Reliable Transport Protocol

RELDAT

Design Report

CS 3251

Jihai An, Yuanjie Sun

jan61@gatech.edu
ysun371@gatech.edu

2017-04-03

Instruction for Compiling and Running

Files:

Reldat_Server.java : The server side.
Reldat_Client.java : The client side.
Packet.java : The packet we used, including modified headers.
Encoder.java : The encoder. Please refer to the Algorithmic Description section.
Decoder.java : The decoder. Please refer to the Algorithmic Description section.
State.java : Macrostates.
README.pdf : This document.
sample.txt : Sample output after transmitting "short.txt"
short.txt: A small test file.
bible.txt: A large test file.

To compile the project, use the command:

```
javac *.java
```

To run the RELDAT server:

```
java Reldat_Server <port> <window_size>
```

To run the RELDAT client:

```
java Reldat_Client <Address/Host> <port> <window_size>
```

Avaliable commands under the RELDAT client:
To disconnect:

```
disconnect
```

To transform a file:

```
transform <filename.txt>
```

Note:

1.Current sequence number will be printed out at the sender side.
2.Since the RELDAT uses Go-Back-N protocol, the performance will be bad if the window size is too big. To test the correctness, please use the file "short.txt", or use small window size.

Design Specification

High Level Description

RELDAT is a reliable transport-layer protocol with following features:

1. It provides byte-stream communication semantics.
2. It is a pipelined protocol with sliding-window.

3. It is connection-oriented, using go-back-N algorithm.
4. It is bi-directional.

In general, RELDAT provides a reliable data transfer between two endpoints. The following paragraphs will specify how RELDAT works in different stages at a high level.

Since RELDAT is connection-oriented, the client must establish a connection with the server before initiating the data transfer. Just like the TCP protocol, RELDAT uses a 3-way handshake process to establish a connection. First, as a connection request, the client sends a SYN packet with the sequence number equals to 0 to the server. The window size will be included in this packet inside the data field. Once the server receives this packet, it will send a packet with SYN bit set, and its sequence number equals 0. The server-desired window size will reside in the data field. This packet serves as an acknowledgment to client's connection request. Once the client receives this acknowledge packet, it will respond the server with a third packet. This packet has its SYN bit cleared, and its sequence number equals to 0. This third packet will piggyback the first chunk of data, identified by its sequence number: zero. The actual data transfer process has started at this point.

RELDAT's communicating stage uses the modified Go-Back-N algorithm to perform a pipelined data transfer. After entering this stage, RELDAT client and RELDAT server will exchange data simultaneously. In other words, the server will piggyback the modified data with the acknowledge packet. The sender will respond to several events:

1. When current window is not full, a packet will be generated and send, and the next sequence number will be increased by one.

2. When receiving an expected acknowledgment(non-corrupted), RELDAT client will slide the window by one(base number increased by one).

3. When a timeout event occurs, RELDAT client will send packets from current base number to next sequence number - 1.

On the server side of RELDAT, it simply waits for incoming packets. When it receives a non-corrupted packet with the expected sequence number, it will send an acknowledgment piggybacked with a modified data back to the sender. Otherwise, if the arriving packet's sequence number is not as expected, the server will send back the acknowledgment of the most recently received in-order packet.

RELDAT client takes the responsibility to end the session. Once the client notices that all data has been transferred, it will initiate a closing request by sending a special packet. This packet will have its FIN bit set, sequence number equals -1. After receiving this packet, the server will generate an acknowledgment and send it back to the client. This acknowledge packet also has its FIN bit set, and sequence number equals -1. When the client receives this acknowledgment, it will start a timer for 30 seconds. At the same time, the client will also respond this acknowledgment, generate a packet with FIN bit set, sequence number equals -2, and send it back to the server. The server will close itself when it receives client's respond. If nothing was received by the client during 30 seconds, the client will shut down. Otherwise, it will send the response packet once again to the server.

Header Structure

RELDAT header is 22 bytes long. We use the ‘Source Port’ and the ‘Destination Port’ fields provided by the UDP packet, and modify UDP packet’s data field to add more header fields, including Sequence Number, Checksum, etc.

