Georgia Institute of Technology
CS 3251 – A

Reliable Transfer Protocol (RxP)

Homework 4:
Protocol Specification

Tyler M. Smith
Kyle Rabago-Banjo
November 4, 2015

Introduction

Reliable Transfer Protocol (RxP) is designed to allow users to reliably transfer files across a network. As such, it behaves as an Application Layer extension to TCP, even though it is built upon UDP. Consequently, it must handle data loss and corruption in the following ways:

1. Lost packets are retransmitted after a timeout period (calculated in a similar manner to TCP)
2. Corrupted packets are detected by a checksum value transmitted within each packet. Upon receipt of a corrupted packet, the receiver notifies the sender with a NACK packet (see Packet Types).
3. All non-corrupt packets are ACKnowledged by the receiver. Only one ACK is accepted per sent packet. All subsequent ACKnowledgments for the same packet are ignored.
    a. Duplicate data packets are simply ignored when receiving data.
4. All packets are sent with an order number. The receiver buffers packets according to their order numbers as they are received.
5. Once connected, both the server and client can send DATA, FIN, RST, ACK, and NACK packets. No distinction is made between client and server after connection.
6. Each packet has a checksum attached which provides protection against corruption. See Appendix A: Algorithms for the Checksum Algorithm.
7. RxP uses the Sliding Window, Selective Repeat ARQ to transfer data. The window size can be specified by the user (see API), and specific packets are retransmitted upon receipt of a NACK.

Segment Header

The Reliable Transfer Protocol uses packets to transfer data. Each packet can be up to 512 bytes long. The header for these packets is described below:

Source IP Address	4 bytes
Source Port Number	2 bytes
Destination IP Address	4 bytes
Destination Port Number	2 bytes
Sequence Number	4 bytes
Number of Segments	4 bytes
Checksum	2 bytes
Window Size	2 bytes
Payload Size	9 bits
Flags	6 bits
Unused	1 bit
Payload	486 bytes

Source IP Address gives the IP address that this packet originated at.

Source Port Number gives the port from which this packet was sent.

Destination IP Address gives the IP address that this packet is being sent to.

Destination Port Number gives the port to which this packet is being sent.

Sequence Number is a number assigned to each packet, describing the order of the packets.

Number of Segments gives the total number of segments sent, so that the receiver knows if any are missing.

Checksum uses the algorithm described in Appendix A to verify the data in the segment is not corrupt.

Window Size indicates the remaining size of the receive buffer.

Payload Size gives the actual size of the payload, in case it is less than 498 bytes.

Flags has six (6) distinct fields:

1. ACK: this field is set (=1) when the packet is acknowledging the receipt of a data packet.
2. NACK: this field is set (=1) when the packet is reporting the corruption or loss of a data packet.
3. SYN: this field is set (=1) when the packet is attempting to connect a server and client.
4. FIN: this field is set (=1) when the packet is attempting to close an existing connection.
5. DATA: this field is set (=1) when the packet contains data which is being transferred.
6. RST: this field is set (=1) when the packet is indicating that the connection has been reset.

There is one (1) unused bit.

Payload can be up to 486 bytes in length and contains the data which is being sent. In the case of SYN packets, Payload will contain information used to establish a connection (see Appendix A).

Establishing a Connection

The Reliable Transfer Protocol uses a four-way handshake in order to establish a connection between a server and client. This handshake is as follows (See Appendix B for the Connection State Diagram):

1. The client sends a "connection request" packet which has the SYN flag set.
2. The server responds to the request with a packet having SYN, ACK, and DATA flags set containing a 256-byte random key.
3. The client then sends in a packet, with SYN and ACK flags set, a 16-byte MD5 hash consisting of:

   random_key, client_ip, client_port

   in that order.
4. The server compares this hash to its own locally generated one, and responds with an ACK if the connection is accepted, or NACK if the connection is rejected.

