CSE 123 Discussion

Oct 16th 2018

Today’s Agenda

- Usage of CRC in Project 1
- Buffer Window in Sender/Receiver
- Timeout handling
- Fragmentation for long messages
- Homework Discussion

Before we start …

- ll_get_length, ll_append_node, ll_pop_node, ll_destroy_node
- These functions are in util.c and they are given to you to help you interact with the data structure linked list which the project largely depends on.
- input_framelist_head in both sender/receiver is the linkedList for you to get the frame. After you pop a node from the list, you will get a node. (LLnode_t)
- void * value in LLnode_t is the char * you want to work with.
- input_cmdlist_head in sender is where you get input commands. Cmd_t contains information you need.

CRC – Cyclic Remainder Check

- Table 2.3 in P&D page.102 to check the common CRC Polynomials.
- Create crc.h and crc.c. Or put them in the utils.h and utils.c
- You will need to implement:
  - char get_bit (char byte, int pos); // return a char with a value of 0 or 1 depending on whether the bit in the pos is 0 or 1
  - char crc8 (char* array, int byte_len); // or crc16, crc32 etc.
  - void append_crc (char* array, int array_len); // append crc remainder to the char array
  - int is_corrupted (char* array, int array_len); // return 1 if a frame is corrupted, otherwise return 0

Steps For Using CRC

- Sender
  - When constructing the frame (header and payload), set CRC field to zero.
  - Convert frame to char array by calling convert_frame_to_char()
  - Append CRC into your frame by calling append_crc(). Append_crc() will call crc8() to compute the CRC remainder.
  - Send the resulting char array from frame + the crc remainder by appending to outgoing_frames_head_ptr.
- Receiver
  - Check for corruption when you receive a packet by calling is_corrupted() function.
  - If not corrupted, convert the char array to the frame. Otherwise, drop it.

Bitwise Operators in C

- X AND Y -> X & Y
- X OR Y -> X | Y
- X XOR Y -> X ^ Y
- NOT X -> ~X
- Shift X by Y bits to the left -> X << Y
- Shift X by Y bits to the right -> X >> Y
- Note: Bitwise operators have lower priority than comparison operators.
  - Exp. If we want to check whether the most significant bit of a byte is 0 or not.
  - If (byte & 0x80 == 0) Wrong !!
  - If ((byte & 0x80) == 0) Correct

CRC 8 Computation

```
// Function returns the remainder from a CRC calculation on a char* array of length byte_len
char crc8(char* array, int array_len){
    // The most significant bit of the polynomial can be discarded in the computation, because:
    // (1) it is always 1
    // (2) it aligns with the next '1' of the dividend; the XOR result for this bit is always 0
    char poly =0x07;  //00000111
    char crc = array[0];
    int i, j;
    for(i = 1; i < array_len; i++){
        char next_byte = ith byte of array;
        for(j = 7; j >= 0; j--){  // Start at most significant bit of next byte and work our way down
            if(crc's most significant bit is 0){
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j); // get_bit(next_byte, j) returns the a bit in position j from next_byte
            } else{ // crc's most significant bit is 1
                left shift crc by 1;
                crc = crc OR get_bit(next_byte, j);
                crc = crc XOR poly;
            }
        }
    }
    return crc;
}
```

Sender Buffer/Window

- Sender need to maintain window(buffer) while sending packets out
- The window is like this:
  - struct sendQ_slot {
    struct timeval* timeout; // event associate with send timeout
    Frame frame;
    } sendQ[SWS];
- Timeout is of type struct timeval (declared in sys/time.h)

Receiver Buffer/Window

- Similarly, it is better for receiver to maintain a window too.
- Example:

```
struct recvQ_slot {
    struct Frame_t* frame
} recvQ[RWS]
```

- Why don’t we need a timeout here ?

struct timeval in C

- struct timeval {
  time_t tv_sec; //seconds
  suseconds_ tv_usec; //microseconds
  }
- To calculate the timeout, get the current time and add 0.1s to it.

```
void calculate_timeout(struct  timeval* timeout) {
   gettimeofday(timeout,NULL);
   timeout->tv_usec  += 100000;
   if (timeout->tv_usec  >= 1000000) {
      timeout->tv_usec  -= 1000000;
      timeout->tv_sec  += 1;
   }
}
```

- Take a look at time_val_usecdiff() in util.c

Fragmentation

- One way to implement fragmentation is to use the sender's input command list.
- sender->input_cmd_list_head is a doubly linked list whose nodes are of type struct cmd (in common.h)
- Before popping off the head node from sender->input_cmd_list_head check if the message > your_payload_size.
- If yes, then split the head node into multiple nodes where each node contains part of the message, length <= your_payload_size
- You can also maintain your own data structure in sender_t to handle this.

