Adam Mooers

Luke Kledzik

4/11/2017

System and Networks 2

**Algorithm Descriptions**

**Message Format:**

The messages are stored in packed struct rdpPacket described in rdtIPConverter.h where it is shared by the sender, receiver, and network programs. Each field in the rdpPacket struct is a char array that mirrors the 54-byte layout described in the project description. Chars were selected since they have a well-defined length. srcIP and destIP are dotted-quad formatted IPV4 address strings of the source and destination respectively. srcPort and destPort are char arrays holding the string-formatted ports of the source and destination. Two byte-length fields are reserved for the ack and corrupt flags. The ack flag can be 0 or 1 depending on the sender/receiver’s state. The network does not change it. The corrupt flag is always cleared by the source node and possibly set to 1 by the network. 8 bytes are remain for the message fragment. In all packets but the last, the msgFrag fills completely and is not null terminated. To end transmission, the sender sends a null-filled msgFrag.

The rdpPacket struct is included for reference below:

struct rdpPacket

{

char srcIP[16]; // The dotted-quad IPv4 address of the sender UDP socket

char srcPort[6]; // The port number of the sender UDP socket

char destIP[16]; // The dotted-quad IPv4 of the receiver UDP socket

char destPort[6]; // The port number of the receiver UDP socket

char ack; // The expected boolean ack value (from sender) or ack value (from reciever)

char corrupt; // Indicates whether the packet is corrupted the network (set by network)

char msgFrag[8]; // The message fragment to send to the user

}\_\_attribute\_\_((packed));

Packet buffers are reused by the nodes, so they are zeroed each time to prevent information from leaking between packets.

