

**CS 471 Spring 2008**  
**Programming Project 3**  
**Reliability and Flow Control**  
Due: Friday, April 18, 2008

## 1 Overview

The goal of this project is to write a transport layer protocol that offers reliability and flow control using a sliding window protocol. To simplify your implementation, you will build your protocol on top of the UDP protocol (rather than IP) so you don't need to change anything in the kernel. You will write a sender program that will transmit data (reliably) to a receiver program that you will also write. For purposes of this project, we will call your new protocol the Reliable Flow Control Protocol (RFCP).

## 2 The Packet Format

You will send RFCP packets in the payload portion of UDP datagrams. The format of an RFCP packet is shown in Figure 1.

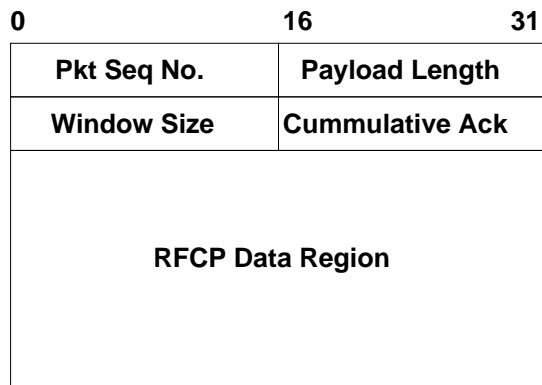


Figure 1: RFCP packet format.

RFCP packets have a fixed length header (8 octets). All fields are used in both direction: from sender to receiver and from receiver to sender.

The sequence number is recorded in the *Pkt Seq No.* field. Data packets are sent with increasing sequence numbers – i.e., the sequence number is incremented by one each time a new data packet is sent. The Pkt Seq No. is not incremented if the packet has a zero length payload. In that case, the

packet is considered to be an ACK packet and the Pkt Seq No. is not incremented. The *Payload Length* field indicates the size of the RFCP data region (payload) in bytes.

The *Window Size* and *Cumulative ACK* fields are primarily used to ACK data packets. Each ACK packet sent from the receiver to the sender will identify the highest *cumulative* Pkt Seq No. seen by the receiver (i.e., the *Cumulative ACK*). It will also indicate the number of packet buffers still available at the receiver (i.e., the receiver *Window Size* measured in packets).

For the purposes of this project, your sender program will fill the data region with all zero's (i.e., we are not interested in the content of the packets). You may randomly pick the length of the data region for packets going from the sender to the receiver (but it must be greater than zero). For the ACK packet sent from the receiver to the sender, the Payload Length field should be zero.

The initial Pkt Seq No. (in both directions) should be zero. The initial advertised window size (in both directions) should be 10.

### 3 The RFCP Sliding Window Protocol

The RFCP sliding window protocol has similarities to TCP's sliding window protocol in the sense that the receiving side advertises its *Window Size* (measured in packets) to the sending side. The sending side will not send more packets than the number specified in the *Window Size*. The *Window Size* will shrink (or grow) based on the number of available slots at the receiver. The receiver will ACK a packet as soon as it receives the packet, so there is no danger of the window size going to, and remaining at, zero forever – retransmissions will eventually open up the window. However, the receiving side will not “move” the window, until the lowest numbered missing frame (i.e., the next frame expected) arrives.

### 4 Reliable Delivery

To implement reliable delivery, your RFCP protocol will retransmit packets that have not been ACKed within a timeout period equal to 5 round trip times (RTT). You must monitor the RTT and keep a running average. In particular, you should use the function

$$\text{EstimatedRTT} = 0.8 \times \text{EstimatedRTT} + 0.2 \times \text{SampleRTT}$$

to compute the estimated RTT.

Your sender will continue to retransmit data packets until the packet finally gets through. ACK packets (from the receiver to the sender) are not acknowledged by the sender, nor are they retransmitted by the receiver.

You will need to use Unix's timing mechanisms to measure RTT's and to wake-up when a timeout occurs. Two routines that you should find useful are:

- **gettimeofday():** returns the current time in a structure that contains seconds and microseconds
- **select():** waits for input on a socket for a specified amount of time.

The *gettimeofday()* call will be useful for finding out the current time. You can call this to find out how much time has elapsed between two points in the program. The *select()* call will be useful

while waiting for an ACK packet from the receiver. The `timeout` parameter will allow you to wait for a specified amount of time. If no packets arrived before that amount of time elapses, select will unblock and you can process the events (timeouts) that need to be handled.

#### 4.1 Causing Packet Loss

To ensure your protocol experiences packet loss, your sender and your receiver will intentionally drop packets. The percentage of packets that should be dropped (intentionally not sent) will be specified on the command line when you run your program. For each packet, your program will decide (with the specified probability) whether to send the packet or not.

## 5 The Sender

You will launch the sender program using the command

```
senddata DstAddr DstPort [LossRate]
```

where *DstAddr* is the IP address (in dotted decimal) of the machine where the receiver process runs, *DstPort* is the UDP port number that the receiver is listening to, *LossRate* is the percentage of packets that should be discarded – specified as an integer between 0 and 100.