Skeleton Code for Fragmentation

```
void ll_split_head_if_necessary(LLnode ** head_ptr, size_t cut_size){
    //TODO: check if head is NULL
    LLnode* head = *head_ptr;
   Cmd* head_cmd = (Cmd*) head -> value;
   char* msg = head_cmd -> message;
   if(strlen(msg) < cut_size) {
    return;
    }
   else{
    size_t i;
    LLnode* curr, * next;
    Cmd* next_cmd;
    curr = head;
    for(i = cut_size; i < strlen(msg); i += cut_size) {
                  // TODO: malloc  next, next_cmd
                char* cmd_msg = (char*) malloc((cut_size + 1) * sizeof(char)); // One extra byte for NULL character
               memset(cmd_msg, 0, (cut_size + 1) * sizeof(char));
                strncpy(cmd_msg, msg + i, cut_size);
               // TODO: flll the next_mcd
              //  TODO: fill the next_nose and add it to the linked list
    }
    msg[cut_size] = '\0';
   }
}
```

The End

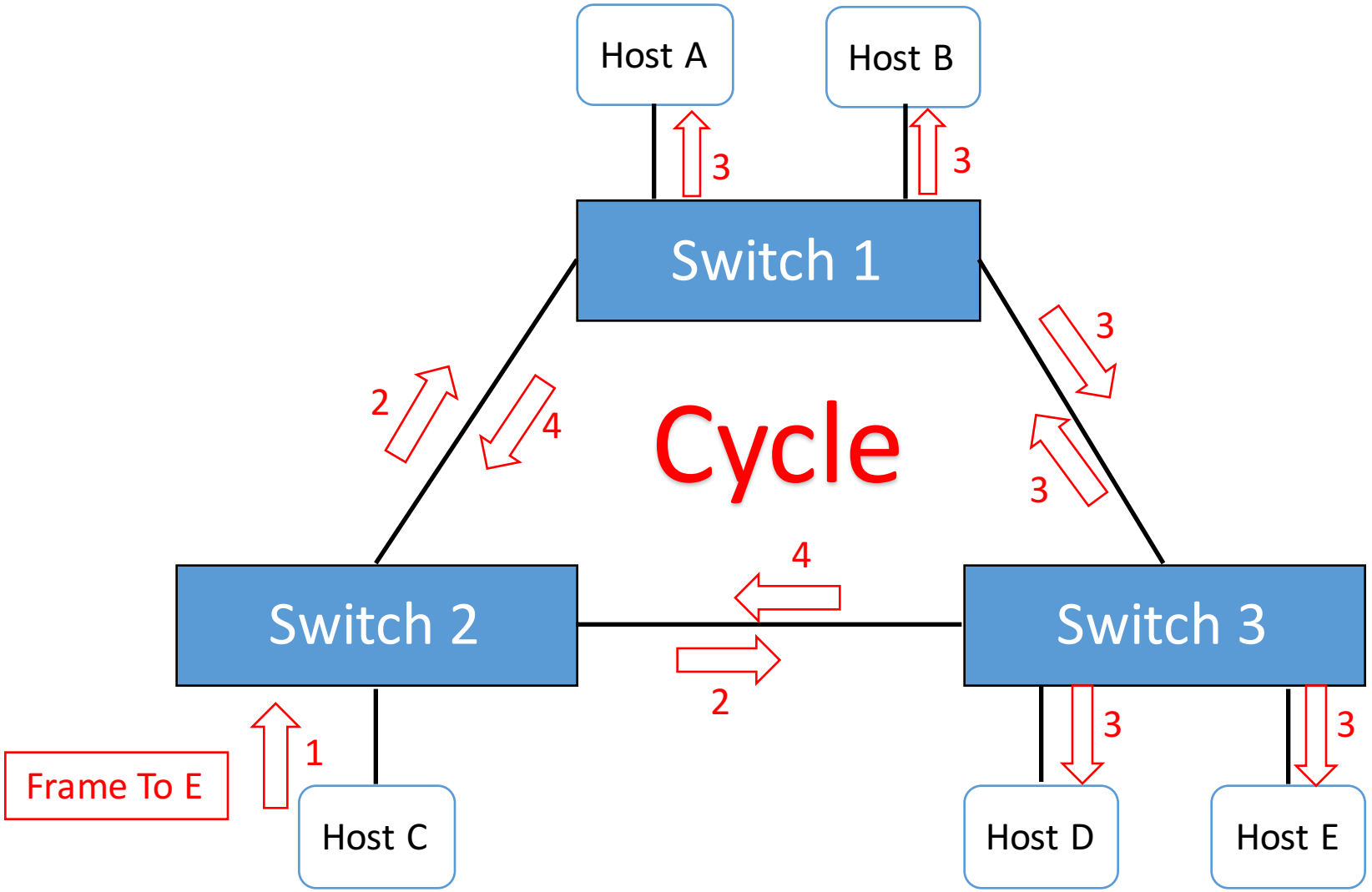
Extra Office Hour on Thursday, Oct 18th from 2:00 to 3:00pm in CSE basement.

