UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Fábio Alves Bocampagni

Lista 05, capítulo 03

Implementação das soluções da quinta lista, capítulo 03.

RIO DE JANEIRO
2022

1)

P6. Consider our motivation for correcting protocol *rdt2.1*. Show that the receiver, shown in **Figure 3.57**, when operating with the sender shown in **Figure 3.11**, can lead the sender and receiver to enter into a deadlock state, where each is waiting for an event that will never occur.

O receptor mencionado (figura 3.57) é o da seguinte imagem:

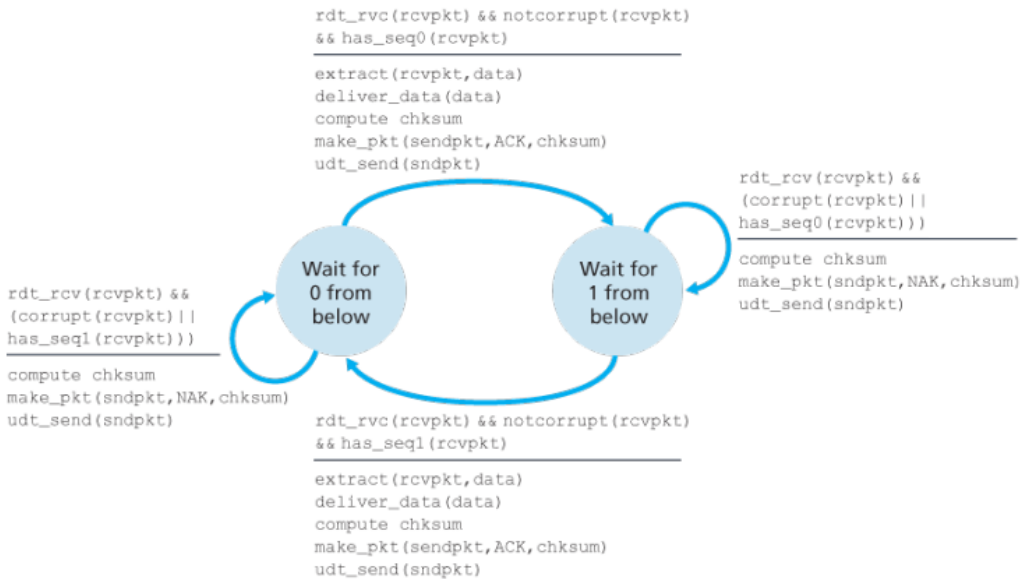


Figure 3.57 An incorrect receiver for protocol *rdt 2.1*

E esse é o seguinte transmissor mencionado na questão:

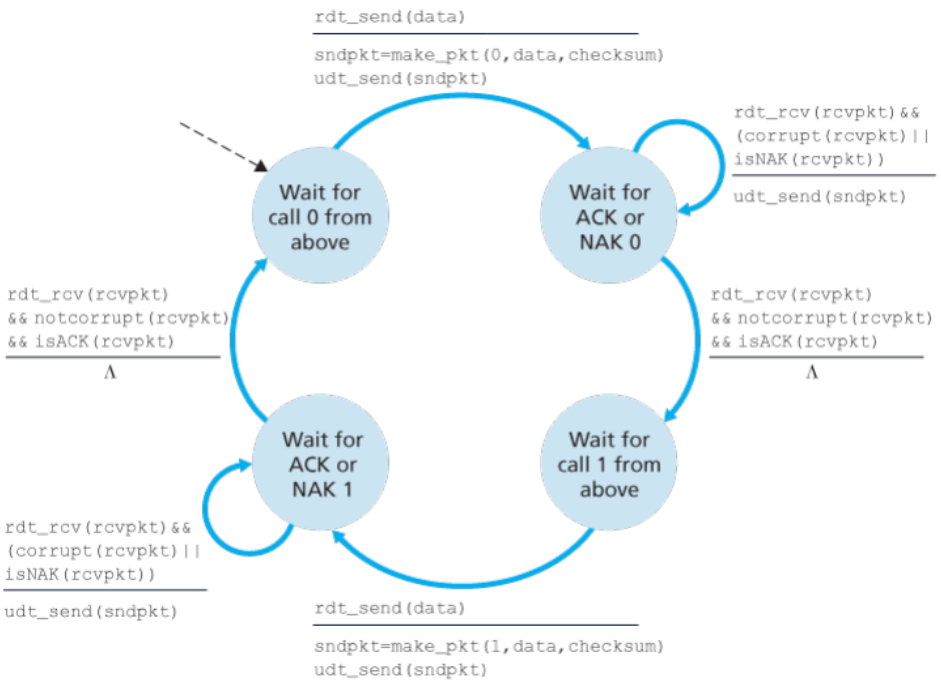


Figure 3.11 *rdt2.1* sender

É solicitado que mostre a possibilidade dessa combinação gerar a entrada em um estado de deadlock, estado o qual ambos as máquinas de estados estão esperando por um evento que nunca ocorrerá.

Suponhamos que o transmissor enviou o primeiro pacote e o mesmo tenha chegado corrompido. Como sabemos, quando o transmissor envia esse primeiro pacote, ele parte para o segundo estado, no caso, “Wait for ACK or NAK0”. Como o pacote enviado pelo transmissor foi corrompido, o receptor irá identificar esse corrompimento e não enviará um ACK, ou seja, não confirmando a boa conformidade desse pacote, e sim, entrará na passagem de estado que leva para o mesmo estado inicial, ou seja, mandando um NAK para o transmissor. O transmissor, por sua vez, aguarda o estado ACK ou NAK0, porém, nenhum NAK é esperado. Assim, ele ficará esperando um ACK ou NAK0, o qual nunca chegará, pois o receptor apenas envia ACK e NAK.

Em outras palavras, o receptor não enumera quais são seus ACKs e NAKs, enquanto que o transmissor espera por essas mensagens enumeradas.

2)

P7. In protocol `rdt3.0`, the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

Não precisamos identificar qual o pacote que estamos reconhecendo devido ao fato da forma que o transmissor muda seus estados. Não existe a possibilidade de termos 2 pacotes sendo enviados ao mesmo tempo, ou seja, pacotes recebendo o mesmo ack, devido ao fato do rdt3.0 ser stop-and-wait. Ou seja, ele executa uma ação e espera a resposta dela.

Após o transmissor enviar o pacote, ele parte para o estado de esperar um ACK0 e ali fica até que haja um timeout, ou chegue um ACK0, confirmando que o pacote chegou.

Dessa forma, não é necessário que haja um número de série do ACK dizendo qual pacote está sendo confirmado, porque só existe a possibilidade, dado o comportamento stop-and-wait, de um pacote ser confirmado (Aquele do estado atual do transmissor).

3)

P11. Consider the `rdt2.2` receiver in **Figure 3.14** , and the creation of a new packet in the self-transition (i.e., the transition from the state back to itself) in the Wait-for-0-from-below and the Wait-for-1-from-below states: `sndpkt=make_pkt(ACK, 1, checksum)` and `sndpkt=make_pkt(ACK, 0, checksum)` . Would the protocol work correctly if this action were removed from the self-transition in the Wait-for-1-from-below state? Justify your answer. What if this event were removed from the self-transition in the Wait-for-0-from-below state? [*Hint*: In this latter case, consider what would happen if the first sender-to-receiver packet were corrupted.]

A figura mencionada é está:

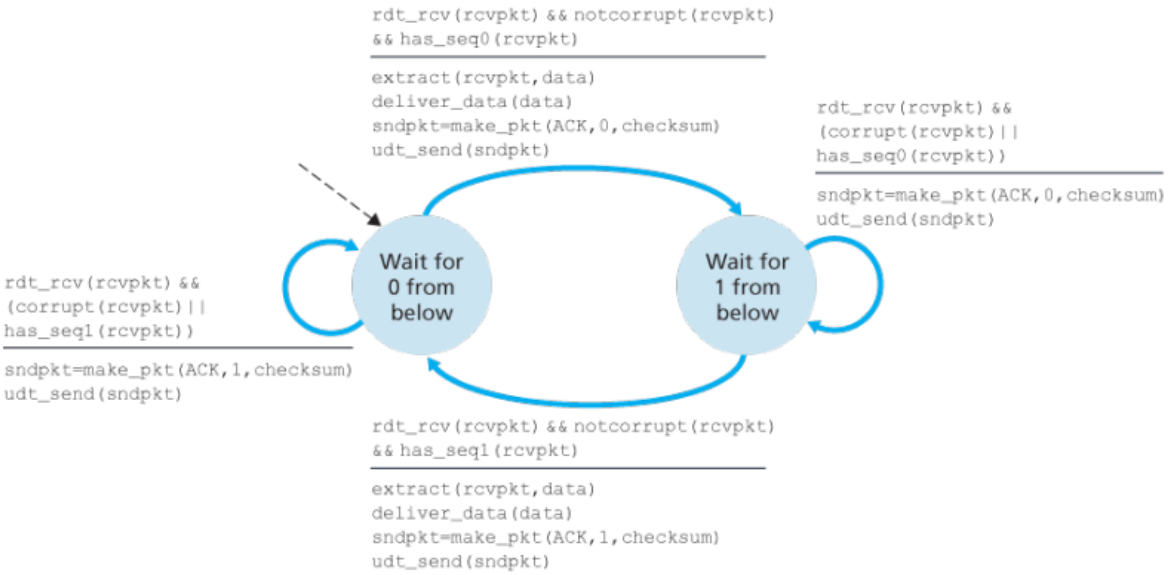


Figure 3.14 *rdt2.2* **receiver**

Não funcionaria em ambos os casos. Tanto removendo do primeiro estado, quanto do segundo devido ao fato do transmissor necessitar de uma confirmação do pacote quando pula do estado de envio para o estado de espera. Se não houver confirmação, seja ela positiva ou negativa, o transmissor ficará preso na espera pela resposta do receptor.

4)

P12. The sender side of *rdt3.0* simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the *acknum* field of an acknowledgment packet. Suppose that in such circumstances, *rdt3.0* were simply to retransmit the current data packet. Would the protocol still work? (*Hint*: Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the *n*th packet is sent, in the limit as *n* approaches infinity.)

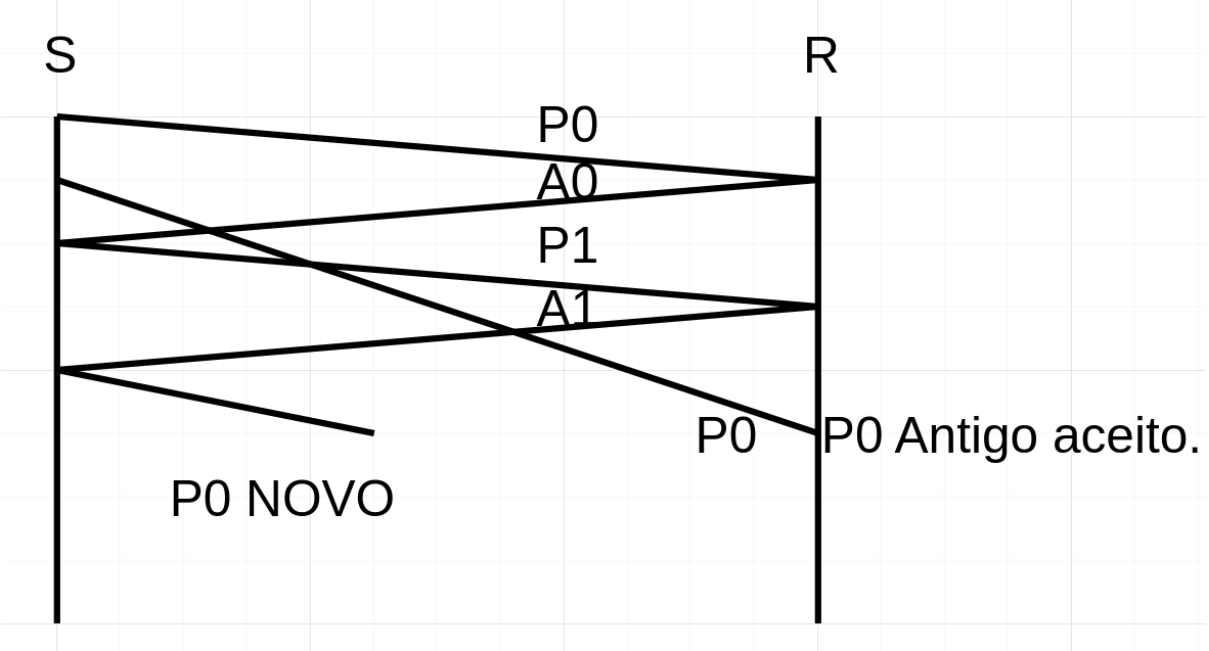
O protocolo ainda funcionaria devido ao fato da retransmissão ser algo natural ao lado do transmissor. Ela já aconteceria se o pacote, que chegou com erro, nunca chegasse.

