RDTP - Reliable Data Transfer Protocol

Protocol Specification

Introduction

This specification describes a Reliable Data Transfer Protocol(RDTP). RDTP is a lightweight packet transfer protocol with the basic aim to provide in-order reliable packet delivery layered over traditional UDP/IP Protocol.

This protocol is suitable for simple communication protocols like instant messaging or moderately sized file transfer. RDTP aims to provide a flexible and minimal design over existing UDP transport protocol with support for in-order and reliable delivery using a sliding window protocol.

Specification

API Specification

The RDTPConnection library provides 2 classes: RDTPSender and RDTPReceiver which implement the RDTP specification.

RDTPSender

RDTPSender is a class that needs to be instantiated by the sender. This class provides 3 simple methods which abstract the complete protocol:

- **connect(ip_address):** Open a connection to target ip_address
- **send(data):** send data(binary format) to the receiver
- **close():** terminate the connection and close the socket

RDTPReceiver

RDTPReceiver is also a class which provides the following 2 methods:

- **listen(handleData_func):** Listen for any incoming connections and run the given handleData function when data is received. This is a blocking function call. There is also an optional 2nd parameter which takes boolean value to decide whether to exit after data transfer.
- **close():** stop listening, and close the socket.

Protocol Overview

The **Sender** performs the following tasks:

1. The sender calls the send method from the RDTPConnection library.
2. Application data is divided into chunks of fixed size.
3. A header with packet info is attached to each chunk.
4. A connection with the receiver is opened.
5. Packets are sent out using a reliable and in-order delivery protocol.
6. Terminate the connection with the receiver.

The **Receiver** performs the following tasks:

1. Receiver uses the *listen* method from the RDTPConnection library
2. A socket is created and binds to port 8448
3. Receiver listens for any inbound connection requests.
4. When an inbound connection is detected, it is accepted
5. All the received data is collected according to the delivery protocol
6. Collected data is sorted in-order, stripped of its headers and sent to the application

Packet Format

Sequence Number	Acknowledgment Number
Flag Bits	Data Index
Content-Length	
Checksum	
Payload (data)	

Header Format (7 Bytes**)**:

- 1st byte: Sequence number.
- 2nd byte: Acknowledgement number.
- 3rd byte: Protocol flags.
- 4th byte: Data-Index.
- 5th and 6th byte: Content-length.
- 7th byte: Checksum.

Sequence Number: A number used to denote/identify a packet sent. Also serves the purpose of denoting present and past packets

Acknowledgment Number: Denotes the sequence number of the next packet to be received. It can also denote the acknowledgment of the previous sent packet.

Flag Bits: An octet where each bit is a flag

- 1st bit - ACK bit: This bit is set when the received packet is an acknowledgment packet.
- 2nd bit - SYN bit: This is the synchronize bit, set when starting a new connection.
- 3rd bit - FIN bit: This is the finish bit, set when ending a connection.
- 4th bit - NUL bit: This is a null bit, set when pinging to check if the host is online.
- 5th bit - BEG bit: This is the data begin (signal) bit. Packet with this bit set is sent before data is sent.
- 6th bit - END bit: This is the data end (signal) bit. Packets with this bit set are sent after data transfer is complete.

The remaining 2 bits are unused.

Data Index: An index number used to represent the position of packet in the current window

Content-Length: Length of the payload in the current packet(in bytes).

Checksum: A number that uniquely denotes the content of each packet. Any corruption of packet data can be verified using this checksum.

Payload: The data chunk carried in the current packet.

Connection Establishment

RDTP uses a 3-way handshake to establish a connection:

1. First, the sender sends a SYN packet with initial random sequence number
2. The receiver accepts the request and sends a SYN-ACK packet with its sequence number
3. The sender sends back ACK packet to confirm the connection

After performing above 3 steps, both sender and receiver now have the required parameters to communicate with each other. The connection is now established on both sender and receiver

Data Transfer

For data transfer, RDTP uses a selective repeat protocol to provide reliability.
Steps followed for Data Transfer between client and server:

1. Sender sends a BEG packet (which has the length of complete data to be sent in payload) to indicate start of data transfer
2. Receiver akcs the BEG packet, notes the data length and is now ready to receive
3. Sender sends all the packets in the current window and waits for ACK's for each packet form receiver. After timeout, it resends all un-ACKed packets. When all packets in current window are ACKed, the window slides forward to the next unsent data
4. Receiver ACK's all the received packets and saves their data in a cache/data structure. Receiver's window only moves forward if all packets in the current window are received.
5. When the data transfer is complete, sender sends an END packet to denote the end of data transfer
6. Receiver ACKs the END packet, and now arranges the data received in order and sends it to the application

After the data transfer is complete, the connection is still open and further transfers can also be performed until sender or receiver terminates the connection

Connection Termination

RDTP uses a similar 3-way handshake to signal connection termination:

- First, the sender denotes the end of connection with a FIN packet
- The remote receiver sends back a FIN-ACK packet to acknowledge termination
- The sender ACKs the FIN-ACK packet and terminates

After the above 3 steps, both the sender and receiver successfully terminate and the connection is broken down.

Network Tolerance

In the real world, the internet does not always work in the ideal assumed way. There are various problems associated with a network connection like packet delay, packet loss, packet reorder, packet duplication, packet corruption and network jitter. RDTP follows the following standards to resolve each of this issue:

- **Packet Loss:** As RDTP follows a selective repeat protocol, all the packets which are not ACKed by the receiver are assumed to be lost by the sender and the unAcked packets in the same window are sent until all packets in the window are ACKed and then the window shifts to a new data segment

- **Packet Delay:** No specific solution is implemented for packet delay. If the sender times out before a packet reaches, it resends all the unAcked packets in the current window.
- **Packet Reorder:** Packet reorder doesn't really affect the protocol since selective repeat is used. The receiver always takes note of any packet received in the current window and drops all other packets
- **Packet Duplication:** All the packets which are already received by the sender are silently dropped whereas duplicates received by receiver will be ACKed.
- **Packet Corruption:** Each packet whose calculated checksum does not match up with the included checksum are dropped.

RDTP follows this rules in case of handshake or termination:

- **Establishment:** if the handshake packets are lost are corrupted, the receiver terminates the connection. If the receiver is not reachable by the sender, after 3 retries, the sender terminates.
- **Termination:** The sender sends the FIN packet until it receives FIN-ACK packet. Then it sends 3 ACK packets hoping at least one will reach the receiver and terminate. The receiver, after receiving FIN packet, sends out FIN-ACK packet until it receives an ACK packet or timeout causing the receiver to terminate.
- **No Data Transfer:** if no data is transferred for 30sec, the receiver sends a FIN packet and terminates the connection.

Assumptions

Network Assumptions

- No error correction is provided. Therefore, any corrupted packet received will be dropped.
- No flow control is provided as of current specification. So data transfer is not regulated either at sender or receiver
- Transferred packets are of fixed size.
- Packets which have not been ACKed before timeout are assumed to be lost
- A full-duplex network link is assumed for bi-directional data transfer.