Try to find me in the hallway first.

We won’t have a room since they are all reserved already.

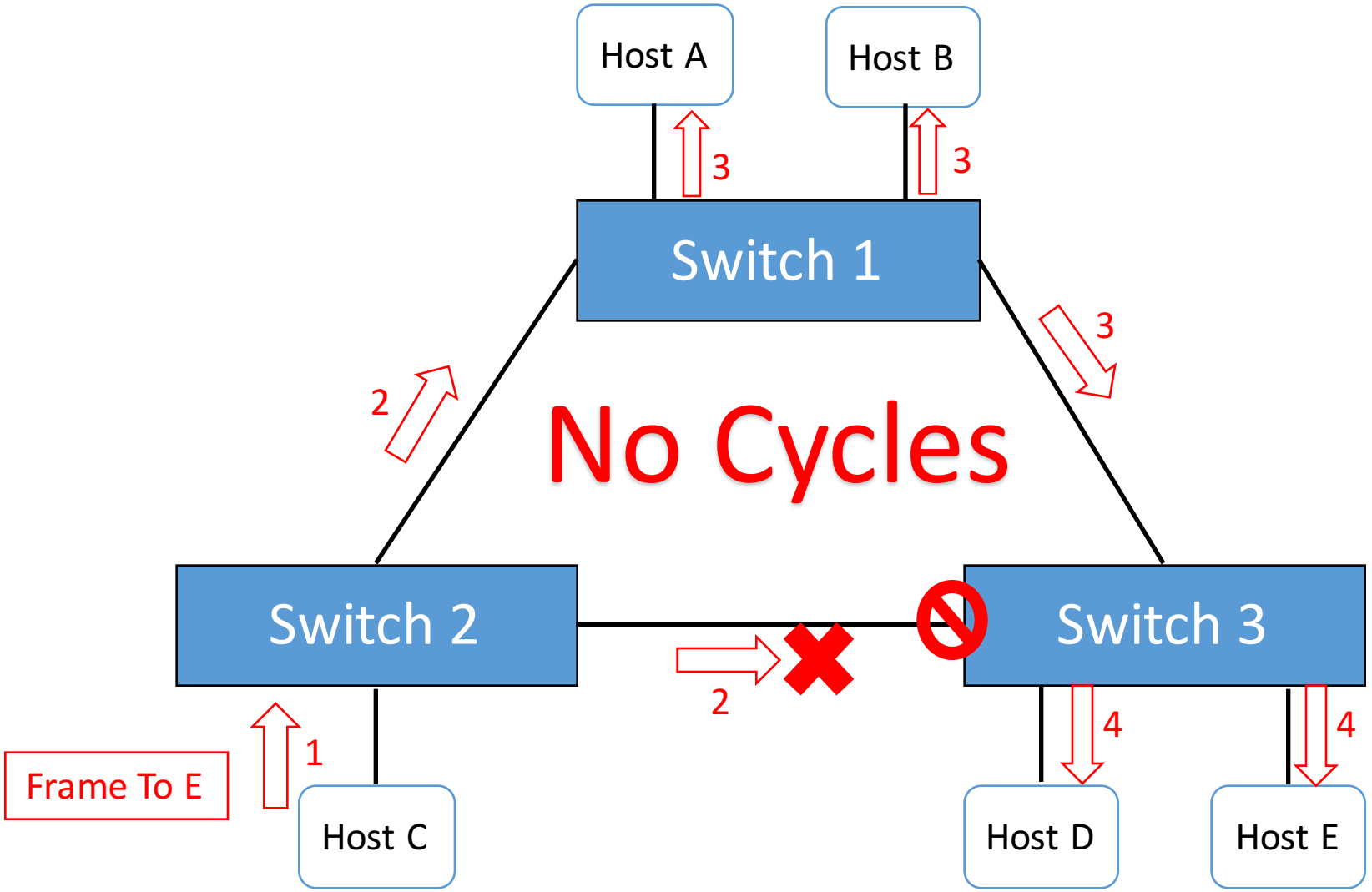
Thank you!

