Reliable Bytestream Transmission Protocol

CS 3251 Homework 4

Protocol Specification

November 2015

prepared for

Georgia Institute of Technology
CS 3251 - Networking 1
801 Atlantic Drive NW
Atlanta, Georgia 30332

by

Roi Atalla, Evan Bailey
Georgia Institute of Technology
College of Computing, School of Computer Science
801 Atlantic Drive NW
Atlanta, Georgia 30332

1. INTRODUCTION

1.1. Summary

RBTP is a reliable, connection-oriented, full-duplex, byte-stream protocol for use over an unreliable packet-switched network. It is meant as a bridge between the user and a datagram network. The protocol is exposed to the user as an API: a set of functions with clear contracts regarding their behaviors.

RBTP fragments the outbound stream into packets that will be sent efficiently over the underlying network. Each packet of data is prepended with a 16-byte header that contains the source and destination ports, a sequence number, a header length, a receive window length, a checksum, and identifying flags. The header can also be expanded to include a list of acknowledged packets. The protocol implements reliability through selective repeat, whereby the sender pipelines the packets up until the receive window length and the receiver uses selective acknowledgement by sending the aforementioned list of sequence numbers that have been successfully received and validated. Thus lost packets can be inferred by the sender as the packets that have not been acknowledged within a certain period of time, called a timeout. The usage of sequence numbers furthermore aid in detecting duplicate and out-of-order packets. When those are detected, the affected packets are simply retransmitted. Since the underlying network provides no guarantees about the validity of the data in a packet, corruption must be detected using a checksum, which is computed using CRC16 over the entire packet, including the header and payload and compared with the stored checksum in the header of each packet.

2. RBTP HEADER

2.1. Header Format

RBTP packets are sent as internet datagrams, and is designed to run on top of Internet Protocol (IP), whose header handles the source address and destination address. The RBTP header follows the Internet Protocol header, and supplies information that is relevant to the transport layer and RBTP protocol.

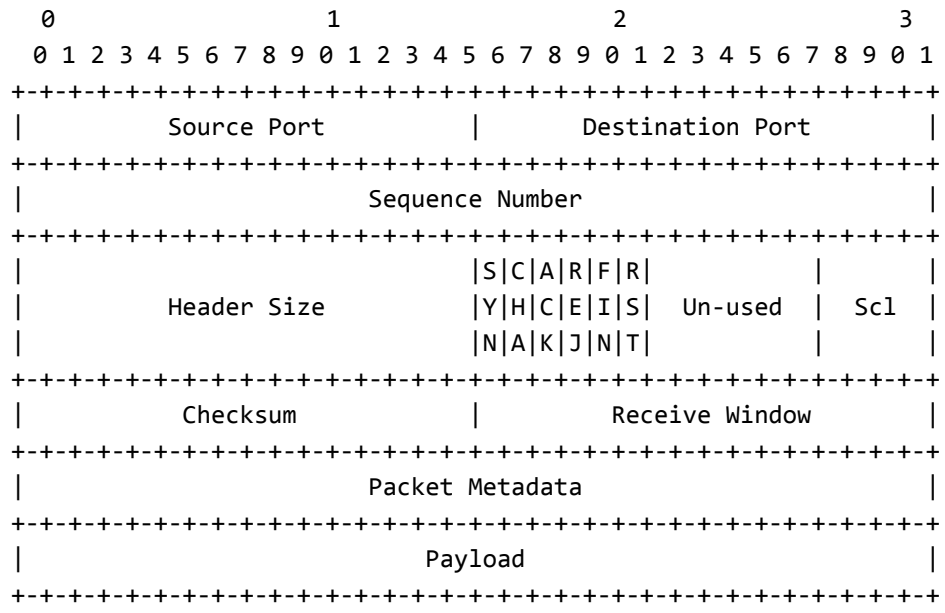


Figure 2.1 - RBTP Header format
Note that one '-' represents one bit

2.2. Header Fields

All fields are big-endian.

