FivR

Protocol Design Report

Authors	Shiv Patel, Tim Kerr
Last Edited	20 October 2014
Class	CS 3251 A
Professor	Constantine Dovrolis

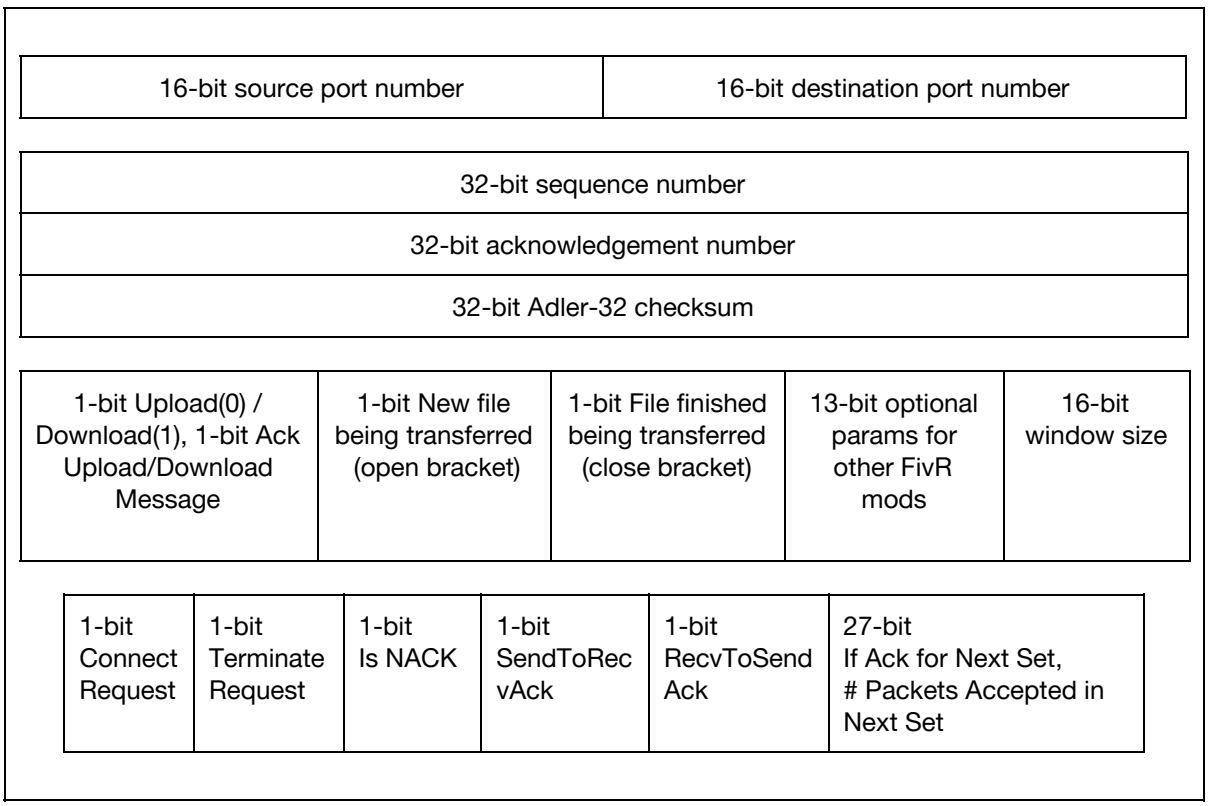
High-Level Description

FivR is a reliable transport protocol that relies heavily on the cummalitive acknowledgement concept that can is found in traditional RTP protocols such as TCP. However, FivR differs by selectively sending acknowledgement messages every so often, therefore reducing the number of communication packets sent between the sender and receiver. The name of the protocol derives from the default number of packets (5) that the receiver must receive before an acknowledgement is sent back to the sender. Note that FivR refers to the number of packets sent each time as a “set”.

Additional features of FivR are described below in the **Detailed Protocol Breakdown** section.