Spanning Tree Protocol


Host A
Host B
3
3
Switch 1
2
4
Cycle
3
3
4
Switch 2
Switch 3
2
1
3
3
Frame To E
Host C
Host D
Host E

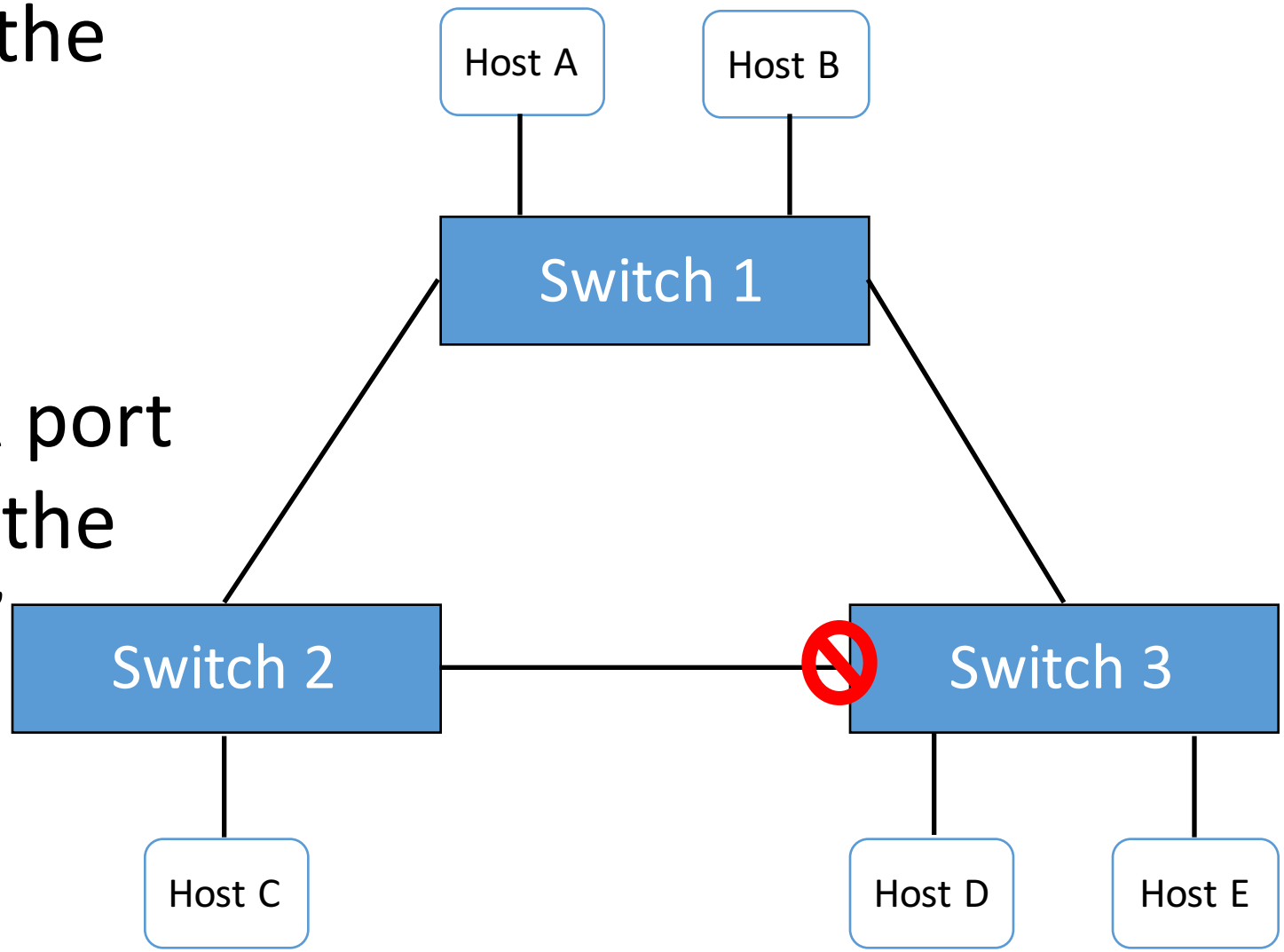
Spanning Tree Protocol

- 1. Detect loops in the network topology
- 2. Deactivate some ports to break loops
- 3. recalculate spanning tree instances when topology changes


