RxP Documentation

Samir Jain

Anthony Tsou

CS 3251: Computer Networking I

Fall 2015

Foreword FAQ

Is your protocol non-pipelined (e.g. Stop-and-Wait) or pipelined (e.g. Selective Repeat)?

Our protocol is Stop-and-Wait.

How does your protocol handle lost packets?

Once a packet is sent, a timer is started for that packet. The timer will wait for the incremented ACK for the respective packet and will expire if it is not received within the allotted time. This timeout signifies that the either the sent packet or the received packet was lost. If this is the case, the sent packet will be resent. Packets can be resent 5 times before connection is assumed to be terminated.

How does your protocol handle corrupted packets?

The protocol will perform a checksum and drop the packet if corrupted.

How does your protocol handle duplicate packets?

Our protocol handles duplicate packets with sequence numbers. Sequence numbers state byte order on each packet. The server will expect the next packet to have a sequence number equivalent to the last one incremented by the packet size. If this is not the case, the receiving end will send an ACK with the expected sequence number of the packet to receive, requesting the sender to resend the expected packet.

How does your protocol handle out-of-order packets?

Our protocol also handles out-of-order packets with sequence numbers, much like with duplicate packets. If the receiving end receives the wrong sequence number, the receiving end will send an ACK with the expected sequence number of the packet to receive, requesting the sender to resend the expected packet.

How does your protocol provide bi-directional data transfers?

The client can accept commands for ‘get’ and ‘put’ to initiate download transfers or upload transfers

Does your protocol use any non-trivial checksum algorithm (i.e., anything more sophisticated than the IP checksum)?

We use MD5 as part of the authentication in the four-way handshake for establishing a connection. The checksum uses CRC32, splits the long value in halves and adds them together to fit into a four byte checksum in the RXP header.

Design Specification

High level description

In our construction of RxP, our team seeks not to recreate the wheel but to implement a system similar to TCP. Initially, datagrams will be sent to listening hosts for connection establishment in a secure four-way handshake. With an established pipelined connection, each host will maintain variables to understand byte stream status. Hosts will maintain byte stream semantics for accurate data delivery in uniform 512 byte segments and utilize a Stop-and-Wait system. We maintain a secure connection by checking each packet's source port number, destination port number, as well as its checksum. We also maintain a reliable connection by implementing a maximum of five timeouts for each packet, in the case that it is lost in transmission. In closing, the protocol provides control to the user for closing the connection.

This protocol's four-way handshake ensures safety of server memory and information. Every step of the handshake carries information about source and destination numbers port that feed into a port and challenge string hashmap to maintain authentication control for unique ports. After the host and server have set up their port and the server is listening, the following conversation will occur as shown in the finite state diagram:

1. Client sends request to connect with SYN
2. Server replies with SYN ACK and a 64-bit challenge string (after storing the received Port in a hashmap with this challenge string).
3. Client replies with ACK and the MD5-encoded challenge string
4. Server replies with ACK after confirming the MD5-encoded challenge string, or drops the request if the check fails

The client can request a file for download using the 'get' command followed by a filename. The client will send a packet with the GET flag and the filename to the server to initiate the download process.

The client can send a file to the server using the 'put' command followed by a filename. The client will send a packet with the POST flag and the filename it will upload. This will prompt the server to respond with a POST+ACK, which then prompts the client to send the file.

The transfer protocol sends files by breaking up the specified file into a byte array and then into different data packets. Traditional sequence numbers and acknowledgement numbers maintained between the client and server are used to keep track of which data packets have been received and which data packets are expected next. For example, the client may be downloading a file from the server and the client

will receive a file of sequence number 5. The client will ACK back with acknowledgement number 6, per traditional TCP protocol.

On the receiving end of a file transfer, the host is compiling all of the collected data into a byte array, which will output into a new file with the specified filename appended with “downloaded”.

The maintenance of acknowledgement numbers and sequence numbers will help to deter issues with packet loss, duplication, and rearrangements. Corrupted packets will be caught by the initial checksum at every packet receival.

The protocol allows the either side to send request to close. Per the finite state diagram, the hosts will send FIN packets after data transfer has finished for graceful close.