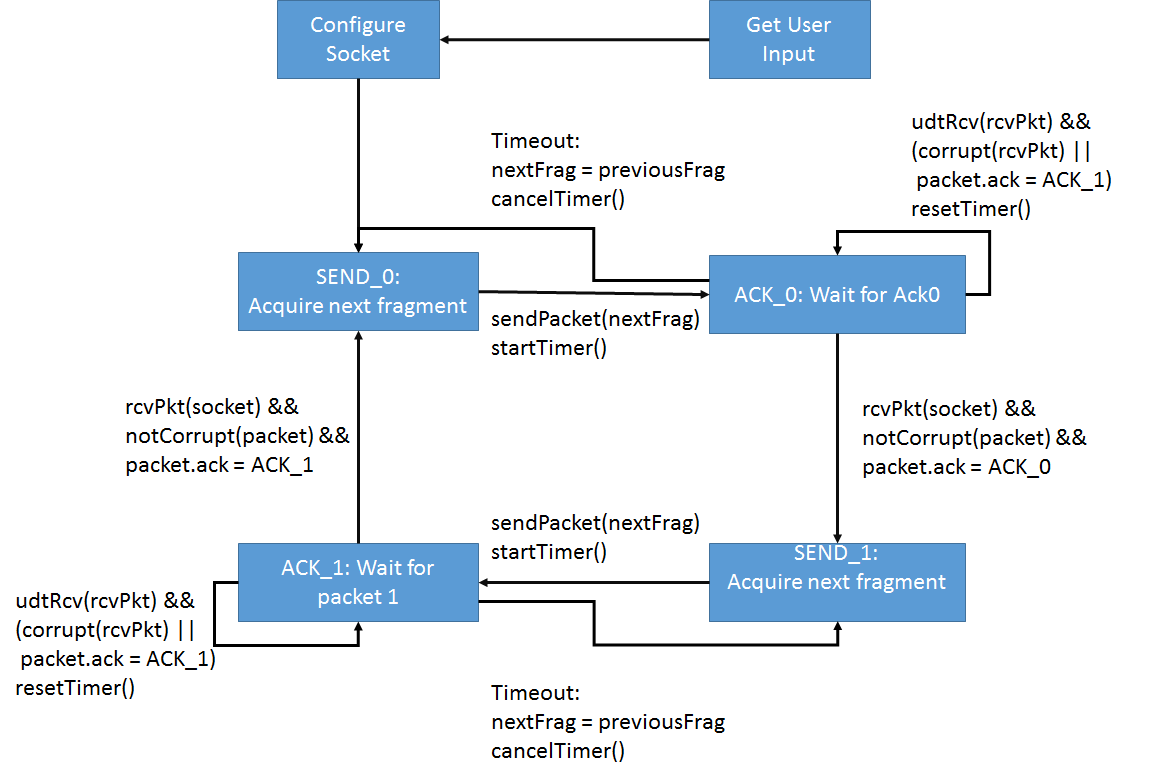
**Sender:**

The sender takes input from the user and attempts to send it to the receiver using a combination of acks and timeouts. Since the network may drop, corrupt, or delay message packets and acknowledgments, the sender must be fault-tolerant. In RDP 3.0, the sender takes full liability for lost packets.

When the sender starts, it first accepts a single line of input from the terminal. This message (including the final newline) is then fed into the sender state machine algorithm. If the user’s message exceeds a maximum buffer size set by the program, no more input is accepted, and the full buffer is sent.

When the socket is configured, it resolves the host names of the sender and receiver and binds the associated UDP socket. The sender simply assumes the network and receiver exist. In this way, the RDP 3.0 sender is connectionless. The destination and network can also be changed at any time if the state machine is reset.

The Mealy finite machine of the sender (shown in fig. 1) sends packets and then waits for uncorrupted acks in reply. The states ACK\_0 and SEND\_0 are identical to states ACK\_1 and SEND\_1 respectively, except for the ack sequence number which alternates between the state pairs. If the wrong ack is received, a corrupted ack is received, or no ack is received before a timeout, the previous fragment is resent. If there are any errors with the underlying socket, the any state of the finite state machine will jump to a final terminating state that closes the socket. The fig. 1 finite state machine assumes that the input is infinitely long. In the case of the actual sender program, the final null-terminating character is sent in an empty packet. At this point, the sender waits for a final ack to terminate. If no matching ack is received after n (set to 3 by default) successive timeouts and retransmissions, the sender terminates (even if the receiver is still open). Corrupt packets and mismatched acks reset the sender’s timer, so there is no guarantee that the sender will exit after its final packet is sent.



**Figure 1:** Simplified finite state machine of the sender

**Network:**

The network is simply an infinite loop which accepts packets and forwards them to their destination based on their RDP header. To simulate imperfect network conditions, a percentage of incoming packets are corrupt. Of those that are not dropped, a percentage of outgoing packets are corrupted and delayed. All three random variables are independent, but the number of corrupted and delayed (outgoing) packets depends on the number of dropped incoming packets.

By default, the delay time of the network is set to 3 seconds, while the timeout of the sender is 2 seconds. This means that the sender will timeout and resend the packet before the receiver receives the first packet. If the resent packet is not delayed, it will arrive at the receiver before the first transmission is received. Normally, the receiver will simply drop the delayed transmission when it arrives. However, if the retransmission is acked and a next fragment transmitted before the first resent packet is received, the receiver will accept the resent packet as the next fragment since its ack is correct. It is possible for even more complex errors to occur, especially if the delay time is more than double the retransmission time. All errors occur because there are simply not enough states in the receiver to guarantee the correct order errors beyond single neighbor swaps.

**Receiver:**

The receiver is simpler than the sender since it is not responsible for retransmission. If it receives a corrupt packet or a packet with the wrong sequence number, it simply drops the packet and waits for the sender to timeout and retransmit. An ack is sent to the sender if and only if the sequence number is correct and the packet is not corrupt. The receiver will not ack and accept the same fragment twice under normal conditions. If an empty fragment is received (where the first char is null-terminating), the receiver acks it and ends the transmission. The receiver concatenates accepted fragments into a buffer. If the buffer is full, the receiver exits the state machine. The only fragment with a null-terminating character in it should be the last. If a packet has a null-terminating char in any position but the first, the receiver will concatenate only the part of the fragment up to the null-terminating character.