Se cada pacote que chegou com erro, ou está com o acknum errado, fosse retransmitindo cada cópia receberia um ACK e cada extra ACK recebido causaria outra cópia do pacote a ser enviado, desta forma, o número de pacotes n a serem enviados iria crescer sem limitações, já que n tende ao infinito.

5)

P13. Consider the `rdt 3.0` protocol. Draw a diagram showing that if the network connection between the sender and receiver can reorder messages (that is, that two messages propagating in the medium between the sender and receiver can be reordered), then the alternating-bit protocol will not work correctly (make sure you clearly identify the sense in which it will not work correctly). Your diagram should have the sender on the left and the receiver on the right, with the time axis running down the page, showing data (D) and acknowledgment (A) message exchange. Make sure you indicate the sequence number associated with any data or acknowledgment segment.

Utilizando o draw.io o seguinte diagrama foi criado:



7)

P22. Consider the GBN protocol with a sender window size of 4 and a sequence number range of 1,024. Suppose that at time *t*, the next in-order packet that the receiver is expecting has a sequence number of *k*. Assume that the medium does not reorder messages. Answer the following questions:

a. What are the possible sets of sequence numbers inside the sender's window at time *t*? Justify your answer.

b. What are all possible values of the ACK field in all possible messages currently propagating back to the sender at time *t*? Justify your answer.

a. Se todos os pacotes antes de k forem confirmados pelo transmissor, a janela do transmissor é [k,k+N-1]. Do contrário, supondo que nenhum desses pacotes anteriores a k forem confirmados, a janela do transmissor é [k-N,k-1]. Assim, a janela do transmissor começa em algum lugar no intervalo [k-N,k].

8)

P24. Answer true or false to the following questions and briefly justify your answer:

a. With the SR protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

b. With GBN, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

c. The alternating-bit protocol is the same as the SR protocol with a sender and receiver window size of 1.

d. The alternating-bit protocol is the same as the GBN protocol with a sender and receiver window size of 1.

a. Sim, se a rede da camada de baixo re-ordenar os pacotes, ou caso haja timeout de todos os pacotes antes que o ACK chegue ao transmissor.
Por exemplo, o transmissor manda 1,2,3 no momento 0, no momento 1 o receptor reconhece e manda ack para os 3 pacotes. No momento 2, o transmissor aciona o timeout para os 3 pacotes e os reenvia. Logo, os 3 primeiros pacotes serão reenviados para o receptor, que já pulou uma janela. Assim, reconhecendo os pacotes antigos como pacotes novos.
b. Sim, mesma justificativa da resposta acima.
c. Sim.
d. Sim. Com uma janela de tamanho 1, SR e GBN são essencialmente um protocolo de alternância de bits. A janela de tamanho 1 contempla a possibilidade de recaída de um ack fora da janela, um ack cumulativo é somente um ack nessa situação, haja vista que só pode se referir para o pacote dentro da janela.