Host A
Host B
3
3
Switch 1
3
2
No Cycles
Switch 2
Switch 3
2
1
4
4
Frame To E
Host C
Host D
Host E

Why do we need loops in the network topology?

What is the meaning of “A port is deactivated/blocked by the spanning tree algorithm?”



Spanning Tree Algorithms

- 1. Each node is a computer, so the problem is Not equivalent to “Find a cycle in a linked list”.

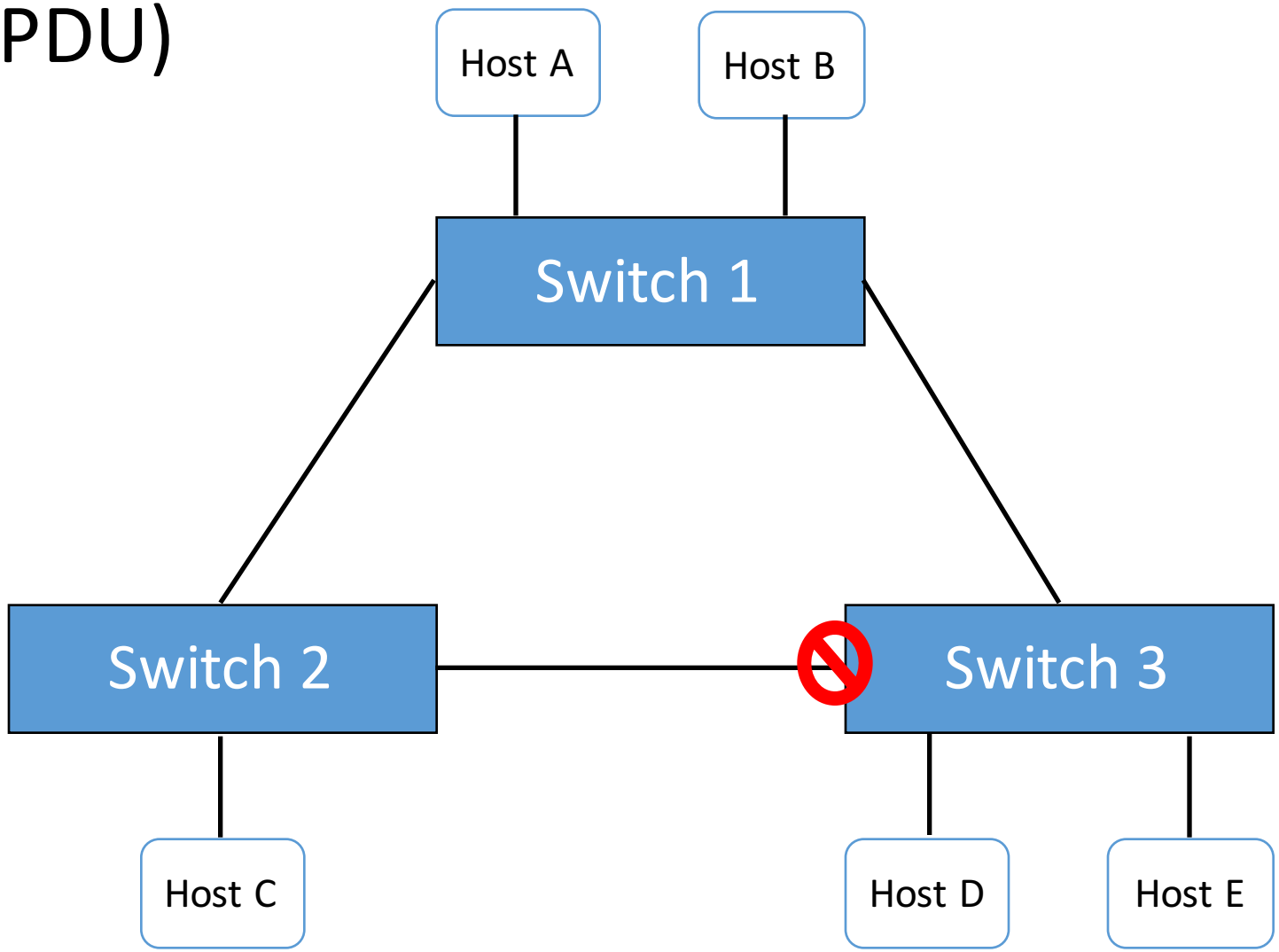
- 2. Each node only knows about itself and its neighbors.

