Distributed Systems Exercise 2 Design Document

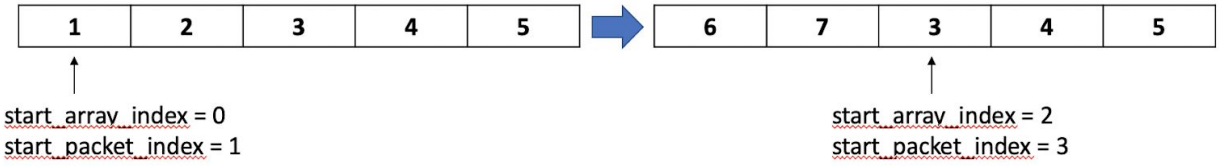
Minqi Ma (JHED: mma17), Jerry Chen (JHED: gchen41)

General Approach

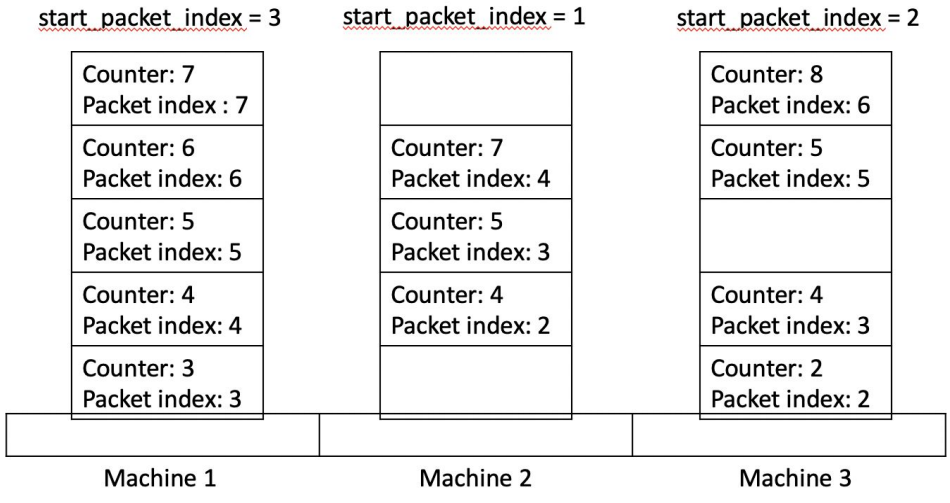
- Use Lamport timestamps to deliver packets in agreed and FIFO order
- Create a table to store received packets; deliver packets with the same counter at once; stop delivering if one packet index is missing
- Send NACK packets in multicast; any machine can respond if it has the packet
- Exit the program if every machine has the same last counter

Data Structures

- *int* created_packets*: array to store created packets
  - Note: to discard a delivered packet and create a new one (i.e sliding window), we wrap around and directly replace the old packet with the new one
  - For example: WINDOW_SIZE = 5, and we want to discard packet 1 and 2 and create packet 6 and 7



- *int* acks*: array of integers to record for machine i, how many packets from the current machine has delivered
  - For example: machine_index = 1, acks = [1 2 3] means machine 1 (i.e itself) has delivered 1 packet from machine 1; machine 2 has delivered 2 packets from machine 1; machine 2 has delivered 3 packets from machine 1.
- *int** table*: table of size num_machine * WINDOW_SIZE to store received data packets from other machines
  - Packets are placed in the table according to its machine_index and packet_index
  - At the spot for current machine, it is always empty, since created packets are stored in *created_packets* array. (We put *created_packets* array at the spot just for demonstration)
  - Each array use wrap-around technique (same as above) to avoid shifting entries
  - For example, num_machines = 3, machine_index = 1, WINDOW_SIZE = 5. *table* can look like:



*This is in fact *created_packets* array

- *int* end_indices*: array to store last packet index for each machine

- *bool* finished*: array to indicate for each machine i, if current machine has finished deliver its packets
- *int counter*: current counter to support Lamport timestamps
- *int last_delivered_counter*: the last counter that current machine has finished delivering
- *int* start_array_indices, int* start_packet_indices*: parameters stored to help wrap around arrays
- *bool ready_to_end*: if current machine has finished delivery and received all acks (explained in Algorithms part)

Message formats

struct packet:

- *unsigned int tag*: indicate the type of the packet
- *unsigned int counter*: counter of the packet
- *unsigned int machine_index*: index of the machine which created the packet
- *unsigned int packet_index*: packet index (or sequence) of the packet
- *unsigned int random_data*: a random integer between 1-1000000
- *int* payload*: unused data, or information about ack, nack or last counter (as specified below)

Types of packets:

- TAG_START: machines can start transmission
- TAG_DATA: data packet which has counter, machine_index and random_data. *payload* is not used.
- TAG_ACK: first <num_machines> integers in *payload* record, for each machine i, the number of packets from machine i that current machine has delivered
    - For example, num_machines = 3, machine_index = 1, payload = [1 2 3] means that machine 1 has delivered 1 packet from machine 1; machine 1 has delivered 2 packets from machine 2; machine 1 has delivered 3 packets from machine 3.
- TAG_NACK: first <num_machines> integers in *payload* record, for each machine i, packet index of the missing packet; -1 indicates no missing packet from machine i
    - For example, num_machines = 3, machine_index = 1, payload = [-1 3 5] means that machine 1 does not miss any packet from machine 1; machine 1 misses packet no.3 from machine 2; machine 1 misses packet no.5 from machine 3.
- TAG_END: *packet_index* contains the last packet index of the machine
- TAG_EMPTY: only used in *table*, indicates the cell is empty
- TAG_COUNTER: first <num_machines> integers in *payload* record the last counter for each machine. -1 indicates unknown.

