Grupo: João Vítor Lioti Nunes, Josué da Silva Nascimento

Nesse relatório documentamos a lógica e as estruturas utilizadas no desenvolvimento do trabalho. Conforme os protocolos se tornam mais complexos, muitas lógicas continuaram as mesmas porém de forma expandida, a cada novo protocolo documentamos apenas as expansões realizadas.
A grande maioria dos comentários foram removidos no relatório, já que estamos explicando o passo a passo.

1. Estrutura do código:

Pegamos os argumentos e iniciamos o contador de pacotes. O contador de pacotes será somado sempre que houver um envio, falho ou bem sucedido. Em seguida chamamos a função flow_control_simulation() que será responsável por realizar o processamento.

```
arguments = sys.argv
if len(arguments) != 5:
    print("python flowctrlsim <protocol> <seqbits> <num_frames> <lost_pkts>")
    exit()

arg_protocol = ProtocolsEnum(arguments[1])
arg_sequence_of_bits = int(arguments[2])
arg_number_of_frames = int(arguments[3])
arg_lost_packets = [int(i) for i in arguments[4].split(",")]

global_packet_counter = 0

flow_control_simulation(
    protocol=arg_protocol,
    sequence_of_bits=arg_sequence_of_bits,
    number_of_frames=arg_number_of_frames,
    lost_packets=arg_lost_packets,
)
exit()
```

Temos um Enum para os protocolos assim como uma classe para os Packets.

```
class ProtocolsEnum(Enum):
    STOP_AND_WAIT_ARQ = "saw"
    GO_BACK_N_ARQ = "gbn"
    SELECTIVE_REPEAT_ARQ = "sr"


class Packet:
    payload: int
    retransmission: bool
    sequence_number: int
    time: int = 0
    max_time: int = 2
```

```
    ack: str = None
    timed_out = False
```

Dentro da função flow_control_simulation() temos o dicionário protocol_functions_dictionary que tem como chave opções do ProtocolsEnum e valores as funções de execução dos protocolos.

As três funções são funções aninhadas da função flow_control_simulation()

```
protocol_functions_dictionary = {
    ProtocolsEnum.STOP_AND_WAIT_ARQ: saw,
    ProtocolsEnum.GO_BACK_N_ARQ: gbn,
    ProtocolsEnum.SELECTIVE_REPEAT_ARQ: sr
}

protocol_functions_dictionary.get(protocol)()
```

Ao iniciar, antes de executar o protocolo, realizamos algumas tarefas comuns a todos protocolos:

```
max_sequence_number = 2 if protocol == ProtocolsEnum.STOP_AND_WAIT_ARQ else
(2 ** sequence_of_bits)
if protocol == ProtocolsEnum.SELECTIVE_REPEAT_ARQ:
    sequence_of_bits -= 1
window_size = 2 if protocol == ProtocolsEnum.STOP_AND_WAIT_ARQ else (2 **
sequence_of_bits)

packets = []
# create all packets for sending
for frame_number in range(number_of_frames):
    packets.append(
        Packet(
            payload=frame_number + 1,
            retransmission=False,
            sequence_number=frame_number % max_sequence_number,
        )
    )
```

Calculamos o max_sequence_number e tamanho da janela e criamos os packets para enviar.

2. Stop-and-wait ARQ

Antes de iniciar o loop de send e receive declaramos a global_packet_counter como uma variável global e inicializamos o contador de pacote atual.

```
global global_packet_counter
current_packet = 0
```

O loop executa até que o contador de pacote atual seja maior do que o total de pacotes.

```
while current_packet < number_of_frames:
    global_packet_counter += 1
```

O sender, a cada iteração, pega um pacote da lista, caso esse pacote deva ser perdido, é marcado para uso futuro como retransmissão, perdido e dado como timed out, caso contrário é enviado com sucesso.

```
    # sender
    packet = packets[current_packet]
    if global_packet_counter in lost_packets:
        packet.retransmission = True
        print(f"A -x B : ({packet.payload}) Frame {packet.sequence_number}")
        print(f"Note over A : TIMEOUT ({packet.payload})")
        continue
    print(
        f"A ->> B : ({packet.payload}) Frame {packet.sequence_number} {'(RET)'
if packet.retransmission else ''}"
    )
```

O receiver, a cada iteração envia um ACK com sucesso ou falha, dependendo do contador.

```
    global_packet_counter += 1
    # receiver
    if global_packet_counter in lost_packets:
        packet.retransmission = True
                    print(f"B   --x   A   :   Ack   {packet.sequence_number   %
max_sequence_number}")
        continue
    print(f"B -->> A : Ack {packet.sequence_number % max_sequence_number}")
    current_packet += 1
```