Spanning Tree message (BPDU)

BPDU are Frames with multicast destination MAC Address 01:80:C2:00:00:00

A BPDU contains the following information:

1. Who is the Root Bridge (bridge ID)
2. What is my distance to the root bridge
3. What is my bridge ID (priority, MAC address)
4. What is my port ID (tie breaker)
5. Message Age
6. Max Age
7. Hello Time
8. Forward Delay



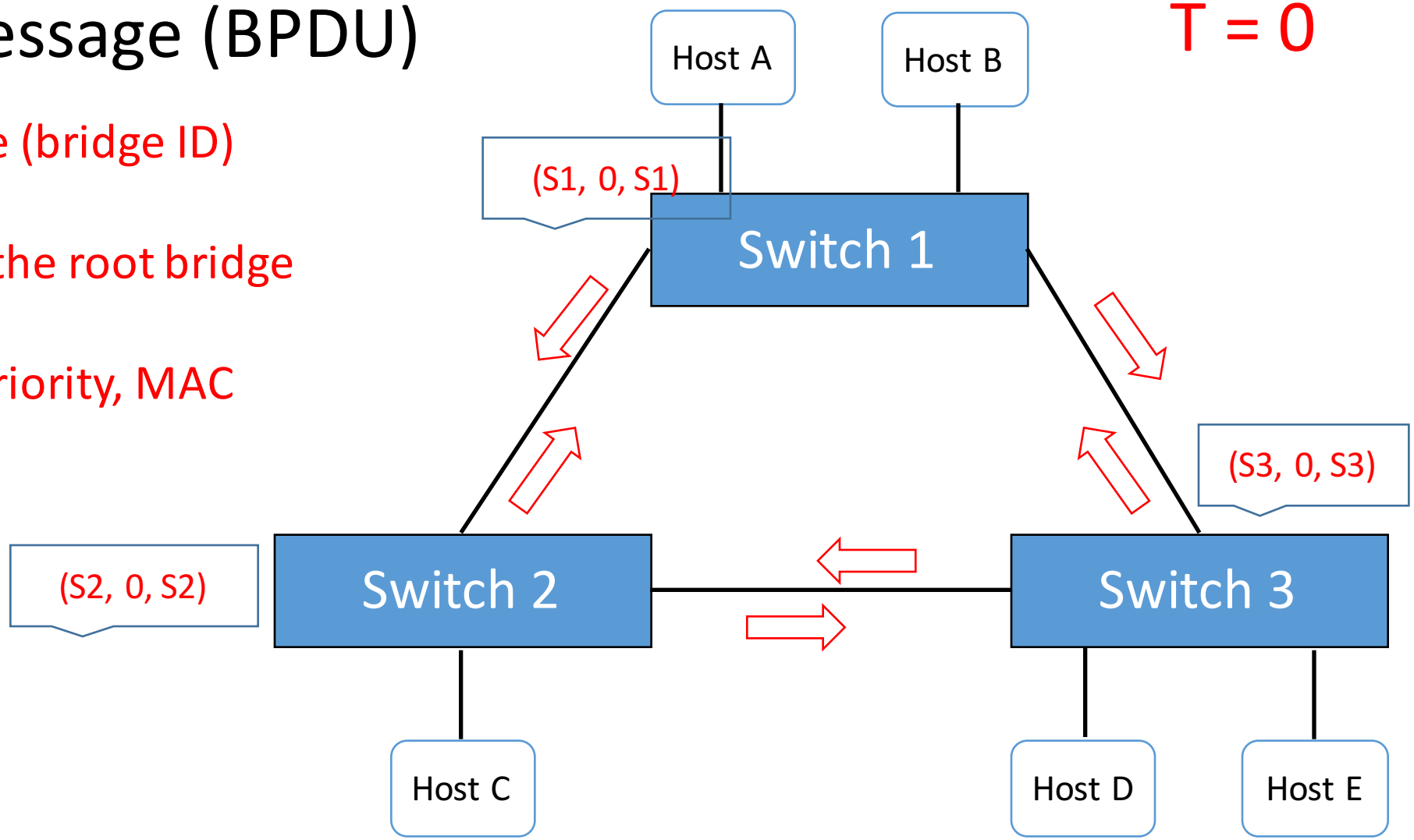
Spanning Tree message (BPDU)

