MTCP : TRANSPORT LAYER SUPPORT FOR HIGHLY AVAILABLE NETWORK SERVICES

Kiran Srinivasan
New Brunswick, New Jersey
October, 2001

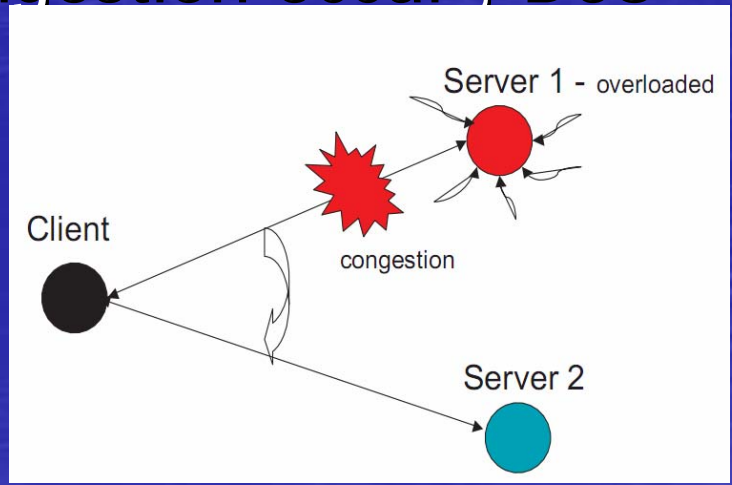
Shing-Guo Chang

Outline

- Introduction
- Implementation
- Experiment
- conclusion

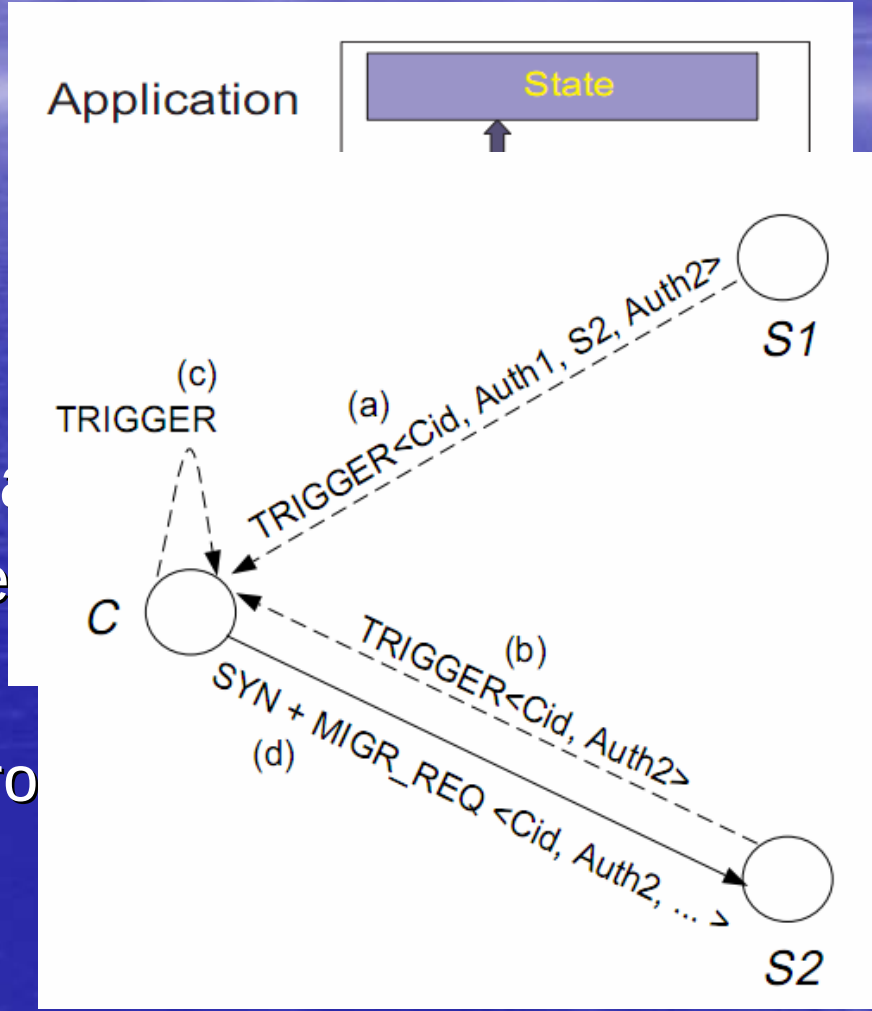
Introduction

- What is migration ?
  - change the connection by binding App. State , socket state , TCP state etc..
- When do migration ?
  - Server overload , congestion occur , DoS attack etc..



