Reliable Multicast tool using UDP/IP

David Gong, Stephen Hamilton

Abstract—Our goal is to develop a multicast system capable of reliably sending packets in agreed order utilizing the UDP/IP protocol. In order to accomplish this task, we will implement a ring protocol where a token is passed in order to establish the control of sending packets. A node will only be able to send packets when the node is in possession of the token.

1 Introduction

This multicast ring protocol works by utilizing a token to control the flow of packets multicasted to all processes. There are three stages to this protocol. First is the startup which involves initializing the processes and determining the ring-path for the token.

The second stage is data transmission. Once the processes are initiated, the first process will generate the first token. If that machine has data to send, it will send data up to the maximum flow control value (which is pre-defined but also adapts as the program progresses). Once it has sent the data, it will send an updated token to the next machine. The next process then writes from its received data packets up to the min(token_aru, previous_token_aru). It resends any requests contained in the rtr and it sends the packets it needs to send, updates the token sequence number, rtr, an finally sends the token. And so the cycle repeats. Due to the potential for loss over UDP, this protocol has an additional field on the token called loss_level which determines the number of times a token is sent per attempt from one process to another. This protocol will attemp to adapt the loss_level by detecting when a token is lost. The intent is to increase the reliability of the token by sending it multiple times.

Finally, because properly ending is provably unsolvable, we adopt the method of termination that relaxes the requirement for theoretical, finite termination, but in practical settings will work by drastically reducing the chance of failure after every attempt further.

2 Design

2.1 Assumptions

Before delving into the specifics of our protocol, there are a few assumptions on which the success of our protocol depends.

- Machine ids will be in sequential and continuous order up to the maximum number of machines (10).

With these assumptions, we can describe a successful multicast protocol for agreed data ordering over UDP.

2.2 Token Design

```
// Structure of token.
int type;//Type of packet
int sequence;//Sequence of last message
int aru;//Sequence of all rcv up to
int fcc;//Flow control/Max send size
int rtr[MAX_RTR];//Array of Retransmit
    requests
int rtrcount;//Number of entries in rtr
    array
int loss_level;//Increases on token
    regeneration
int nodata[10];//Process completion flag
int round;//Number of ring traversals
int aru_lowered_by;//Flag for lowering
    the aru
```

2.3 Data Structure

The data structure we plan to implement is a linked list. This list will contain all the sent packets that are greater than min(token.aru, previous token.aru). It will be implemented in a struct as follows:

```
struct packet_structure {
int type;//Type of packet
int sequence;//Sequence of packet
int received;//Flag for when globally
    received
int machine_index;//Index of packet's
    sender
int packet_index;//Index of packet from
    the sender
int random_number;//Data
char data[packet_size];//Contents
}
```

The packet structure contains the overall token sequence, whether or not the packet was received, the machine_index of the sender, the packet_index of the sender's sequence, the random number, the 1200 additional payload bytes, along with the pointer to the next packet.

2.4 State Machine

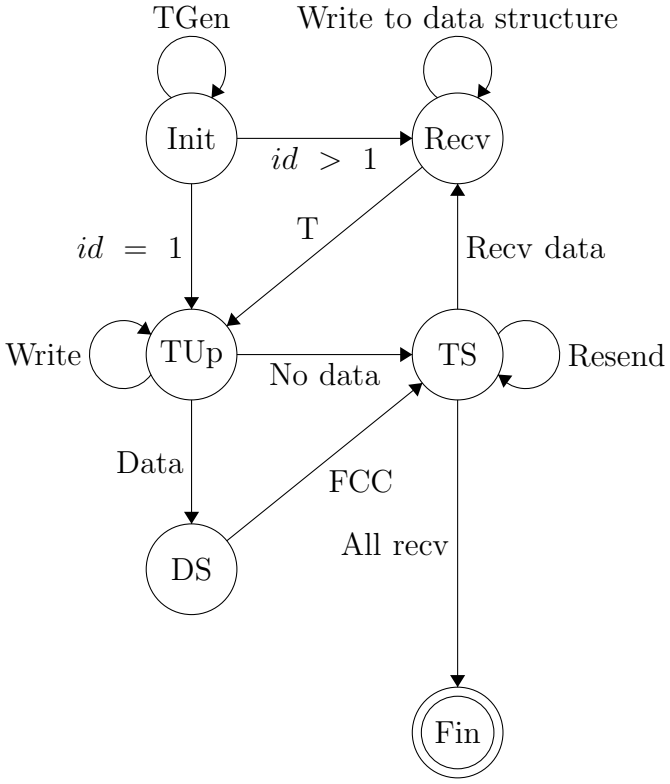

Fig. 1. A general state machine for the program. T is for token, D is for data, S is for send,.

2.5 Algorithm

We dedicate a section to explaining the details of the protocol. As stated before, the protocal is split into three main parts: startup, message transmission, termination.

The startup is difficult because we must find a way to successfully establish the ring through which the token is passed (using unicast). Our approach to this is a simple ack routine. Every machine starts out by multicasting its own information so that everyone (ncluding the target) receives the information. In the meantime, every machine also filters through the received packets, looking for its target (the machine with the next sequential ID). Once it successfully receives the information, it sends a message to the target so the target knows to stop. If the machine receives conformation, then it sends a message to machine 1 (who is essentially the controller). Until all machines are done, machine 1 will not create the token therefore the transmission process will not start. Once all acks are received, the process starts with machine 1 sending data (note, once the other machines start receiving data packets, they know for sure that they're in the messaging stage).

The transmission proceeds as discussed in the introduction essentially covers everything. A note is that flow control is determined by how often various limits are reached. For example, if the rtr ever reaches a state of being completely full, then we lower the fcc meaning that people must send less (giving more opportunitites time for the machines to catch up).

Finally, termination is done by sending a mass of tokens before finaly closing. That way the probability of none of them being received is 0 in a practical setting.

3 Results

4 Conclusion

TBD

5 Discussion

TBD

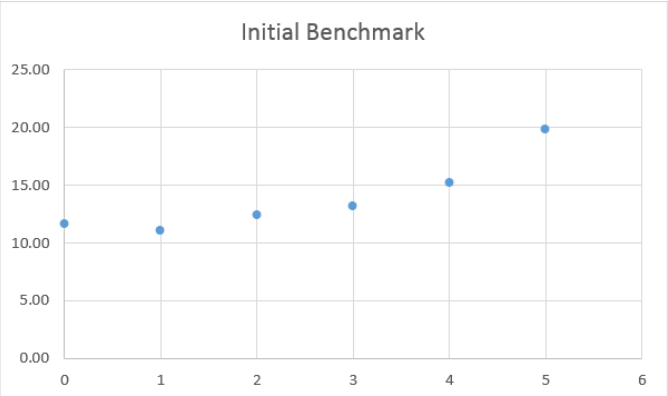

Fig. 2. Inital benchmark results. The y-axis is the number of seconds for transmission and the x-axis is the loss rate which goes from 0%, 1%, 2%, 5%, 10%, 20%.

Loss	Times
0	11.61
1	11.11
2	12.43
5	13.17
10	15.19
20	19.80

Fig. 3. Hard results.