Sequel Course Part 2

Developing TCP/IP Stack Part B

1. Interface Management & statistics
2. Dynamic L3 Route Calculation
   No More Manual installation of L3 routes
3. Making TCP/IP stack dynamic
   Dynamic ARP table Entries
4. Develop Logging Infra
   Packet Captures
5. Sample L2 Layer Application & Working with Timers
6. Programmable TCP/IP Stack

LIVE**

Pre-Requisite :

Must have completed Part A

Agenda

Developing TCP/IP Stack Part B

1. Interface Management & statistics
2. Dynamic L3 Route Calculation
   No More Manual installation of L3 routes
3. Making TCP/IP stack dynamic
   Dynamic ARP table Entries
4. Develop Logging Infra
   Packet Captures
5. Sample L2 Layer Application & Working with Timers
6. Programmable TCP/IP Stack

Pre-Requisite :

Must have completed Part A

- Fast Paced Course
- Do CLI dev self

TCP/IP Stack Part B End Product

Enhanced TCP/IP Stack Library

- Automatic Routing Table construction
- Timer management
- More User Control over Networking Device
- Logging Infrastructure
- Appln Development
- Programmable TCP/IP Stack
- Event Notification

Interface Management

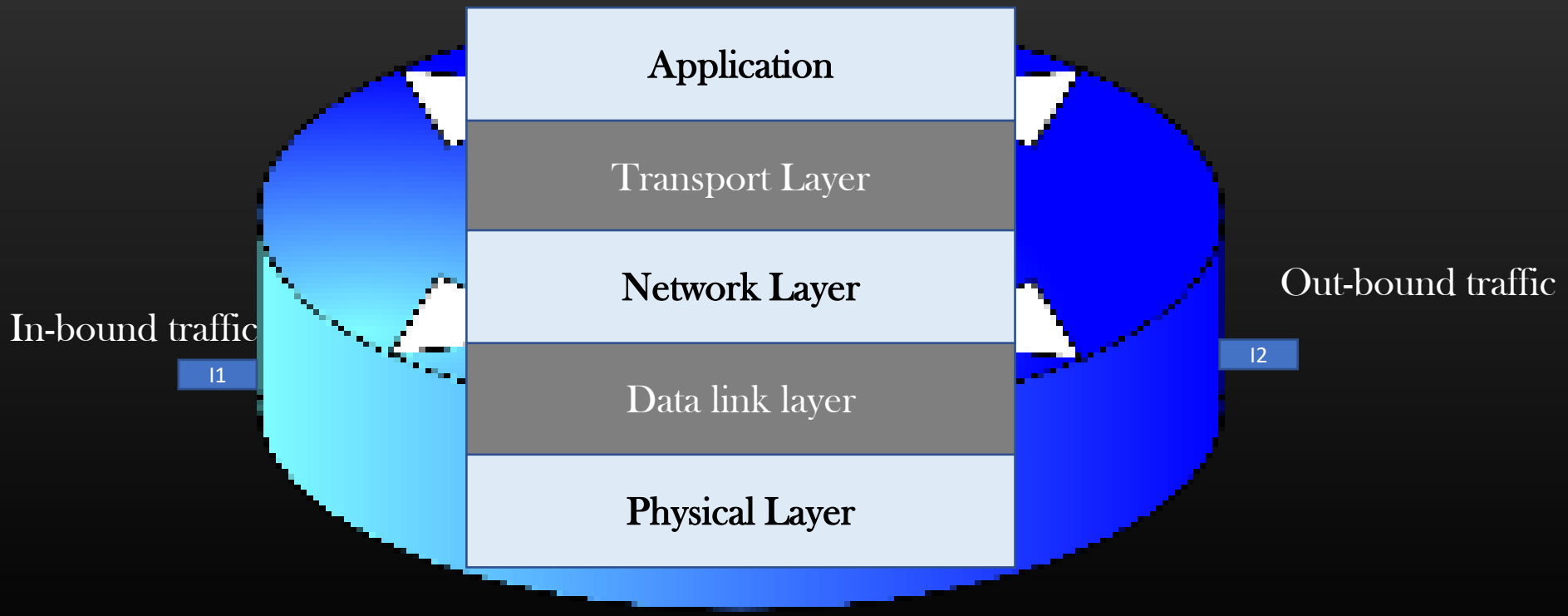
TCP/IP Stack Project -> Interface Management