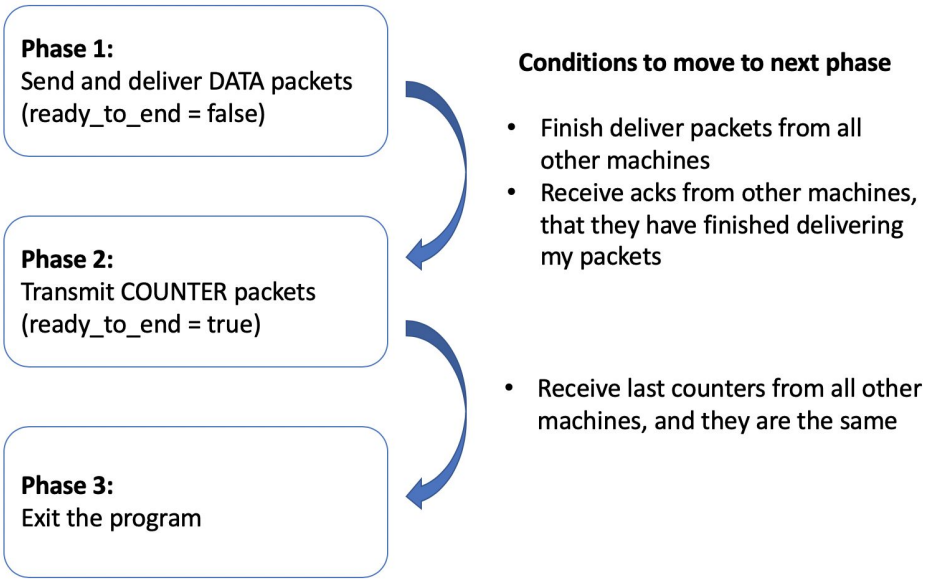
Algorithm

Lamport timestamps

- When creating a packet, increment counter and stamp the packet with counter, machine_index and packet_index
- Adopt the larger counter when receiving a DATA packet
- Deliver in lexicographical order of <counter><machine_index>

Termination logic

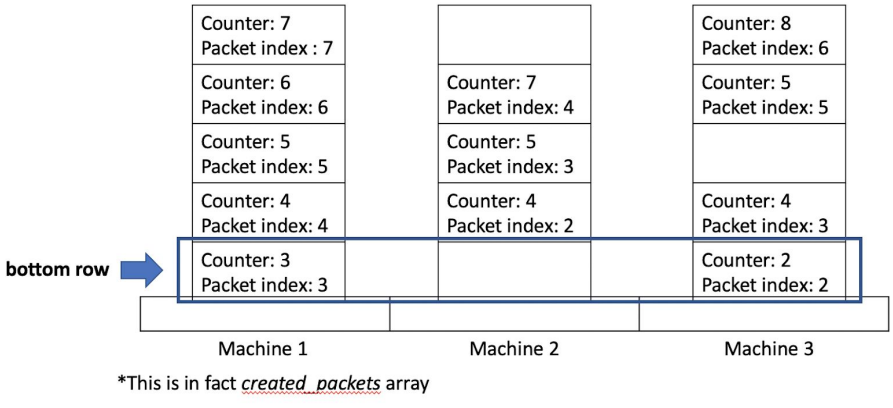
- Each machine goes through 3 phases (to satisfy 3 termination conditions):



- In Phase 1, every time after delivery or updates on *acks*, we check if the machine satisfies 2 conditions to move on to Phase 2.
- In Phase 2, every time after receiving COUNTER packet, we check if the machine satisfies the condition to exit the program.
- Note that *ready_to_end* distinguishes Phase 1 from Phase 2.

