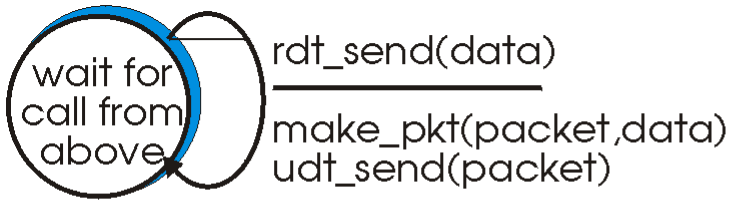
Redes de Computadores

Camada de Transporte

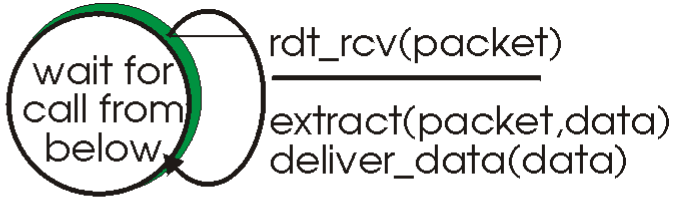
Implementação de um Transporte Confiável

Transferência Confiável de Dados sobre um Canal Confiável rdt1.0

- Uma vez que o canal é confiável, não existe complicações na implementação do protocolo de transporte para operar nele;
- Transmissor envia dados para o canal (a);
- O emissor lê dados do canal (b).



(a) rdt1.0: sending side

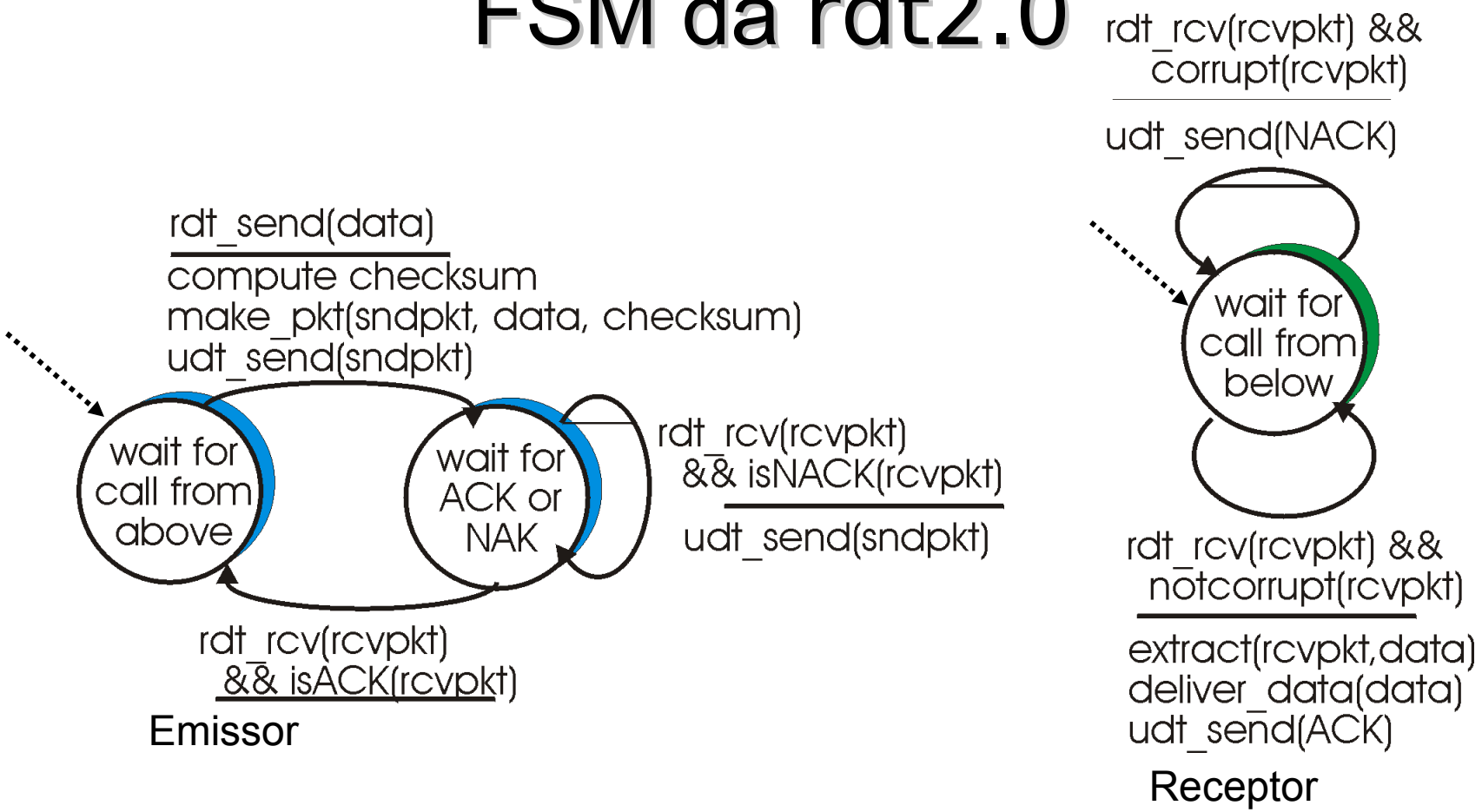


(b) rdt1.0: receiving side

Transferência Confiável de Dados por um Canal com Erros de bits rdt2.0

- O canal inferior pode trocar valores dos bits num pacote;
- Utiliza-se as mensagens de reconhecimento para controle:
  - ACK – reconhecimento positivo, enviado para confirmar o recebimento correto do segmento;
  - NAK – reconhecimento negativo, enviado para informar que o segmento foi recebido com erros;
- Evolução do rdt2.0 em relação ao rdt1.0:
  - Verificação de erros;
  - Mensagens de retorno do receptor (ACK e NAK);
- Protocolo ARQ (*Automatic Repeat reQuest* – solicitação automática de repetição).