Source Port: 16 bits

This field contains the source port number.

Destination Port: 16 bits

This field contains the destination port number.

Sequence Number: 32 bits

This field contains the packet sequence number, which specifies the location of the first byte of this packet's payload relative to the bytestream. If the SYN control bit is set, this field represents the initial sequence number (ISN), and the first byte of the stream will be located at ISN + 1.

Header Size: 16 bits

This field contains the length of the RBTP header, measured in 32-bit (4-byte) words.

Control Bits: 6 bits (from left to right)

SYN: Synchronize sequence numbers
CHA: Metadata is relevant to authentication challenge
ACK: Metadata is relevant to acknowledgement numbers
REJ: Failed authentication
FIN: Sender has finished transmitting data
RST: Reset connection (optional - implementation specific)

Un-used: 6 bits

This field is not currently used. It must be set to all zeros. This space is reserved for future flags.

Scl: 4 bits

This field contains the scaling factor for the receive window field. This value denotes by how much the receive window field should be bit-shifted to the left. This means the maximum value the receive window can obtain is approximately 2 gigabytes.

Checksum: 16 bits

This field contains the result of the checksum applied over the entire header (with all zeroes in the checksum field) and payload. This checksum is computed using the CRC16 algorithm.

Receive Window: 16 bits

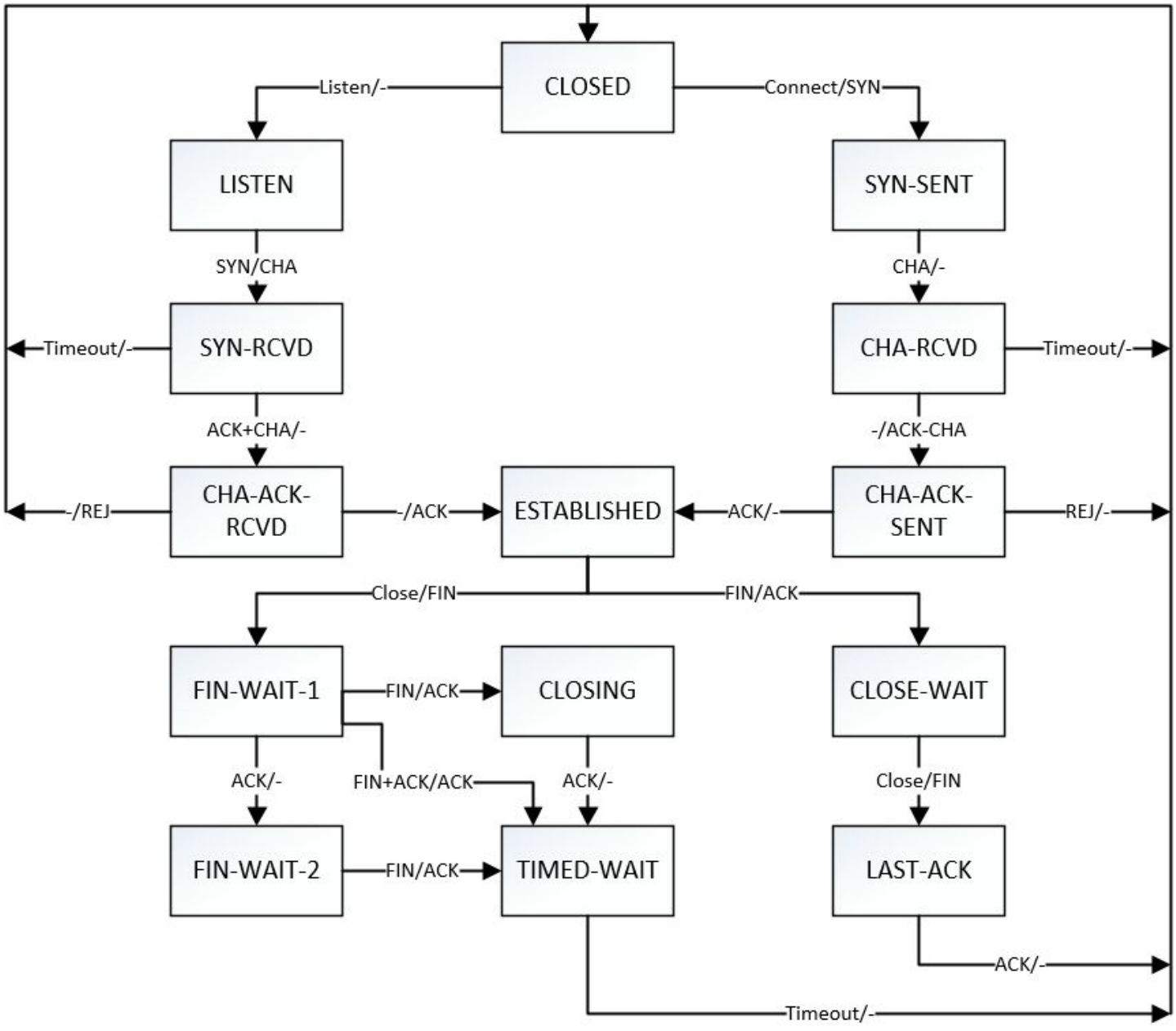
This field contains the maximum number of bytes the sender of the packet is willing to accept in 1 window.

