Reliable Data Transfer Protocol

Release 1.0

Ethan Iannicelli

Mar 14, 2025

CONTENTS:

CHAPTER ONE

RDT PROTOCOL

To get the checksum of an ICMP packet based on the string representation of the packet, use the `udp_checksum()` function:

`rdt_protocol.`**`udp_checksum`**`(`*data*`)`

perform a psuedo udp checksum by reducing the data to 4 bytes and taking one's complement

Parameters
`data` (*`bitstring`*) – the data that the checksum is created from

Returns
generated checksum

Return type
int

To create a packet using a sequence number, acknowledgment number, and data, use the `create_packet()` function:

`rdt_protocol.`**`create_packet`**`(`*seq_num*, *ack_num*, *data*`)`

create a packet using a sequence number, ack number, and data

Parameters

- **`seq_num`** (*`int`*) – the sequence number of the packet
- **`ack_num`** (*`int`*) – the ack number of the packet
- **`data`** (*`bitstring`*) – the data to be included in the packet

Returns
bitstring representing formed packet

Return type
bitstring

To parse a packet into its sequence number, acknowledgment number, checksum, and data, use the `parse_packet()` function:

`rdt_protocol.`**`parse_packet`**`(`*packet*`)`

extracts seq, ack, checksum, and data from a packet

Parameters
`packet` (*`bitstring`*) – formatted packet

Returns
4 tuple of seq, ack, check, data

Return type
tuple

To split a bitstring of data into multiple parts of a given size, use the `split_data()` function:

`rdt_protocol.`**`split_data`**`(`*data*, *chunk_size*`)`

splits a bitstring of data into multiple parts of a given size

Parameters

- **`data`** (*`bitstring`*) – bitstring of the full data
- **`chunk_size`** (*`int`*) – maximum chunk size

Returns
array of data split up into chunk_sizes

Return type
array

The `ReliableDataTransferEntity` class represents an RDT entity that can act as a sender or receiver:

`class` `rdt_protocol.`**`ReliableDataTransferEntity`**`(`*inter_address*, *entity_address*, *window_size=4*, *timeout=True*`)`

Bases: `object`

class for a RDT entity, either a client or server. different types of entity are differentiated by their actions and behaviors

`receive``()`

recieves data from a network

Parameters
`self` (`ReliableDataTransferEntity`) – the receiver object

Returns
the data in the packet

Return type
bitstring

`send``(`*data*`)`

sends data based on the entity of the sender

Parameters

- **`self`** (`ReliableDataTransferEntity`) – the sender object
- **`data`** (*`bitstring`*) – data to be sent

To simulate packet loss, use the `simulate_loss()` function:

`intermediary.`**`simulate_loss`**`(`*packet*`)`

simulate packet loss

Returns
the packet if no loss, None if else

Return type
bitstring?

To simulate packet corruption, use the `simulate_corruption()` function:

`intermediary.`**`simulate_corruption`**`(`*packet*`)`

simulate packet curruption

Returns
the packet

Return type
bitstring

To simulate packet reordering in the packet queue, use the `simulate_reordering()` function:

`intermediary.`**`simulate_reordering`**`(`*packet_queue*`)`

simulate packet queue reordering

Returns
packet queue

Return type
array

To simulate packet delay via sleep, use the `simulate_delay()` function:

`intermediary.`**`simulate_delay`**`()`

simulate packet delay via sleep

To handle a packet by applying network conditions and forwarding it to an address, use the `handle_packet()` function:

`intermediary.`**`handle_packet`**`(`*packet*, *packet_queue*, *inter_socket*, *forward_address*`)`

handles a packet by undergoing network conditions and forwarding to address

Parameters

- **`packet`** (*`bitstring`*) – the packet to be handled
- **`packet_queue`** (*`array`*) – queue of packets to be delivered
- **`inter_socket`** – the socket of this script
- **`forward_address`** (*`2 tuple of ip and port`*) – the address to forward the packet to

To run the intermediary that simulates network conditions and handles forwarding of packets, use the `run_intermediary()` function:

`intermediary.`**`run_intermediary`**`()`

runs the intermediary that acts as a network for this project. simulates network conditions and handles forwarding of packets

To send all data from a given file to the server, use the `send_file()` function:

The `FileTransferClient` class represents a client for the file transfer procedure:

`class` `client.`**`FileTransferClient`**

Bases: `object`

`send_file``(`*file_path*`)`

sends all the data from a given filepath to the server

Parameters

- **`self`** (`FileTransferClient`) – client in the file transfer procedure
- **`file_path`** (*`String`*) – relative filename to this program being run

To receive a file and save it to a designated folder, use the `receive_file()` function:

The `FileTransferServer` class represents a server for receiving and saving files:

`class` `server.`**`FileTransferServer`**

Bases: `object`

`receive_file``()`

receiver function for a file transfer destination/server saves data to a file destination (constant)

Parameters

`self` (`FileTransferServer`) – the server object