T = 0

1. Who is the Root Bridge (bridge ID)

2. What is my distance to the root bridge

3. What is my bridge ID (priority, MAC address)



Root bridge has the smallest bridge ID

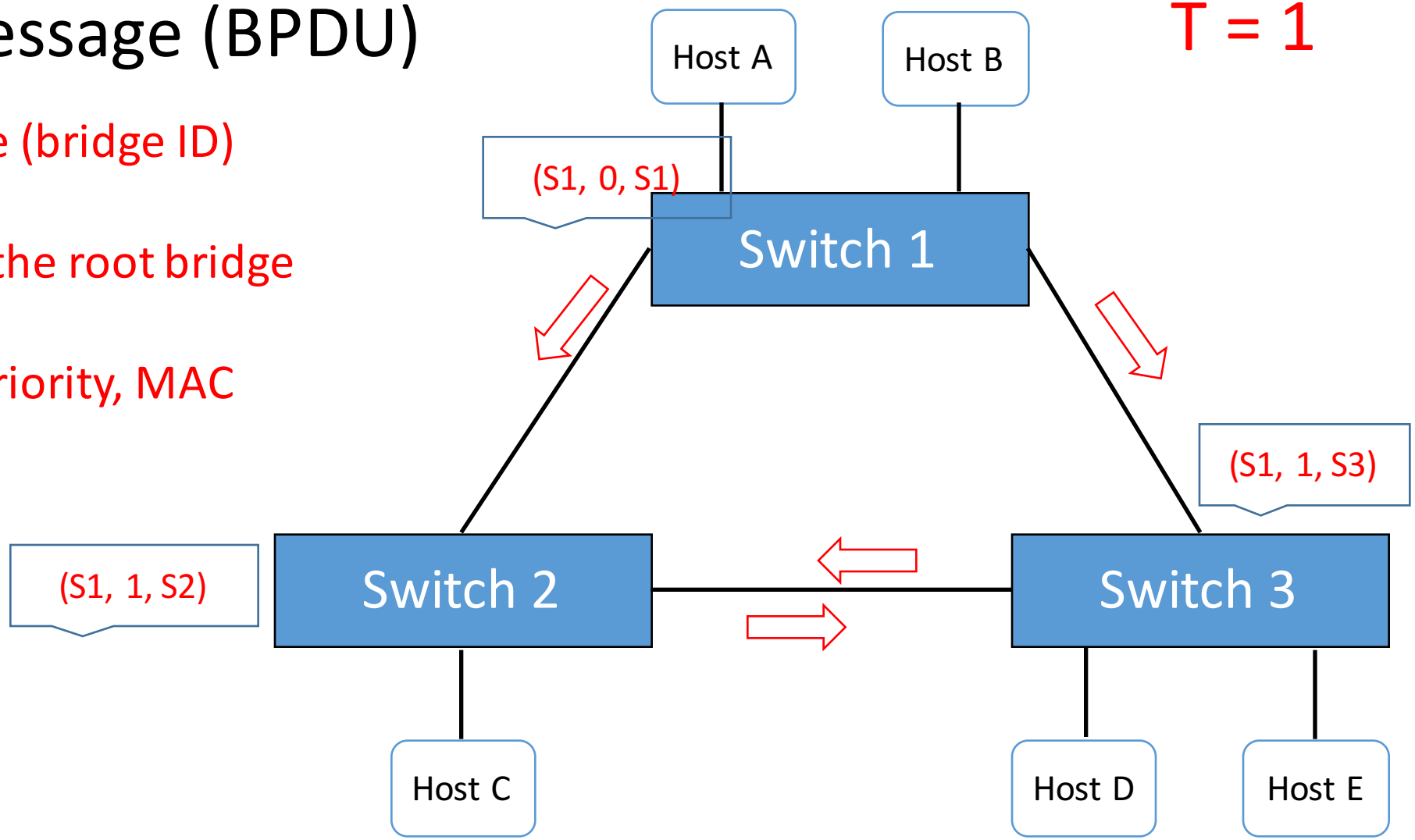
Spanning Tree message (BPDU)