Packet Metadata: varies

This field contains header metadata relevant to the control bit(s) set. If the CHA control bit is set, this field contains data relevant to the authentication challenge. If control bit is set, this field will contain acknowledgement numbers (except in the special case where both the CHA and ACK control bits are set).

3. RBTP STATE DIAGRAM

3.1. State Diagram



4. CONNECTION ESTABLISHMENT / TEARDOWN

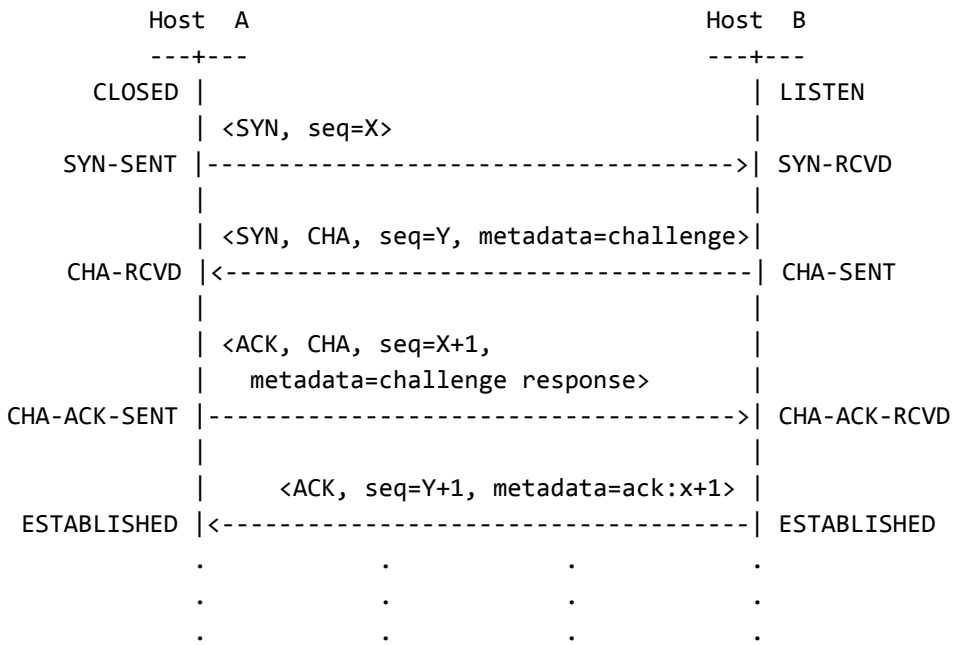
4.1. Establishing a Connection

A 4-way handshake with challenge-based authentication is used to establish a connection. This procedure is initiated by one host, labelled “client,” and responded to by another, labelled “server.” However, once a connection has been established, the client-server distinction goes away as both hosts are able to send and receive data simultaneously.