9)

a. For a given TCP connection, suppose four acknowledgments have been returned with corresponding sample RTTs: $SampleRTT_4$, $SampleRTT_3$, $SampleRTT_2$, and $SampleRTT_1$. Express $EstimatedRTT$ in terms of the four sample RTTs.

b. Generalize your formula for *n* sample RTTs.

c. For the formula in part (b) let *n* approach infinity. Comment on why this averaging procedure is called an exponential moving average.

a. Iremos somar os samples da seguinte forma:

$$EstimatedRTT_4 = xSampleRtt_1 + (1-x)[xSampleRtt_2 + (1-x)[xSampleRtt_3 + (1-x)SampleRtt_4]]$$

$$= xSampleRtt_1 + (1-X)xSampleRtt_2 + (1-x)^2 xSampleRtt_3 + (1-x)^3 SampleRtt_4$$

b. Genericamente, podemos expressar da seguinte forma:

$$EstimatedRtt_n = x\sum_{j=1}^{n-1}(1-x)^{j-1}SampleRtt_j + (1-x)^{n-1}SampleRtt_n$$

c. $EstimatedRtt_\infty = \frac{x}{1-x}\sum_{j=1}^{\infty}(1-x)^j SampleRtt_j$

O peso dado às amostras anteriores decai exponencialmente, por isso o nome exponencial.

10a) The Chebyshev Bound dictates that the probability for a random value exceeding a range around a mean depends on the standard deviation. Jacobson's Algorithm uses the Chebyshev Bound to produce reasonable retransmission time-out (RTO) values.

O algoritmo de Jacobson é utilizado para estimar valores convenientes para RTO, ou o tempo máximo de espera que o transmissor irá aguardar um ACK. Dado uma distribuição de rtts, é calculado a média e o desvio padrão. Jacobson estima esses valores das observações da distribuição, e então computa um "overbound" para o RTO usando Chebyshev. Após ter esperado um valor equivalente à média, depois o produto de uma constante K com o desvio padrão e o ACK ainda não chegou, logo, deve ter sido perdido. Esse cálculo de RTO é utilizado pelo transmissor para cada transmissão.

11)

P37. Compare GBN, SR, and TCP (no delayed ACK). Assume that the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments and their corresponding ACKs can be received (if not lost in the channel) by the receiving host (Host B) and the sending host (Host A) respectively. Suppose Host A sends 5 data segments to Host B, and the 2nd segment (sent from A) is lost. In the end, all 5 data segments have been correctly received by Host B.

a. How many segments has Host A sent in total and how many ACKs has Host B sent in total? What are their sequence numbers? Answer this question for all three protocols.
b. If the timeout values for all three protocol are much longer than 5 RTT, then which protocol successfully delivers all five data segments in shortest time interval?

a. No caso do GBN, como o segundo pacote foi perdido, o timeout irá estourar para ele. E será reenviado todos os pacotes após o segundo até o último enviado. Assim, como foi enviado $p1, p2, p3, p4, p5$, e como o pacote $p2$ foi perdido, $p2, p3, p4$ *e* $p5$ serão reenviados. O host B enviará 4 Acks na primeira leva de pacotes, haja visto que o segundo foi perdido. Depois mais 4 na segunda leva de pacote devido a retransmissão. Assim, o host B enviará 8 acks.

   No caso do SR, será retransmitindo apenas os pacotes que foram perdidos. Dessa forma, somente o pacote 2 será reenviado. O host B irá enviar 5 acks.

   O host A, no Tcp, enviará 6 segmentos no total. Eles são inicialmente 1,2,3,4,5 e depois o segmento 2 é reenviado.
   Já o host B, no Tcp, enviará 5 Acks. Eles são 4 acks com sequência numérica 2, dando a entender que precisa que o host A mande o pacote de sequência numérica 2. Depois, um ack com sequência numérica 6, solicitando o pacote com sequência numérica 6.

b. Tcp. Devido ao fato do tcp utilizar uma retransmissão rápida sem aguardar até o timeout.