Terminating a Connection

The Reliable Transfer Protocol allows either endpoint to terminate a connection. This could be due to the server shutting down, the client closing, or at the end of a data transmission session. The termination of a connection occurs after a three-way handshake (See Appendix B for the Connection State Diagram):

1. One peer sends a packet with the FIN flag set, indicating to the other peer that it wishes to close the connection.
2. The other peer then sends a packet with the FIN and ACK flags set.
3. The first peer responds with an ACK packet and frees resources used to maintain the connection.
4. The second peer receives the ACK and can then free its resources.

Application Programming Interface (API)

The Reliable Transfer Protocol has several methods to allow the user to transfer data in a reliable, connection- and data-stream-based manner:

rxp.connect

Parameters: *address* – An IP address, port number pair which describes the server to connect to.

Returns: *connection* – A Connection object which allows the user to send and receive data or to manage the connection. If no connection can be made, then *connect* returns *NULL*.

Attempts to establish a connection to the server located at *address*.

rxp.listen

Parameters: *port* – The port on which this server is to listen for connection requests.

Allows the server to detect and temporarily queue incoming connections.

rxp.accept

Returns: *connection* – A new connection object which can be used to send information and manage the connection by the user. Returns *NULL* if no incoming connections are available.

Takes the next available incoming connection from the queue and passes it to the user.

rxp.ConnectionClosedError

Raised by a *Connection* when it has been closed by the other peer. Users should catch this *Exception* to know when the *Connection* gets closed.

***Connection* Object Methods:**

close

Returns: *code* – A Boolean code: True if the connection was closed correctly, False if the connection was closed abruptly.

Attempts to gracefully close this connection.

send

Parameters: *data* – The data payload to send to the host on the other side of this connection.

Returns: *success* – True if the data was sent successfully, False otherwise.

Sends *data*, guaranteeing that the information has been sent without corruption.

recv

Parameters: *size* – The amount of data to receive over this connection. Defaults to 486 bytes.

Returns: *data* – The data received over the connection.

Receives *size* bytes from the connection and stores returns the information as a byte array.

setWindow

Parameters: *size* – The size (in packets) that this connection can accept without emptying the input buffer.

Sets the window size of this connection. i.e. the number of packets that this connection can accept before any data is received. Defaults to 4 packets.

getWindow

Returns: *size* – The size (in packets) that this connection can accept without emptying the input buffer.

Gets the window size of this connection. i.e. the number of packets that this connection can accept before any data is received. Defaults to 4 packets.