3. Go-back-n ARQ

Nessa implementação adicionamos, além do pacote atual, um contador de qual deve ser o próximo pacote a ser enviado.

```
global global_packet_counter
current_packet = 0
next_packet_in_window = 0
```

Torna-se necessário também filas para envio e recepção dos pacotes.

```
sender_queue = Queue()
receiver_queue = Queue()
while current_packet < number_of_frames:
```

Antes de realizar envios, checa se foram recebidos ACKs. ACKs são cumulativos, logo não verificamos uma ordem.

```
    # check for acks
    while not receiver_queue.empty():
        ack = receiver_queue.get()
        if ack > current_packet:
            current_packet = ack
```

Pacotes que passarem pela etapa anterior e não receberem um ACK chegarão mais próximo de um timeout.

```
    # if any packets went unacked
    for i in range(current_packet, next_packet_in_window):
        packets[i].time += 1
        if packets[i].time > 2:
            if current_packet < next_packet_in_window:
                print(f"Note over A : TIMEOUT ({current_packet + 1})")
            next_packet_in_window = current_packet
```

Enquanto houverem pacotes para enviar, e espaço na janela o sender continuará no loop de envio.

```
    # sender
    while (
             not next_packet_in_window - current_packet >= max_sequence_number
- 1
            and not next_packet_in_window > frame_number
    ):
        global_packet_counter += 1
        packet = packets[next_packet_in_window]
        if global_packet_counter in lost_packets:
            print(
                 f"A -x B : ({packet.payload}) Frame {packet.sequence_number}
{'(RET)' if packet.retransmission else ''}")
        else:
            print(
                f"A ->> B : ({packet.payload}) Frame {packet.sequence_number}
{'(RET)' if packet.retransmission else ''}"
            )
            sender_queue.put(packet)
        packet.time = 1
        packet.retransmission = True
        next_packet_in_window += 1
```

O receiver realiza a leitura da fila de envio e verifica se o pacote recebido é o esperado pelo seu sequence_number. Se sim, tenta realizar o envio do ACK.

```
    # receiver
    ack = current_packet
    while not sender_queue.empty():
        packet = sender_queue.get()
        if packet.sequence_number == ack % max_sequence_number:
            global_packet_counter += 1
            ack += 1
            if global_packet_counter in lost_packets:
                print(f"B --x A : Ack {ack % max_sequence_number}")
            else:
                print(f"B -->> A : Ack {ack % max_sequence_number}")
                receiver_queue.put(ack)
```

4. Selective repeat ARQ

Além do pacote atual, agora guardamos o ACK atual.

```
global global_packet_counter
current_packet = 0
current_ack = 0
next_packet_in_window = 0
```

Além das filas de envio e recepção, temos uma fila de reenvio para o sender, fila de espera de pacotes e buffers de ACK e NACK para o receiver.

```
sender_queue = Queue()
receiver_queue = Queue()
packets_to_resend = Queue()
packets_in_waiting = Queue()
ack_buffer = Queue()
nack_buffer = {}

while current_packet < number_of_frames:
```

O sender permanece o mesmo

```
    # We send the packages
    while (
            not next_packet_in_window - current_packet >= window_size
            and not next_packet_in_window > frame_number
    ):
        global_packet_counter += 1
        packet = packets[next_packet_in_window]
        if global_packet_counter in lost_packets:
            print(
                f"A -x B : ({packet.payload}) Frame {packet.sequence_number}
{'(RET)' if packet.retransmission else ''}")
        else:
            print(
                f"A ->> B : ({packet.payload}) Frame {packet.sequence_number}
{'(RET)' if packet.retransmission else ''}"
            )
            sender_queue.put(packet)
        packet.time = 1
        packet.max_time = 2
        packet.retransmission = True
        next_packet_in_window += 1
```

Temos diversas alterações no receptor

Temos três possíveis cenários quando recebendo um pacote:

- Recebemos o pacote esperado
- Recebemos um pacote já ACKed
    - Nesse caso utilizamos o ACK buffer para enviar o ACK mais cumulativo
- Recebemos um pacote fora da ordem esperada e enviamos um NACK
    - Nesse caso utilizamos o NACK buffer para enviar múltiplos NACKs se necessário.

```
    # We receive the packages
    while not sender_queue.empty():
        packet = sender_queue.queue[0]
```