UDP packet header:

Source Port	Destination Port
UDP_Data	

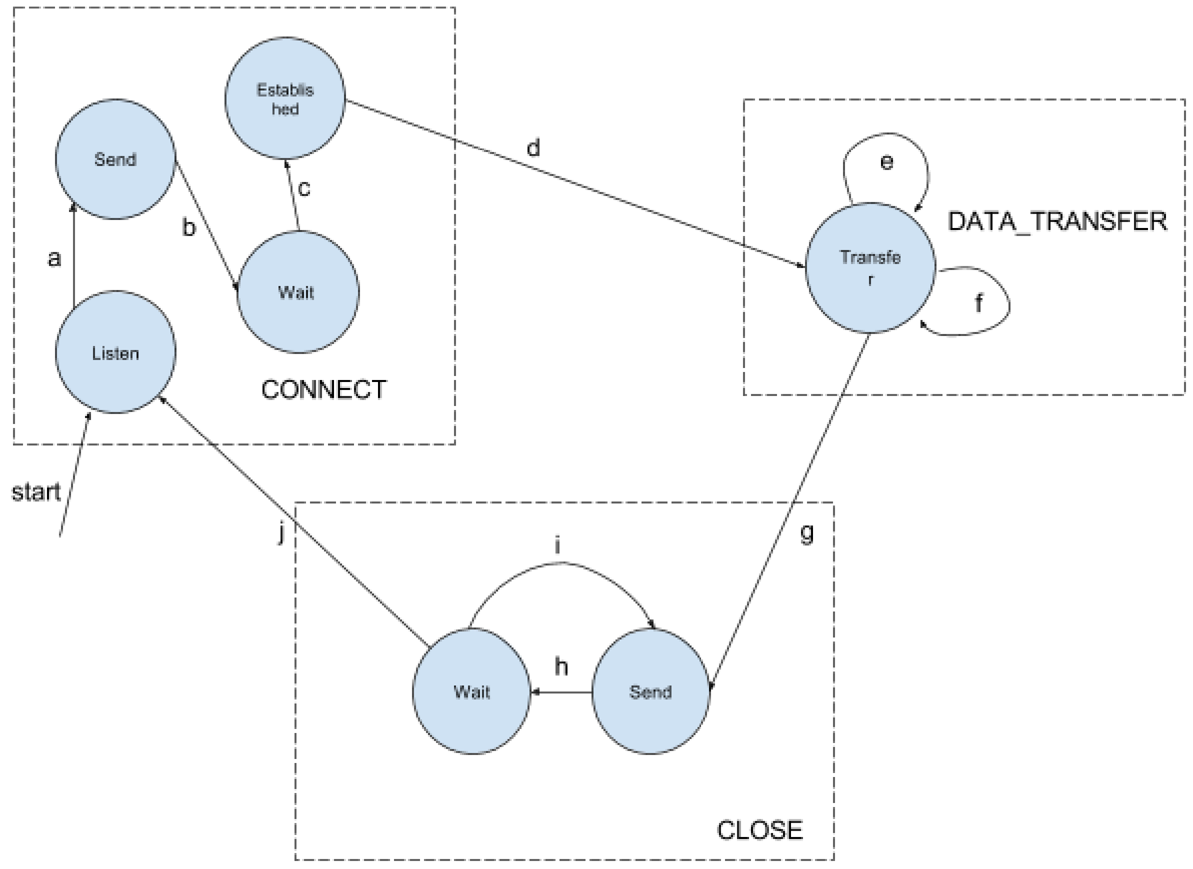
Within the ‘UDP_Data’ field, we have:

Sequence		
Checksum		
Acknowledge		
Data Size		
SYN	FIN	
Data		

Field’s name	Length	Usage
Source Port	2 bytes	The port number of the sender.
Destination Port	2 bytes	The port number of the receiver.
Sequence Number	4 bytes	This number indicates the sequence of this packet. It is used to identify each data packet sent to receiver.
Checksum	4 bytes	This number is used to verify if the packet is corrupted

Acknowledge	4 bytes	This number is the sequence number of the next byte receiver is expecting from the sender. The receiver uses this number to acknowledge the packet whose sequence number is this number.
Data Size	4 bytes	The length of the data byte array.
SYN	1 bytes	If the SYN bit is set, the packet is a SYN packet used in the initial 3-way handshake process.
FIN	1 bytes	If the FIN bit is set, the packet is a FIN packet used to end the connection.
Data	Variable length	This is the application-layer data payload.

RELDAT Server Life Cycle and Finite State Machine



RELDAT server has three macrostates, each consists several micro states.