Appendix A: Algorithms

Checksum Algorithm

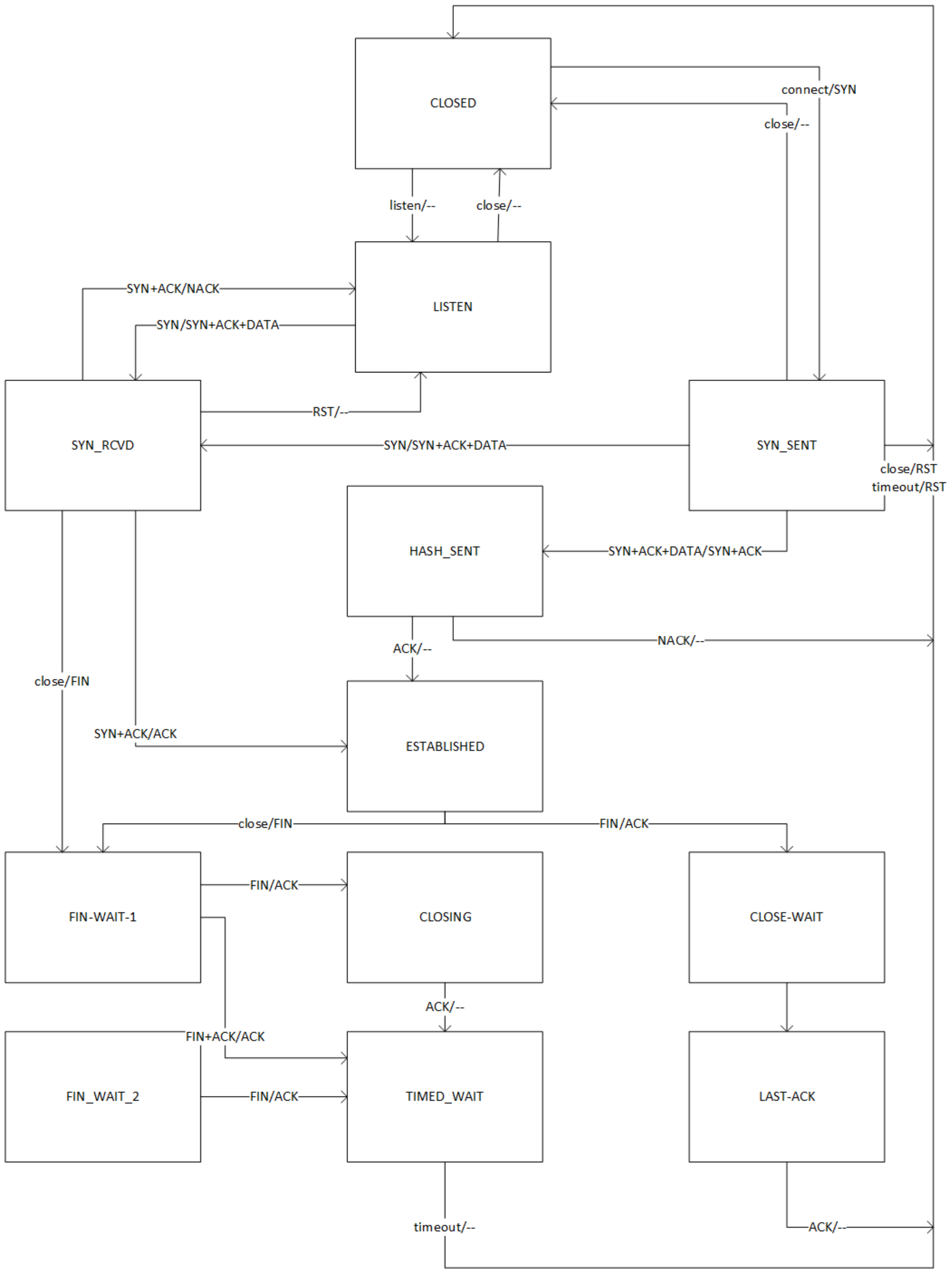
The Reliable Transfer Protocol uses a checksum to ensure that packets and data are delivered without being corrupted. The checksum is calculated in the following manner:

1. The packet is separated into 16-bit words, with the checksum word being set to zero (ignored).
2. These words are summed together, and overflow is "wrapped-around" and added as its own 16-bit word to the sum.
3. This total is inverted (one's complement) and stored into the checksum field.

In order to verify a packet against its checksum, the following procedure is used:

1. The packet is separated into 16-bit words, with the checksum field included.
2. These words are summed together, and overflow is "wrapped-around" and added as its own 16-bit word to the sum.
3. This total is inverted. If the value is zero, then no corruption is detected. Otherwise, the packet has been corrupted.

Appendix B: Connection State Diagram



CLOSED
connect/SYN
close/--
listen/--
close/--
LISTEN
SYN+ACK/NACK
SYN/SYN+ACK+DATA
SYN_RCVD
RST/--
SYN/SYN+ACK+DATA
SYN_SENT
close/RST
timeout/RST
HASH_SENT
SYN+ACK+DATA/SYN+ACK
ACK/--
NACK/--
close/FIN
SYN+ACK/ACK
ESTABLISHED
close/FIN
FIN/ACK
FIN-WAIT-1
FIN/ACK
CLOSING
CLOSE-WAIT
ACK/--
FIN+ACK/ACK
FIN_WAIT_2
FIN/ACK
TIMED_WAIT
LAST-ACK
timeout/--
ACK/--