1.INTERNET CHECKSUM

COMPUTING AN INTERNET CHECKSUM

Consider the two 16-bit words (shown in binary) below. Recall that to compute the Internet checksum of a set of 16-bit words, we compute the one's complement sum [1] of the two words. That is, we add the two numbers together, making sure that any carry into the 17th bit of this initial sum is added back into the 1's place of the resulting sum); we then take the one's complement of the result. Compute the Internet checksum value for these two 16-bit words:

10100111 11101110 *this binary number is 42990 decimal (base 10)*

00000101 11100001 *this binary number is 1505 decimal (base 10)*

Câu 1: What is the sum of these two 16 bit numbers? Don't put any spaces in your answer

10101101 11001111

Câu 2: Using the sum from question 1, what is the checksum? Don't put any spaces in your answer

01010010 00110000

RELIABLE DATA TRANSFER: RDT2.2 (SENDER AND RECEIVER ACTIONS)

Consider the rdt2.2 protocol from the text (pages 209-212). The FSMs for the sender and receiver are shown below:

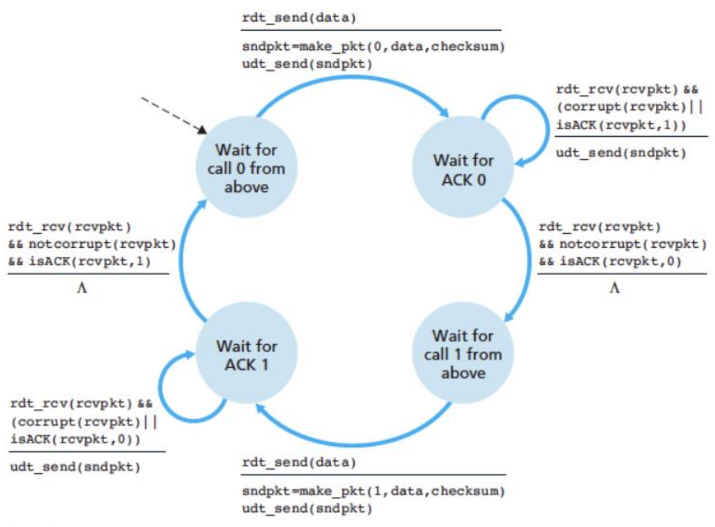
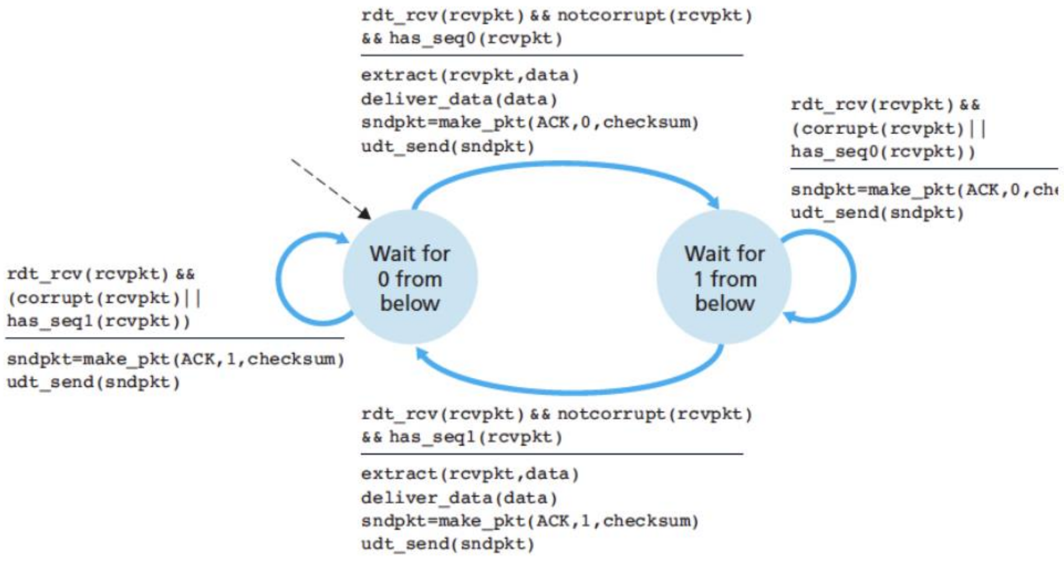

rdt_send(data)
sndpkt=make_pkt(0,data,checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| isACK(rcvpkt,1))
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
Λ
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
Λ
Wait for ACK 1
Wait for call 1 from above
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| isACK(rcvpkt,0))
udt_send(sndpkt)
rdt_send(data)
sndpkt=make_pkt(1,data,checksum)
udt_send(sndpkt)