CONNECT Macrostate:

In this macrostate, RELDAT server will continuously wait for any incoming connection request, acknowledge it, and establish a connection between itself and the source of the request.

i. Listen Microstate:

RELDAT server waits for a connection request.

ii. Send Microstate:

RELDAT server receives one request, sending back an ackowledge packet.

iii. Wait Microstate:

RELDAT server waits for an acknowledge packet from the client.

iv. Established Microstate:

RELDAT server receives the acknowledge packet, establishing the connection.

DATA_TRANSFER Macrostate:

In this macrostate, RELDAT server will perform a bi-directional, pipelined, piggybacked streaming data transfer with the client.

i. Transfer Microstate:

RELDAT server performs a data transfer, and handles several destructive situations.

CLOSE Macrostate:

In this macrostate, RELDAT server will receive an end request from the client then close itself upon this request.

i. Listent Microstate:

RELDAT server receiced a FIN packet, which indicates a close request.

ii. Send Microstate:

RELDAT server sends a FIN packet to acknowledge the request.

iii. Wait Microstate:

RELDAT server waits for an acknowledgment, and closes the connection when receives it.

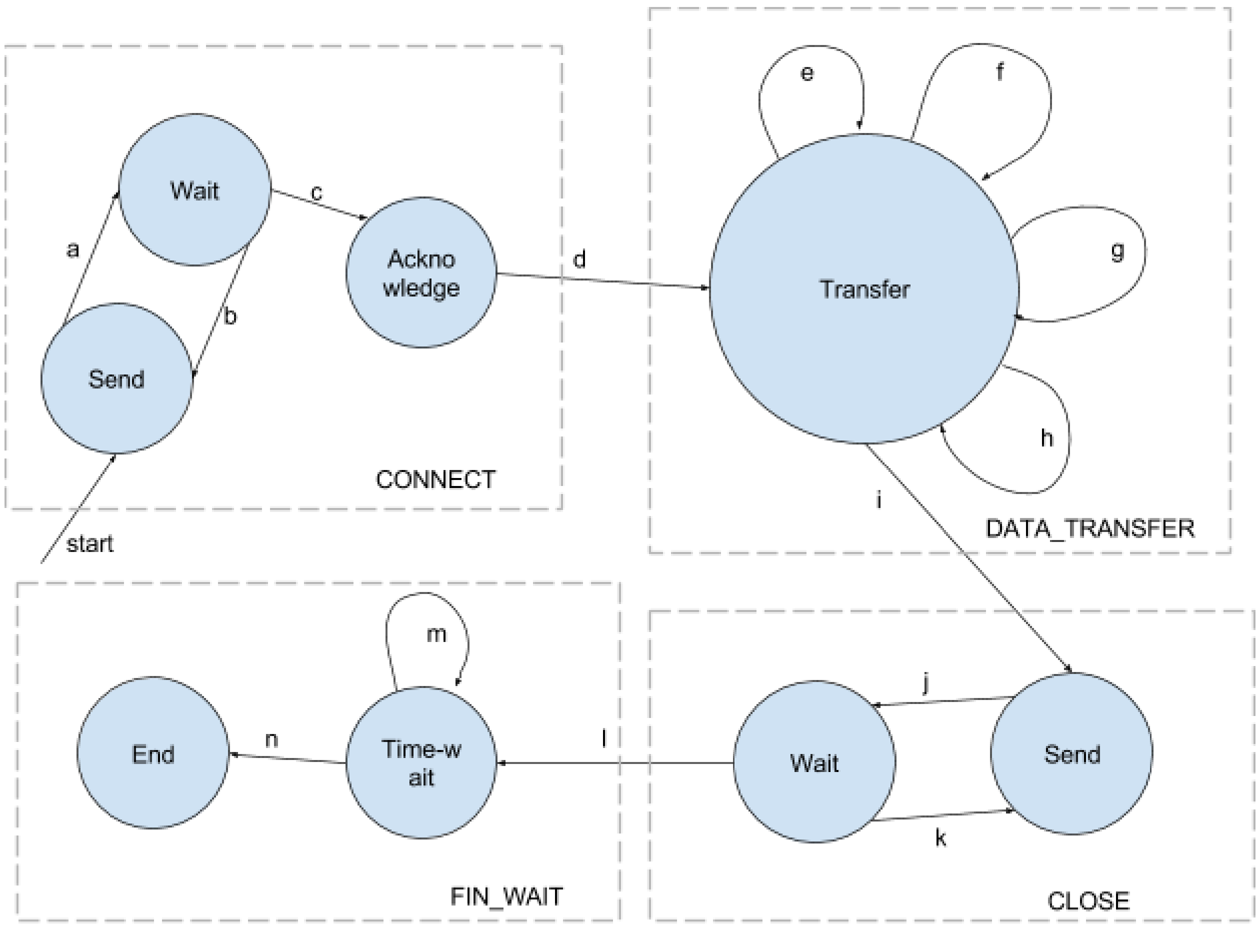
Specific Transit Actions:

start: Use the command line to start the server. RELDAT server is now listening.

a. When the server receives a packet with SYN bit set, sequence number is 0, the RELDAT server will go to the next state.
b. After the server sends an acknowledge packet, it will go to the next state, waiting for any further response. This acknowledge packet will be configured as: SYN bit = 1, sequence number = 0, data = user’s inputted window size.

c. When the server receives a packet with SYN bit cleared, and sequence number equals to 0, the RELDAT server will go to the next state. Notice this packet will piggyback the first chunk of data.
d. After the server gets into the ‘Established’ state, it will transit automatically to the CONNECT macrostate. The expected sequence number will be set here. Since RELDAT server starts with a sequence number of 0, the expected sequence number will be set as 1 at here.
e. If the server successfully receives a non-corrupted packet with the desired sequence number, RELDAT server will decode the packet, extract the data, change all letters to their uppercase, make a new send packet with this modified data, set the sequence number to current expected sequence number, update the checksum, send this packet to the client and finally, increment the expected sequence number.
f. In all other cases, the receiver discards the packet and resends the most recently received in-order packet to the client. If current received packet’s sequence number is larger than the expected sequence number or current received packet is corrupted, RELDAT server will send back the latest in-order packet. If current received packet’s sequence number is smaller than the expected sequence number, RELDAT server will do the same work as in ‘e’ -- modify this packet and send it back.
g. If the server receives a packet with FIN bit set, and the sequence number equals to -1, it will transit to the CLOSE macrostate.
h. The server will prepare a packet with FIN bit set, and let sequence number equals to -1. This packet is used to acknowledge the previously received FIN packet. The server will send this packet back to the client, set a timer and go to the next state.
i. If the server hasn’t heard anything from the client until the timer counts down to 0, it will return to the Send microstate and send the acknowledge packet again.
j. Once the server receives a packet with FIN bit set, and sequence number equals to -2, it can go back to the CONNECT macrostate and wait for the next connection request.

RELDAT Client Life Cycle and Finite State Machine



RELDAT client has four macrostates, each consists several micro states.

CONNECT Macrostate:

In this macrostate, RELDAT client will send a connection request to the server, waiting for the response, and then acknowledge that response.

i. Send Microstate:

RELDAT sends a connection request.

ii. Wait Microstate:

RELDAT waiting for server's response.

iii. Acknowledge Microstate:

RELDAT sends an acknowledge packet to the server.

DATA_TRANSFER Macrostate:

In this macrostate, RELDAT will perform a bi-directional, pipelined, piggybacked streaming data transfer with the server.

i. Transfer Microstate:

RELDAT performs a data transfer, and handles several destructive situations.

CLOSE Macrostate:

In this macrostate, RELDAT client will send an end request to the server then close itself upon this request.

i. Send Microstate:

RELDAT client sends a FIN packet to initiate the closing stage.

ii. Wait Microstate:

RELDAT waits for an acknowledgment.

FIN_WAIT Macrostate:

In this macrostate, RELDAT client will send a final acknowledge packet to the server to let it shuts down.

i. Time-wait Microstate:

RELDAT client waits for 30 seconds to make sure nothing is sent from the server.

ii. End Microstate:

RELDAT client closes.

