# CRP

## A Reliable Network Protocol

### Abstract

This document will discuss CRP, a network protocol like TCP. It will show a high-level description detailing how CRP works, a detailed description of the header structure and fields, a finite state machine diagram for the client and server endpoints, an API of available functions, and algorithmic descriptions of any non-trivial algorithms.

Ashvin Rai (arai35), Vagdevi Kondeti (vkondeti3)

## Connection

CRP is a connection oriented, reliable, bi-directional, window based, pipelined network protocol with byte-stream communication. With this being a preliminary report, the implementation may be different than what is described here. At an abstract level, the protocol must first establish a connection between the server and a client. This is done using a three-way handshake. First the client must send a SYN packet to the already listening server. Then the server must send a SYN/ACK packet back to the client, acknowledging the receiving of the SYN packet and sending its own SYN packet. Then the client sends an ACK packet back to the server acknowledging the receiving of the server SYN packet. Once this three-step process is complete, information can be sent over the user defined ports.

## Packets

Once a connection is established, data is broken down into packets. Each packet size is 537 bytes (25 bytes for header + 512 bytes for data). CRP uses a sliding window based packet sending algorithm. Having a set window that allows limited packets through will allow for flow control within the connection. CRP uses sequence numbers defined by a 32-bit number in a packets header to place packets in order. After a user defined timeout, CRP will again request any missing packets from the sequence. Any packets with a duplicate sequence number will be deleted essentially not allowing any duplicated packets through. CRP is a pipelined information transfer protocol. It uses a selective repeat ARQ to resend any lost or damaged packets. The receiver will request any damage or missing packets and the sender will resend the individual packets in a new window along with any new packets that need to be sent. The client and server will both have access to send and receive methods allowing them both to transmit information to each other resulting in bidirectional communication.

## Validation

CRP will use a basic checksum to validate each packet and to weed out any corrupted packets. The header of each packet will contain a checksum field that will be used to validate packets. The checksum is calculated by taking the one’s compliment sum of all the 16 bit words in the packet and calculating the one’s compliment of it.

## Closing Connection

When either the server or the client decides to close the connection the last packet they send will have the END flag in the header set to 1. Once that packet is received and validated, the other endpoint must also send a packet with the END flag set to 1. This will solve any premature closing on either end of the connection. Any missing or damaged packets at the end of the information stream must be sent before an END flag can be turned on.

## Extra

CRP also contains an options field where a variety of options can be set in the header. These options include resetting the connection, adjusting the window size mid transfer depending on the connection speed, pushing packets into the sequence, and an urgent pointer for placing data in the sequence out of order.

## CRP Header

The CRP header will contain 4 bytes for the source port, 4 bytes for the destination port, 4 bytes for the sequence number, 4 bytes for the acknowledgement number, 4 bytes for the checksum, 4 bytes for window size, and 3 bits for flags including SYN, ACK, and END flags. A diagram of the header is shown below.

| Bits 0-16 | Bits 17-31 | Bits 31-47 | Bits 48-64 |
|---|---|---|---|
| Source | | Destination | |
| Sequence Number | | Acknowledgment Number | |
| Checksum | | | |