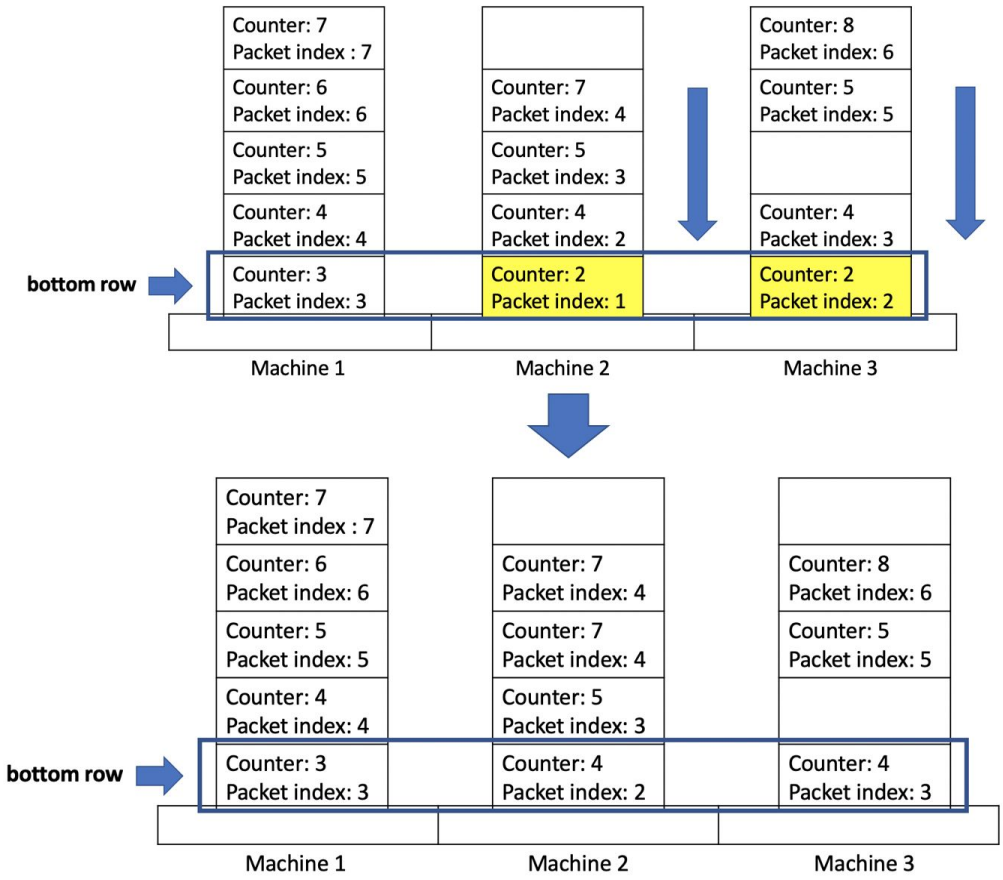
Protocol

- Receive TAG_START packet:
  - Generate WINDOW_SIZE packets in *created_packets* array
  - Send 1/10 of the window in multicast (to avoid overwhelming the network)
- Receive TAG_DATA packet:
  - Insert packet to the corresponding empty cell in the table
  - Adopt the larger counter
  - Attempt to deliver if among unfinished machines, the bottom row in the table are full (i.e has received the next packet to deliver for each machine)
    - Only deliver those with counter == last_delivered_counter + 1
    - If deliver its own packet, slide window and create new packets
    - If deliver other machines' packets, send ACK packet every time last_delivered_counter is increased by 1/10 WINDOW_SIZE (to avoid overwhelming the network)
    - Delivery will stop if the bottom row is not full; then send NACK packet in multicast
    - Examples: last_delivered_counter = 1
      CANNOT deliver because bottom row is empty; send NACK for packet no.1 of machine 2 and packet no.4 of machine 3



*This is in fact *created_packets* array

CAN deliver because bottom row is full. Can ONLY deliver highlighted packet since *counter* == *last_delivered_counter* + 1. After delivery, packet will be discarded and heads of array are slided. (can be interpreted as the rest of cells “fall” down)



  - If deliver packet with the last packet index, mark the machine as finished in *finished* array
- Receive TAG_ACK packet:
  - Update *acks* array from *payload* information. If created packets have been delivered by all machines, slide window and create new packets.
  - Attempt to deliver the same way as above.
- Receive TAG_NACK packet:
  - Look up in *table* and *created_packets* array. If has the requested packet, send it in multicast.
  - If the requested packet index exceeds last packet index for the machine, send END packet.
    - Note: we are using multicast for NACK, so any machine which has the machine or end index information can respond to a NACK packet.
- Receive TAG_END packet:
  - Record last packet index for the machine in *end_packet_indices* array
  - Update *finished* array if necessary.
- Receive TAG_COUNTER packet:
  - Update *last_counters* array accordingly
  - Send its own last_counters array in COUNTER packet if *last_counters* array gets updated
  - If received all last_counters and are the same, send COUNTER packets for 5 times and exit.
    - Note: once one machine receives all last_counters, we would like to inform all machines to exit. To make sure that the packet won’t be lost, we can send it for 5 times, as the highest loss rate is 20%.
- Timeout:
  - In Phase 1:

        - If haven't finished delivery, resend most recently sent NACK packet and current ACK packet
          If have finished delivery, send END packet with last packet index
        - If min(*acks*) != *num_packets*, (i.e not all packets have delivered packets from the current machine), send the packet in the window with the highest packet index
- In Phase 2: send COUNTER packet