Header Structure and Fields

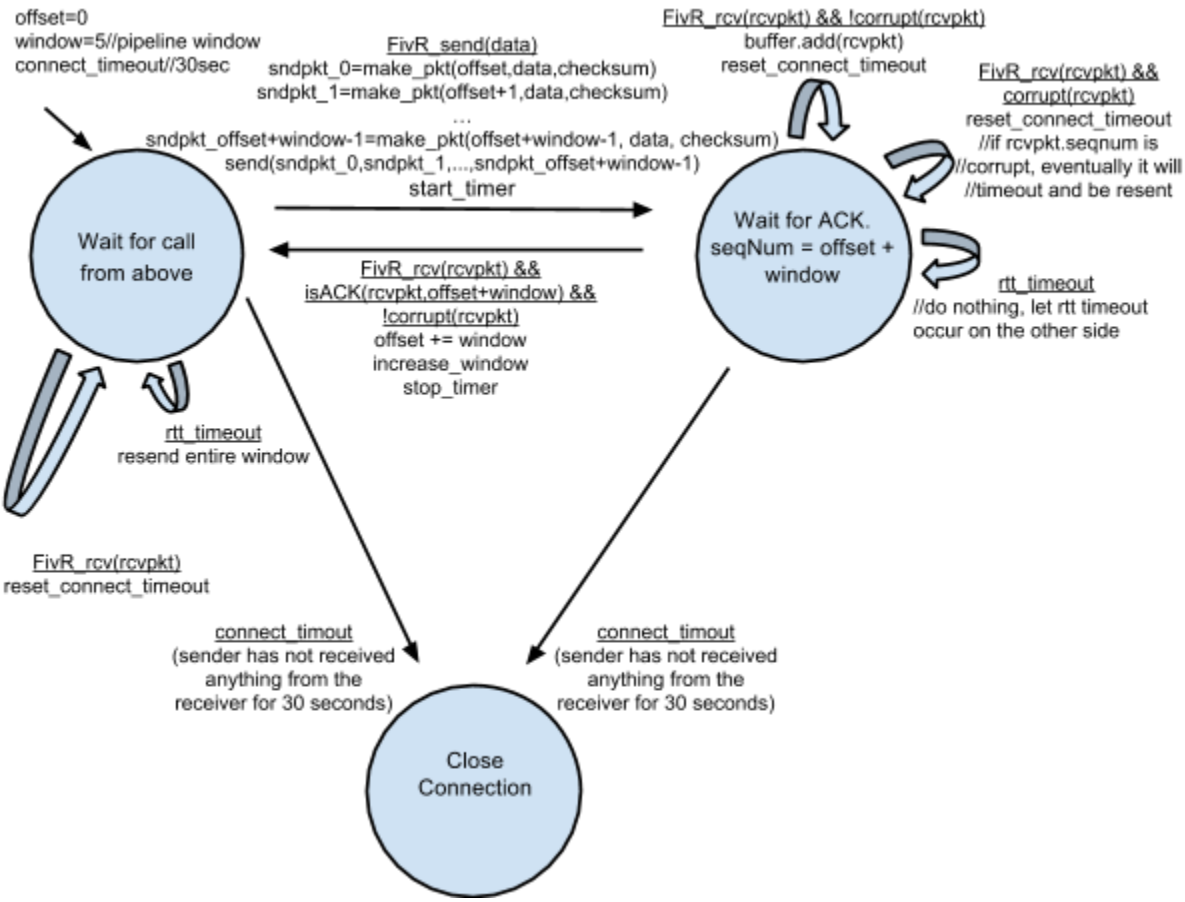
FivR’s header structure use the following format:



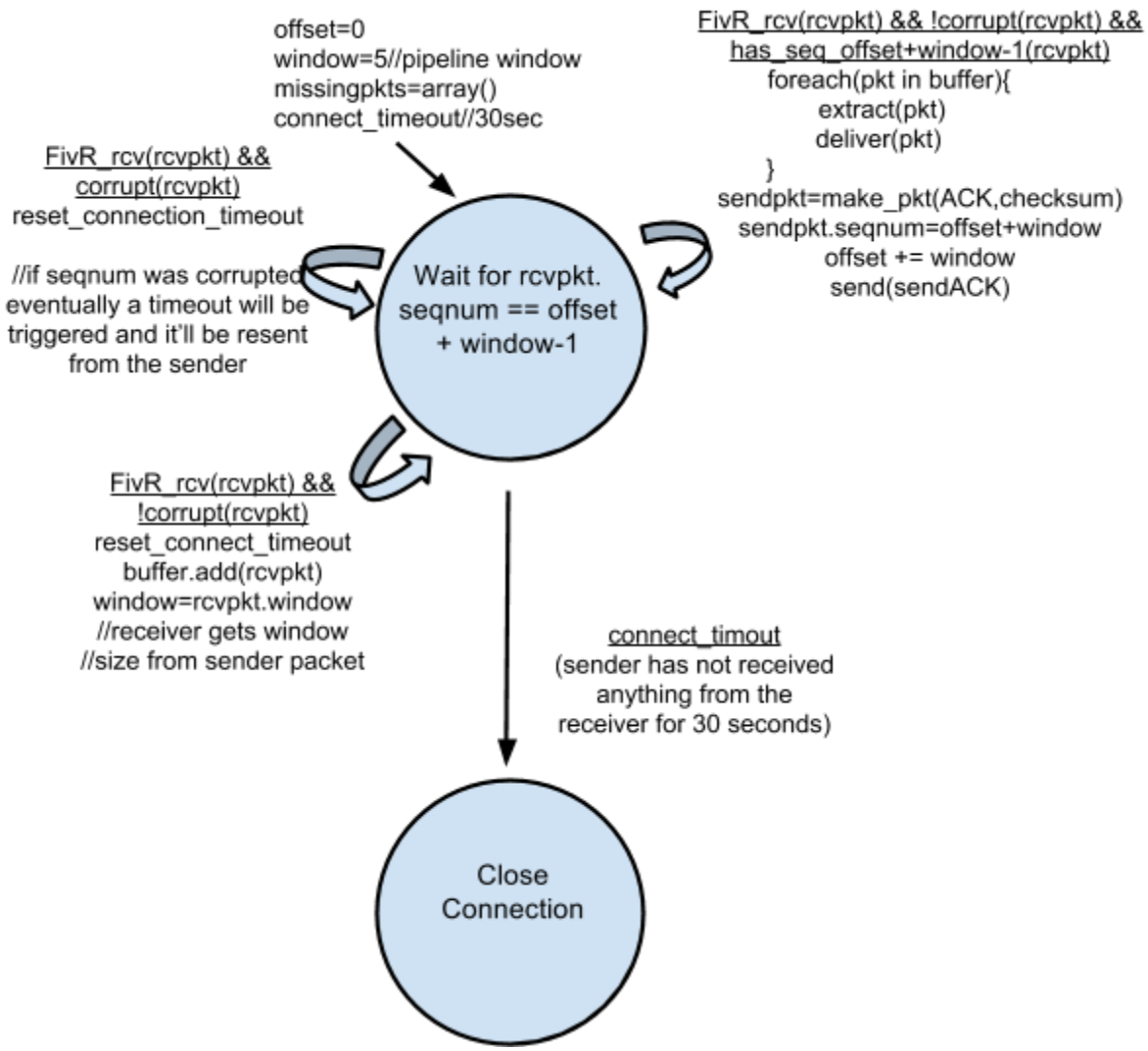
- Source Port Number: The source host port number.
- Destination Port Number: The destination host port number.
- Sequence Number: A unique value representing the packet (used as the packet ID in FivR).
- Acknowledgement Number: Set to the value of the sequence of the next expected packet.
- Checksum: See **Corruption Handling and Checksum Usage**.
- Window Size: The number of segments the sender of this packet is willing receive in return.