FSM da rdt2.0


rdt_send(data)
compute checksum
make_pkt(sndpkt, data, checksum)
udt_send(sndpkt)
wait for call from above
wait for ACK or NAK
rdt_rcv(rcvpkt) && isNACK(rcvpkt)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && isACK(rcvpkt)
Emissor
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
udt_send(NACK)
wait for call from below
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
udt_send(ACK)
Receptor

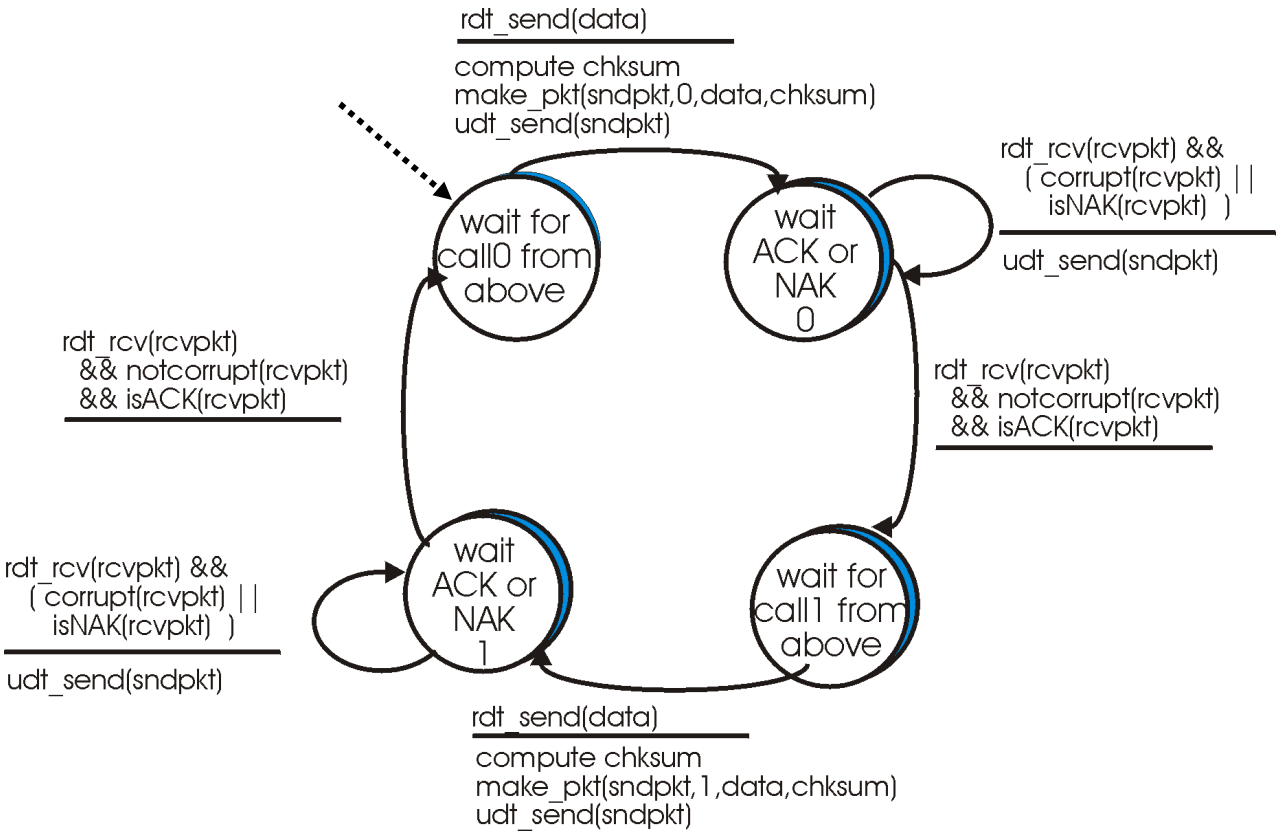
Problema do rdt2.0

- Se o ACK ou NAK estiver corrompido, o que fazer?
  - Não se tem a situação do receptor;
  - Se reenviar o mesmo pacote, corre o risco de haver uma duplicata;
- A solução:
  - O reenvio do arquivo no caso de corrupção do pacote de confirmação ou NAK;
  - O controle da sequência de pacotes enviados;
- Funcionamento *Stop and Wait*, o emissor envia o pacote e fica aguardando a confirmação do receptor.