- One of the most important aspect of Real-World Networking is Interface Management
- Networking is a game of Managing interfaces !
- Interface Management :
  - Gathering Tx & Rx Statistics per Device's interfaces
  - Interface Disable/Enable Or up/down
  - Creating/Updating Configuration per interfaces
  - Notifying interface status to interested parties (applications etc ..)
  - Network Applications need to react to Interface Config change
- In this Section, We shall begin adding a functionality to gather interface statistics
  - Rx Statistics – Number of packets recvd
  - Tx Statistics – Number of packets sent
- We shall enhance this feature in subsequent sections of the course, including
  - Dump packets sent/recvd on interface
  - Making the appln react to interface disable/enable OR Config Change
- This Course will push our TCP/IP Stack library developed in Part - A from being static towards dynamic .. .!
- Real World Demo . .

TCP/IP Stack Project -> Interface Management -> Interface Enable/Disable Problem Statement

- A Typical Routing Device allows the provision to admin to Enable or disable the interface of a Network Device

- Properties of Disabled Interface :
  - Do not send out any data/traffic (discards the outbound traffic just before it is placed on wire)
  - Discards all data / traffic as soon as recvd (at physical layer itself)



TCP/IP Stack Project -> Interface Management -> Interface Enable/Disable-> Code Changes

➢ Config CLI :

nwcli.c

config node <node-name> interface <if-name> <if-up-down>

```
tcp-ip-project> config-node-H1-interface-eth1 $ ?
Parse Success.
nxt leaf -> STRING                    |   <up | down>
```

Add the handler case in intf_config_handler(. . .)

Show interface status :

(enhance dump_intf_props(...))

```
tcp-ip-project> $ show topology node H1
Parse Success.
Topology Name = Dual Switch Topo

Node Name = H1, udp_port_no = 40000
        node flags : 0    lo addr : 122.1.1.1/32
Interface Name = eth1
       Nbr Node L2SW1, Local Node : H1, cost = 1
        If Status : UP
        IP Addr = 10.1.1.1/24    MAC : 82:08:c3:5b:00:0
```

Data Structure Changes :

```
typedef struct intf_nw_props_ {

    /*L1 Properties*/
    bool_t is_up;
  . . .
  . . .
```

Macro:

net.h

Introduce a macro which checks interface Up/Down Status

```
#define IF_IS_UP(intf_ptr)     \
/*provide implementation*/
```

TCP/IP Stack Project -> Interface Management -> Interface Enable/Disable

- Functional Changes

- Depending on the up/down status of the interface :

  Reject In-bound traffic in function :
  _pkt_receive(. . .)

  Reject out-bound traffic in function :
  send_pkt_out(. . .)

- Testing

  Use Ping to check if the changes have taken effect

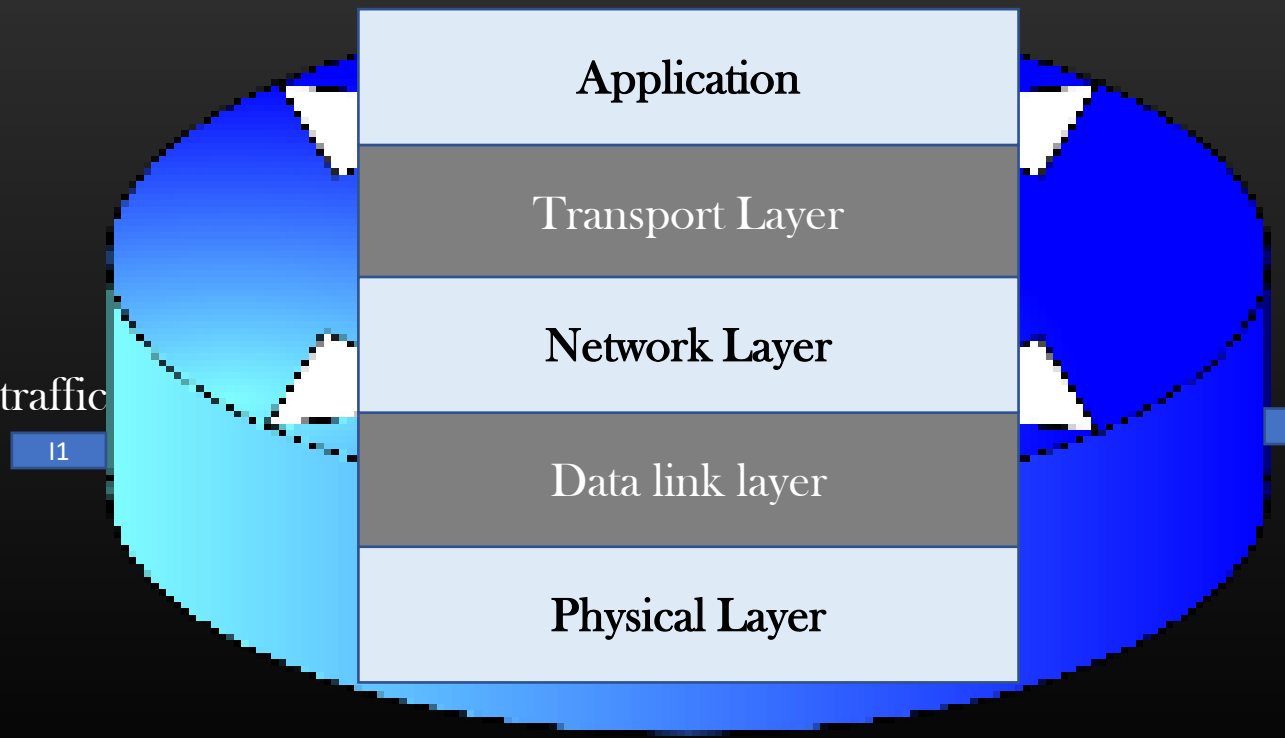
  The enable/disable of an interface applies to all types of interfaces – L2 or L3

