RDT over UDP Report

**UDP Header Format**

|  |  |  |  |
| --- | --- | --- | --- |
| TCP header field | Functionality / purpose | Needed for “rdt over UDP”? | Justification of your choice |
| Source port # | Identifies the sending process | Needed, but not included | Redundant, since source port # is already present in the UDP header |
| Destination port # | Identifies the destination process | Needed, but not included | Redundant, since destination port # is already present in the UDP header |
| Sequence # | Identified the packet sent | Needed | Both GBN and SR rely on the sequence # of a packet to determine if packets are out of order or have been lost. |
| Acknowledgement # | Value of the next sequence # that the receiver is expecting to receive. | Not needed | A window is maintained by the sender. The receiver does not need knowledge on the size of the window in our implementation. |
| Data offset | Specifies the offset so that the package data can be referenced. | Needed, but not included. | Redundant, since data offset is already present in the UDP header as “length.” |
| Reserved | Bits reserved for future use. | Not needed. | We do not require reserved bits. |
| N S | Protection against concealment | Not needed. | Out of scope. |
| C W R | Congestion window reduced flag. | Not needed. | Out of scope. |
| E C E | Indicates TCP is ECN compatible. | Not needed. | Out of scope. |
| U R G | A flag used to indicate that a packet has a higher priority. | Not needed. | Our UDP packets all have the same priority. |
| A C K | A flag used as a receipt for a received packet. | Not needed | ACKS are only sent from receiver to sender in our implementation, so there is no need to include this bit in the header. |
| P S H | Flag to indicate receiver should push to buffered data to application layer. | Not needed. | Out of scope. |
| R S T | Reset flag to indicate packet was sent to different host. | Not needed. | Out of scope. |
| S Y N | Used to establish a three-way handshake. | Not needed. | UDP is a connectionless protocol, no connection establishment is required. |
| F I N | Used to signify the end of a transmission and to end a TCP connection | Not needed. | This is similar to the SYN bit; just as there is no hand-shake, there is no need to signal the end of a transmission in UDP. |
| Window Size | The size of the window for sending packets and receiving packets. | Needed. | Both the sender and the receiver will be required to keep a window to keep track of which packets have been acked and which packets have not. |
| Checksum | A numeric value used to detect lost data or corrupted data. | Needed | Checksum must be included to ensure integrity of packet. |
| Urgent Pointer | Gives priority to packets. | Not needed. | All our data has the same priority. |
| Options | Options available to TCP | Not needed. | We will not require any additional options. |
| Padding | 0 bits used to pat the header so that it ends on a 32 bit length/boundary. | Not needed. | In our implementation, this is not necessary. |

**Validation Scenarios**

GBN Basic

*What is the expected relationship between CORRUPT\_PROBA and (numCorrupts / numTransmits)?*

As the CORRUPT\_PROBA is increased so are the numCorrupts and numTransmits.

*What do you observe?*

The number of numCorrupts is 540 which is about 10% of the total necessary packets needed to transmit the entire 500K file. This is in line with the CORRUPT\_PROBA of 10 or 10%.

*What is the expected relationship between numCorrupts and numErrors?*

They should be directly related. They should be about the same. The numCorrupts indicates the number of packets with corrupted data sent by the sender. The number of packets received with errors at the receiver side is numError. Hence, the numCorrupts and numErrors should be the same.

*What do you observe?*

From the output file of the sender we can see that the number of corrupted packets or numCorrupts is 540. On the output file of the receiver we can see that the number of checksum errors or numErrors is 540. Hence, numCorrupts and numErrors are the same.

*What is the expected relationship between TOevents and numCorrupts?*

There should be no direct relation between the two. The total number of corrupted packets has no effect on the number of packets that have timed out.

*What do you observe?*

Indeed, we see that there were 540 corrupted packets, but the number of times out packets is zero (as the delay on the links is 0ms).

*What is the expected relationship between numRetransmits and TOevents?*

They should be directly related. A retransmission of a packet is caused by a time-out of that packet.

*What do you observe?*

Because the delay on the links are all set to 0ms none of the packets cause a timeout. Hence, TOevents is zero and numRetransmits is also zero.

SR Basic

*What is the expected relationship between CORRUPT\_PROBA and (numCorrupts / numTransmits)?*

Again, as the CORRUPT\_PROBA is increased so are the numCorrupts and numTransmits.

*What do you observe?*

The number of corrupted packets was 556 and this increased the total number of packets transmitted as they had to be re-sent.

*What is the expected relationship between numCorrupts and numErrors?*

The sumcheck function will always indicate an error when a packet with corrupted data is detected. Hence, the numCorrupts and numErrors are directly relation in that they always equal.

*What do you observe?*

The number of corrupted packets sent were 556 and the receiver registered a number of 556 checksum errors. This agrees with our expectation.

*What is the expected relationship between TOevents and numCorrupts?*

There should be no direct relationship between these two statistical variables. A packets corrupted data should have no barring on whether it time out or not.

*What do you observe?*

Although there were 556 packets with corrupted data sent there were no timeout events registered and this confirms our expectations.

*What is the expected relationship between numRetransmits and TOevents?*

They should be directly related. This is because a retransmission occurs when a packet times-out.

*What do you observe?*

In this validation no packets timeout as the delay in the topology was set to zero, but the number of timeout events and retransmitted packet variables also remained zero.

GBN Window

*What do you observe? Explain*

GBN’s throughput remains fairly static even when the window size is increased.