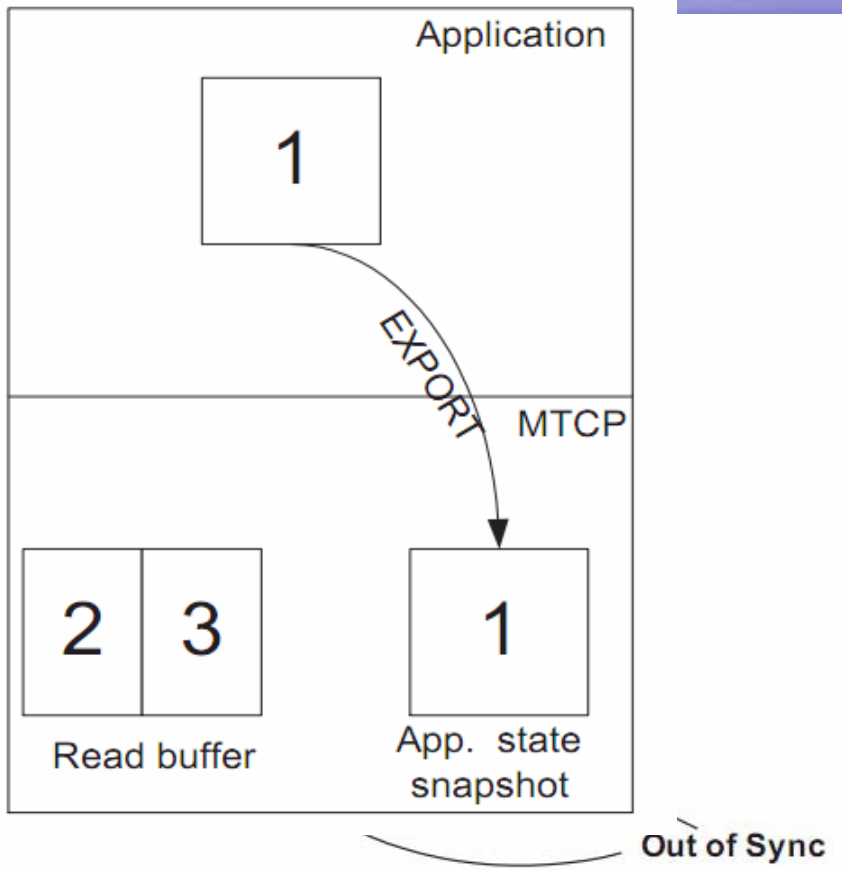
Introduction

- Migration issues
  - Triggers and initiators
    - Client , S1 or S2
  - Export/import state sna
    - App. state, socket state
  - State synchronization
    - Because of App. and pro
  - Buffer synchronization



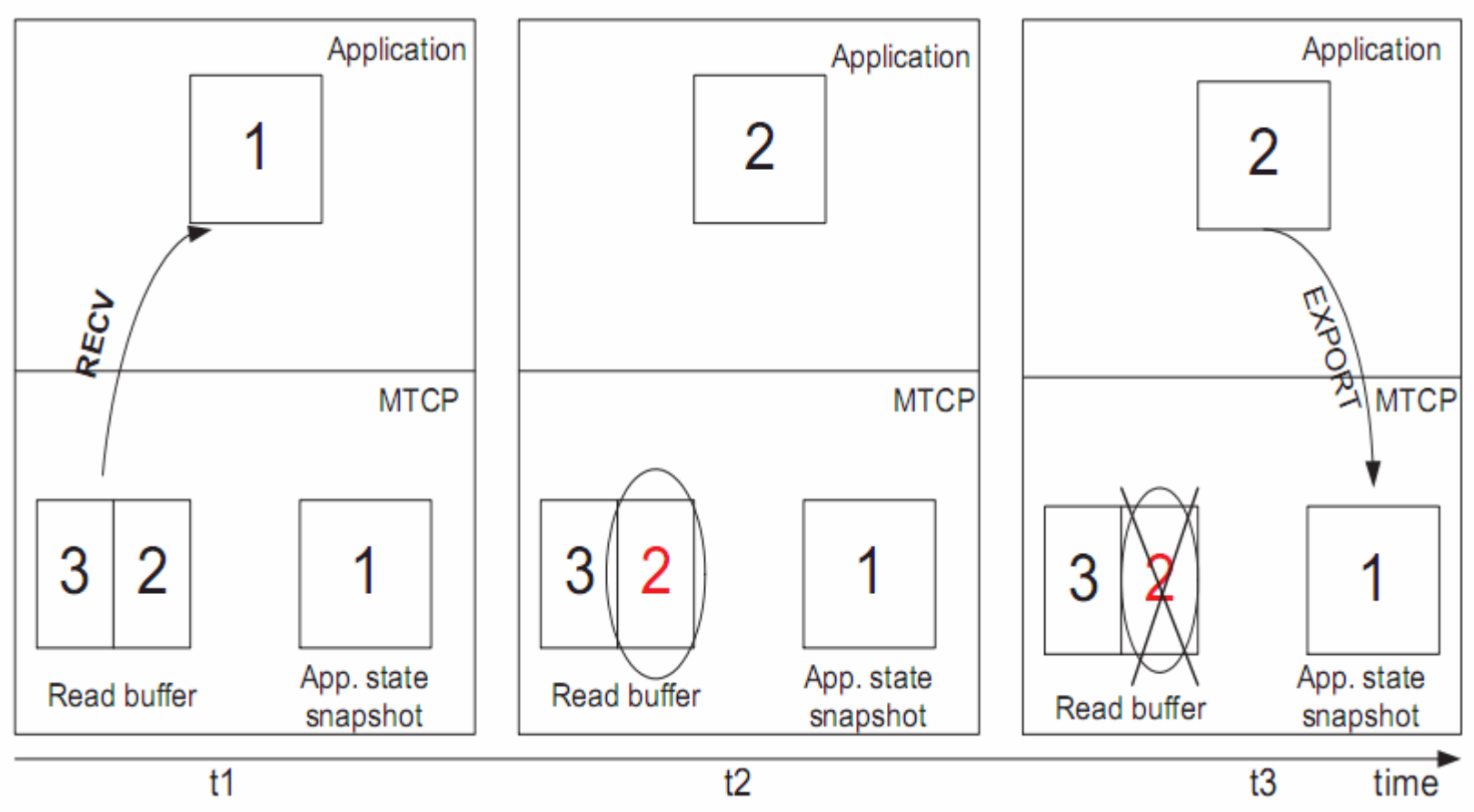
Introduction

```
while((s = accept(ssock)) != -1)
{
        if( import_state(s, &state))
                num = state
        else
                num = 0
        recv (s, &buf)
        state = ++num
        export_state (s, state)
        recv (s, &buf)
        state  = ++num
        export_state (s, state)
          . . .
}
```