TCP/IP Stack Project -> Interface Management -> Interface Send/Recv Statistics

- A Typical Routing Device allows the provision to admin to Check Send & Recv Statistics per interface of a Network Device

- Send Statistics :
  - Number of Network Packets sent out of the interface

- Recv Statistics
  - Number of Packets Recvd on an interface



TCP/IP Stack Project -> Interface Management -> Interface Send/Recv Statistics

- CLI :

  nwcli.c

  show node <node-name> interface statistics

  (must show send and recv counters of all interfaces of a device)

```
tcp-ip-project> $ show node H1 interface statistics
Parse Success.
eth1   ::  PktTx : 0, PktRx : 0
```

Backend handler fn : show_interface_handler(. . .) in nwcli.c

- Data Structure changes :

```
typedef struct intf_nw_props_ {
. . .
 /*Interface Statistics*/
    uint32_t pkt_recv;
    uint32_t pkt_sent;
. . .
. . .
} intf_nw_props_t;
```

- Functional Changes :

Increase the pkt_sent counter as soon as pkt is sent in fn send_pkt_out(. . .)

Increase the pkt_recv counter as soon as pkt is recvd in fn _pkt_receive(. . .)

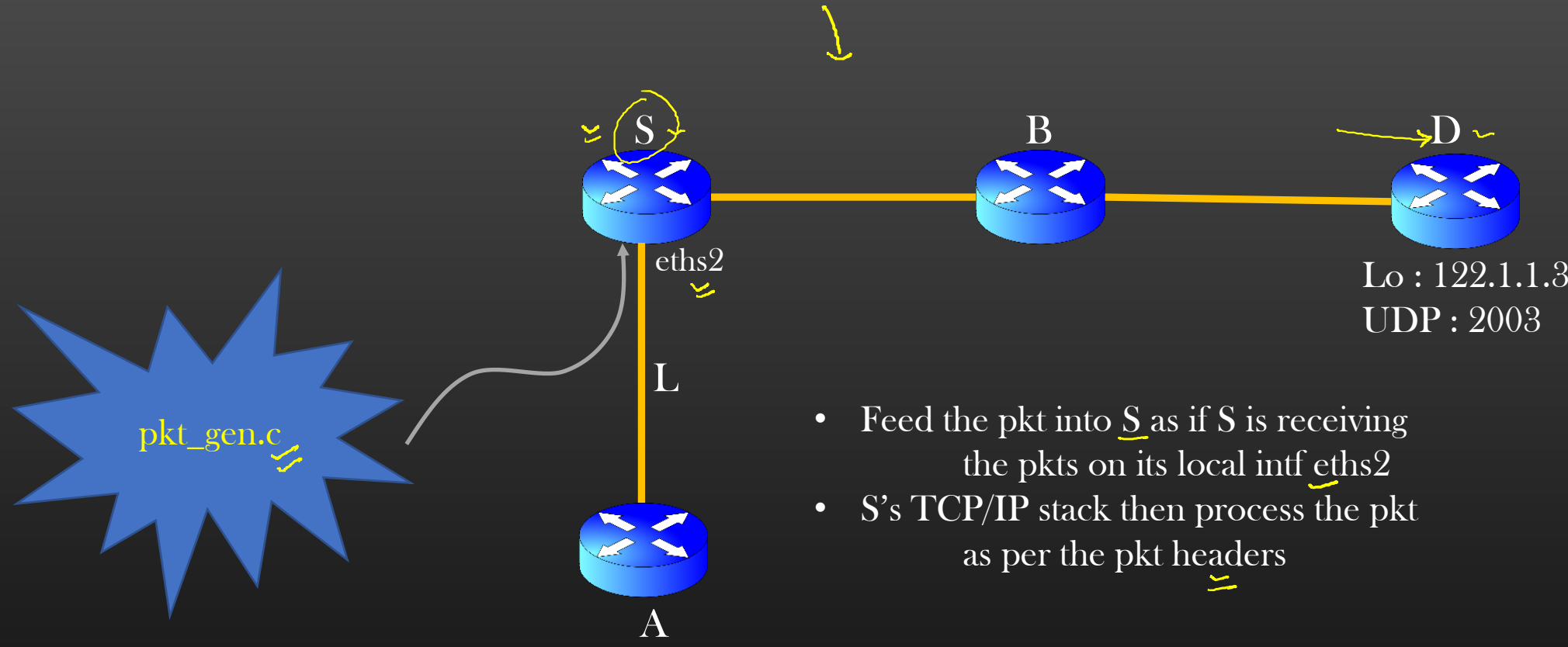
- Test

Use Ping

Interface in *down* state must not increment its send & recv counters

Packet Generator

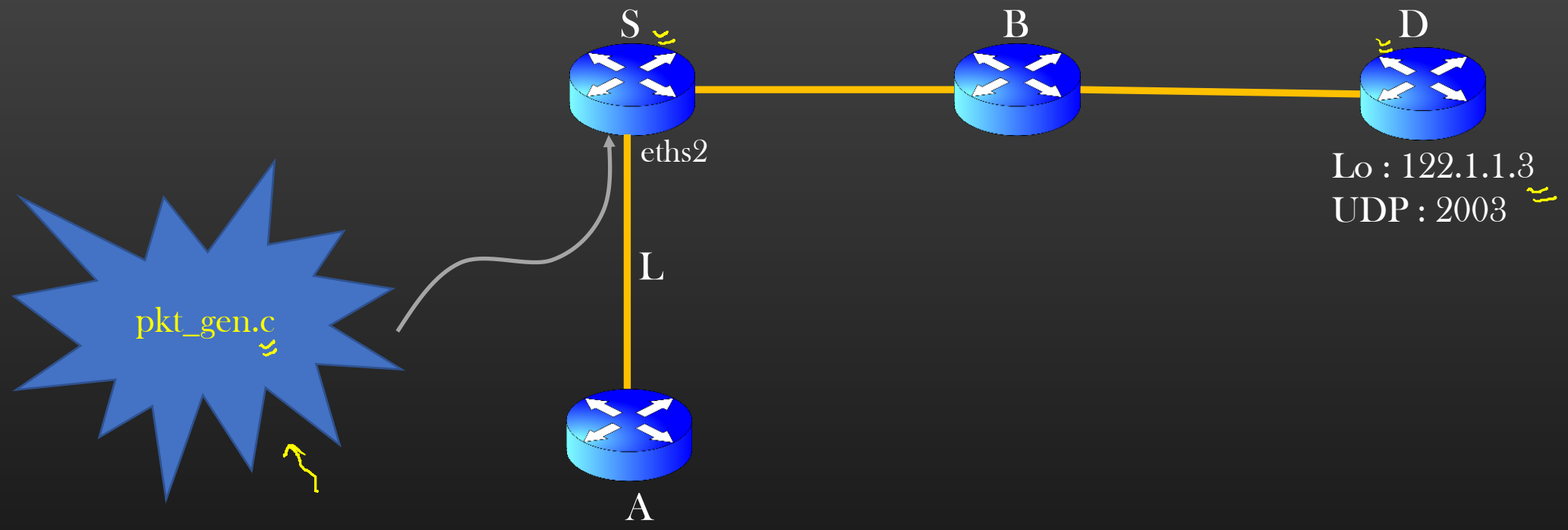
TCP/IP Stack Project -> Packet generator



- Feed the pkt into S as if S is receiving the pkts on its local intf eths2
- S's TCP/IP stack then process the pkt as per the pkt headers