SR Window

*What do you observe? Explain*

SR’s throughput increase as the window size is increased because unlike GBN, SR is only required to retransmit the singular packets that have been lost. Hence, there is no risk of having a larger window size and a larger traffic of packets in the network at anyone time. The higher the window size the higher SR’s effective throughput will be.

*Plot the effective throughput as a function of different window sizes (10, 20, 40 packets). It is expected that the throughput should increase with the window size. All parameters being equal, do you expect the SR throughput to be about the same, or greater than, or less than the GBN throughput?*

SR throughput should be much greater than GBN because of the fundamental way in which GBN operates. If the window is increased, then there are more packets on the network at anyone time. Because GBN forces all unsuccessfully delivered packets to be resent if any preceding packet was lost then the throughput is not going to be very high and could possibly decreased as it is more likely that a larger number of packets must be resent.

GBN Payload

*What do you observe? Explain.*

There is an increase in throughput for GBN. A smaller payload size indicates that more packets will be required to deliver all the data. This larger number of packets in the network increases the number of packets affected by delay and loss. Hence, when the payload for a packet is larger the number of packets that suffer from loss and delay is reduced because we simply have fewer packets in the system. Thus, the effective throughput increases as the payload increases.

SR Payload

*What do you observe? Explain.*

There is a major increase in effective throughput as the payload size increases. This is because any retransmission is only associated with one packet, hence there are no mass-sequential retransmission and the total cost of retransitions is negligible compared to the gains in effective throughput. Additionally, for the same reasons as stated for GBN, simply having fewer packets with higher payload in the network increases throughput because the number of total packets that incur a loss or time out is smaller.

*Plot the effective throughput as a function of different payload sizes (25, 50, 100 bytes) for GBN and SR. It is expected that the throughput should increase with the payload size. All parameters being equal, do you expect the SR throughput to be about the same, or greater than, or less than the GBN throughput?*

Effective throughput should be greater in SR because we only retransmit a packet that was lost rather than the lost packet and all packets that arrived after it as in GBN. GBN suffers from having to retransmit any packet that has successfully arrived after a lost packet.

GBN Delay

*What do you observe? Explain.*

GBN’s throughput remains fairly constant even as the delay is increased. This could be due to the fact that there is no difference when it comes to thought put is the first packet in a sequence of packets is lost versus the entire sequence of packets being lost. In both scenarios all packets must be retransmitted, and so effective throughput remains relatively constant.

SR Delay

*What do you observe? Explain.*

SR’s throughput decreases at a much larger rate than GBN. As stated before this is because SR begins to behave like GBN in that it retransmits an increasingly larger amount of packets.

*Plot the effective throughput as a function of different one-way delay values (0, 5, 10 msec) for GBN and SR. It is expected that the throughput should decrease with the delay value. All parameters being equal, do you expect the SR throughput to be about the same, or greater than, or less than the GBN throughput?*

As the delay increases we expect a convergence in the effective throughput for both GBN and SR. If the delay is large enough (as in the case for 10 ms) then both algorithms must resend a fairly large portion of their total packets. In this case SR begins to behave like GBN in its rate of retransitions. So for low delays SR will have higher throughput, but as the delay increases SR’s throughput will quickly diminish and near GBN’s throughput.

GBN Loss

*What do you observe? Explain*

From the graph we see that the effective throughput did decrease as loss increases. The effective throughput of GBN should decrease because as the loss is increased any packets that have been successfully delivered after a lost packet will have to be retransmitted and this incurs a high cost on throughput. Since the lost rate is increased more and more packets have to be retransmitted hence the throughput decreases.

SR Loss

*What do you observe? Explain*

SR throughput should be greater simply because SR is more efficient and at any loss event only one packet must be resent (the lost packet). The throughput for SR is still higher than that of GBN but there was a much larger drop in throughput for SR as the loss was increased.

*Plot the effective throughput as a function of different packet loss ratios (5%, 10%, 20%) for GBN and SR. It is expected that the throughput should decrease with the loss ratio. All parameters being equal, do you expect the SR throughput to be about the same, or greater than, or less than the GBN throughput?*

SR throughput should be greater simply because SR is more efficient and at any loss event only one packet must be resent (the lost packet).

**Issues and Lessons Learned**

The biggest problem we had in writing the program was with troubleshooting small, seemingly simple syntactical errors. The logic for implementing go-back-n and SR was actually fairly simple, and writing structures that controlled the packets was fairly simple. We found the syntax to be incredibly touchy, and spent upwards of 8 hours troubleshooting things that were not behaving as expected. For instance, there was one particular conditional statement that could absolutely not be changed or the program would cease to function. Even replacing the conditional expression with something like “if true:” instead of “if [expr]:” would completely break the program. Errors like this were not being detected by the python compiler, and the scripts would often run as expected individually, but would break on running the topography python script. Nothing I came across in my research explained why this specific conditional would be so fragile. Troubleshooting errors and bugs like these drew out the experience long after we figured out the logic.

Another issue that we ran into was the lack of online resources for mininet. In troubleshooting the errors expressed above, it was hard to distinguish when an issue was related to a logical or syntax error, python, mininet, or our Linux virtual machine. Googling error codes we were receiving often led to dead ends, which warranted meticulous line-by-line troubleshooting. In retrospect, many of these issues could have been mitigated by seeking advice from the aide or the professor.

Ultimately, the project was an invaluable look into the featured protocols. Despite some frustration with the tools involved, we feel as though we have a deep understanding of UDP, how it differs from TCP, and the methods by which one can implement RDT when sending UDP packets. We also learned about some of the issues that can occur when working on the socket level that will be useful to know when working on other Python networking applications.