Figure 3.13 ♦ rdt2.2 sender

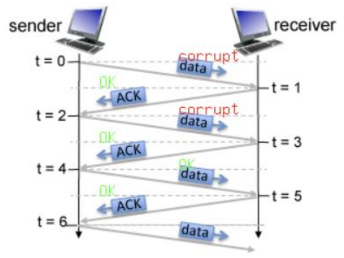
Figure 3.13 ♦ rdt2.2 sender



rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,0,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| has_seq0(rcvpkt))
sndpkt=make_pkt(ACK,0,ch
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)|| has_seq1(rcvpkt))
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,1,checksum)
udt_send(sndpkt)

Figure 3.14 ♦ rdt2.2 receiver

Suppose that the channel connecting the sender and receiver can corrupt but not lose or reorder packets. Now consider the figure below, which shows four data packets and three corresponding ACKs being exchanged between an rdt 2.2 sender and receiver. The actual corruption or successful transmission/reception of a packet is indicated by the corrupt and OK labels, respectively, shown above the packets in the figure below.



Câu 1: At time t=0, what is the sender state?

Wait for ACK0

Câu 2: At time t=0, what is the receiver state?

Wait for 0 from below

Câu 3: At time t=0, what is the sequence/ack # of the packet?

0

Câu 4: At time t=1, what is the sender state?

Wait for ACK0

Câu 5: At time t=1, what is the receiver state?

Wait for 0 from below

Câu 6: At time t=1, what is the sequence/ack # of the packet?

1

Câu 7: At time t=2, what is the sender state?

Wait for ACK0

Câu 8: At time t=2, what is the receiver state?

Wait for 0 from below

Câu 9: At time t=2, what is the sequence/ack # of the packet?

0

Câu 10: At time t=3, what is the sender state?

Wait for ACK0

Câu 11: At time t=3, what is the receiver state?

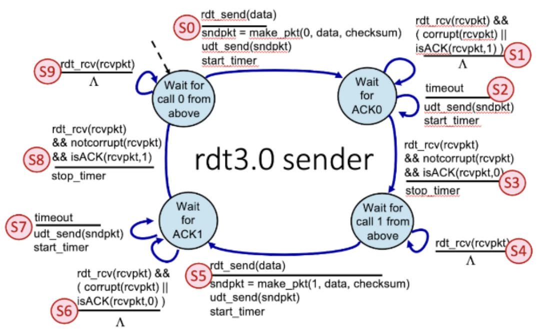
Wait for 0 from below

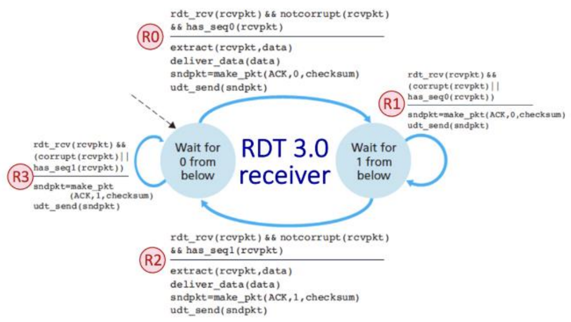
Câu 12: At time t=3, what is the sequence/ack # of the packet?

1

RELIABLE DATA TRANSFER: RDT 3.0

Consider the RDT 3.0 protocol, for reliably communicating data from a sender to receiver over a channel that can lose or corrupt packets in either direction, and when the maximum delay from sender to receiver and back is not known. The FSMs for the sender and receiver are shown below, with their transitions labeled as SX and RY, respectively.





Now let's consider the sequence of sender and receiver transitions that would happen when one or more of the following complications occur: a packet (data or ACK) is lost, a timer times out (prematurely or not), or a message is corrupted. One or more of these events has occurred to produce the sequence of transitions below. In the sequence below, one transition has been omitted and replaced with a "*".

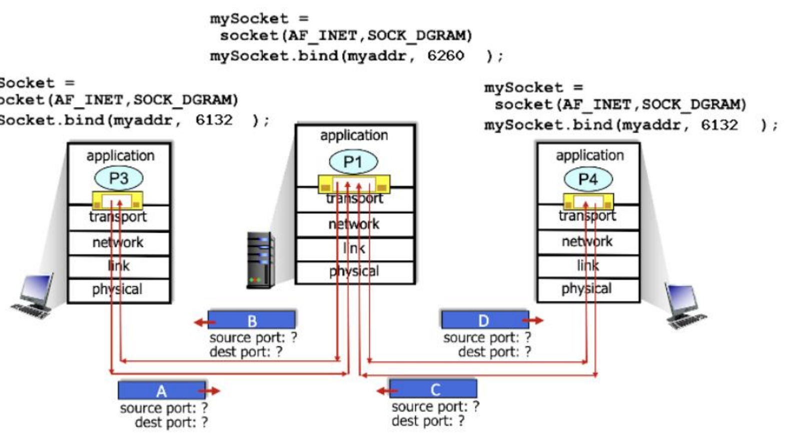
Transition Sequence: S0, R3, S1, S2, R0, S1, *, R1, S3, S5, R1, S6, S7, R2, S8

Câu 1: What is the missing transition? To indicate the missing transition, enter S or R, followed by an index.

