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. The general idea is that once the processes are initiated, the first process (machine_id = 1) will generate the first token. If that machine has data to send, it will send data up to the maximum flow control value (pre defined). Once it has sent the data, it will send the token to the group with the process id set to the next receiving process. The next process then checks the min(token.aru, previous token.aru) and writes all the data it has up to that aru from the data structure to the file. It looks at the rtr and resends any packets that it has. Finally, it sends the packets it needs to send, updates the token sequence number, rtr, and sends the token to the next process. So the cycle repeats. Due to the potential for loss over UDP, this protocol has an additional field on the token called loss_level. In an environment where loss could be large, the token protocol will have an issue due to token loss. This protocol will attempt to overcome that problem by detecting when a token is lost, and not only regenerating the token, but increasing the reliability of the token by sending it multiple times. As loss increases, the number of times the token will be sent by each node will also increase.

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 sequence;//Sequence of last message
int aru;//Sequence of all rcv up to
int fcc;//Flow control/Max send size
int rtr[fcc];//Array of Retransmit
    requests
int rcv_process_id;//Receiving process id
int loss_level; //Increases on token
    regeneration
```

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 token_sequence;
  int received;
  int machine_index;
  int packet_index;
  int random_number;
  char data[packet_size];
  struct packet_structure *next;
  }
```

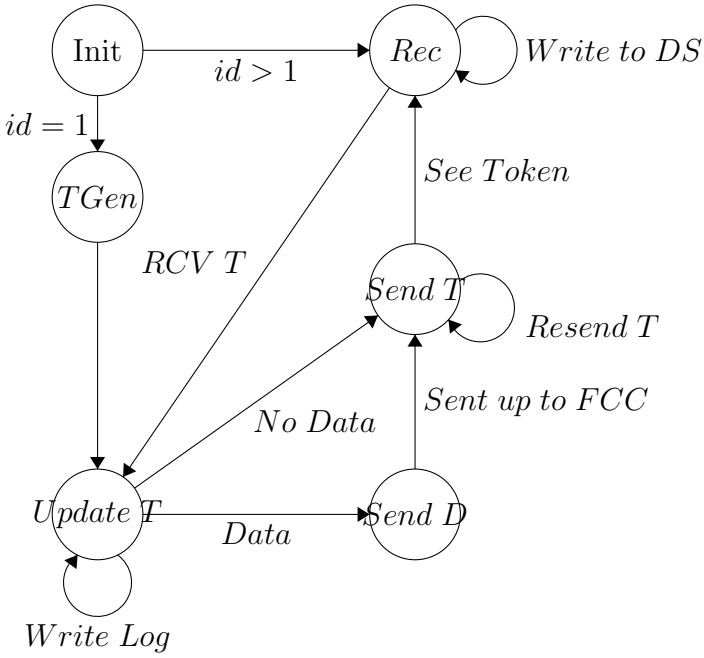
The packet structure contains the overall token sequence, whether or not the packet was received, the machine_index of the sender, the packet_index of the sender's sequence, the random number, the 1200 additional payload bytes, along with the pointer to the next packet.

As data is received, it is put into the linked list of packets. If a packet is received out of sequence, then a placeholder packet is created with received set to 0. Then the packet further in the sequence is populated with the received data. Once the token is received, the packets that are at sequence number min(token.aru, prevtoken.aru) or before are written to the log and deallocated. This allows each process to maintain all the packets that are not verified as as received by all processes in memory. Since they are in memory, they can be re-sent if they appear in the token retransmission request (rtr).

We also plan to utilize the following variables.

```
struct initializers {
struct packet_structure *head;
struct packet_structure *tail;
FILE *logfile;
int packets_to_send;
int machine_index;
int packet_index = 0;
int total_machines;
int loss_rate;
int token_timeout;
int prior_token_aru = 0;
}
```

2.4 State Machine



2.5 Functions

In this design, we plan to utilize many functions in order to carry out the described algorithm.

```
function init()
   Waits for start_mcast to singal start

function token_generate()
  returns token;
  generates the first token

function update_token
(struct token *t, int sequence)
  updates the token with sequence number

function write_log
(struct initializers *i)
  writes to log for all received data

function update_rtr
(struct initializers *i)
  Writes rtr to token based on missing
   packets

function send_data
(struct initializers *i, struct token *t)
   returns new sequence number
```

```
function send_token
(struct token *t)
   sends the current token to the next
    process

function add_packet
(initializers *i,
struct packet_structure *p)
   adds an incoming packet to the data
    structure.

function send_rtr_packets
(initializers *i, struct token *t)
   sends packets needed to be
    retransmitted from token rtr

function generate_packet
(initializers *i)
Generates the next packet, and
  increases packet_index
```

3 RESULTS

TBD

4 CONCLUSION

TBD

5 DISCUSSION

TBD