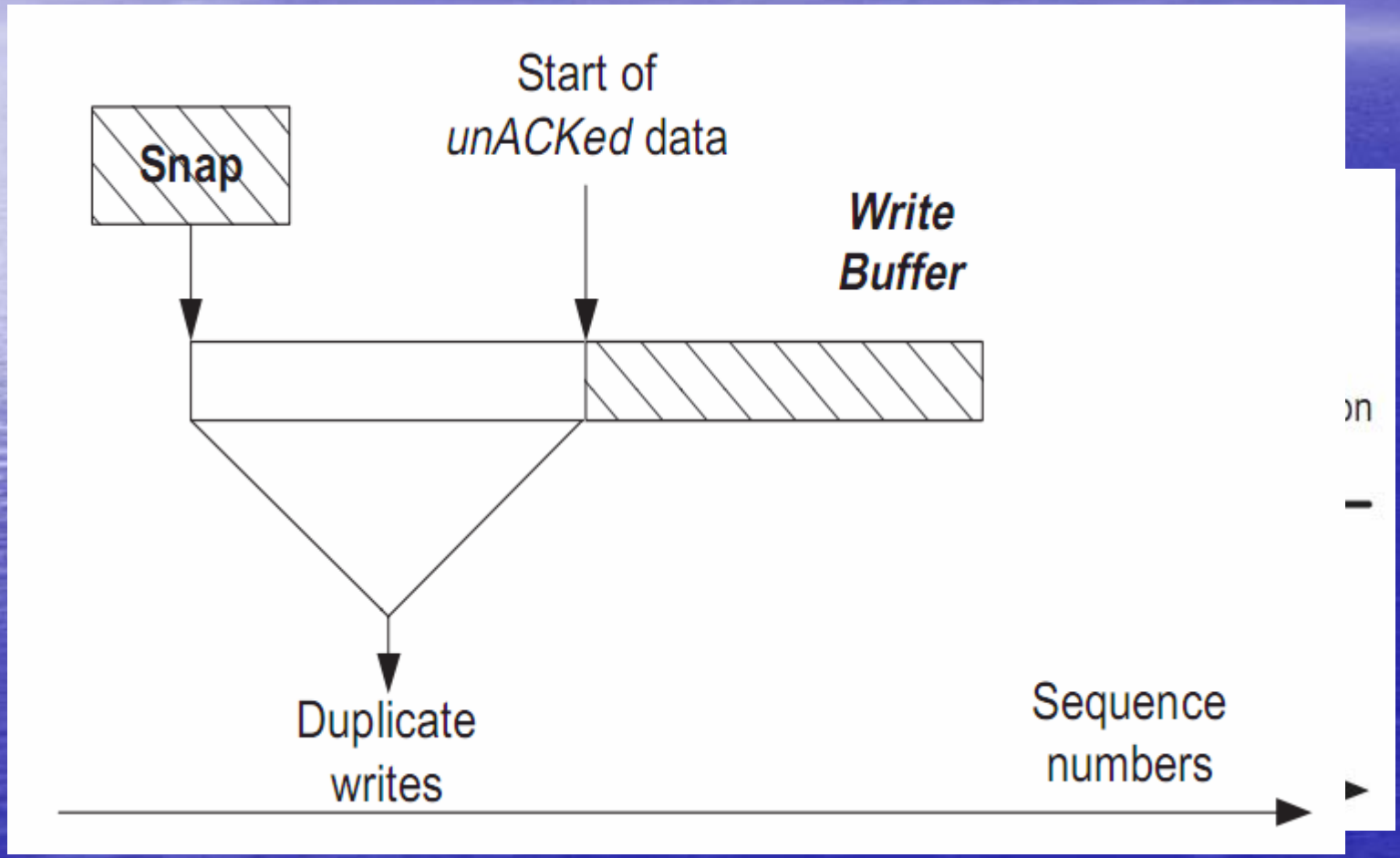
State Synchronization

Introduction



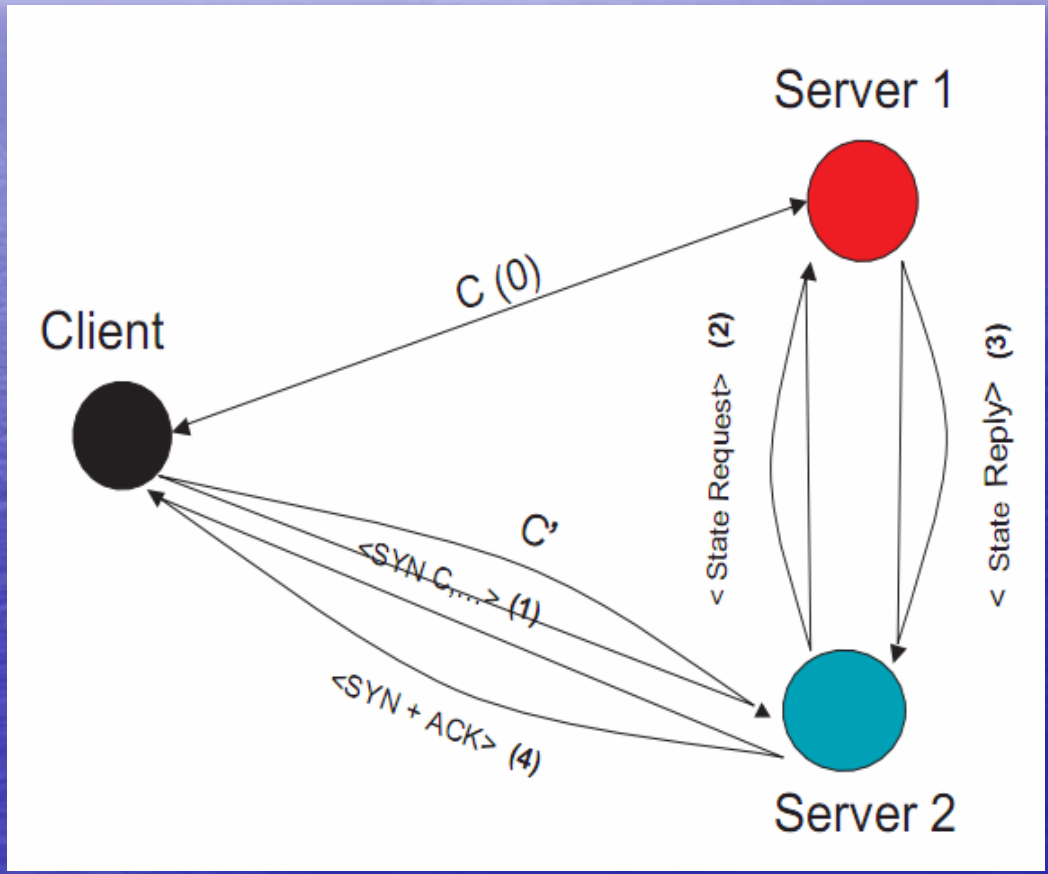
State Synchronization

Introduction



Buffer Synchronization

Implementation



Lazy migration scheme

Implementation

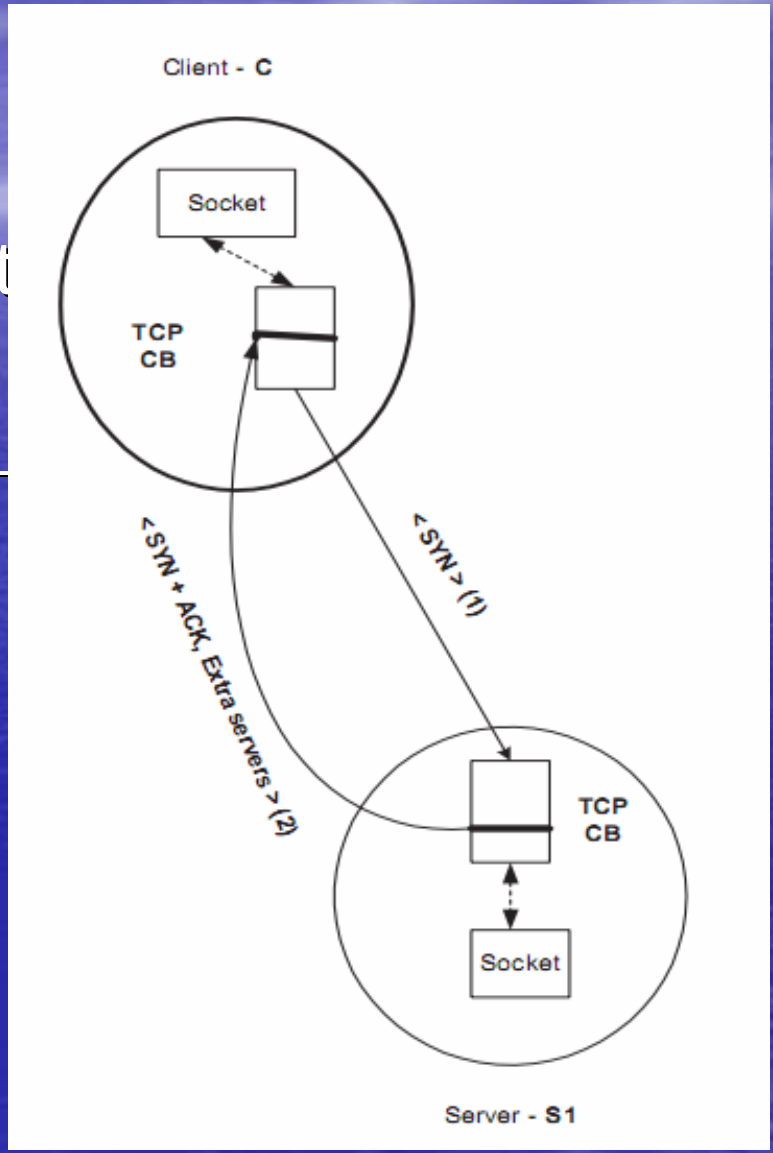
```
typedef struct
{
  u_int32_t server_ip_address;      /* the server's ip address that it
                                     * uses to interact with the client */
  u_int32_t control_ip_address; /* Control connection's IP address */
}ctrl_ip_mapping;


#define MAX_CONTROL_IPS MAX_IP_ADDRS
struct ctrl_mappings_info
{
  ctrl_ip_mapping ctrl_mappings[MAX_CONTROL_IPS\];
                                     /* the mappings */
  char num_maps; /* the number of mappings */
};
```

Implementation

- Read_log_mb
  - A chain of mbufs that stores all complete mbufs read
- So_read_log_len
  - Read_log_mb mbuf chain size
- So_read_log_offset
  - First mbuf present in the read buffer