S2

UDP MULTIPLEXING AND DEMULTIPLEXING

In the scenario below, the left and right clients communicate with a server using UDP sockets. The same socket at the server is used to communicate with both clients. The Python code used to create the sockets is shown in the figure. Consider the four transport-layer packets – A, B, C and D – shown in the figure below.



```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr, 6260  );
```

```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr, 6132  );
```

```
mySocket =
 socket(AF_INET,SOCK_DGRAM)
mySocket.bind(myaddr, 6132  );
```

Câu 1: What is the source port # for packet B?

Port 6260

Câu 2: What is the destination port # for packet B?

Port 6132

Câu 3: What is the source port # for packet D?

Port 6132

Câu 4: What is the destination port # for packet A?

Port 6260

Câu 5: What is the source port # for packet C?

Port 6132

Câu 6: What is the destination port # for packet C?

Port 6260

Câu 7: What is the source port # for packet D?

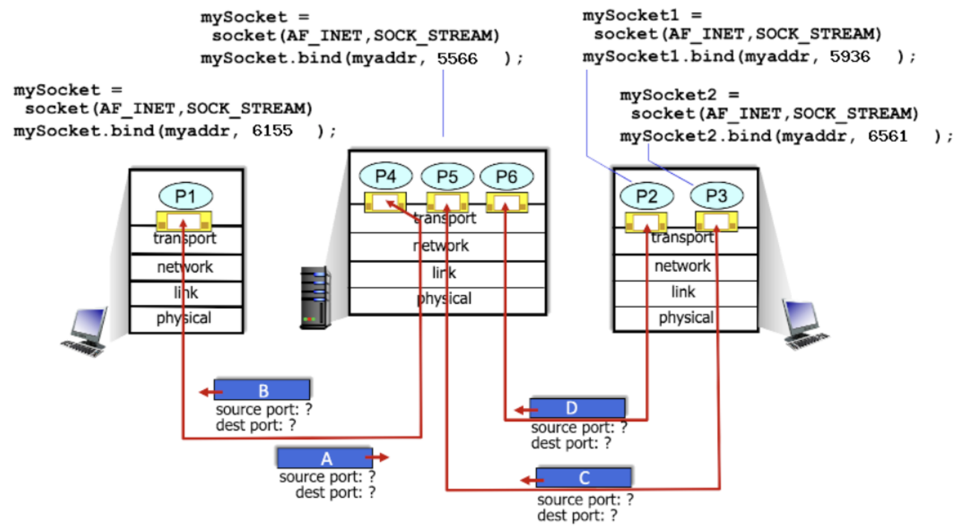
Port 6260

Câu 8: What is the destination port # for packet D?

Port 6132

TCP MULTIPLEXING AND DEMULTIPLEXING

In the scenario below, the left and right TCP clients communicate with a TCP server using TCP sockets. The Python code used to create a single welcoming socket in the server is shown in the figure (the welcoming socket itself is not shown graphically); code is also shown for the client sockets as well. The three sockets shown in server were created as a result of the server accepting connection requests on this welcoming socket from the two clients (one connection from the client on the left, and two connections from the client on the right).


mySocket = socket(AF_INET,SOCK_STREAM) mySocket.bind(myaddr, 5566);
mySocket1 = socket(AF_INET,SOCK_STREAM) mySocket1.bind(myaddr, 5936);
mySocket = socket(AF_INET,SOCK_STREAM) mySocket.bind(myaddr, 6155);
mySocket2 = socket(AF_INET,SOCK_STREAM) mySocket2.bind(myaddr, 6561);
P1
P2
P3
P4
P5
P6
transport
network
link
physical
B source port: ? dest port: ?
A source port: ? dest port: ?
D source port: ? dest port: ?
C source port: ? dest port: ?

Câu 1: What is the source port # for packet A?

Port 6155

Câu 2: What is the destination port # for packet A?

Port 5566

Câu 3: What is the source port # for packet B?

Port 5566

Câu 4: What is the destination port # for packet B?

Port 6155

Câu 5: What is the source port # for packet C?

Port 6561

Câu 6: What is the destination port # for packet C?

Port 5566

Câu 7: What is the source port # for packet D?

Port 5936

Câu 8: What is the destination port # for packet D?

Port 5566