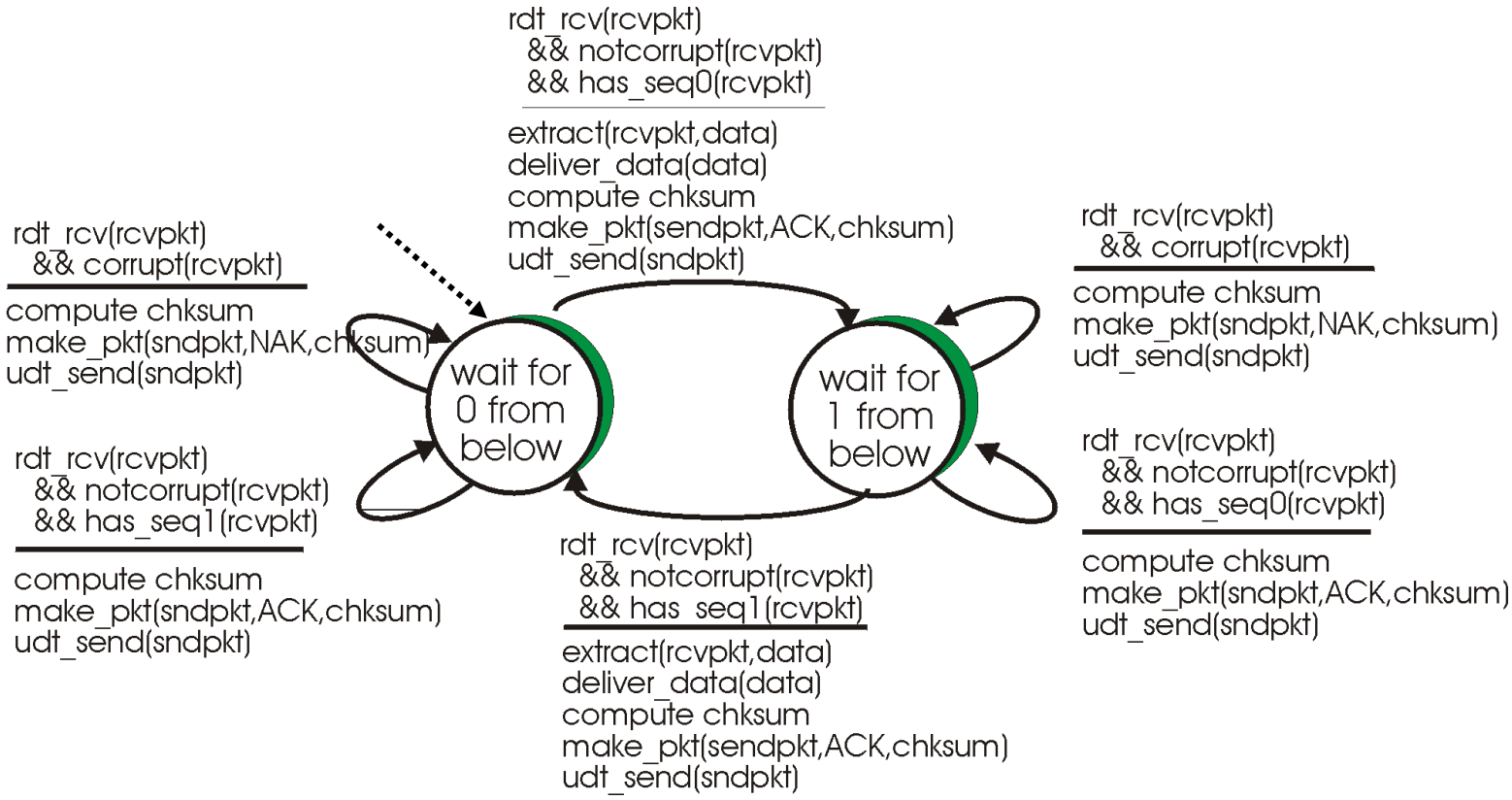
Detalhes do rdt2.1

- Transmissor:
  - Adicionar o número de sequência ao pacote;
  - Verificar se os ACK ou NAK estão corrompidos;
  - Somente um bit para sequência, porque é comparado o valor atual com o anterior;
- Receptor:
  - Verificar se o pacote recebido é duplicado, comparando com o recebido anteriormente;

rdt2.1 – Transmissor, Tratamento de ACK's e NAK's Corrompidos

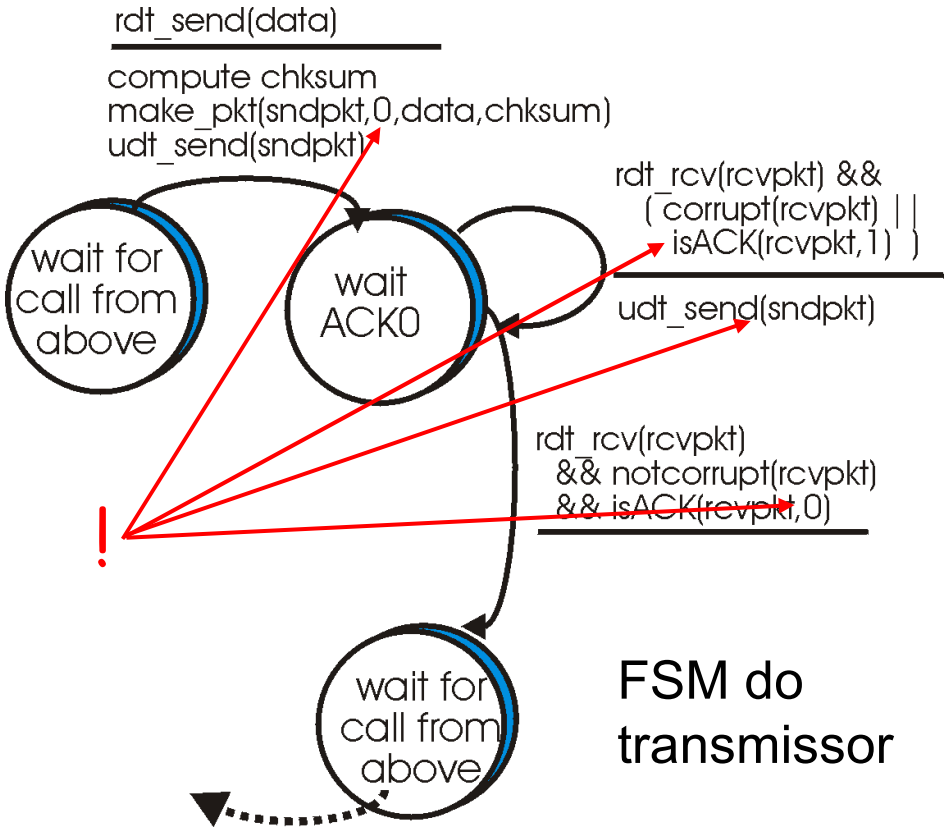


rdt2.1 – Receptor, Tratamento de ACK's e NAK's Corrompidos



rdt2.2 – Um Protocolo sem NAK

- Tem a mesma funcionalidade do rdt2.1, mas utilizando somente ACK;
- Ao invés do receptor enviar um NAK, ele envia um ACK para o último pacote recebido;
- Quanto o emissor recebe o ACK para o pacote anterior, descobre que o atual está corrompido e o reenvia;
- Os ACK's passam a ter um bit de sequência.