FLAG: SYN, ACK, END

**Source:** source port in the form of a 32-bit number

**Destination:** destination port in the form of a 32-bit number

**Sequence Number:** 32-bit integer containing the sequence number that will allow packets to be placed in order

**Acknowledgment number:** 32-bit integer contains the next expected sequence number from the receiving endpoint

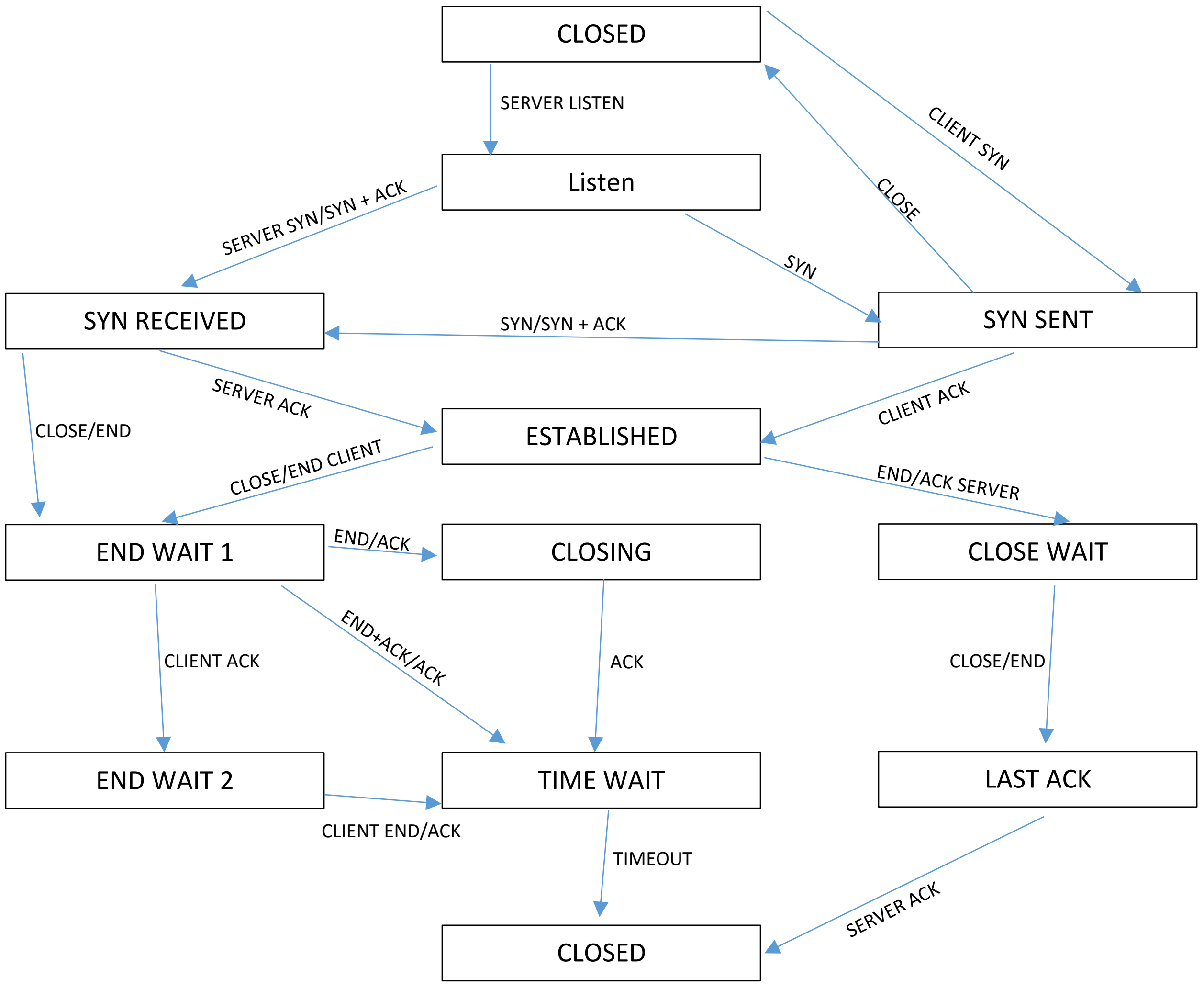
**Checksum:** 32-bit one's compliment number for validating the packet

**[SYN]:** SYN flag set when establishing a connection

**[ACK]:** ACK flag set when establishing a connection

**[END]:** Final packet of new information

## Finite State Machine



## Non-Trivial Algorithms

Sliding Window: Packets are sent in groups of a user defined size. The “window” slides to the next set of packets and missing or damage packets, reported by selective repeat ARQ, are added to the beginning of the window.

Selective Repeat: Uses a user defined timeout to re request packets that were damaged or missing. Once a missing or damaged packet is detected, a new packet is sent with the acknowledgement number reading the new expected packet.

Checksum: CRP takes the one’s compliment sum of all the 16 bit words in the packet and takes the one’s compliment of the result. This number is then sent in the header of each packet to validate the packet.

## CRP API

Described below is the CRP API for anyone wishing to use CRP in their application.

Starting the client takes two arguments: IP and port.

This is the API for the client:

Connect:

- This allows the client to connect to the server

Get(filename):

- This sends the file from the server to the client

Post(filename):

- This uploads a file to the server

Window(int):

- This changes the sliding window size the given input

Disconnect:

- Disconnects the client from the server

Starting the server takes one argument: port.

This is the API for the server:

Window(int):

- This changes the sliding window size the given input

Terminate:

- Disconnects the server

## UTIL API

Described below is the UTIL API for anyone wishing to learn about the util class.

**make_packet(source_port, dest_port, seq, acknum, syn, ack, end, window, data)** *returns packet*

- Creates packet with inputted details

**unpack_packet(packet)***returns header, data, checksum*

- Unpacks the packet to its contents of header, data, and checksum

**create_checksum (packet)** *returns checksum*

- Returns the checksum of the packet

**check_checksum(packet)** *returns Boolean*

- Returns if the calculated checksum equals the checksum attached to the packet

**request_file(filename)** *returns string array*

- Opens the file and converts to packets (each of packet size 512)