Assignment-2 : Report

Anish Teja Bramhajosyula (**IMT2024029**)

EGC-123 : Computer Networks

Question-1 : RDT-2.2

Implementation

The simulation generates a random number of random 16-bit payloads and transmits them one by one sequentially. The sender sends a packet and it is sent through an unreliable channel, which introduces random errors. At the receiver's side, the checksum is calculated and error detection is implemented. Then, the receiver sends an appropriate ACK through the unreliable channel and it may also have errors, which are detected by the sender.

Parameter Choices

The classes `Packet`, `Sender` and `Receiver` are used to encapsulate the behaviour of these respective entities. Here, a payload is chosen as a 16-bit number, and the checksum is calculated as the sum of this value and a 16-bit extended 1-bit sequence number and their one's complement. It has been implemented so that the probability of a bit flip is 50%, corruption of an ACK sequence number is 10%, and payload corruption in an ACK is 30%. The data type `uint16_t` (present in the `cstdint` header) is used for handling all 16-bit values. The headers `chrono` and `thread` are used for time functionality and delays.

Sample Output

```
--------------SIMULATION (0xBF6D)---------------

(SENDER)   -> Sent packet with seq. no.: 0
(RECEIVER) -> Corrupt Packet: Sending cumulative ACK
(RECEIVER) -> Sending ACK for seq. no.: 1
(SENDER)   -> Corrupt ACK: Packet Resent
(SENDER)   -> Sent packet with seq. no.: 0
(RECEIVER) -> Packet with seq no.: 0 received: Delivering data from payload: 0xBF6D
(RECEIVER) -> Sending ACK for seq. no.: 0
(SENDER)   -> Corrupt ACK: Packet Resent
(SENDER)   -> Sent packet with seq. no.: 0
(RECEIVER) -> Duplicate Packet: Sending cumulative ACK
(RECEIVER) -> Sending ACK for seq. no.: 0
(SENDER)   -> Packet with seq. no.: 0 ACK'ed correctly

--------------TRANSMISSION COMPLETE--------------
```

Note that this is only a portion of the actual output displayed when the file `rdt2_2.cpp` is executed.

Question-2 : RDT-3.0

Implementation

The simulation generates a random number of random 16-bit payloads and transmits them one by one sequentially. The sender sends a packet and it is sent through an unreliable channel, which introduces random errors. At the receiver's side, the checksum is calculated and error detection is implemented. Then, the receiver sends an appropriate ACK through the unreliable channel and it may also have errors, which are detected by the sender. In addition to this, timeouts are also triggered whenever packets get dropped/corrupt ACK's are received.

Parameter Choices

The classes `Packet`, `Sender` and `Receiver` are used to encapsulate the behaviour of these respective entities. Here, a payload is chosen as a 16-bit number, and the checksum is calculated as the sum of this value and a 16-bit extended 1-bit sequence number and their one's complement. It has been implemented so that the probability of a bit flip is 50%, corruption of an ACK sequence number is 10%, and payload corruption in an ACK is 30%. The data type `uint16_t` (present in the `cstdint` header) is used for handling all 16-bit values. The headers `chrono` and `thread` are used for time functionality and delays. There is a 25% chance that any packet (either a sender's packet or an ACK) may be dropped at any point during transmission.

Sample Output

```
---------------SIMULATION (0x6F3F)----------------

(SENDER)   -> Sent packet with seq. no.: 1
(RECEIVER) -> Corrupt Packet: Sending cumulative ACK
(RECEIVER) -> Sending ACK for seq. no.: 0
(SENDER)   -> Packet Corrupted in Channel
(SENDER)   -> Timeout: Packet/ACK Lost/Corrupted: Resending Packet
(SENDER)   -> Sent packet with seq. no.: 1
(RECEIVER) -> Packet with seq no.: 1 received: Delivering data from payload: 0x6F3F
(RECEIVER) -> Sending ACK for seq. no.: 1
(SENDER)   -> Corrupt ACK
(SENDER)   -> Timeout: Packet/ACK Lost/Corrupted: Resending Packet
(SENDER)   -> Sent packet with seq. no.: 1
(RECEIVER) -> Duplicate Packet: Sending cumulative ACK
(RECEIVER) -> Sending ACK for seq. no.: 1
(SENDER)   -> Packet with seq. no.: 1 ACK'ed correctly

--------------TRANSMISSION COMPLETE---------------
```