The *senddata* program will reliably transmit data (RFCP payloads filled with zeros) to the receiver identified by *DstAddr* and *DstPort*. You will use UDP sockets to send and receive data. The kernel will assign you a local UDP port to send/receive packets on.

Because your sender will initially not have any information from the receiver – i.e., there is no connection setup like TCP – your sender can simply assume an RTT of 10ms and an advertised window size of 10. Your sender should send 1000 packets and terminate.

To show what is going on inside your sender, your sender program should print out the following information:

- Every 100<sup>th</sup> packet print “Sent packet *N* at time *T*”, where *N* is the packet sequence number and *T* is the current time returned by `gettimeofday` (in seconds and microseconds). For example “Sent packet 0 at time 12345 sec and 4321 usec”.
- Each time you (intentionally) discard a packet, print “Discarding packet *N* at time *T*” where *N* and *T* are defined as above.
- Each time a packet is retransmitted, print “Retransmitting packet *N* at time *T*. The current RTT estimate is *rtt* usec” where *N* and *T* are defined as above, and *rtt* is the current RTT estimate in microseconds.

## 6 The Receiver

You will launch the receiver program using the command

```
rcvdata MyPort [LossRate]
```

where *MyPort* is the UDP port number that the receiver will listen to, and *LossRate* is the percentage of packets that should be discarded – specified as an integer between 0 and 100.

The *rcvdata* program will receive incoming packets from the sender, and send cumulative ACKS to the sender. Since the receiver will not use the data, it does not actually need to buffer it. However, it does need to keep track of the amount of buffer space available at any time. If a packet is lost, subsequent packets with higher sequence numbers will continue to consume buffer space until the missing packet arrives. Your receiver must record which buffers are in use and send the appropriate window advertisements to the sender (even though the receiver does not actually store the packet contents in the buffers). You should set the buffer space (i.e., Window Size) to 10 packets.

Like the sender, your receiver will use UDP sockets to send and receive data. You will use the *recvfrom()* call to discover the sender’s IP address and UDP port number, and then you will use the *sendto()* call to send packets to the sender. (Before sending the ACK, you will decide to discard the ACK with probability *LossRate*). You can assume there will never be more than one sender that contacts your receiver (i.e., you don’t need to worry about multiple senders sending data at the same time).

To show what is going on inside your receiver, your receiver program should print out the following information:

- Every 100<sup>th</sup> packet that it receives print “Received packet *N* at time *T*”, where *N* is the packet sequence number and *T* is the current time returned by *gettimeofday* (in seconds and microseconds). Start by printing packet 0.
- Each time you (intentionally) discard a packet, print “Discarding ACK for packet *N* at time *T*” where *N* and *T* are defined as above.
- Each time a packet is received out of order, print “Missed packet *N* at time *T*” where *N* is the sequence number of the packet that is missing (i.e., the one(s) just before the packet you received), and *T* are defined as above. Note that you might have to print several consecutive missing packets. For example, if you received packets 1, 2, 5, you would print that you missed packets 3 and 4.
- Each time the receive window size opens up to 10 again, print “Opening the window size to 10 from *M*”, where *M* was the previous window size.

## 7 Grading

The project will be worth 100 points. Points will be assigned as follows:

- Program compiles and produces some meaningful output (10 points)
- Program sends RFCP packets to the receiver (10 points)
- Program drops the specified percentage of packets (10 points)
- Receiver sends RFCP ACK packets to the sender (10 points)
- Sender only sends packets within the specified window (20 points)

- Receiver adjusts the receive window size correctly (20 points)
- Sender correctly times-out and retransmits packets (10 points)
- Program prints correct output messages and is well documented (10 points).

## 8 What to Submit

**Your code must compile and run on the multilab machines or you will not receive credit.**

You will submit all your code plus a documentation file. You should **not** submit .o files or other binary files. To create your submission, tar and compress all files that you are submitting (e.g., tar czf proj3.tgz projectdirectory). You should submit the following:

- README file - listing all the files you think you are submitting
- Documentation file - brief description of your project including the algorithms you used. Your documentation should also include a description of any special features or limitations of your program. **This must be a simple text file (.txt).** *Do not submit MS Word, postscript, or PDF files.*
- Makefile - we will type 'make' and it must automatically compile your executable. Name your executables *senddata* and *rcvdata*.
- all your C/C++ files
- all your header files

Once you create the tar file, go to

<https://www.cs.uky.edu/CSAPP/main.php>

to submit your project. You may upload your tar file as many times as you like. Each submission will **overwrite** the previous submission, so we will only have a copy of your last submission.

You must work on this project by yourself. That is, you must write all the code yourself. If you use any code that you did not write yourself (e.g., you found it on the web, read it in a book, etc.) you must cite where you got the code. Even if you found code and then modified it, you must say where it came from.