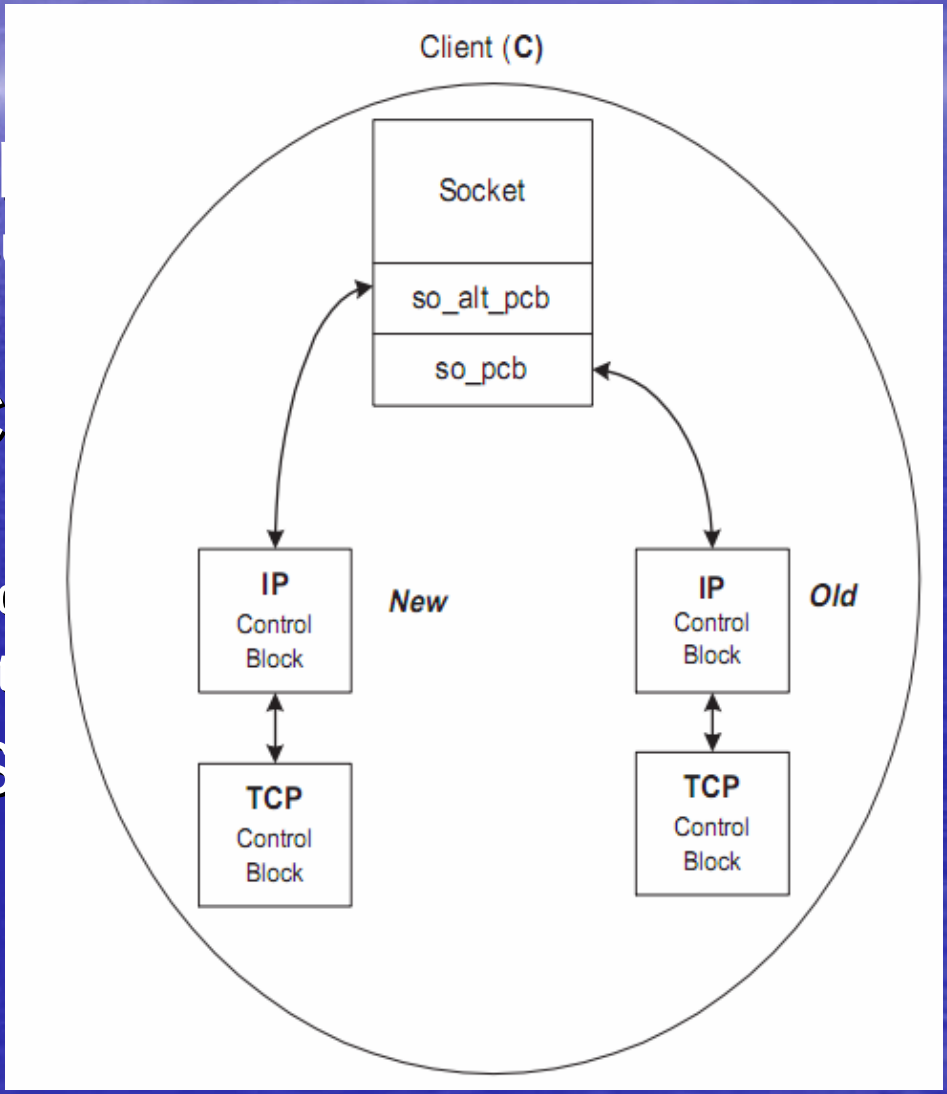
Implementation

- Additional servers informat
  option
  - Client copy the info. to self–extra_



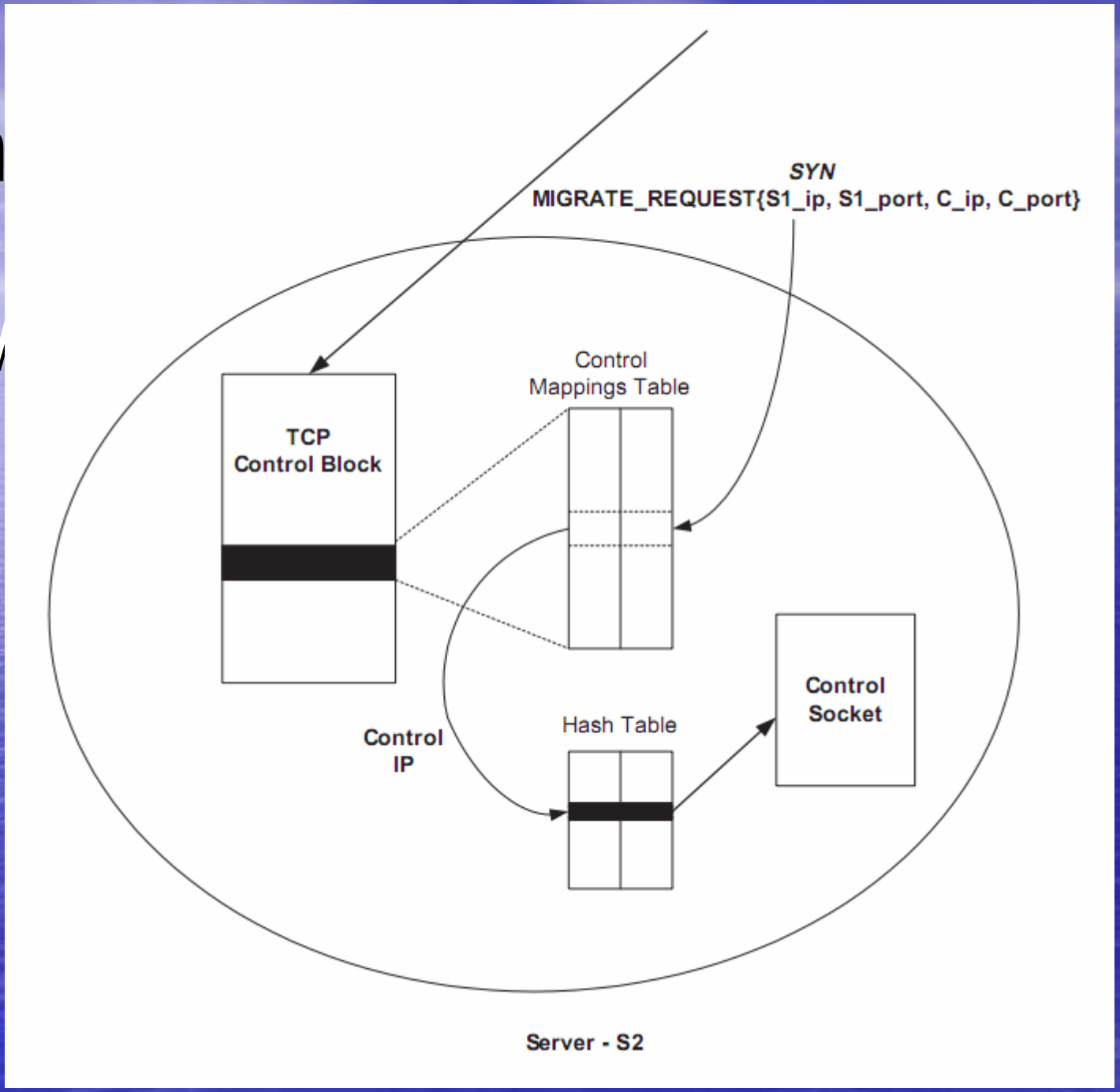
Implementation

- when two flag is setu
  - Server has so_migration_s
  - Client has so_can_migrate
- Client send SYN + TC
  option to S2
  - The connection ID of C an
  - Rcv_nxt in original connect
- Client socket enter SS
  - Don't send any data to S1



Im

- W


SYN
MIGRATE_REQUEST{S1_ip, S1_port, C_ip, C_port}
Control
Mappings Table
TCP
Control Block
Control
IP
Hash Table
Control
Socket
Server - S2

Implementation

- State request
  - 4-tuple connection ID that is migrating
  - 4-tuple connection ID for new server
  - Rcv_nxt from client to s2
- State_reject
  - An integer represent the reason
    - INVALID STATE

