The CRP Protocol

Mani Japra & Alex Kim

CS 3251

11/04/2016

Design Specifications

- Is your protocol non-pipelined (such as Stop-and-Wait) or pipelined (such as Selective Repeat)?

CRP will try to be a selective repeat pipelined protocol that will ensure the reliability of every packet making it to the user-specified destination. A connection is first established in this protocol by implementing a three-way handshake identical to that of TCP. To reiterate this, the sender will initiate the transmission by sending a SYN packet. When the receiver successfully receives this packet, it will send back a SYN/ACK packet to acknowledge the successful SYN. The sender will then send an ACK to tell the receiver it received its SYN/ACK, and at this point both sides will be an established state to be ready for normal data transfer operations.

- How does your protocol handle lost packets?

We will borrow from a parent protocol, TCP, and use a positive ACK that the receiver will send back to the sender upon successfully obtaining a packet. When the receiver does not receive back an acknowledgement, it will try to retransmit that packet. To bound the retransmission time, we will implement a timer that will reset upon each retransmission and bound the resets to a number that will assume that the receiver is unreachable. Once the bounded number of resets is reached, the sender will stop retransmitting and alert the user of the issue. To handle duplicates, the protocol will assign a sequence number to each packet sent and the receiver will then check to see if duplicates or out of order packets are received. If a duplicate packet is received, the receiver will discard the packet.

- How does your protocol handle corrupted packets?

To counteract corrupt packets, the protocol will use a MD5 hash of the data being sent and received to ensure a flawless data transfer. This hash is calculated using a provided key, which is a 16 bit random alphanumeric sequence of characters.

- How does your protocol handle duplicate packets?

Our protocol uses ACKs that are sent to the sender to verify whether a packet was received and if a duplicate packet is received, it is ignored.

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

Our protocol uses the SEQ number and ACK number, similar to the TCP protocol, to verify the packet ordering of each packet it receives and will use both numbers to check the ordering.

- How does your protocol provide bi-directional data transfers?

Our protocol provides bi-directional data transfers by utilizing a universal ‘get’ and ‘rec’ method that allows both the client and server to send/receive packets of data.

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

CRP provides bi-directional data transfer by allowing both the server and client to send and receive data between one another. Thus, both the client and server can perform the same operations. Our protocol implements a MD5 hash of the data being sent and received instead of the IP checksum to provide a more reliable check should any errors or corruptions occur during the data transfer. The robustness of the MD5 hash was why we decided to choose it over the IP checksum, as well as the fact that we have extra header space available to to implement a full 32 bit checksum.

CRP Header Description

0 15 16 31

Source port	Destination port
NACK | ACK | SYN | FIN | END	
MD5 Checksum	
Data	

Source port: 16 bit
- Source port number

Destination port: 16 bit
- Destination port number

Control flags: each 1 bit
- NACK: indicates if packet is a NACK
- ACK: indicates if packet is an ACK
- SYN: synchronize sequence
- FIN: indicates if transmission sequence is finished
- END: indicates the end of a sequence of packets

MD5 checksum: 32 bit
- Expected checksum value

Data:
- Data being sent between the receiver and sender

API

Class: CRPException

Exception class that provides detailed information about various errors that may arise during runtime.

Class: Status

enum values: CLOSED, SEND, RECV, WAIT, LISTEN, IDLE

Class: Socket

socket.bind(addr, port)

Binds a socket instance to an address and port number

socket.connect(addr, port)

Connects to a device with the address and port number

socket.listen()

Listens for a connection request on a port specified in the bind command

socket.accept()

Accepts an incoming connection after successfully listening for a connection request

socket.send(msg)

Sends a message to the receiver

socket.send_packet(msg)

Sends a packet to the receiver

socket.recv_packet()

Receives a packet from the sender

socket.rec(msg)

Receives a message from the sender

socket.close()

Closes a connection and unbinds from the port number

socket.send_FINACK()

Sends a FIN+ACK Packet

socket.send_SYNACK()

Sends a SYN+ACK Packet

socket.send_ACK()

Sends a ACK Packet

socket.send_FIN()

Sends a FIN Packet

socket.send_SYN()

Sends a SYN Packet