Aqui tratamos o cenário 1, ACK em um pacote esperado.

```
        if packet.sequence_number == current_ack % max_sequence_number:

            if packet.sequence_number in nack_buffer.keys():
                nack_buffer.pop(packet.sequence_number)

            sender_queue.get()
            global_packet_counter += 1

            current_ack += 1
            packet.ack = "ACK"
            most_cumulative_ack = packet

            if ack_buffer.qsize() <= window_size:
                ack_buffer.put(packet)
            else:
                ack_buffer.get()
                ack_buffer.put(packet)
```

Após reconhecer o pacote, não enviamos ainda o ACK. Verificamos se o pacote seguinte esperado está na lista de espera. Realizamos essa verificação até que a fila de espera esteja vazia ou o pacote não seja o esperado.

```
            while not packets_in_waiting.empty():
                packet = packets_in_waiting.queue[0]

                                    if packet.sequence_number == current_ack %
max_sequence_number:
                    if packet.sequence_number in nack_buffer.keys():
                        nack_buffer.pop(packet.sequence_number)

                    packets_in_waiting.get()
                    current_ack += 1
                    packet.ack = "ACK"
                    most_cumulative_ack = packet
```

Continuação da iteração pela lista de espera

```
                    # adding to buffer
                    if ack_buffer.qsize() <= window_size:
                        ack_buffer.put(packet)
                    else:
                        ack_buffer.get()
                        ack_buffer.put(packet)

                    continue
                break
```

Após encontrar o ACK mais cumulativo, tentamos realizar o envio.

```
            if global_packet_counter in lost_packets:
                print(f"B --x A : Ack {current_ack % max_sequence_number}")
            else:
                print(f"B -->> A : Ack {current_ack % max_sequence_number}")
                receiver_queue.put(most_cumulative_ack)
            continue
```

Aqui tratamos o cenário 2. Reconhecemos esse cenário pois o sequence_number do pacote está no buffer de pacotes que já enviamos o ACK. Enviamos o ACK mais recente.

```
        elif packet.sequence_number in [buffered_packet.sequence_number
                                                for buffered_packet in
ack_buffer.queue]:
            sender_queue.get()
            global_packet_counter += 1
            if global_packet_counter in lost_packets:
                print(f"B --x A : Ack {current_ack % max_sequence_number}")
            else:
                print(f"B -->> A : Ack {current_ack % max_sequence_number}")
                receiver_queue.put(ack_buffer.queue[-1])
            continue
```

Em último caso, temos o cenário 3. Montamos ou adicionamos ao NACK buffer, e então percorremos ele enviando os NACKs presentes no buffer. O NACK buffer é limpo sempre que um pacote é identificado como esperado no cenário 1.

```
        else:
            # If there are packages in waiting we check to see if extra
NACKs need to be added
            if not packets_in_waiting.empty():
                i = packets_in_waiting.queue[-1].sequence_number
                while i != packet.sequence_number:
                    if i not in [packet_in_waiting.sequence_number for
packet_in_waiting in
                                 packets_in_waiting.queue]:
                        nack_buffer[i] = Packet(payload=i, sequence_number=i %
max_sequence_number, ack="nak")
                    i = (i + 1) % max_sequence_number
```

```
            # If there is no package in waiting we add a single NACK
            else:
                                nack_buffer[current_ack  %  max_sequence_number]  =
Packet(payload=current_ack,
sequence_number=current_ack                %                max_sequence_number,
ack="NAK")

             # After managing the buffers, we add the package to the waiting
list and send NACKs
            sender_queue.get()
            packets_in_waiting.put(packet)
            if sender_queue.empty():
                for packet in nack_buffer.values():
                    global_packet_counter += 1
                    if global_packet_counter in lost_packets:
                        print(f"B --x A : NAK {packet.sequence_number}")
                    else:
                        print(f"B -->> A : NAK {packet.sequence_number}")
                        receiver_queue.put(packet)
```

Finalizamos acima, o receptor. Após envio dos ACKs/NAKs realizamos a leitura deles. Pacotes que não receberem ACK, seja por perda ou NAK vão para a lista de reenvio.

```
    while not receiver_queue.empty():
        ack: Packet = receiver_queue.get()
        if ack.ack == "ACK":
            if ack.payload > current_packet:
                current_packet = ack.payload
            continue
        current_packet = ack.payload
        packets_to_resend.put(packets[ack.payload])
```

