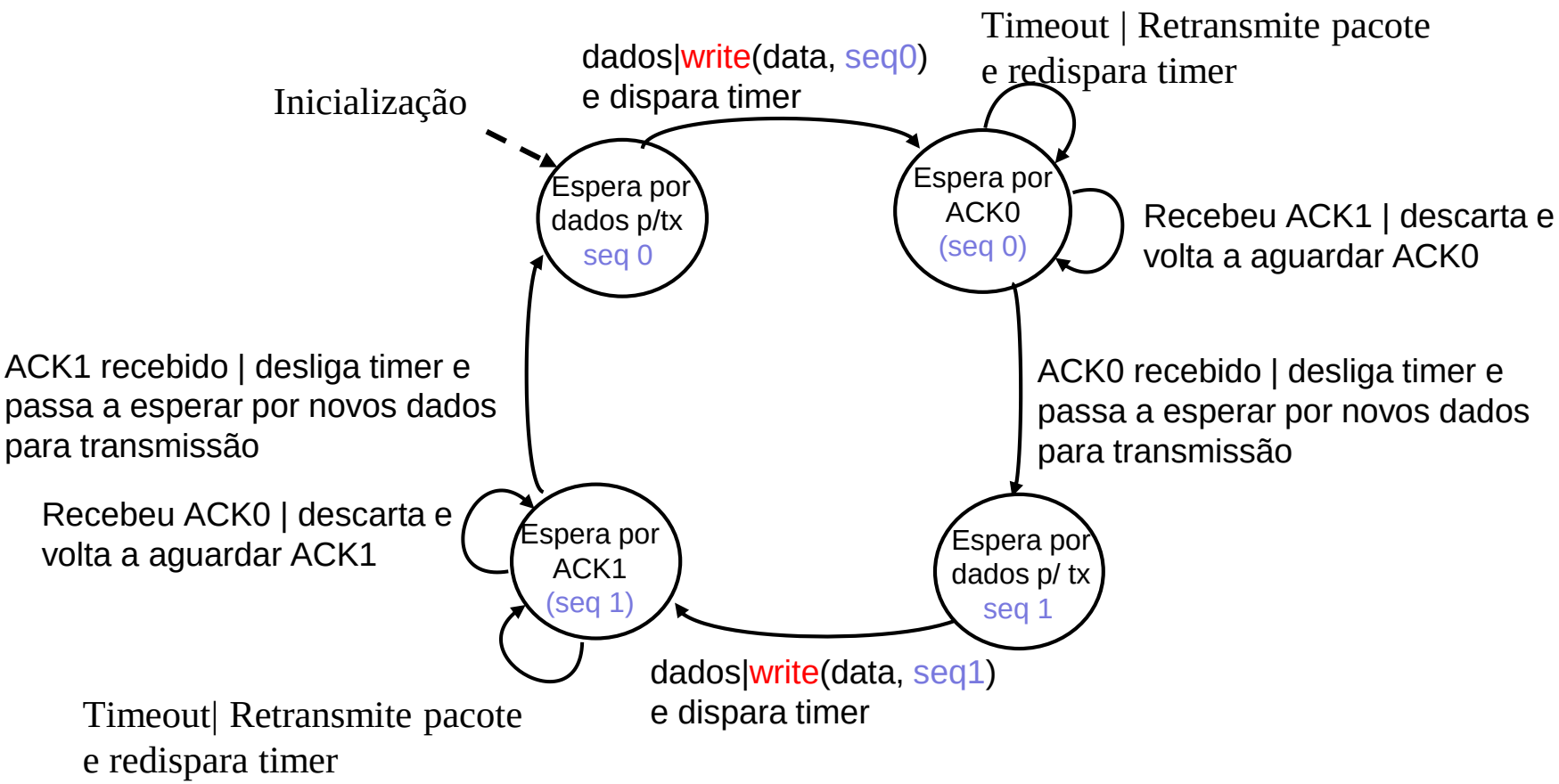
2. Princípios de comunicação confiável

- Surge um **novo problema:** Se a retransmissão fosse para um ACK perdido, teríamos a repetição de um dado como se fosse um novo pacote, o que não é aceitável!
- **Solução**: Enumerar os pacotes do transmissor para que o receptor possa tratar este caso de duplicata
    - Se número do pacote recebido é igual ao anterior, o receptor ignora o pacote, mas envia um ACK para que o transmissor possa ir adiante
    - A enumeração precisa ir até que número?
        - Bastam dois números: (e.g., 0 e 1)
            - Fica-se alternando circularmente entre os dois