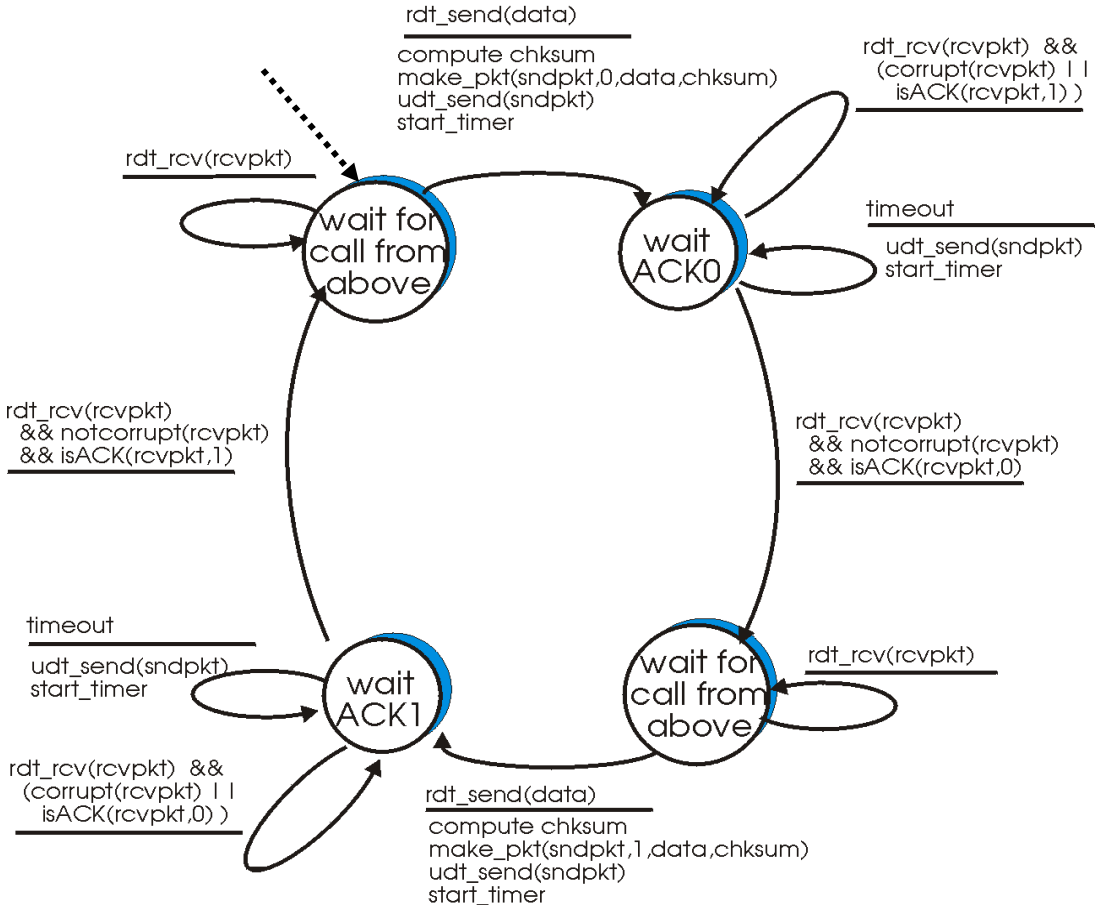


FSM do transmissor

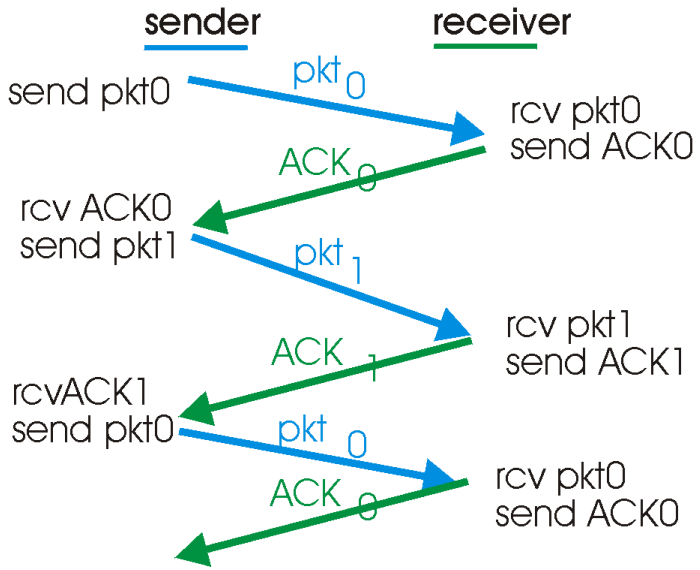
rdt3.0 – Canal de envio com Corrupção e Perda de Pacotes

- Além da Corrupção, o protocolo deverá tratar da perda de pacotes;
- Os recursos presentes nos protocolos rdt2.1 e rdt2.2 são insuficientes para resolver o problema;
- Solução, **reenvio do pacote no caso de não recebimento do ACK depois de um certo tempo**;
- Se houver um grande atraso no recebimento, teremos um pacote duplicado (solução contemplada no rdt2.1 e rdt2.2);
- Deverá ser implementado um temporizador decrescente no emissor.

