RDP

Reliable Datagram Protocol

Eric Buss V00818492 B03

NOTE: if you are viewing this file in raw markdown, it is recommended you view it as a pdf in docs/README.pdf

Table of Contents

Introduction

RDP is a UDP based protocol that provides connection management, flow control, and error control. RDP provides a robust means of sending data across lossy networks. This document details for how to get a demo of RDP running (a simple file transfer application), the RDP design, and the API.

Running the Demo

To run RDP, you will need to compile it from source with a C compiler that supports (eg. GCC):

- `stdint.h`
- `sys/socket.h`
- `sys/types.h>`
- `netinet/in.h>`
- `unistd.h`
- `arpa/inet.h`
- `time.h`

To compile the source for the demo run

```
$ make # or
$ make -B
```

The demo is composed of two binaries `rdpr` (the receiver) and `rdps` (the sender). These two files allow you to transfer a file over a lossy network. You will need to run `rdpr` first to wait for the connection. `rdpr` expects the following command line arguments

```
rdpr [options] <IP> <port> <file>
```

Where `IP:port` is the socket to listen on and `file` is the write destination. Optionally you can run `rdpr` with the `-t` command to run it with preset arguments.

Once `rdpr` is running you can run `rdps` in a separate terminal. `rdps` expects the following command line arguments

```
rdps [options] <IP> <port> <receiver IP> <receiver port> <file>
```

Where `IP:port` is the socket to listen on and `receiver IP:receiver port` is the socket to send to. `file` is the file to transfer. Optionally you can run `rdps` with the `-t` command to run it with preset arguments.

Both `rdpr` and `rdps` log sent and received packets. For further details check `docs/requirements.pdf` in the root directory of the repository. You can also run either binary with the `-h` or `--help` command to get a full list of options.

Configuration

Performance of RDP can be modified by changing several constants in `rdp/net_config.h`. These are the relevant default values

```
#define TIMEOUT           8
#define MAXIMUM_TIMEOUTS 256
#define WINDOW_SIZE      16
```

These values were selected to provide a reasonable transfer speed with a wide range of network speeds. For high quality networks its recommended to raise the WINDOW_SIZE to 64 or even 128. Note that for 128 you should also raise the TIMEOUT for most networks. If your network has more than a 10% drop rate it is recommended you raise MAXIMUM_TIMEOUTS.

Design

Header

RDP is managed using a small 15-byte header. The header is used to carry meta data about each packet in order to coordinate data transfer. The header is implemented in rdp/protocol.c. Below shows the encoding of the header with each section and the number of bytes allocated to each.

```
     1(6b)       2(1b)       3(4b)      4(2b)      5(2b)
+----------+-------------+-----------+---------+----------+
|          |   |R|R|D|F|A|S|           | Window/ |          |
| "CSC361" |00|E|S|A|I|C|Y| Seq/Ack # | Payload | Checksum |
|          |   |S|T|T|N|K|N|           |   Size  |          |
+----------+-------------+-----------+---------+----------+
```

The table below describes each field.

	Section	Description	Bytes
1	Prefix	The bytes "CSC361" indicating this is an RDP packet	6
2	Flags	Flags indicating the packet type	1
3	Seq/Ack Number	Seq number for Sender, Ack for Receiver	4
4	Payload/Window size	Payload number for Sender, Window for receiver	2
5	Checksum	Checksum for header and payload	2

The flags indicate the following.

Flag	Purpose
SYN	For connection synchronization
ACK	For acknowledging a SYN, DAT, or FIN packet
FIN	For closing a connection
DAT	For sending a packet with attached payload data
RST	For resetting a connection (currently not used)
RES	For indicating the packet is being resent

The use of each section is detailed in the following sections.

Control Flow

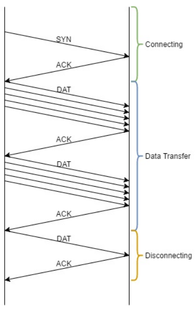
RDP's base packet flow functions very similarly to TCP. RDP's implementation can be found primarily in rdp/receiver.c and rdp/sender.c.

A connection is started when a RDP receiver receives a packet with the SYN flag set (In the future packets with a particular flag set will be referred to as a FLAG packet eg. a SYN packet). The RDP receiver will now begin repeatedly sending ACK packets on a timeout indicating it received the SYN packet.

Once the SYN packet receiver has received an ACK they can send DATpackets up to and including their last received ACKs window size. These packets should be sent in order and be filled to the max packet size. The sender should then wait for an ACK packet and then repeat this process.

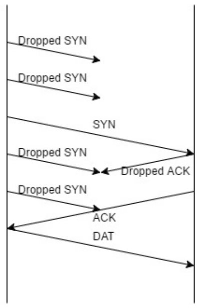
Once the sender is done sending DAT packets they send FIN packets until they receive and ACK packet in response. They can immediately close the connection once they receive the appropriate ACK packet. The receiver on the other hand must continue to wait and respond to FIN packets until they timeout.