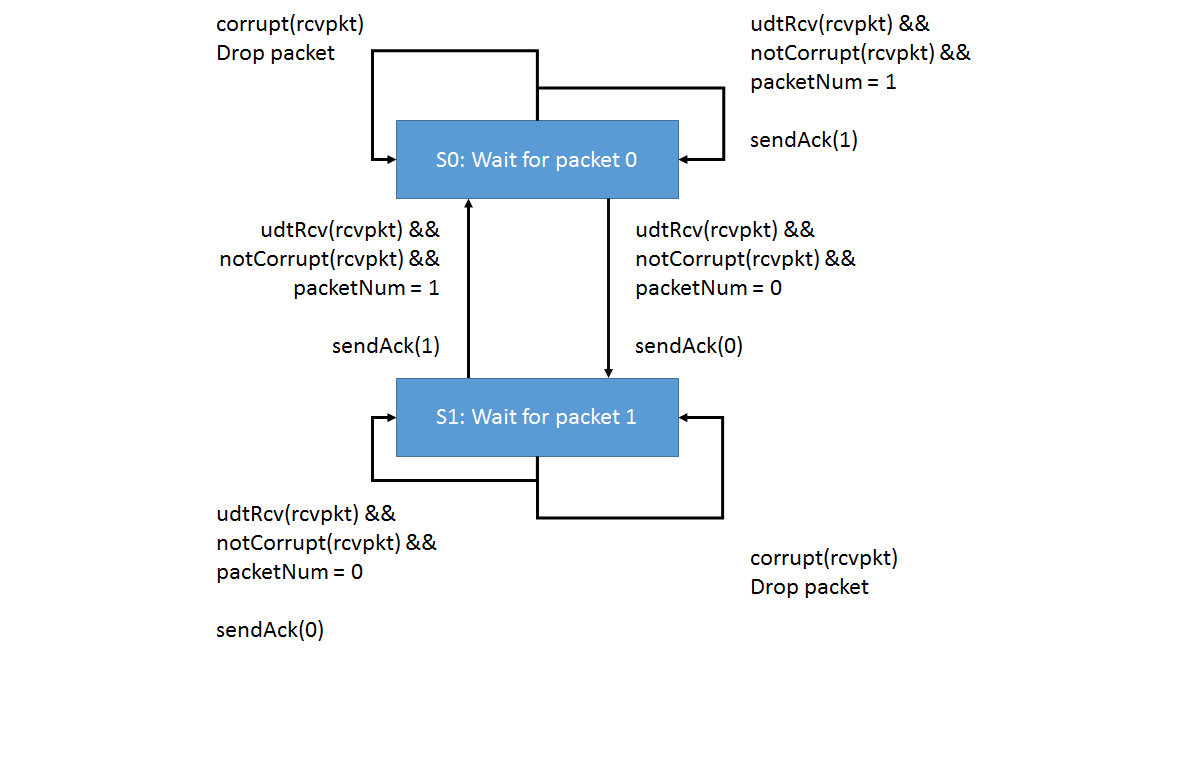
There is a chance that the receiver will be left hanging if packet loss is high. If the sender transmits the final empty fragment three times, and it is dropped by the network each time, the receiver will continue waiting after the sender exits. The chance of the above happening is:

Where Cdropped = the chance that the packet is dropped

Ccorrupt = the chance that the packet is corrupt

For three final send attempts and a 50% drop rate and 50% corrupt rate, the chance that the receiver is left hanging is 42.5%. This can be decreased by increasing the number of final retransmission attempts.

The Mealy finite state machine of the receiver is shown in figure 2.



**Figure 2:** Simplified Mealy finite state machine of the receiver