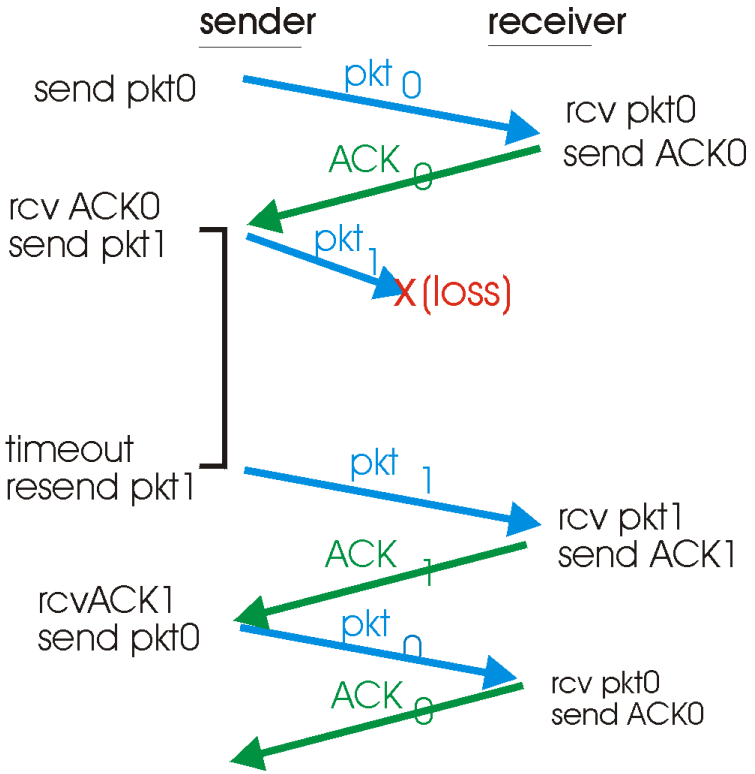
rdt3.0 - Transmissor



Funcionamento do rdt3.0

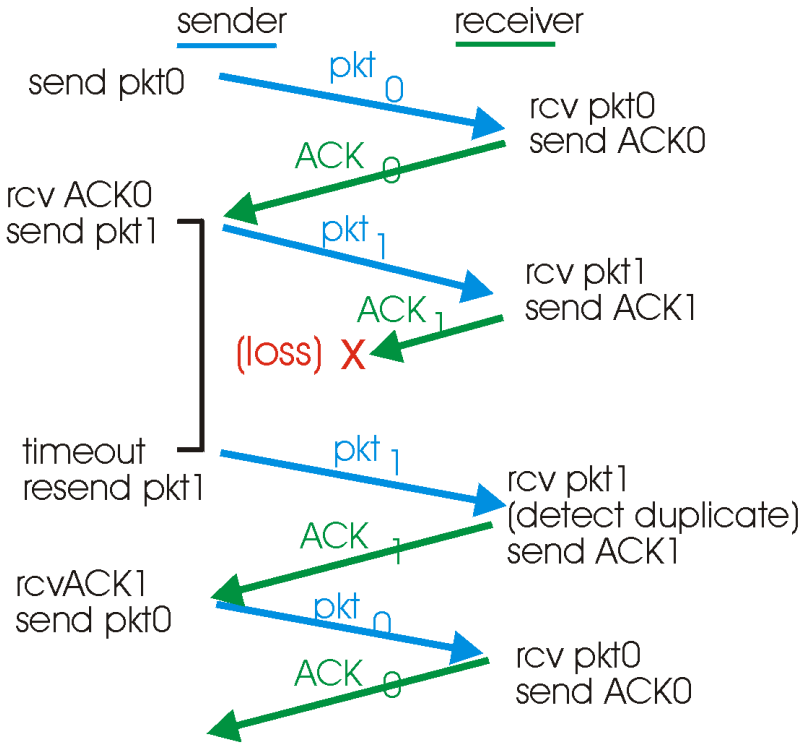


(a) operação sem perda

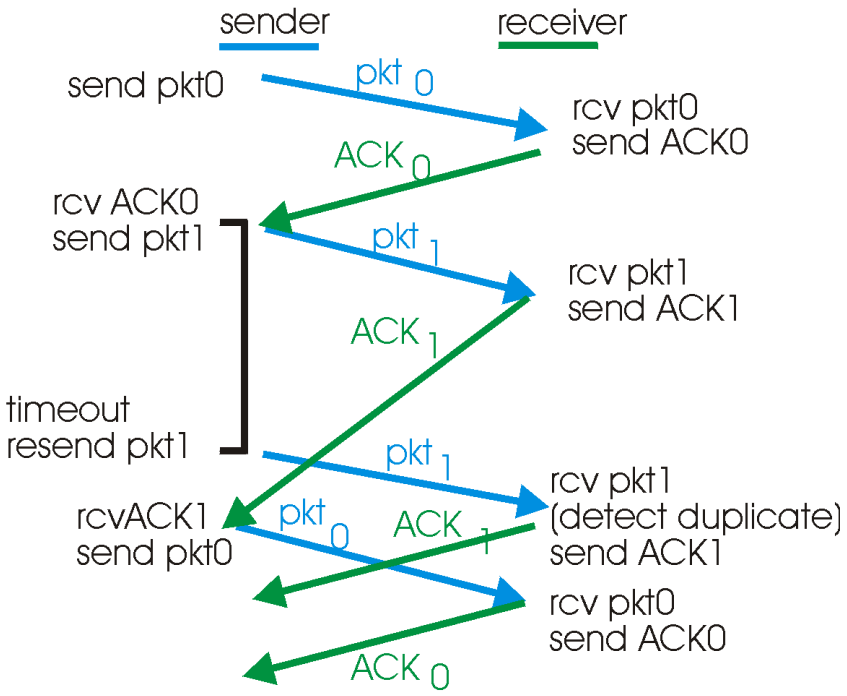


(b) pacote perdido

Funcionamento do rdt3.0



(c) perda do ACK



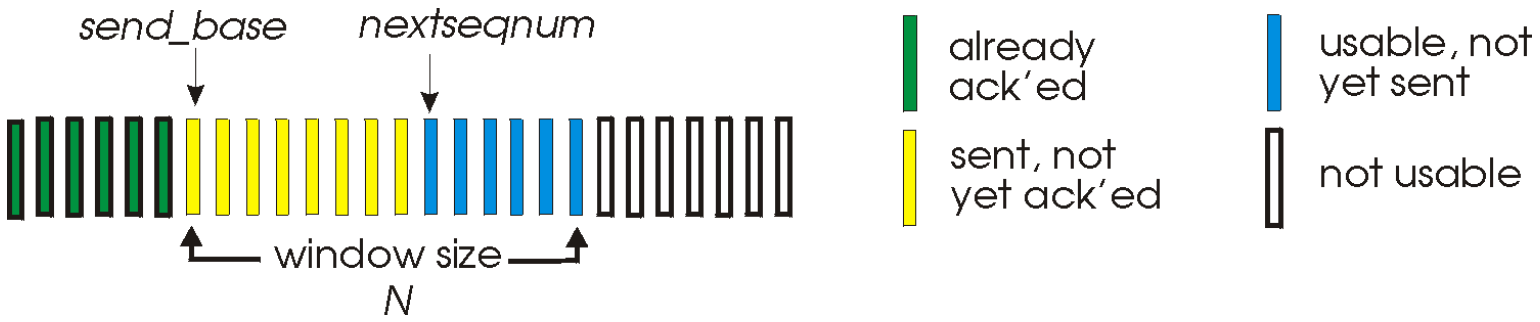
(d) atraso do ACK

Paralelismo (*Pipelining*)