If at any point in establishing a connection a timeout occurs, corrupt packet is received, or the challenge is failed, the connection is aborted.

4.2. 4-Way Handshake

A successful 4-way handshake example:



4.3. Authentication Challenge

The authentication challenge is based on the Hashcash proof-of-work algorithm. The server will send a 56-bit random value followed by an 8-bit value indicating the number of zeroes required. The client will create the header and set these two values in the metadata as received. Then the client will calculate the 160-bit SHA-1 hash of the header (with checksum field set to zero). If the hash begins with the required number of zeroes, the header is acceptable and can be sent. Otherwise, the client will increment the random value and repeat this process. The server shall not require an amount of zeroes such that on average it takes more than 1 second or less than 50 milliseconds to solve. An acceptable range of average times is 100-200 milliseconds.

4.4. Closing a Connection

A connection can be closed in multiple ways. Refer to the state diagram for examples of this.

4.5. Connection Timeout

At various points while a connection is being established, losing connectivity will cause a timeout. During SYN-SENT and CHA-SENT, a timeout must be set such that if the receiver does not reply, the sender does not wait for a response forever. If this timeout triggers, the host moves to the CLOSED state.

5. BUFFERS / SLIDING WINDOW

5.1. Sliding Window

RBTP incorporates a sliding-window protocol to address flow control. Since the connection is full-duplex, both the sender and receiver can have their own window sizes.

In addition to this, selective-reject is implemented. As such, RBTP uses cumulative acknowledgements. Two cases will trigger the receiver to ACK all the packets it has received, either the sliding window is at capacity or a timeout is reached. In either case, the receiver will build a special ACK packet, denoted by the ACK control bit being set.

This special ACK packet will contain a list of 32-bit (4-byte) acknowledgement numbers in the packet metadata field. These acknowledgement numbers denote the sequence numbers of the data packets the receiver has successfully received. The sender can then re-send any data which may have been lost, dropped, corrupted, or otherwise not received.

6. FUNCTION INTERFACES

6.1 Server Interface

class rbtp_server_socket

This class provides an interface around an rbtp_socket behaving as a server.

void set_blocking(boolean)

This function sets whether the accept call is blocking or non-blocking.

boolean is_blocking()

This function returns a boolean value denoting whether or not the server is set to be blocking or non-blocking.

void bind(RBTPSocketAddress)

This function binds the server to listen on a specific address and port. The SocketAddress parameter contains both the address and port on which to bind.

void listen()

This function starts listening on the bound port and begins accepting connections.

rbtp_socket accept()

This function returns an instance of rbtp_socket when a connection is made. In blocking mode, this function blocks until it has a connection to accept, otherwise it returns immediately with a null pointer.

void close()

This function closes the server so as to not accept any more connections.

6.2 Client Interface

class rbtp_socket

This class provides a functional interface for an RBTP socket.

void set_blocking(boolean)

This function sets whether the socket is blocking or non-blocking.

boolean is_blocking()

This function returns a boolean value denoting whether or not the socket is set to be blocking or non-blocking.

boolean connect(RBTPSocketAddress)

This function attempts to connect this socket to the specified socket address. It returns a boolean value denoting whether or not the connection was successful.

boolean is_connected()

This function returns a boolean value denoting whether or not the socket is currently connected to a host.

long recv(ByteBuffer)

This function receives bytes into the ByteBuffer. If this connection is marked as blocking, the function does not return until at least 1 byte has been sent or the connection is closed. Returns the number of bytes read.

long send(ByteBuffer)

This function sends bytes from the ByteBuffer. If this connection is marked as blocking, the function does not return until at least 1 byte has been written to the internal buffer or the connection is closed. Returns the number of bytes written.

void close()

This function gracefully closes the connection to the remote.