RTP for IPv4

Authors: **Tim Chu**, **Steven Han**

**Brief Overview**

RTP is the same as TCP except in a few aspects. The window size header is changed to be dynamic to accommodate larger window sizes. The checksum is larger to decrease the chance of an incorrect checksum, and the checksum uses a different algorithm. There is no options field, and also the header length field has been reserved, since the header length can be extrapolated from just the window size header length, since that’s the only variable length field. It is stop-and-wait to make it more convenient to implement. 4 reserved bits for easier implementation as well.

**Header structure**

16 bits -- Source port

16 bits -- Destination port

32 bits -- sequence number

32 bits -- ACK number

4 bits -- reserved (future flags?)

4 bits -- reserved (flags)

ACK/RST/SYN/FIN

8 bits -- window size header length

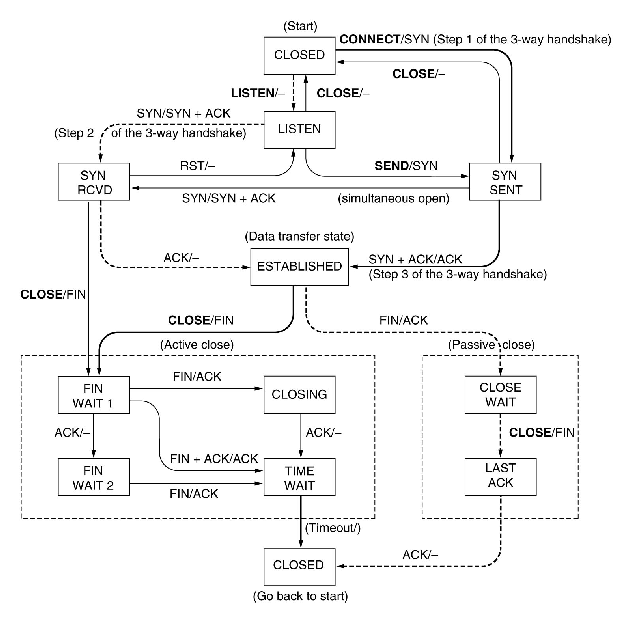
“window header length” bits -- window size header

32 bits -- CRC32 checksum

data...

**Finite state-machine**

It’s exactly the same as TCP.



**Questions**

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

It is stop-and-wait instead.

*Can your protocol handle lost packets?*

Yes, the protocol has a retransmit timer specification the same as TCP.

*Can your protocol handle corrupted packets?*

Yes, better than TCP can, actually. The checksum will be wrong, and the receiver will send a NACK. There are now 32 bits in the checksum header to make it more reliable.

*Can your protocol handle duplicate packets?*

Yes, they get dropped.

*Can your protocol handle out-of-order packets?*

Yes, except it uses stop and wait instead of pipelining.

*Can your protocol provide bi-directional data transfers?*

Yes, same as TCP.

*Does your protocol use any non-trivial checksum algorithm?*

Yes, it uses CRC32 which takes up 32 bits of header instead of 16 bits. The sender calculates it and puts it in the RTP header. Upon receiving, the receiver calculates it to verify that no errors have taken place.

**Interface:**

bind(): Binds to a port

listen(): Opens a buffer for collecting potential connection opening packets

accept(): Takes one of the SYN packets, returns a SYNACK and waits for an ACK, resending the SYNACK as needed.

connect(): Sends a SYN, waits for a SYNACK, then sends an ACK

send(): Nonblocking, Sends a byte array

receive(): Blocking, receives a byte array, sends ACK

close(): closes, sending FIN