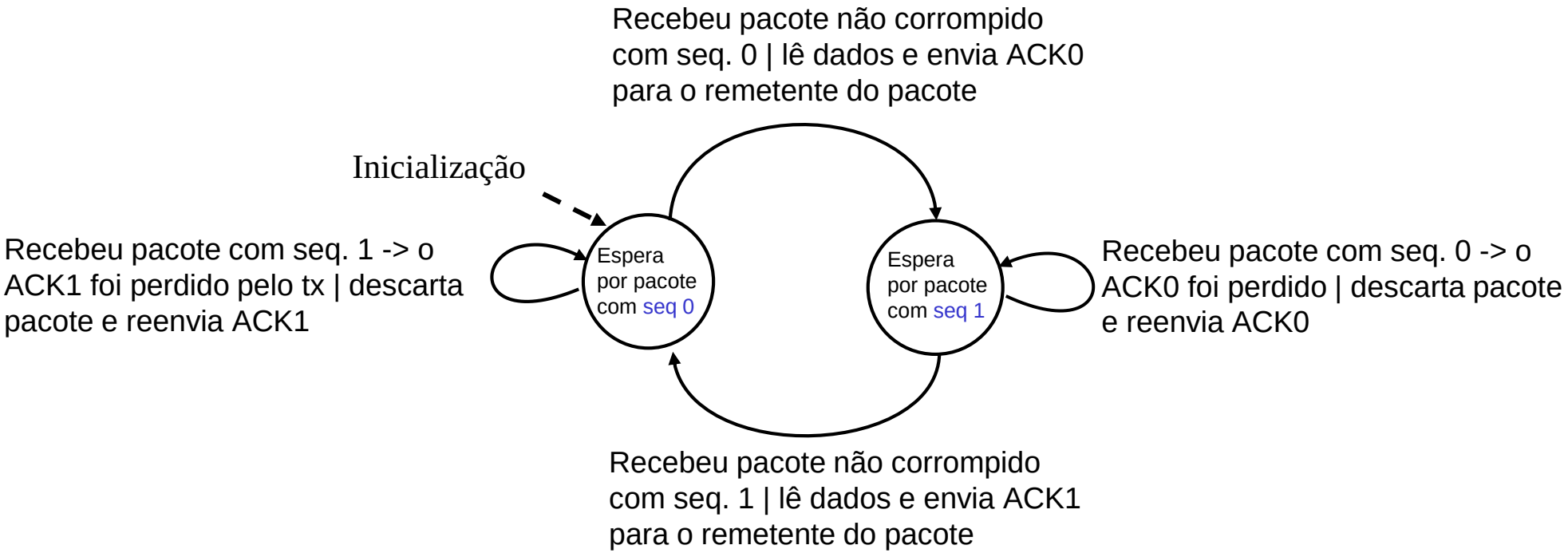
2. Princípios de comunicação confiável

- Particularidade do nosso sistema: Quando o CRC não confere, o hardware não avisa o recebimento do pacote
- Deste modo, ao invés de fazer uma solução com CRC em software, sem a implementação restritiva do hardware, simplificaremos a solução não enviando NACK e trataremos perdas apenas pelo *timeout*
- Perde-se em eficiência: *timeout* é maior que o tempo necessário para receber o NACK -> temos que esperar mais para retransmitir e resolver a perda do pacote
- Por outro lado, se a probabilidade de perda de pacote é baixa, o tempo de *timeout* não é exagerado, e se o intervalo de transmissão costuma ser bem maior que o *timeout,* a perda é pequena