Protocol Visualization (FSMs)

FivR Sender



FivR Receiver



Application Layer Interface and Functionality

File Transfer Application (FTA) Server Commands

start <server_port, client_port> //starts the server on specified port

window <# of bytes> //adjusts pipeline window size

diagnostics-mode <true|false> //toggle debug info on/off

stop //stops the server

File Transfer Application (FTA) Client Commands

connect <client_port, server_port, emulator_ip, emulator_port> //connects to server via the emulator

get-file <filename> //gets a file with specified filename

post-file <filename> //posts a file with specified filename

disconnect //disconnects from the server

Detailed Protocol Breakdown

Below is a detailed breakdown of each unique functionality and feature found in the FivR reliable transport protocol.

Pipelined Structure

FivR will support a piplelined structure . The window variable on both the sender and receiver keep track of how many packets are allowed to be in flight at once without acknowledgement of delivery. The window sizes dials up and dials down based on network conditions in order to get the best possible performance.

Bi-Directional Capabilities

FivR will support Bi-Directional capabilities. A sender will also have all the capabilities of the receiver and vice versa. Both the sender and receiver in FivR will implement both state machines provided above in order to allow a bidirectional flow of data.

Variable Segment Size

Segments can be any length between 24 bytes (header size) and 512 bytes depending on the size of the data is contained in its payload.

Cummalitive Acknowledgments

As mentioned previously, FivR attempts to reduce packet congestion by limiting the number of outgoing acknowledgements sent to the sender from the receiver. This is done by sending acknowledgements only after the correct packets in the window of packets are received. Note that this acknowledgement contains the sequence number of the first packet in the next set of expected packets. This means only one single acknowledgement is sent from the receiver to the sender for a given set of packets.

Timeout Handling

There are two separate timeout scenarios in FivR, **RTT_Timeout** and **Connection_Timeout**.

RTT_Timeout occurs when a set of packets are sent from the sender to the receiver and the sender does not receive an ACK for the set of in flight packets before a given amount of time. This causes the sender to resend the entire window of packets.

Connection_Timeout occurs when neither party (sender or receiver) hear anything from the other for more than 30 seconds. This causes the connection to terminate. (This situation might occur if one side is suddenly switched off before a graceful close to the connection could be performed)

Out-of-Order and Duplicated Packet Handling

FivR allows the receiver to receive packets in almost any order. The only restriction is that only packets with unique sequence numbers that fit in the range for the expected set of packets will be saved in the buffer. FivR is then able to sort these unique packets to determine if any are missing. The following two

scenarios would work correctly with FivR:

Scenario 1 Packet 1 > Packet 2 > Packet 3 > Packet 4 > Packet 5
Scenario 2 Packet 1 > Packet 3 > Packet 4 > Packet 2 > Packet 5

In the event that duplicate packet arrive, FivR will cross check the packet with each unique packet in the connection's buffer. If the packet already exists, FivR will simply discard the packet.

Scenario 3 Packet 1 > Packet 2 > Packet 3 > Packet 4 > ~~Packet 4~~

Corruption Handling and Checksum Usage

FivR uses the Adler-32 checksum algorithm to check for any corruption within packets. If a packet is determined to be corrupted, the receiver will do nothing and wait for the sender to timeout and resend the packet.

Establishing and Terminating Connections

FivR follows a very similar structure to standard TCP when establishing and terminating connections. To establish a connection the following steps are taken:

1. Sender sends a packet to receiver with ConnectRequest flag enabled.
2. Receiver sends ACK message to sender with ConnectRequest and RecvToSendAck flags enabled.
3. Sender verifies for receiver that he/she is still there by sending an ACK message back to the receiver with the ConnectRequest and SendToRecvAck flags enabled.
4. Both hosts are now successfully connected.

To terminate a connection the following steps are taken:

1. Sender sends a packet to receiver with the TerminateRequest flag enabled.
2. Receiver sends ACK message to sender with TerminateRequest and RecvToSendAck flags enabled.
3. Sender verifies for receiver that he/she is received the ACK by sending an ACK message back to the receiver with the TerminateRequest and SendToRecvAck flags enabled.
4. Both hosts can now terminate the connection from their ends.