Specific Transit Actions:

a. After sending an SYN packet, the client will move to the next state.
b. The client waits for an acknowledgment. If nothing is received, return back to the previous state.
c. The client waits for an acknowledgment. If an acknowledge packet is received, move to the acknowledge state.
d. Sending the third packet as a response to the previously received acknowledgment. Notice this packet contains data with a sequence number 0. At the same time, initialize the base number and the next sequence number.
e. If the next sequence number is still within the base+window size, prepare the next packet and send it, and increment the next sequence number.
f. If an in-order, non-corrupted acknowledgment is received, increment the base number. If the base number equals the next sequence number, stop the timer. Otherwise, start the timer.
g. If a corrupted acknowledgment is received, discard this packet.
h. If a timeout event happens, send packets from the base number to the next sequence number - 1.
i. When the data transfer reaches its end, the client moves to CLOSE stage to prepare for a closing.

j. The client sends a FIN packet to indicate its closing request.
k. The client waits for an acknowledgment from the server. If nothing is received before the timeout, the client will return to the Send state to send again.
l. The client waits for an acknowledgment from the server. If an acknowledge packet is received, the client will proceed to the FIN_WAIT macrostate.
m. If the client receives a new acknowledge packet within a 30 seconds time-wait, it will respond that acknowledge packet, send an acknowledgment, and restart a 30-seconds timer.
n. If nothing is received during 30 seconds, the client will close itself.

Algorithmic Description

Checksum

RELDAT uses a simple checksum algorithm to verify the integrity. There are two functions related to the checksum process:

int getDataSum(): return the arithmetic sum of all packet fields.
void updateCheckSum(): set the checksum field according to values of other fields.

When creating a packet, after filling all needed fields, we always call the updateCheckSum() function to update the checksum field of this packet. This function will make the checksum equals to the 1s complement of the sum of all other fields.
Once a new packet is received, its integrity must be checked before further operations. To perform the checksum process, use getDataSum() to get the a sum, then add this sum to the checksum field. If the result is -1(0b11111111111111111111111111111111 in binary), then the packet is not corrupted. Otherwise, this packet is corrupted and should be taken special consideration.

Encoding and Decoding

As mentioned at the Header Structure section, RELDAT modified the UDP packet from the Java.net library to achieve the reliability. An encoder is implemented to embed our own header to the UDP packet and a decoder is implemented to extract our own header from the default UDP packet. The encoder takes in our own packet and turns it into a byte array, therefore it could be used as the data field of the default UDP packet. The decoder takes in a default UDP packet, gets its data byte array and partitions this array into several variables representing different header fields.

Required Questions

Q: How does RELDAT perform connection establishment and connection termination?
A: Please refer to the High Level Description section.

Q: How does RELDAT detect and deal with duplicate packets?

A: At the server side, once it receives a packet with the expected sequence number, it will slide the window by one(Notice, the window size at the server is always one). This window-sliding process is implemented by increment the expected sequence number. The following duplicate packets will be handled just as out-of-order packets.

Q: How does RELDAT detect and deal with corrupted packets?
A: A checksum process(please refer to the Algorithmic Description section) is implemented on both hosts. When a corrupted packet was found at the server side, the server will print the error message and do nothing, therefore the client will not receive an acknowledgment and the client will handle it as a timeout event.

Q: How does RELDAT detect and deal with lost packets?
A: Lost packets are handled by timers. There is a time on the sender side. This timer will keep monitoring the oldest unacknowledged packet. If the sender does not successfully receive an acknowledgment of its oldest unacknowledged packet from the server within the countdown period of its timer, it will retransmit all packets within current window size to the server.

Q: How does RELDAT detect and deal with re-ordered packets?
A: The order of packets will be maintained by the Go-Back-N methodology. At the receiver side, the arriving packet will be discarded if its sequence number is not the same as the expected sequence number of the receiver. If the arriving packet's sequence number is as expected, then this packet will be buffered. In both cases, the server will send an acknowledgment with its expected sequence number.

Q: How does RELDAT support bi-directional data transfers?
A: There are two keys to achieving this bidirectional data transfer. First, each host has a send buffer and a receive buffer. By doing this, both the client and the server can buffer data that need to send and buffer data that are received. Second, each RELDAT packet has a sequence number field and an acknowledge field. In this way, the role of a sender and the role of a receiver can be achieved by a host simultaneously in a single packet.

Q: How does RELDAT provide byte-stream semantics?
A: On each host, the data was collected by unpacking(we called it decoding) the packet, and loaded by packing(we called it encoding) into a packet. The connection-oriented feature allows two endpoints to transmit packets simultaneously.

Q: Are there any special values or parameters in your design (such as a minimum packet size)?
A: The sequence number starts from 0; The sequence number of the first FIN packet during the closing stage is -1; The sequence number of the second FIN packet during the closing stage is -2.