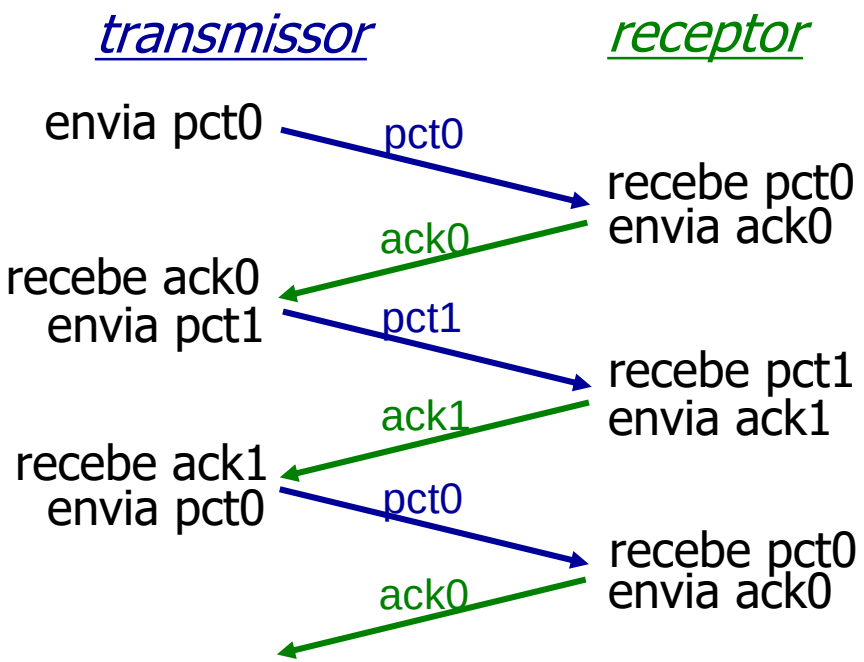
Máquina de Estados do Transmissor



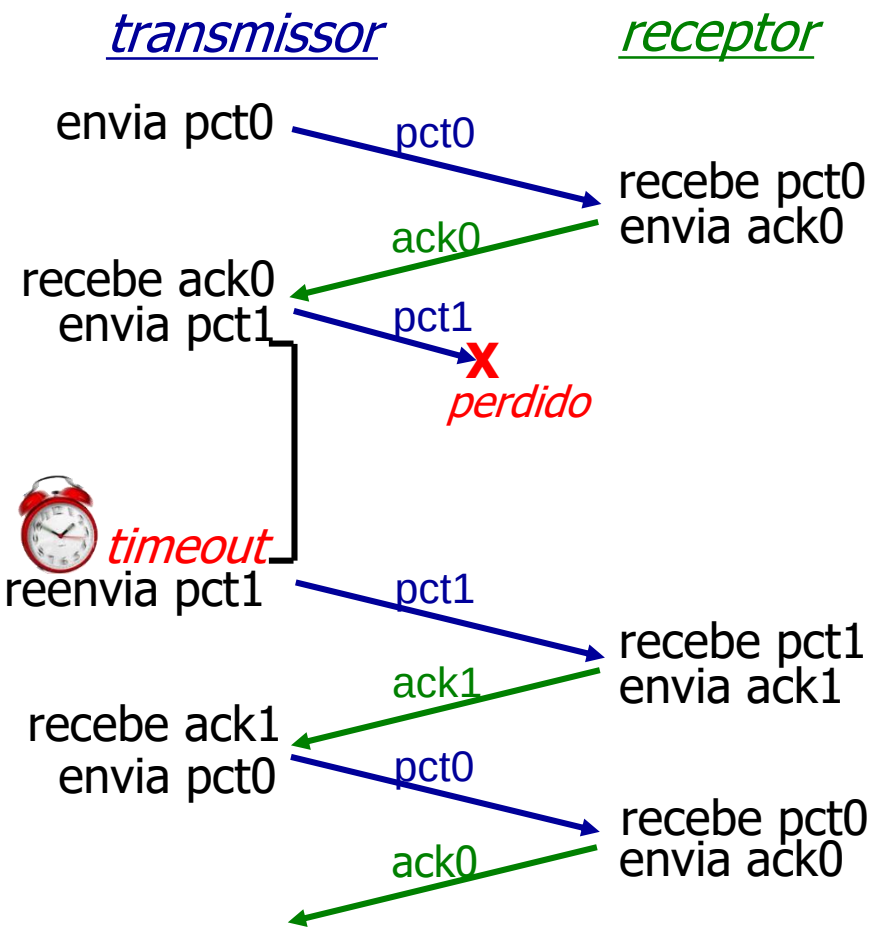
Máquina de Estados do Receptor



Ex.: Protocolo em ação

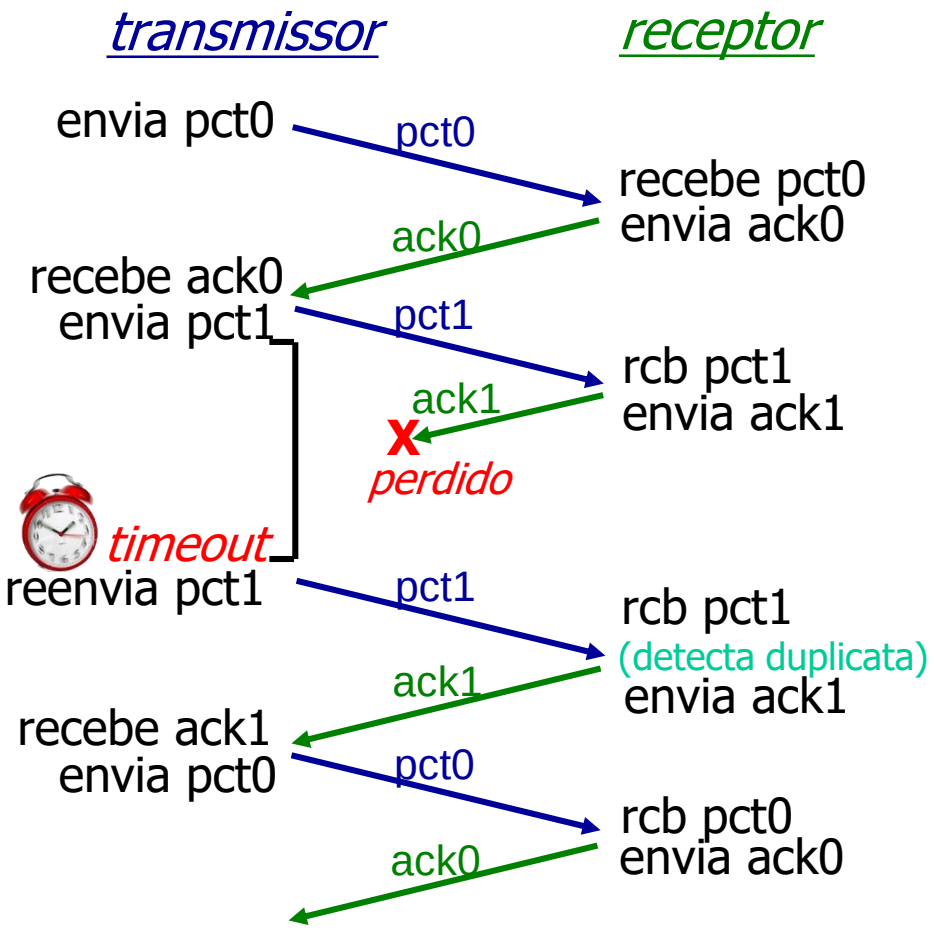


(a) sem perda

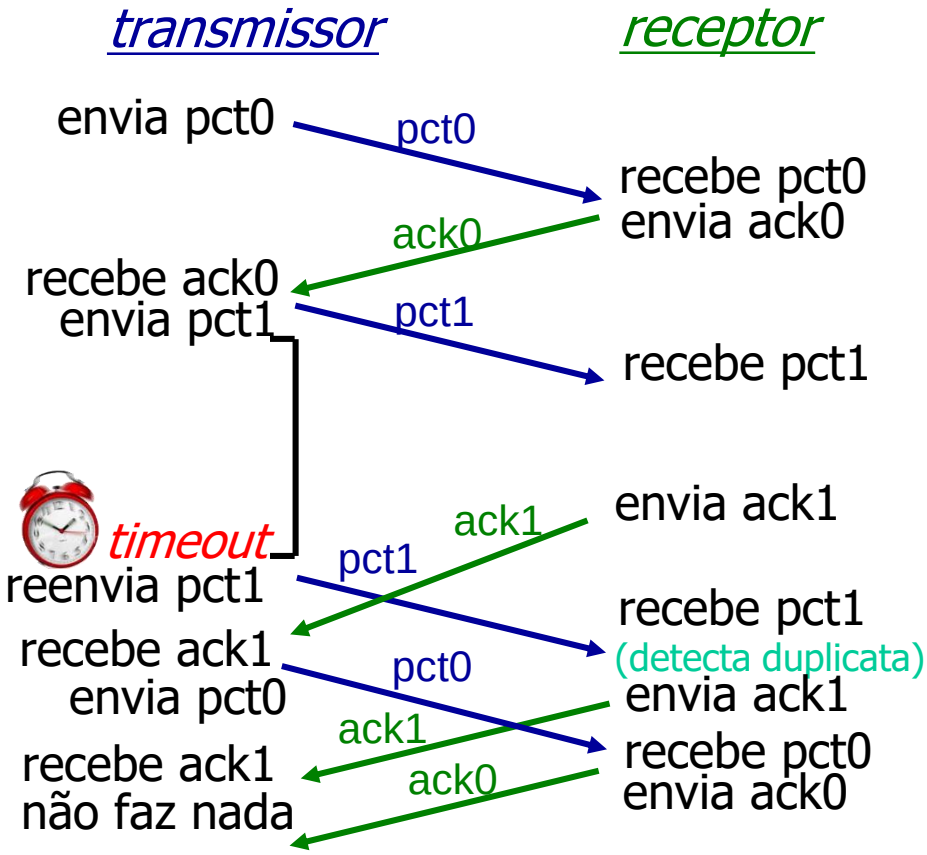


(b) perda de pacote

Ex.: Protocolo em ação



(c) perda de ACK



(d) *timeout* prematuro/ ACK atrasado

2. Princípios de comunicação confiável

Qual é a melhor eficiência dessa proposta no nosso contexto?