Implementation

- State reply
  - Application-level state lengt

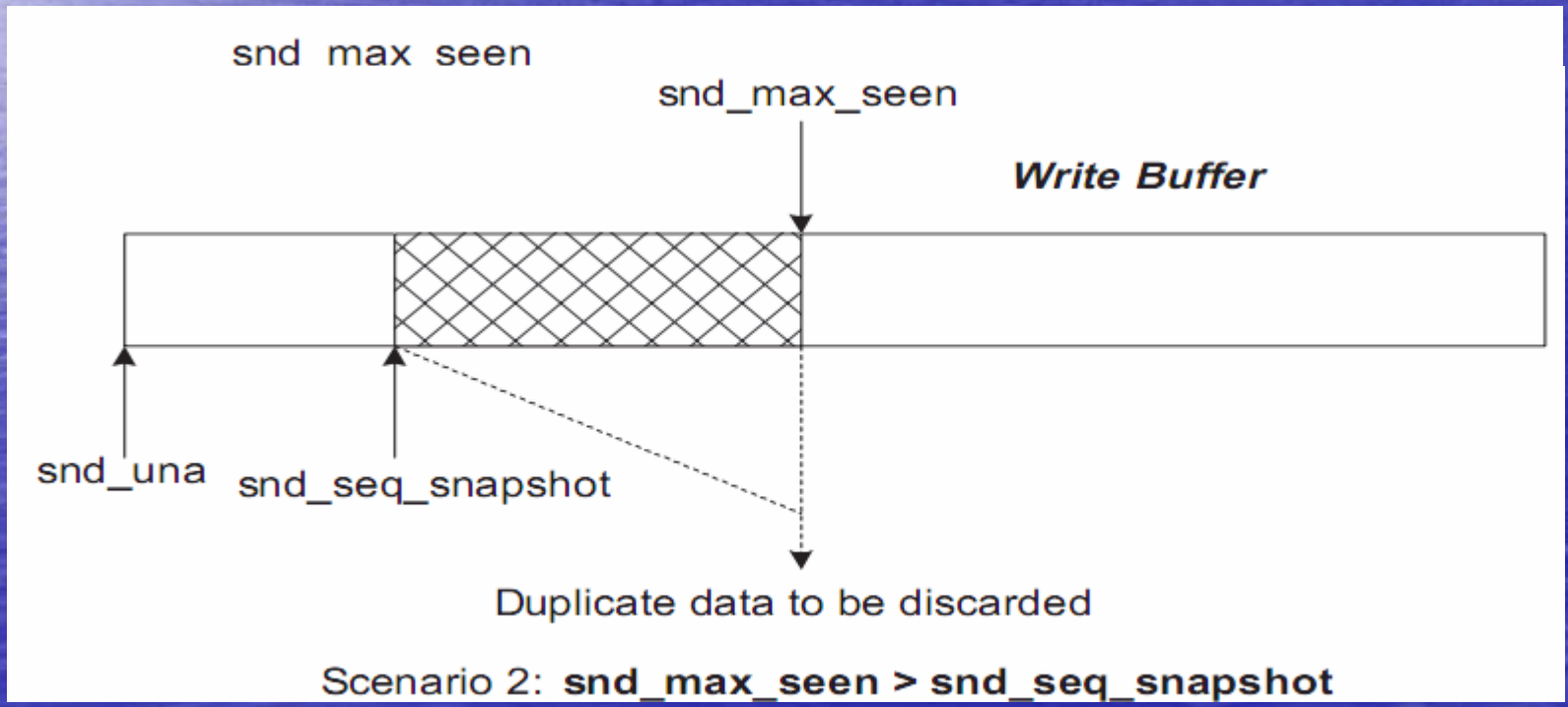
```
struct socket_snapshot
{
        short so_state;
        short so_options;
        short so_linger;
};
```

```
struct tcp_conn_state
{
        struct mbuf
        int
        tcp_seq
        tcp_seq
        struct socket_snapshot
        struct tcp_snapshot
} t_conn_state
```

```
struct tcp_snapshot
{
        int             t_state;
        short           t_state_transitions;
        u_int32_t       t_flags;
        u_int32_t       t_maxseg;
};
```

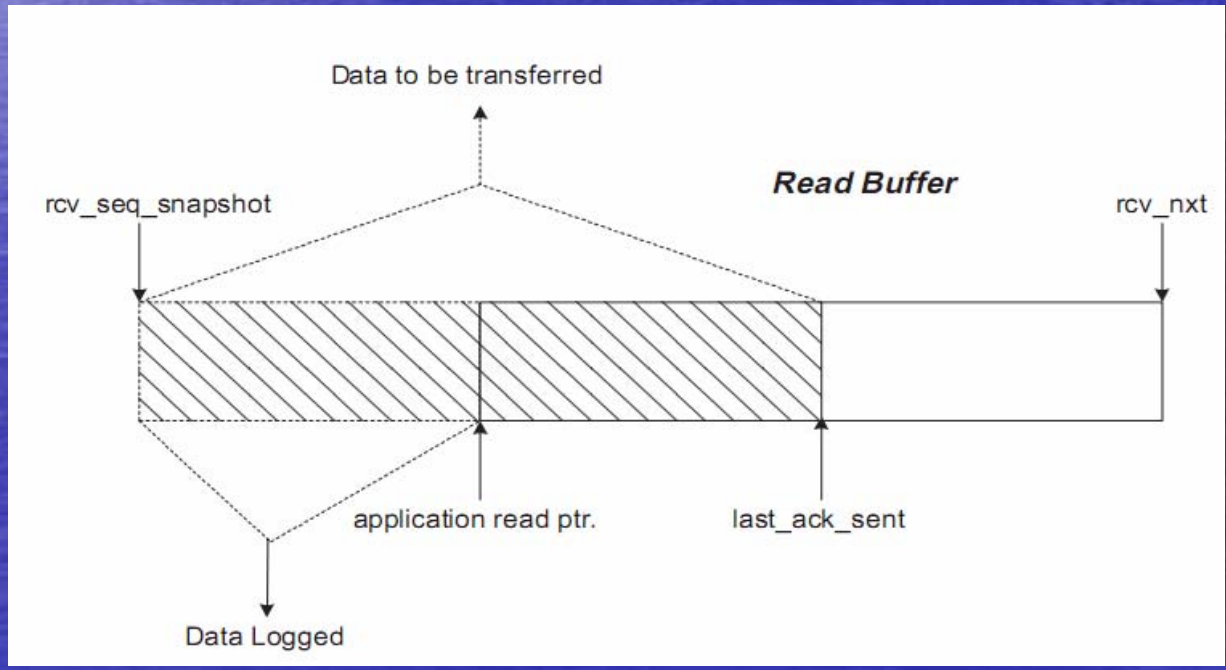
Implementation

- Write buffer state length
  - Snd_max_seen=max(rcv_nxt,syn_una)