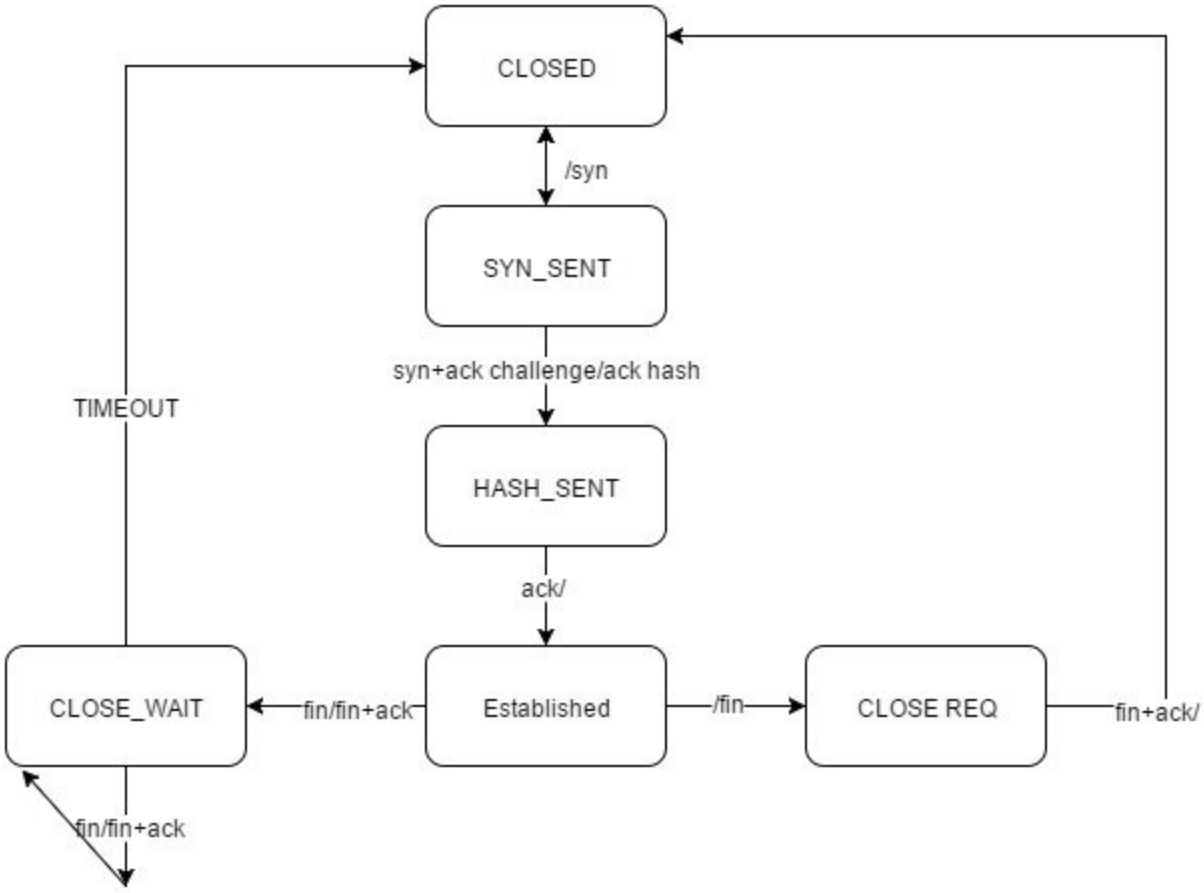
Header Structure

Our additional header structure seeks to add above the UDP header enough information to create a reliable data protocol. This involves 2 byte port numbers for source and destination, as well as sequence numbers and acknowledgement numbers to maintain byte stream status. A segment length variable is utilized to understand the length of the data in the packet. There are single bits used to indicate signals for ACK, SYN, and FIN. Packets are checked with the 4 byte checksum. Since UDP packets can safely contain 512 bytes, the data portion will be the remaining bytes besides this 16 byte header: 496 bytes. The data passed from the application layer will be split into these data segments in the communication stream.