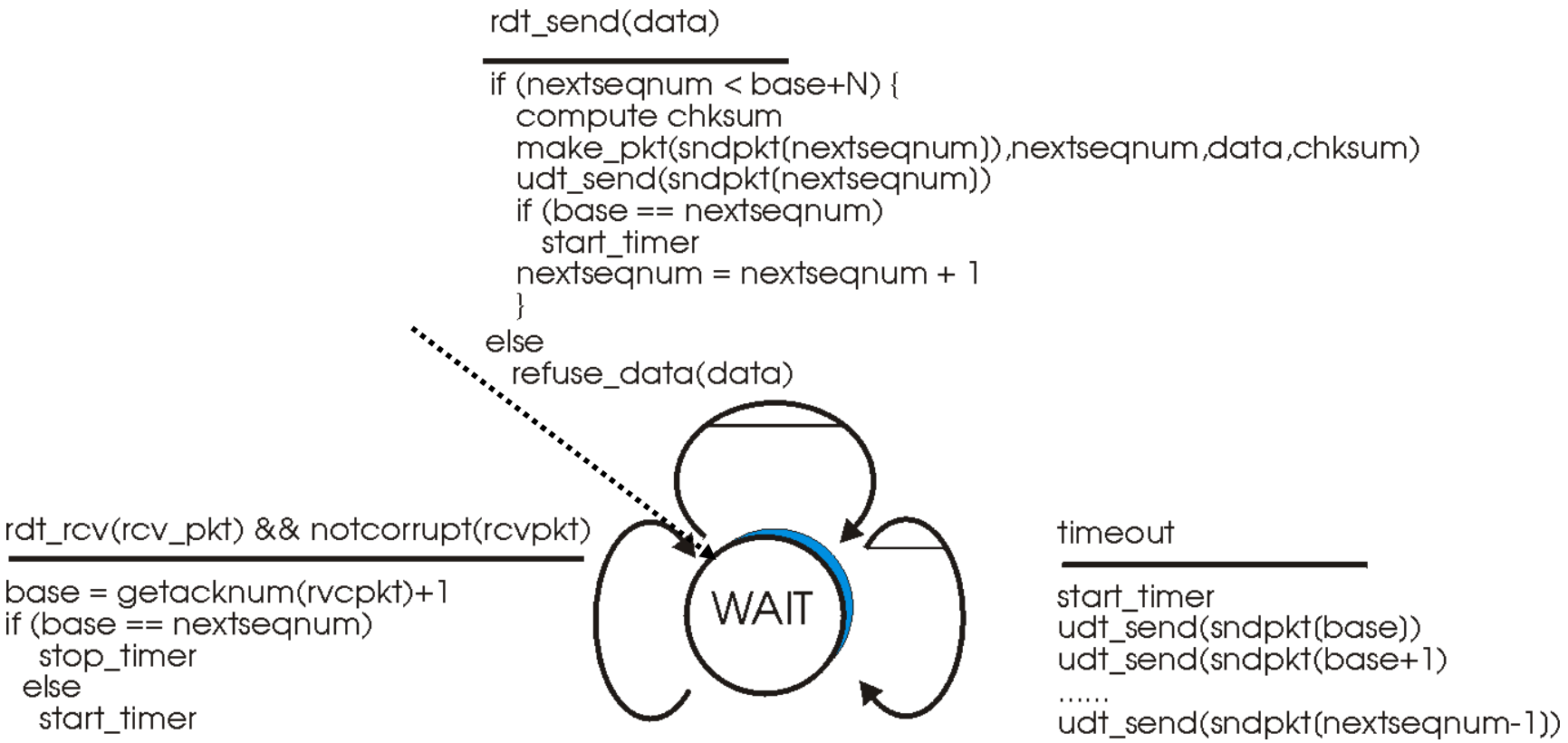
- O transmissor envia vários pacotes antes de receber os devidos ACK até um certo limite;
- Todos os pacotes enviados ficam esperando confirmação de recebimento;
- Consequência:
  - Há a necessidade de aumentar o número de sequência;
  - Utilização de *buffers* para emissão e recepção dos dados;
- Recuperação de erros com paralelismo, duas formas básicas: Go-Back-N e Repetição Seletiva;

Go-Back-N

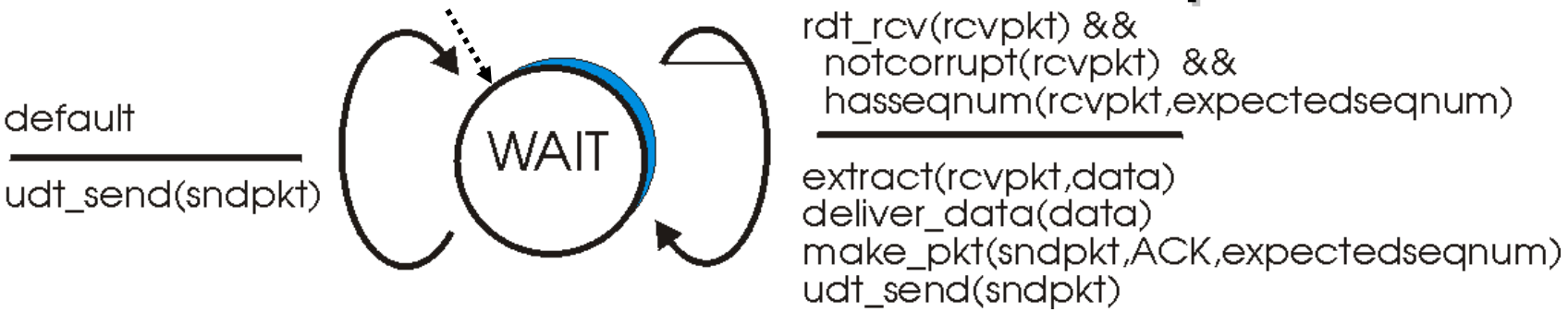
- Transmissor
  - Número de sequência com k bits no cabeçalho do pacote;
  - "janela" de até N, pacotes não reconhecidos, consecutivos, são permitidos;
  - ACK(n): reconhece todos os pacotes até o número de seqüência N (incluindo este limite). "ACK cumulativo"
    - pode receber ACKS duplicados (veja receptor);
  - temporizador para cada pacote enviado e não confirmado;
  - timeout(n): retransmite pacote n e todos os pacotes com número de seqüência maior que estejam dentro da janela;