Implementation

- Recvive buffer state length
  - Rcv_state_length=(last_ack_sent – rcv_seq_snapshot)



Implementation

- State reply packet header
  - A tag indicating the packet is state reply
  - The application state length
  - Write buffer state length
  - Read buffer length
  - Rcv_seq_snapshot and last_ack_sent
  - Snd_bytes_discard
  - Snd_max_seen
  - T_snapshot
  - So_snapshot

Implementation

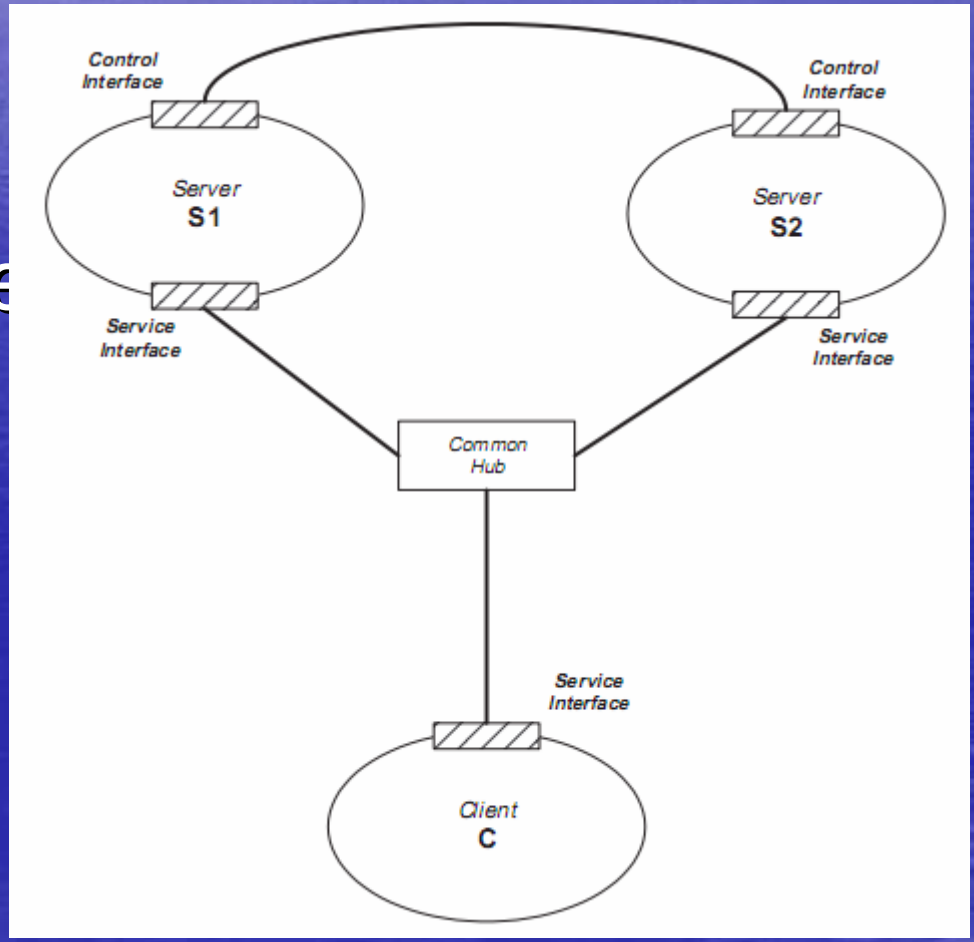
- Header and logged data are organized as a chain of mbufs is sent over the control connection

Implementation

- S2 send ACK
  - S1 enter passive close state
  - When S1 receive implicit ACK , closed
- S2 send SYN, rcv_nxt(S1) to client
  - Client terminate the SS_ISMIGRATING state
  - Client resume the connection