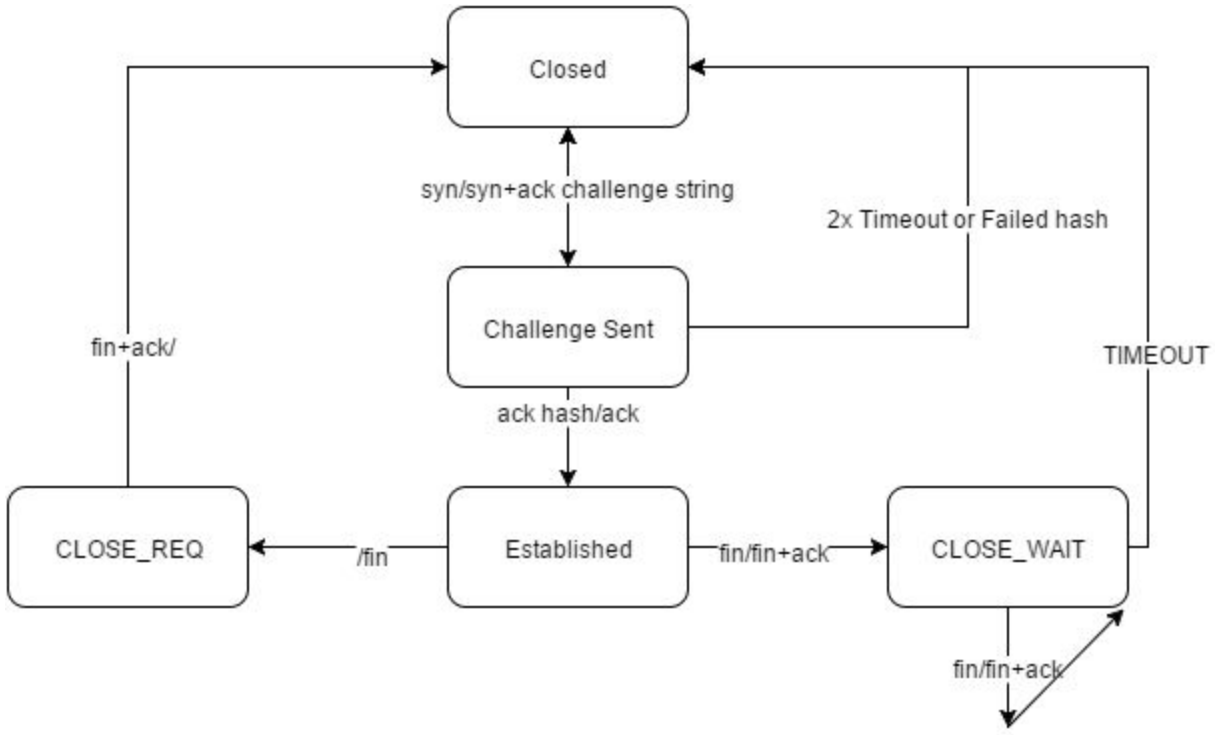
4 bytes per row

Source Port	Destination Port
Sequence Number	Acknowledgement number
SegLen	Flags: ACK, SYN, FIN, GET, POST, LAST | 9 Unused bits
Checksum	
Data (496 bytes)	

Finite State Diagram
Client


CLOSED
/syn
SYN_SENT
syn+ack challenge/ack hash
TIMEOUT
HASH_SENT
ack/
CLOSE_WAIT
fin/fin+ack
Established
/fin
CLOSE REQ
fin+ack/
fin/fin+ack

Server



API to Application Layer

Client Methods

1. RXPClient(int clientPort, String serverIPAddress, int serverPort)
    a. initializes a client and sets up the datagram socket that RXP will operate with
2. boolean setupRXP()
    a. initiate RXP connection to server
    b. Returns success/failure
3. boolean download(String filename)
    a. Request to download specified file
    b. Will generate file in the same directory with “downloaded” appended to the filename
    c. Returns success/failure
4. ClientState getClientState()
    a. Return finite state of client in custom enum “ClientState” (See flow diagram for potential states)
5. void sendFileNameUpload(String filename)
    a. prompts client to send name of the file to the server that it will start uploading to
6. boolean upload(String filename)
    a. Request to upload specified file and then send the file by packets

    b. Returns success/failure
7. void tearDown()
    a. Send signal to shutdown

Server Methods

1. RXPServer(int serverPort, String clientIPAddress, int clientPort)
    a. initializes a client and sets up the datagram socket that RXP will operate with
2. void connect()
    a. Begin connection to continuously listen for prompts from the client
3. void terminate()
    a. Send signal to shutdown

Algorithm

Basic Authentication

In connection establishment, the four-way handshake is authenticated through a challenge string. For every new IP/PORT that the server receives, a 64-bit challenge string is randomly generated and assigned to it in a table. The server will ask the client to send back an MD5 hash of this challenge string. This verifies that the source PORT truly is going to the same end user and the end user is allocating resources and maintaining connection states to reply, preventing DDOS. The server will clear the table entry after three timeouts of the challenge request, or when the challenge is met.

Checksum

RXP uses CRC32 checksum which outputs a long, so significant half is added to the lower half to create a four byte int checksum that will fit into the RXP header.