Avançamos os timers dos pacotes e separamos o mais antigo que já expirou e não está marcado para reenvio

```
    oldest_expired_package = None
for i in range(current_packet, next_packet_in_window):
    packets[i].time += 1
        if  packets[i].time  >  packets[i].max_time  and  packets[i]  not  in
packets_to_resend.queue:
        if oldest_expired_package is None:
            oldest_expired_package = i
            continue
        if packets[oldest_expired_package].time < packets[i].time:
            oldest_expired_package = i
```

Caso o pacote mais antigo expirado esteja marcado para timeout, efetivamos o timeout dele, senão ele é marcado para timeout.

```
 if oldest_expired_package is not None:
    if packets[oldest_expired_package].timed_out:
        print(f"Note over A : TIMEOUT ({oldest_expired_package + 1})")
        packets[oldest_expired_package].time = 1
        packets_to_resend.put(packets[oldest_expired_package])
    else:
        packets[oldest_expired_package].timed_out = True
```

Por fim, reenviamos os pacotes marcados para reenvio.

```
    # Before starting the next iteration, we resend packages marked for
resending
    while not packets_to_resend.empty():
        global_packet_counter += 1
        packet = packets_to_resend.get()
        if global_packet_counter in lost_packets:
            print(
                f"A -x B : ({packet.payload}) Frame {packet.sequence_number}
{'(RET)' if packet.retransmission else ''}")
        else:
            print(
                f"A ->> B : ({packet.payload}) Frame {packet.sequence_number}
{'(RET)' if packet.retransmission else ''}"
            )
            sender_queue.put(packet)
```

5. Exemplos de execução
   SAW [Link para o Mermaid]

```
saw 1 4 3,6
A ->> B : (1) Frame 0
B -->> A : Ack 0
A -x B : (2) Frame 1
Note over A : TIMEOUT (2)
A ->> B : (2) Frame 1 (RET)
B -->> A : Ack 1
A -x B : (3) Frame 0
Note over A : TIMEOUT (3)
A ->> B : (3) Frame 0 (RET)
B -->> A : Ack 0
A ->> B : (4) Frame 1
B -->> A : Ack 1
```

GBN [Link para o Mermaid]

```
gbn 3 10 2,3,8,10
A ->> B : (1) Frame 0
A -x B : (2) Frame 1
A -x B : (3) Frame 2
A ->> B : (4) Frame 3
A ->> B : (5) Frame 4
A ->> B : (6) Frame 5
A ->> B : (7) Frame 6
B --x A : Ack 1
Note over A : TIMEOUT (1)
A ->> B : (1) Frame 0 (RET)
A -x B : (2) Frame 1 (RET)
A ->> B : (3) Frame 2 (RET)
A ->> B : (4) Frame 3 (RET)
A ->> B : (5) Frame 4 (RET)
A ->> B : (6) Frame 5 (RET)
A ->> B : (7) Frame 6 (RET)
B -->> A : Ack 1
A ->> B : (8) Frame 7
Note over A : TIMEOUT (2)
A ->> B : (2) Frame 1 (RET)
A ->> B : (3) Frame 2 (RET)
A ->> B : (4) Frame 3 (RET)
A ->> B : (5) Frame 4 (RET)
A ->> B : (6) Frame 5 (RET)
A ->> B : (7) Frame 6 (RET)
A ->> B : (8) Frame 7 (RET)
B -->> A : Ack 2
B -->> A : Ack 3
B -->> A : Ack 4
B -->> A : Ack 5
B -->> A : Ack 6
B -->> A : Ack 7
B -->> A : Ack 0
A ->> B : (9) Frame 0
A ->> B : (10) Frame 1
B -->> A : Ack 1
B -->> A : Ack 2
```

SR [Link para o Mermaid]

```
sr 3 10 3,7,8,11,15
A ->> B : (1) Frame 0
A ->> B : (2) Frame 1
A -x B : (3) Frame 2
A ->> B : (4) Frame 3
B -->> A : Ack 1
B -->> A : Ack 2
B --x A : NAK 2
A -x B : (5) Frame 4
A ->> B : (6) Frame 5
B -->> A : NAK 2
B --x A : NAK 4
A ->> B : (3) Frame 2 (RET)
B -->> A : Ack 4
A ->> B : (7) Frame 6
A -x B : (8) Frame 7
B -->> A : NAK 4
A ->> B : (5) Frame 4 (RET)
B -->> A : Ack 7
A ->> B : (9) Frame 0
A ->> B : (10) Frame 1
B -->> A : NAK 7
A ->> B : (8) Frame 7 (RET)
B -->> A : Ack 2
```