- Eficiência é o tempo em que se passa transmitindo informação pelo tempo total para efetuar uma transmissão

Ex.: Pacotes de 12 bytes a 250 kbps e ACKS do tamanho do pacote de dados (o que não é a melhor coisa a fazer, veja a seguir)

Atraso de transmissão é o tempo para enviar um pacote de dados no transmissor:

$$d_{\text{trans}} = \frac{L}{R} = \frac{12 \times 8 \text{ bits}}{250000 \text{ bits/s}} = 0,384 \ ms$$

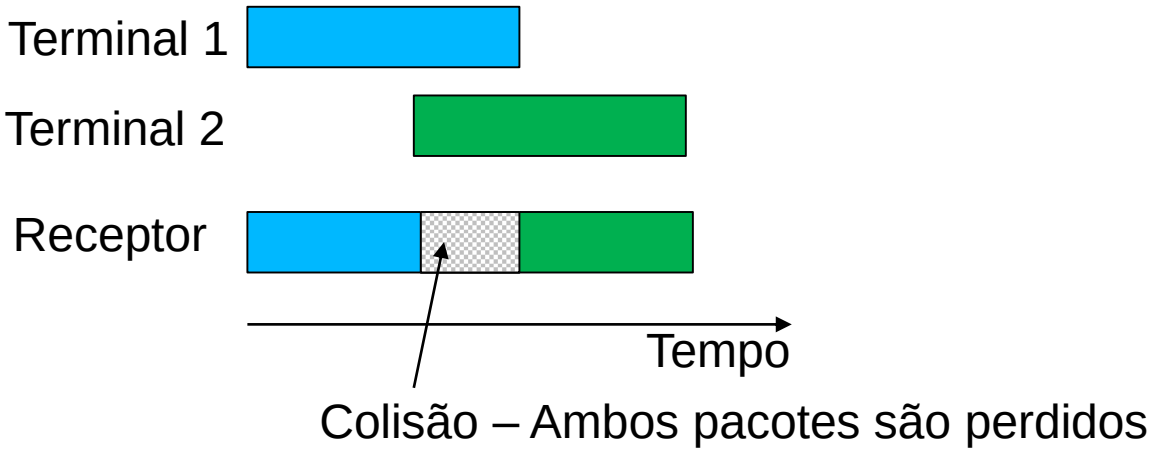
Eficiência no melhor caso (sem perder tempo com processamento) é:

$$\eta = \frac{d_{trans}}{2 \times (d_{prop} + d_{trans})} = \frac{0,38\,ms}{2 \times \left(\frac{15}{3 \times 10^8} + 0,38\,ms \right)} \cong 0.5$$

- Conclusão: Não é tão ruim (50%) se $d_{prop} << d_{trans}$ e poderia ser melhor se ACK fosse menor
- Se d_{prop} é da ordem ou bem maior que d_{trans}, a eficiência tenderá a ser bem pequena
- A vazão líquida (taxa efetiva de informação) do sistema é, no melhor caso, η x R