Go-Back-N – FSM Emissor

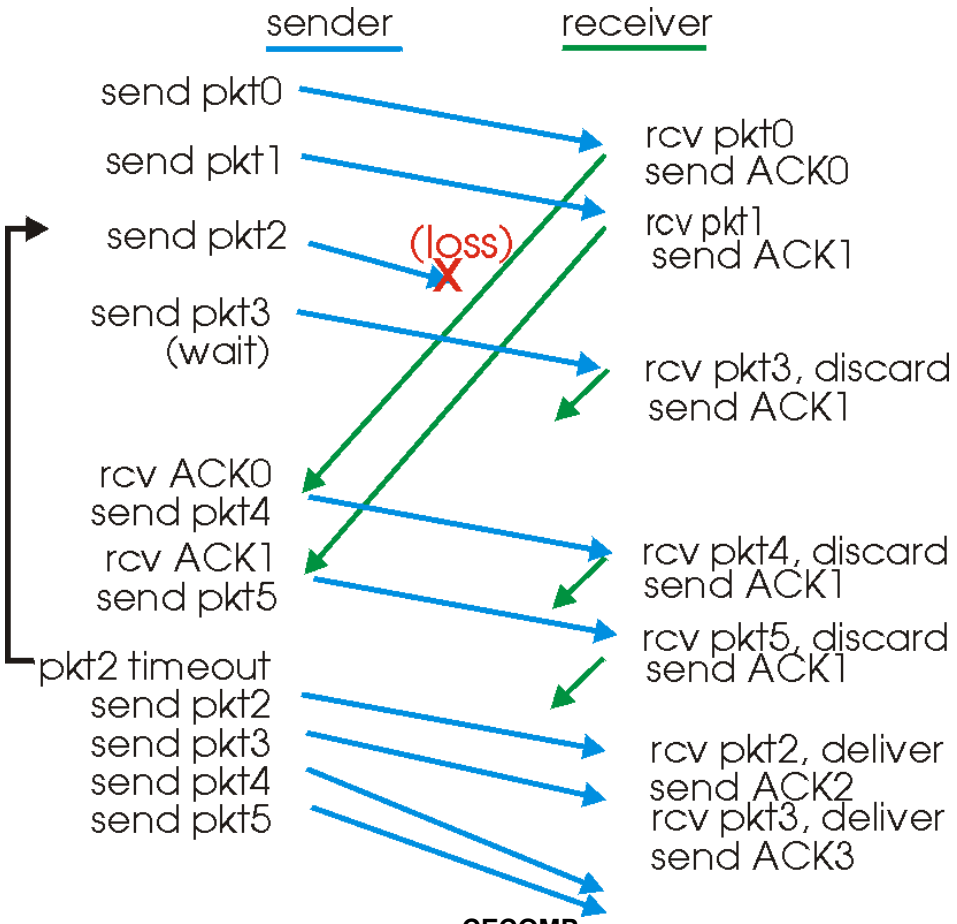


Go-Back-N – FSM Receptor



- Receptor
  - somente ACK: sempre envia ACK para pacotes corretamente recebidos com o mais alto número de seqüência *em ordem*
    - pode gerar ACKs duplicados;
    - precisa lembrar apenas do número de seqüência esperado (**`expectedseqnum);`**
  - pacotes fora de ordem:
    - descarte (não há buffer de recepção);
    - reconhece pacote com o mais alto número de seqüência em ordem;