Ideally this process is ideally described by the diagram below. Note that the number of DAT packets sent by the sender on the left corresponds to the default WINDOW_SIZE set in rdp/net_config.h.



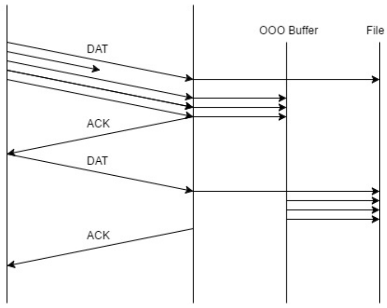
Ideal packet flow

On a poor network packets can be dropped at any part of this process. RDP handles each phase (connecting, data transfer, and disconnecting) differently. When connecting the sender will continue to send SYN packets until they receive a ACK packet with an ack number equal to 1 + the expected sequence number. The receiver will continue to send ACK packets from this point onwards until they receive a FIN packet during the disconnecting phase. This process is illustrated in the diagram below.



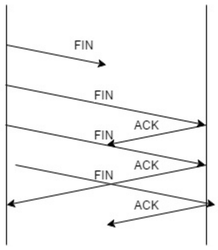
Non-ideal connecting packet flow

During data transfer RDP must deal with dropped and out of order packets. RDP's window size corresponds to a buffer that supports out of order packets (OOO buffer). When the receiver receives a DAT packet with a sequence number it is not expecting (ie. the next byte it hopes to write) it places that packet in the OOO buffer. Once this buffer is full packets will be dropped. Packets with sequence numbers less than the expected sequence number are also dropped. When a DAT packet with the desired sequence number arrives the OOO buffer is emptied of all DAT packets that can be streamed to the file. While there are items in the buffer the receiver sends ACK packets with their desired sequence number and a window size of OOO buffer capacity (ie. Max Window size - # of out of order packets). This process is illustrated in the diagram below.



Non-ideal data transfer packet flow

During disconnection it is up to the receiver to close the connection responsibly. The moment the sender receives an ACK packet responding to its FIN packet it closes its connection. The receiver is expected to stay open until if fully timeouts responding to FIN packets in the meantime. Every time it receives a new FIN packet it resets it timeout counter.



Non-ideal disconnecting packet flow

Error Control

Outside of out of order packets (whose handling is described prior in the section on Control flow), corrupted packets are handled via a checksum. The checksum is a standard inversion of the sum of 16 bit words of the packet. This sum is run on the entire packet minus the checksum. The packet is then reconstructed on the other side with everything except the checksum and then recomputed. The checksum is run on the payload as well as the header is relatively small and a checksum on it alone would increase the chances of a collision.

Statistics

The statistics provided at the end of running the rdpr and rdps demo files are accumulated by the underlying rdp net_config implementation. This results in a different meaning of some fields. The number of unique DAT packets for instance is not a tracking of how many different byte patterns have been received, but instead how many packets without the RES flag have been received. As such the unique DAT packets sent will not equal the unique `DAT packets received on a lossy connection. This implementation is favored for two reasons. First it is more performant. Second its implementation can be hidden from both sender and receiver.

API

The following is a short description of the RDP API. It is not at all exhaustive or detailed. For a better description please read the source.

Creating a Packet

Packets can be created using rdp_pack() function. Note that flags must be but encoded at this point and all integers are expected to be unsigned.

```
rdp_pack(buffer, flags, seq number, size, payload);
```

Buffer will now contain the packet. Buffer is expected to be of at least size rdp_MAX_PACKET_SIZE. If you need the size of a packet you can get it using

```
rdp_packed_size(size);
```

Where size is the payload size. If you have received a packet and wish to parse it you can call

```
rdp_parse(packet);
```

At this point rdp_flags(), rdp_seq_ack_number(), rdp_window_size(), rdp_payload_size(), rdp_payload() will all return relevant information to the last parsed packet. If the packet was corrupted or is not a valid RDP packet rdp_parse() will return 0.

Opening a File

To open a file for reading

```
rdp_filestream_open(filename, 'r');
```

To open a file for writing

```
rdp_filestream_open(filename, 'w');
```

To close any files

```
rdp_filestream_close();
```

Creating a Sender

To create a sender use rdp_sender()

```
rdp_sender(source IP, source port, destination UP, destination port);
```

Each phase can be called with rdp_sender_connect(), rdp_sender_send(), and rdp_sender_disconnect(). For live stats you can call rdp_sender_stats() or rdp_stats(). The first will print the sender stats the second will provide an array of stats.

Creating a Receiver

To create a sender use rdp_receiver()

```
rdp_receiver(source IP, source port);
```

The receiver can than be set to listen using rdp_receiver_receive(). For live stats you can call rdp_receiver_stats() or rdp_stats(). The first will print the sender stats the second will provide an array of stats.

Contact & Credits

My name is Eric Buss. This project was done for the University of Victoria's CSC 361. If you have questions about this or other projects you can reach me at ejrbuss@gmail.com.