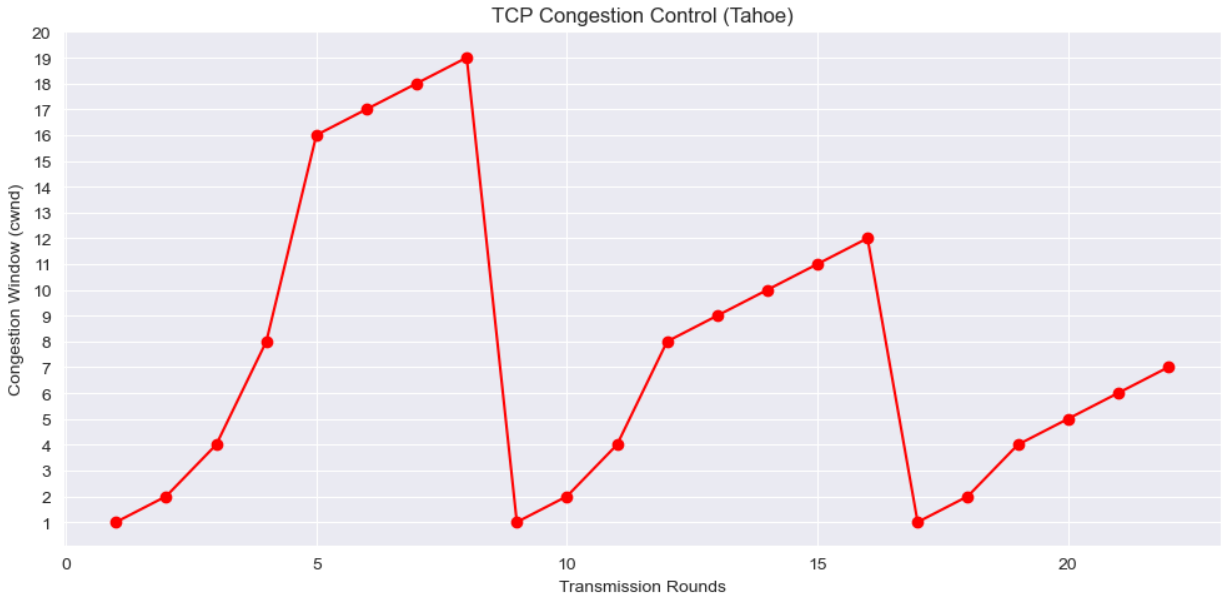
Note that this is only a portion of the actual output displayed when the file `rdt3_0.cpp` is executed.

Question-3 : TCP (Tahoe) Congestion Control

Implementation & Parameter Choices

The simulation begins by setting the congestion window size (`cwnd`) to 1 and a randomly generated threshold value `ssthresh`. The transmission rounds at which loss occurs are predefined. The simulation runs for at most 30 transmission rounds, or until `ssthresh` reaches 1. During each simulation, the value of `cwnd` is checked to determine whether it is in the slow start phase or in the congestion avoidance phase, and `cwnd` is updated as specified: Double if in slow start, increment by 1 if in congestion avoidance and drop to 1 and halve `ssthresh` if packet loss or duplicate ACK is detected.