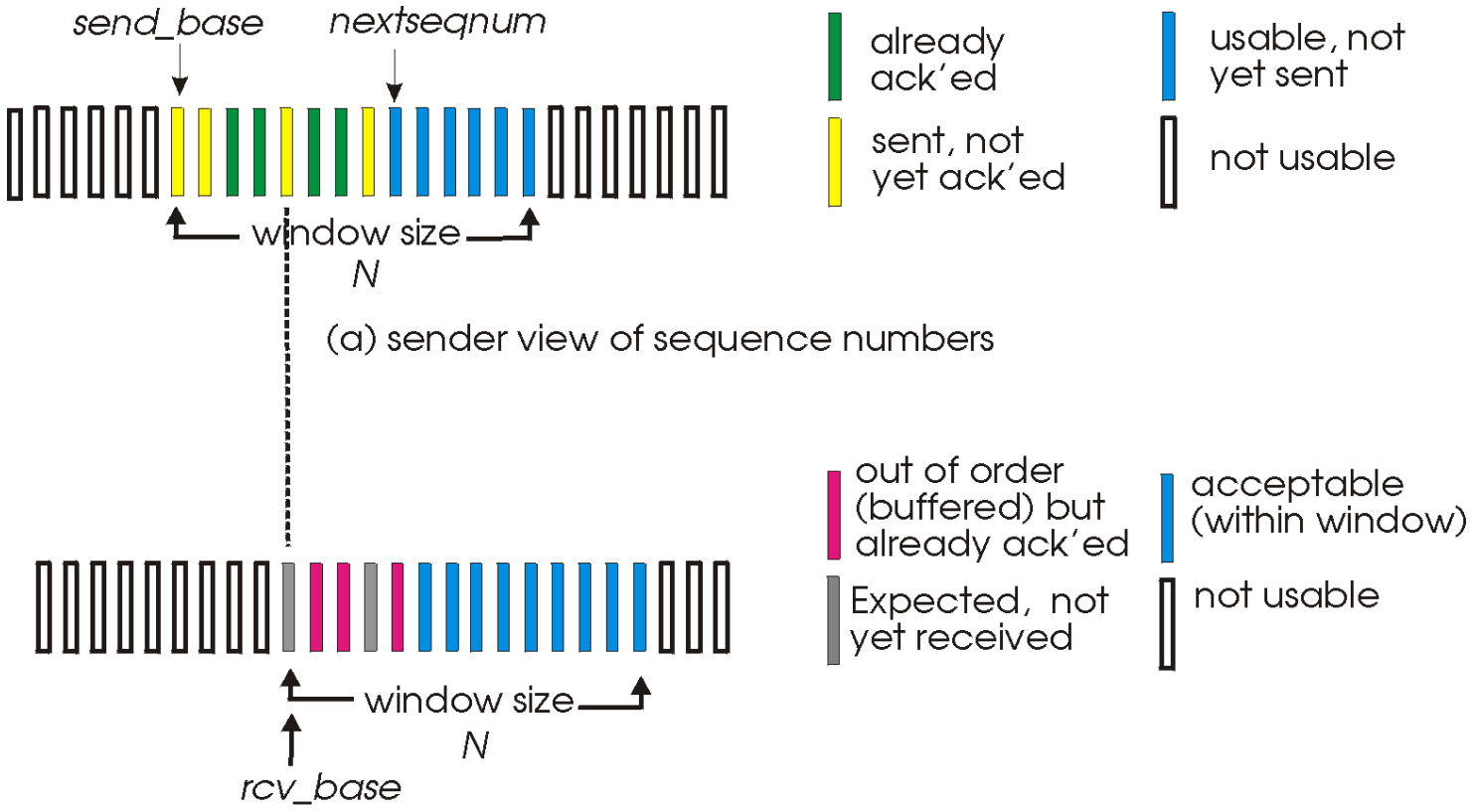
Funcionamento do Go-Back-N



Repetição Seletiva

- receptor reconhece *individualmente* todos os pacotes recebidos corretamente;
  - armazena pacotes, quando necessário, para eventual entrega em ordem para a camada superior;
- transmissor somente reenvia os pacotes para os quais um ACK não foi recebido;
  - transmissor temporiza cada pacote não reconhecido;
- janela de transmissão:
  - N números de seqüência consecutivos;
  - novamente limita a quantidade de pacotes enviados, mas não reconhecidos;

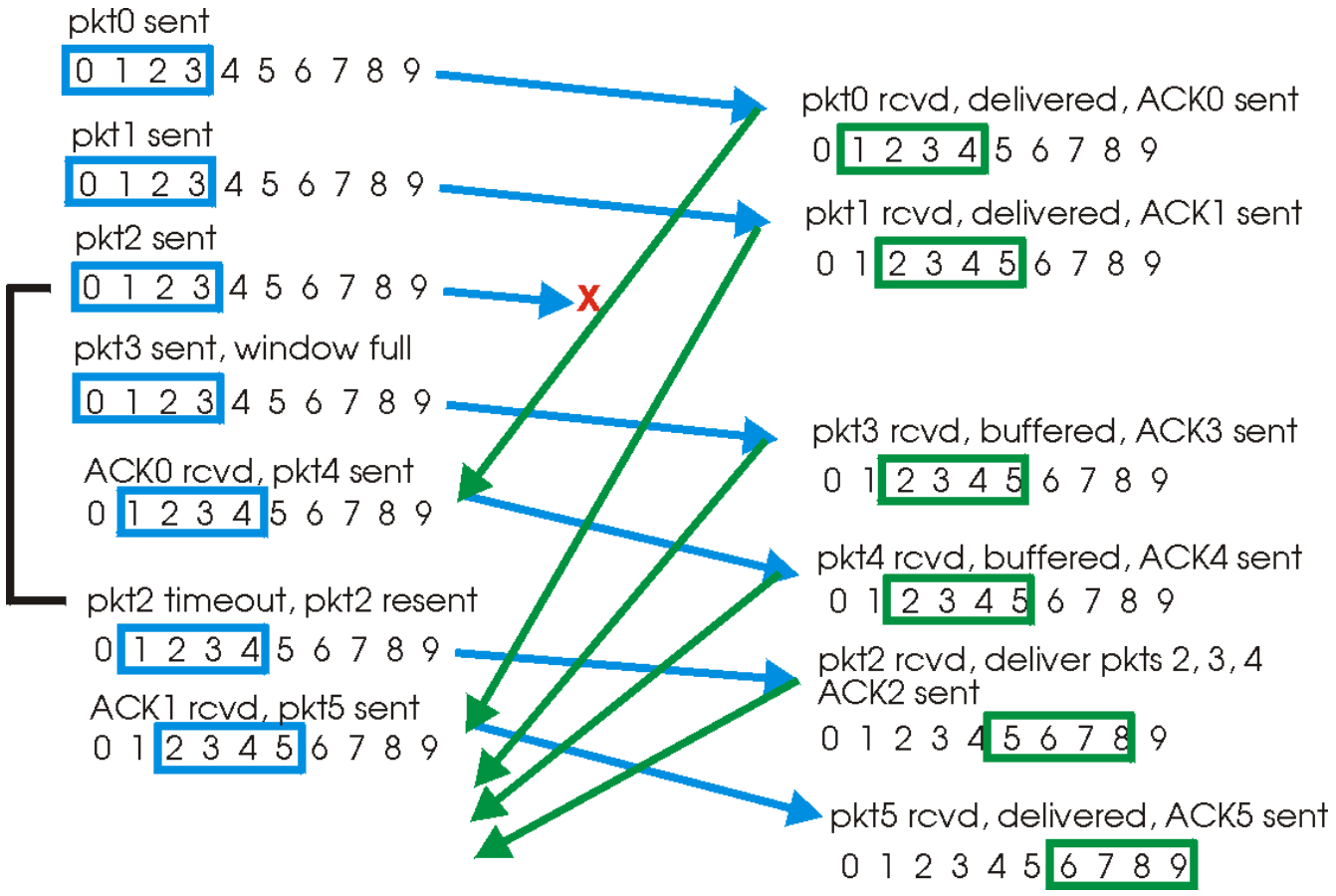
Repetição Seletiva



Repetição Seletiva

- Transmissor
  - Dados da camada superior
    - Se o próximo número de seqüência disponível está na janela, envia o pacote;
  - timeout(n)
    - reenvia pacote n, reinicia o *timer*;
  - ACK(n), no intervalo [sendbase,sendbase+N]
    - Marca pacote n como recebido;
    - Se n é o menor pacote não reconhecido, avança a base da janela para o próximo número de seqüência não reconhecido;
- Receptor
  - Pacote n no intervalo [rcvbase,rcvbase+N-1]
    - envia ACK(n);
    - fora de ordem: armazena;
    - em ordem: entrega (também entrega pacotes armazenados em ordem), avança janela para o próximo pacote ainda não recebido;
  - Pacote n no intervalo [rcvbase-N,rcvbase-1]
    - ACK(n);
  - Caso contrário
    - Ignora

Funcionamento da Repetição Seletiva



Repetição Seletiva - Problema

- Exemplo
  - seqüências: 0, 1, 2, 3;
  - tamanho da janela=3;
  - receptor não vê diferença nos dois cenários!
  - incorretamente passa dados duplicados como novos (figura a);