2. Princípios de comunicação confiável

- Contudo, as coisas ainda não estão totalmente resolvidas
- Temos um contexto de canal compartilhado, em que dois ou mais usuários podem transmitir simultaneamente
- Assim, há um **outro problema – caso acesso múltiplo** :
  - Se dois ou mais usuários transmitem em instantes próximos um dos outros, fazendo com que seus sinais (pacotes) se sobreponham no receptor, os pacotes serão perdidos – há o que se chama de colisão



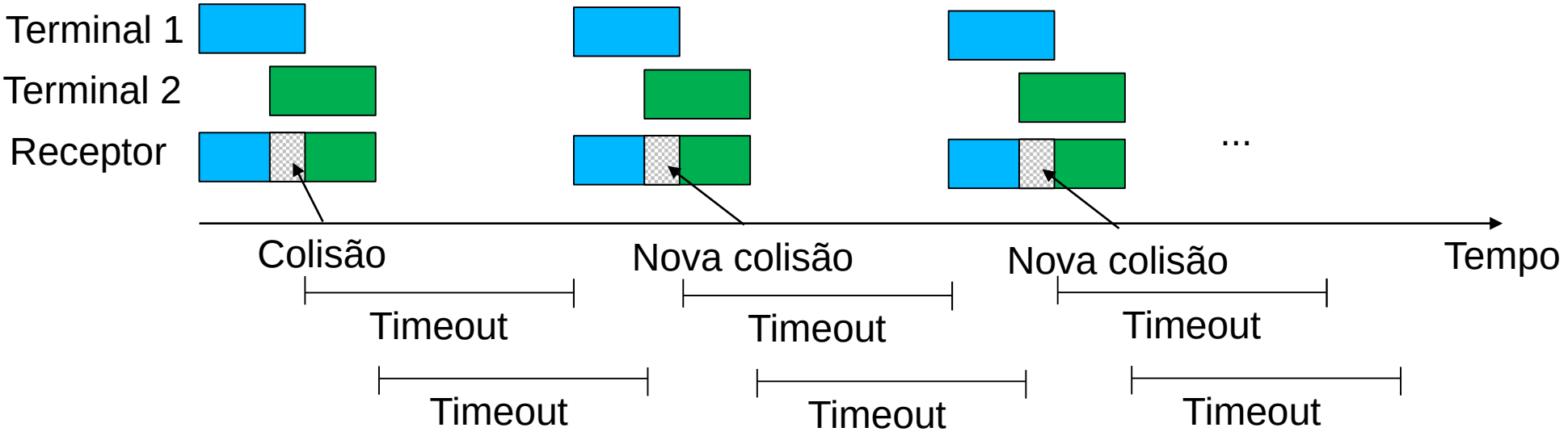
2. Princípios de comunicação confiável

- Note que o *timeout* (ou NACK) não resolve o problema!
- Os terminais envolvidos na colisão irão retransmitir os pacotes com a mesma pequena diferença de tempo (ou sem diferença com o recebimento de um NACK), gerando uma nova colisão e assim por diante...
- Resolução de colisão:
  - O receptor poderia dizer quem deve retransmitir, mas:
    - É bem provável que não se possa dizer quais foram os terminais que colidiram
    - E ainda que fosse, demanda um receptor mais inteligente e complexo para coordenar o sistema
  - A técnica de se adicionar um atraso aleatório e independente para cada *timeout* de cada terminal permite resolver isso localmente, sem centralização

