RDP PROTOCOL OVER A RINGO NETWORK: DESIGN DOCUMENT

by John JaeMin Baek and Thomas J. Bright

ARCHITECTURE

The architecture of the ringo network can be divided into several parts. The first to be used, even before a user is allowed to interact with the ringo, is the **peer discovery** architecture. This uses a python socketserver to send and receive information from all x ringos in the network (x being a user-specified number from 2 to 5). Peer discovery proceeds into **optimal ring calculation** as it collects the peers' sent distance vectors into a distance matrix, which is then passed into an optimal-ring finding, **dynamic programming algorithm**. This gives the ringo the next ringo in its path, it's **neighbor**, in the event of data transfer.

As a user, the first piece of the ringo architecture seen is the **command-line**. Here, the user can input commands, including "send." Send utilizes the **data transfer** architecture built into the ringo. The network contains one sending ringo, one receiving ringo, and 0-3 forwarding ringos. The sender initiates transfer at the user's command by reading a specified file into a python list whose elements are then encoded and sent to the ringo's neighbor. The receiving ringo populates its python list with the pieces of data from these packets. If they are Forwarders, they then send them to their own neighbors. Otherwise they are Receivers, and the data's journey ends there as it is written into a file in the Receiver's home directory.

To ensure reliable data transfer, the ringo network utilizes a **Go-Back-N** re-transmit policy and a **keep-alive** signal periodically sent to every node in the network. Both of these protocols utilize a timeout. In Go-Back-N (GBN), after a certain without acknowledgment of the least unacknowledged packet, the ringo resends an entire window of packets. In keep-alive (KA), after a certain amount of time without receiving a signal from a certain ringo on the network, other ringos assume it has gone offline, and excommunicate it from the network.

HEADERS

The different parts of the architecture require different types of packets, each of which has its own headers.

PEER DISCOVERY

The packets used for peer discovery transport no data, only the information in their headers. The fields are as follows:

*command***:** used by all packets; tells the receiving socket what the purpose of the packet is; this packet will have the command of "peer_discovery"

*peers***:** list of known peers of the sending ringo; upon reception, a ringo updates its own peers list in response to the new information; peer discovery continues until all ringos have peer lists of the expected size

*ttl***:** this ensures that the packets don't continue forth forever; upon arriving at a packet, this value is decremented; packets stop propagating after arriving six times

FIND RTT
This is the point at which the ringos are trying to find the round-trip times to each other.

command: "find_rtt"

rtt_count: used to determine what leg of the trip this packet is on

created: time of packet's creation; used to calculate the rtt

SEND RTT VECTOR
These packets send distance vectors, which update in response to new information and are stored by the receiving ringos in a RTT-matrix.

command: "send_rtt_matrix"

peers: peers and their distance vectors

ttl: trips to live

FILE
These are used to transport data

command: "file"

filename: name of file being transferred

file_length: length of file in bytes

seq_number: number of current packet

data: payload

FILE ACKNOWLEDGMENT
These are the ack packets of the file packets

command: "file_ack"

filename: name of file being acked

file_length: length of file being acked

ack_number: sequence number of the file being acked

KEEP-ALIVE
These are used to tell peers that a ringo is still on the network

command: "keepalive"

seq_id: used to identify the thread of origin for this keep-alive signal

created: time created

ttl: maximum allowed re-transmissions

TIMING

The GBN protocol uses a simple approach: whatever sequence number reaches the receiver gets acked back to the sender, regardless of whether it was expected. The sender is the only discriminant party, and will only reply with a new sequence for the acknowledgment it wanted. Eventually, the sender will timeout, and resend the entire window. This method protects against the edge case in which a sender's final packet is received, but the ack message is lost. In such a case, the sender would timeout and send another, as seen in fig.s 1 and 2.

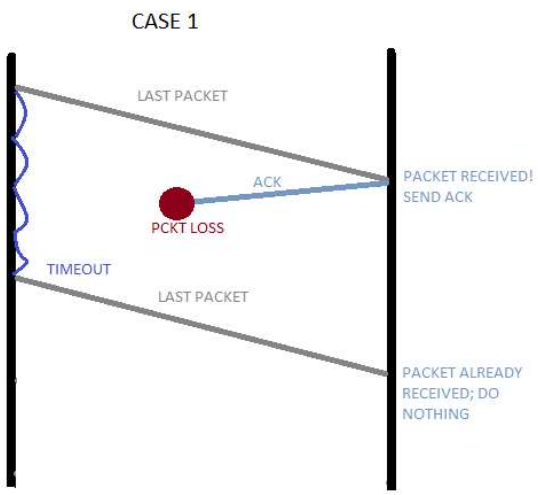

fig 1

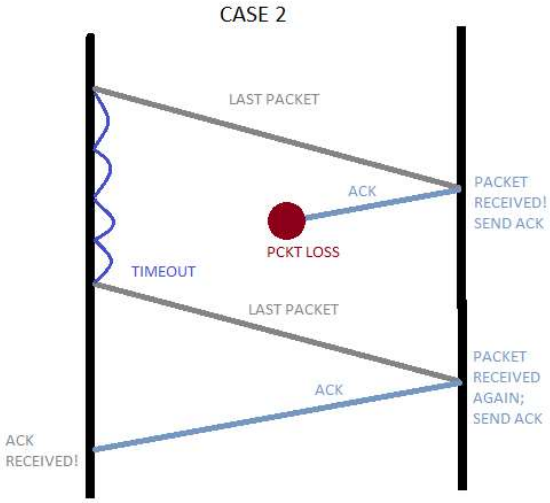

fig 2

PSEUDOCODE

The most difficult and non-trivial piece of code used in the ringo is without a doubt the function findRing(), which computes the optimal-ring, given a fully stocked matrix. Though the network is always of a reasonably small size, optimal-ring is still an NP-hard problem with no intuitive solution. The psuedocode lies below:

```
findRing(node, nodes, currentPath, distance, routes):
        add node to currentPath
        if length of currentPath > 1:
                distance →distance + nodes[currentPath[-2]][node]
        if length of nodes is equal to length of currentPath and currentPath[0] is in nodes[currentPath[-1]]:
                add currentPath[0] to currentPath          #this makes it circular
                distance →distance + nodes[currentPath[-2]][currentPath[0]
                add (distance, path) to routes
                return
        if node is in nodes:
                if node is not in currentPath and node is in nodes[node]:
                        findRing(node, nodes, currentPath, distance, routes)
```

KEY DATA STRUCTURES

This ringo network implementation depends on several data structures. Here is a review of the most important:

peers:	a ringo's distance vector, which stores distances between itself and its peers
rtt_matrix:	a matrix containing the complete **peers** vectors of all peers; this will be identical in all ringos on the same network
routes:	python list used to store possible routes while calculating optimal-ring
file_chunks:	python list used to store the transferred file in 1024-byte chunks
window:	GBN packet window

THREAD ARCHITECTURE

Threads are used in multiple places throughout the ringo architecture. Perhaps most importantly, a single thread is dedicated to the python socketserver's MyHandler() function, allowing it to run, receive packets, and process packets indefinitely.

Alongside that are threads created to serve a purpose before dying off. The timeout function used in for the GBN protocol is such a thread. An instance is created upon transmission of the first packet of a file; it loops for as long as it takes to receive the final file ack, constantly measuring the current time against the time the presently-expected packet was sent.

Similar to the timeout threads are the keep-alive threads, who die off after not receiving a keep-alive signal from their designated peer after a certain amount of time.