Sample Output



Graph generated using `matplotlib`

TR	CWND
1	1
2	2
3	4
4	8
5	16
6	17
7	18
8	19
9	1
10	2
11	4
12	8
13	9
14	10
15	11
16	12
17	1
18	2
19	4
20	5
21	6
22	7

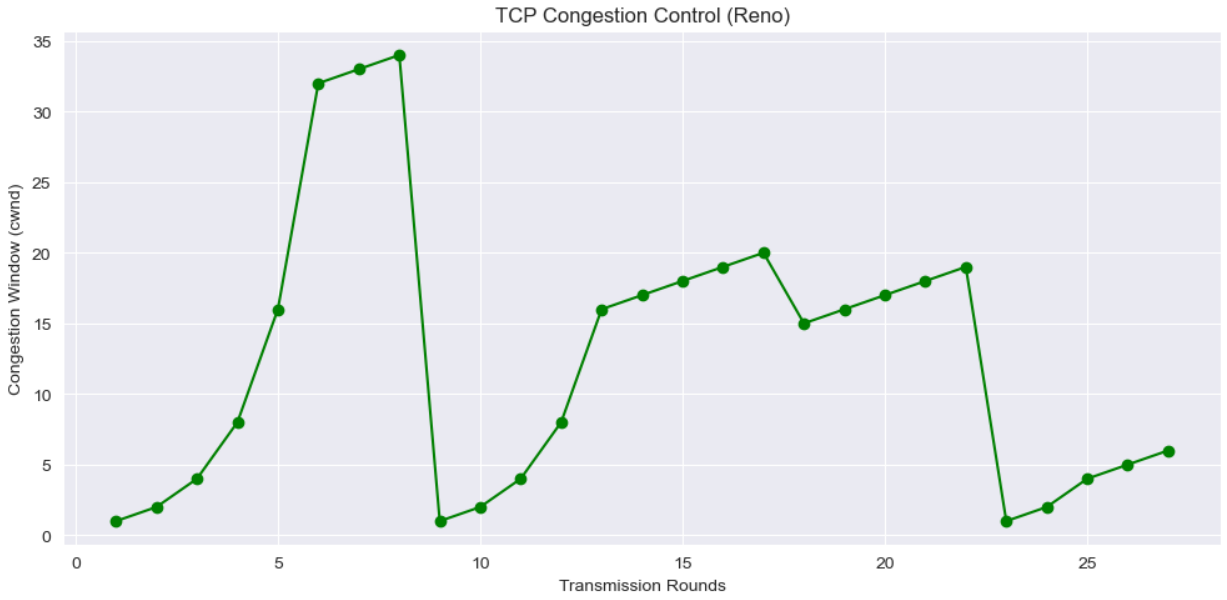
CSV File generated by `tahoe.cpp`

Question-4 : TCP (Reno) Congestion Control

Implementation & Parameter Choices

The simulation begins by setting the congestion window size (`cwnd`) to 1 and a randomly generated threshold value `ssthresh`. The transmission rounds at which loss occurs are predefined. The simulation runs for at most 30 transmission rounds, or until `ssthresh` reaches 1. During each simulation, the value of `cwnd` is checked to determine whether it is in the slow start phase or in the congestion avoidance phase, and `cwnd` is updated as specified: Double if in slow start, increment by 1 if in congestion avoidance and drop to 1 and halve `ssthresh` if packet loss is detected, and set `cwnd` to `ssthresh + 3` if duplicate ACK is detected.

Sample Output



Graph generated using `matplotlib`

TR	CWND
1	1
2	2
3	4
4	8
5	16
6	32
7	33
8	34
9	1
10	2
11	4
12	8
13	16
14	17
15	18
16	19
17	20
18	15
19	16
20	17
21	18
22	19
23	1
24	2
25	4
26	5
27	6

CSV File generated by `reno.cpp`

Warning

The programs `rdt2_2.cpp` and `rdt3_0.cpp` generate a random number of random messages and transmit them. The errors introduced by the channel are also random. As a result, in rare cases, it may happen that they may only transmit a single packet, or packets may not have any errors at all, or packets may take too long to be transmitted successfully. In such cases, please re-execute the program, and the output should be reasonable. Care has been taken so that no infinite loops are generated.