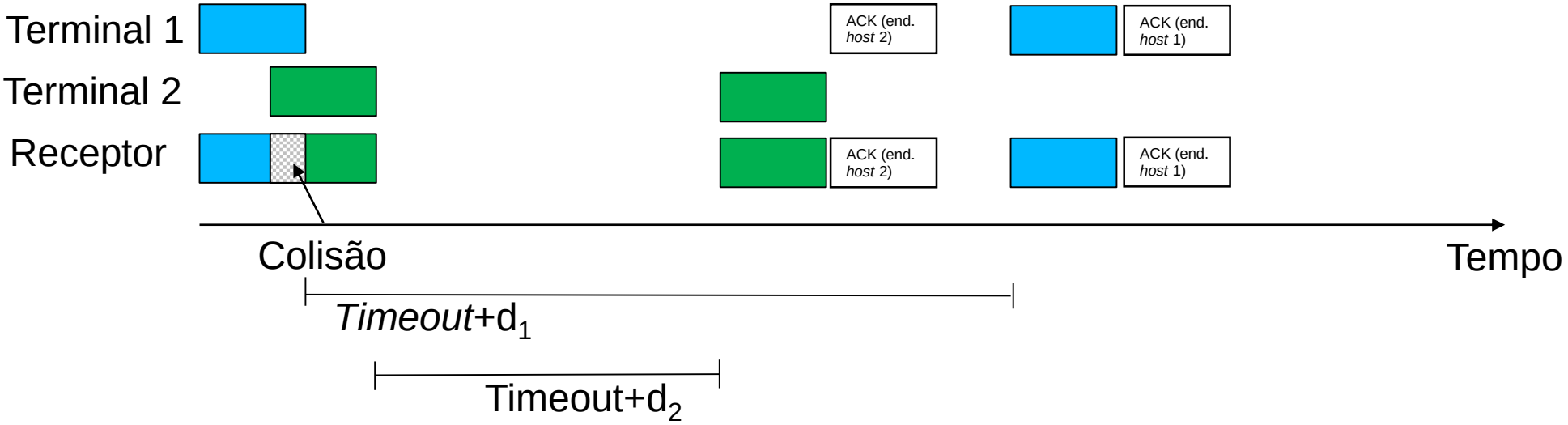
2. Princípios de comunicação confiável

- Veja o exemplo abaixo:

Sem controle de acesso:



Com controle proposto (resolução de colisão):



2. Princípios de comunicação confiável

- Como escolher os atrasos aleatórios?
- Quanto maior for o intervalo do qual o atraso é sorteado, menos provável é a ocorrência de novas colisões entre os terminais que colidiram
  - Todavia, é provável que se espere também mais tempo sem nada transmitir -> baixa eficiência
  - Existem técnica mais avançadas, adaptativas, que permitem lidar com essa questão, como, por exemplo, a técnica *binary exponential backoff* usada no WiFi e Ethernet
- Ainda assim, por simplicidade, usaremos um intervalo razoável de [0, x us]

3. Experimento – Protocolo para Comunicação Confiável

- Roteiro do Experimento:
  - Você deve implementar a máquina de estados do transmissor
  - Use frequência 2402MHz, 1 Mbps de taxa, 0 dBm de potência de transmissão e CRC-8 bits. Não altere o endereço que está no driver da NRF24L01 – a forma de endereçamento será igual ao da aula/experimento anterior
  - Ao pressionar uma tecla (ou botão), você deve transmitir o seu pacote e aguardar o ACK deste seu pacote para poder voltar a transmitir um novo pacote. Imprima o conteúdo dos pacotes recebidos para efeito de debug/avaliação

3. Experimento – Protocolo para Comunicação Confiável

- Os pacotes terão três bytes de comprimento com a seguinte configuração:
  - No primeiro byte, envie o número do seu grupo como endereço
  - No segundo byte, envie o número de sequência do seu pacote
  - No terceiro byte, envie o valor que representa quantos pacotes *novos* você enviou

3. Experimento – Protocolo para Comunicação Confiável

- Para implantar a resolução de colisão, use o comando `rand()%valor` para gerar uma variável aleatória entre [`0 valor-1`]
- Lembre de inicializar o gerador aleatório com `srand(seed)` em que `seed` é um natural e diferente dos outros grupos, caso contrário, não a resolução de colisão não irá funcionar em caso de colisão de grupos com mesmo seed

3. Experimento – Protocolo para Comunicação Confiável

- Use a classe **Timeout** para gerar o evento timeout (vide a descrição da classe na biblioteca do Mbed)
  - Use o evento (interrupção) para habilitar uma *flag*, i.e., variável a ser checada, para ver se é necessária realizar a retransmissão
  - Para disparar o temporizador (obj. `timeout`) , use:
    `timeout.attach_us(&função_que_será_executada,tempo em µs)`
  - Para desabilitar o timer (após sua ativação e da sua execução) use o comando `timeout.detach()` em que `timeout` é o objeto criado com a classe **Timeout** do Mbed
- Sugestão: use `switch` e `case` para trabalhar com os estados
- Extra: como você faria para escolher um tempo de timeout mínimo, sem gerar timeout precoces?

Bom trabalho

Bibliografia

- J.F Kurose e K.W. Ross, “Redes de Computadores e a Internet”, Pearson, 6a. edição