T = 1

1. Who is the Root Bridge (bridge ID)

2. What is my distance to the root bridge

3. What is my bridge ID (priority, MAC address)



Root bridge has the smallest bridge ID

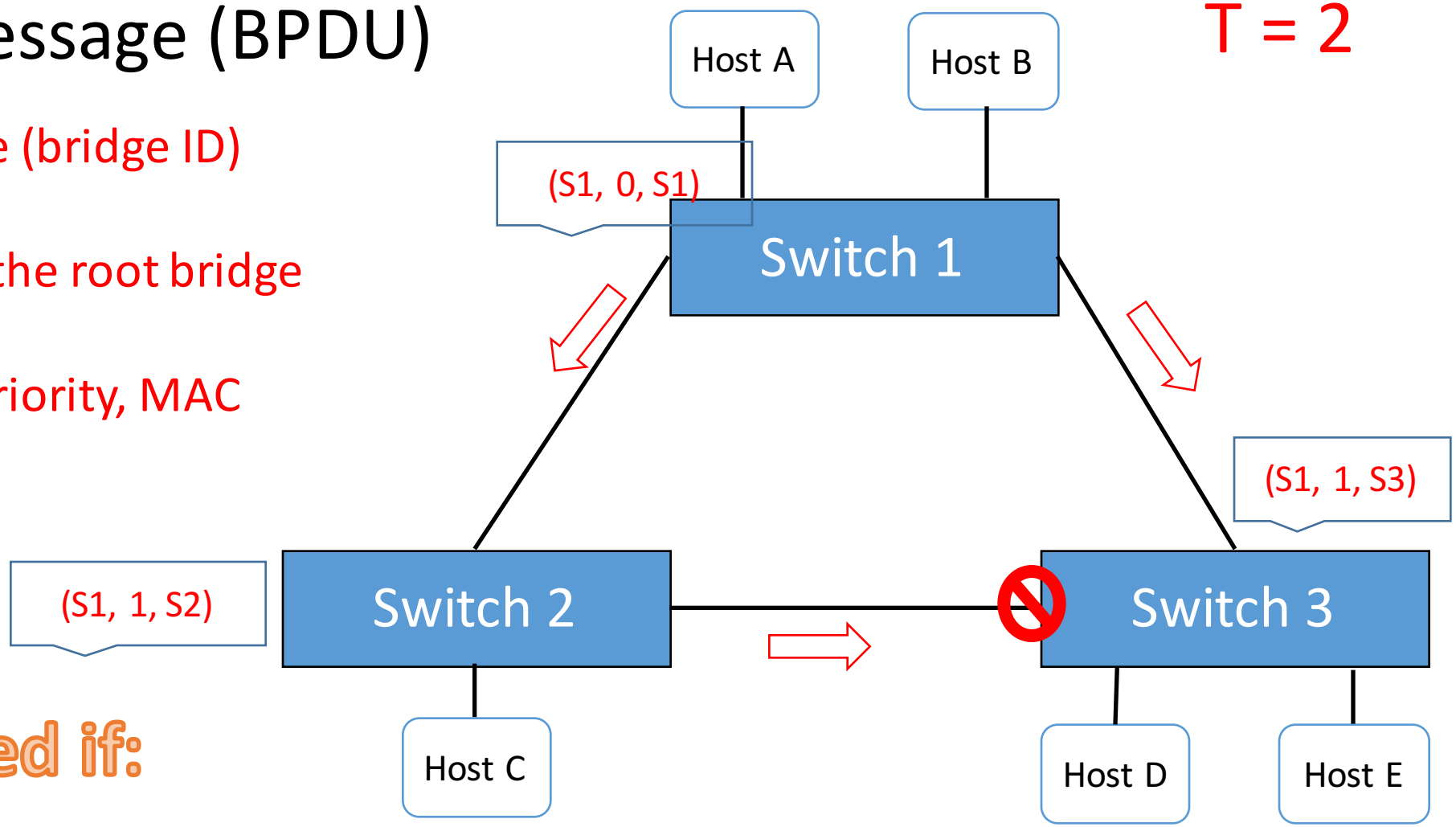
Spanning Tree message (BPDU)

T = 2

1. Who is the Root Bridge (bridge ID)

2. What is my distance to the root bridge

3. What is my bridge ID (priority, MAC address)



A Port is blocked if:

(1) It is not a root port, and (2) the BPDU message it intends to send is inferior to the one it receives

Please Pick up your Homework,

Thank you