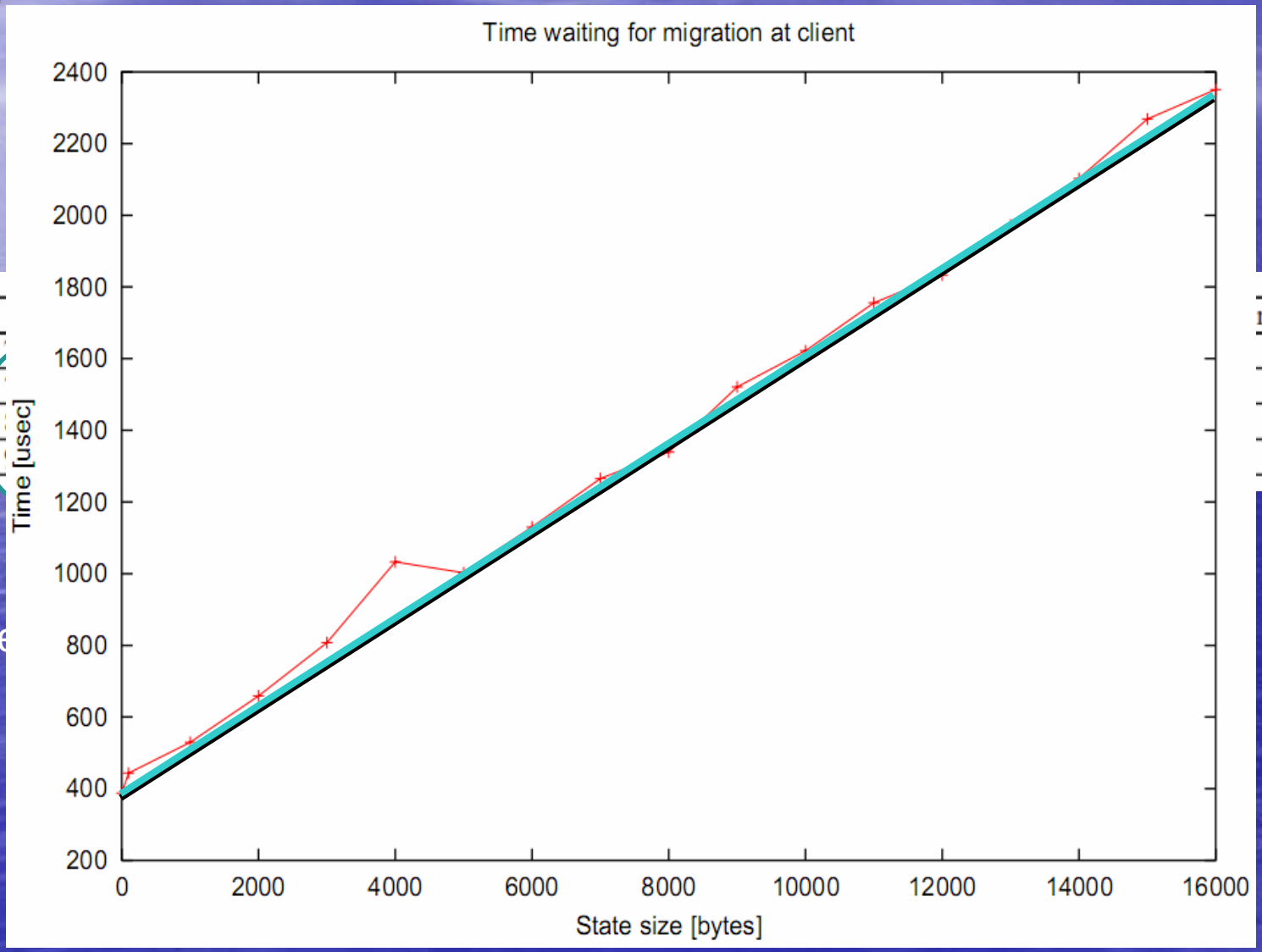
Experiment

- Celeron 450MHz
- 256MB RAM
- 100Mbps Etherne

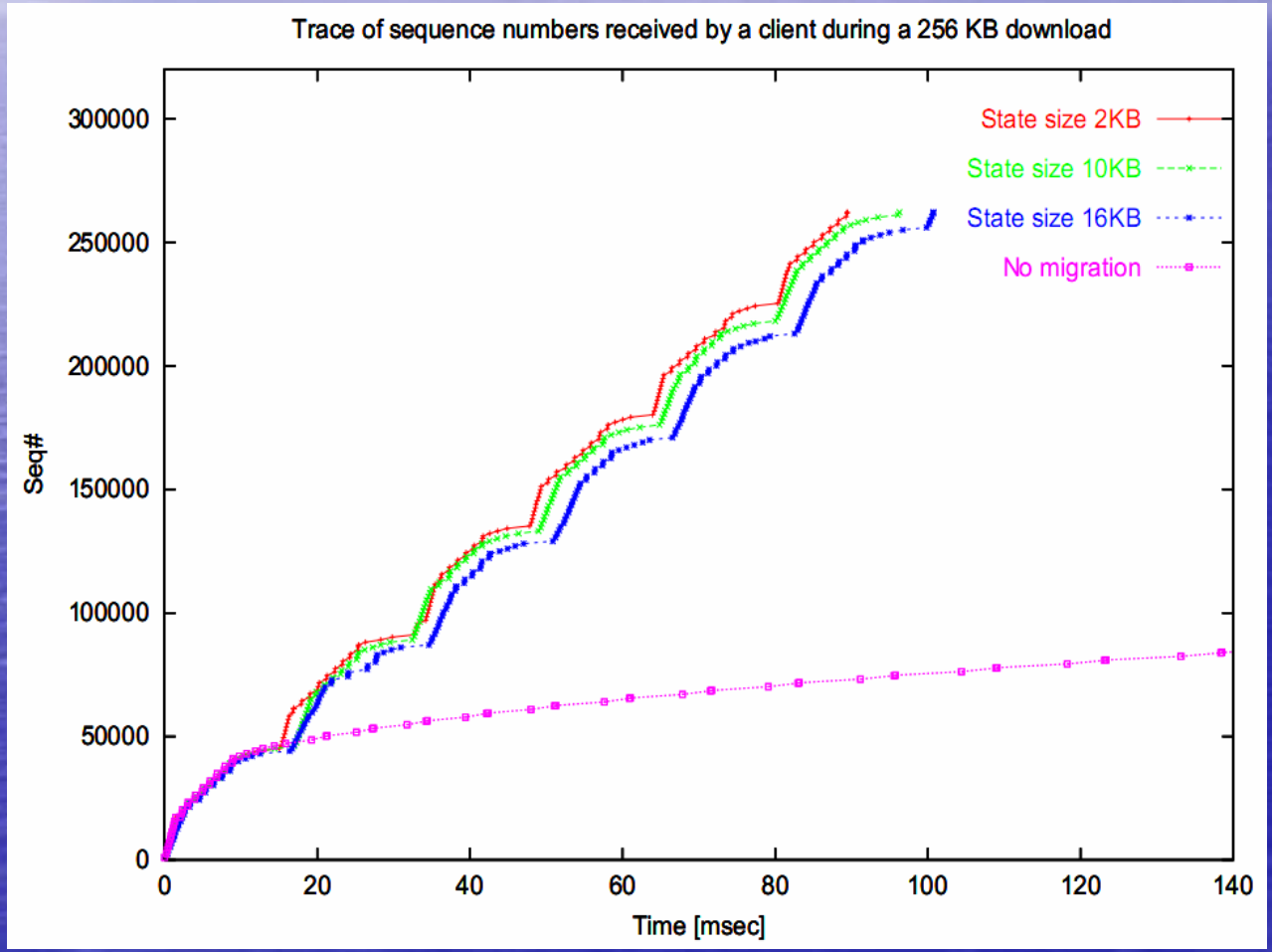


Experiment

State size	
0	
5K	
10K	



Experiment



When server send data , client do migration every time

Conclusion

- MTCP allows dynamic connection migration at any point of time during the session
- Basic assumption : per-connection state is represented as a chunk of memory
  - Recursive migration