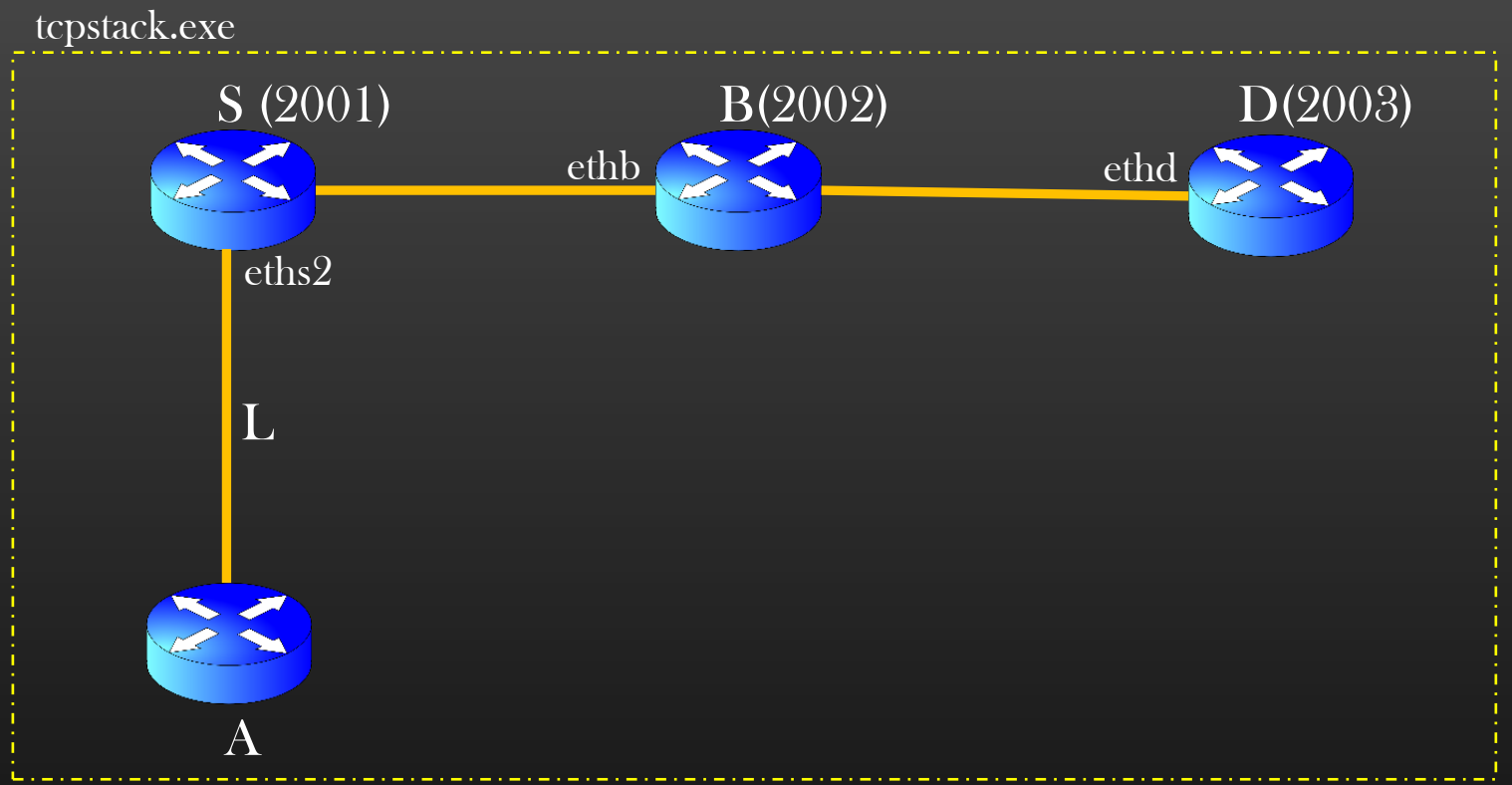
- So far, we have been using ping to test the new feature we implement in TCP/IP stack
- Instead, now we write a pkt generator - small separate program which would feed stream of pkts into our TCP/IP stack
- Terminology :
  - Pseudo TCP/IP Stack - Our TCP/IP stack Library
  - Actual TCP/IP Stack - The Actual TCP/IP Stack running on your local machine in kernel space

TCP/IP Stack Project -> Packet generator



- We want to generate stream of pkts : S sends Pseudo ICMP packet to Destination D with dest ip address : 122.1.1.3
- Equivalent to run node S ping 122.1.1.3
- pkt_gen.c is a simple UDP program which generate and sends UDP packets
- Revise how we implemented the packet exchange infrastructure in Part-A

The Bigger Picture (Recap)

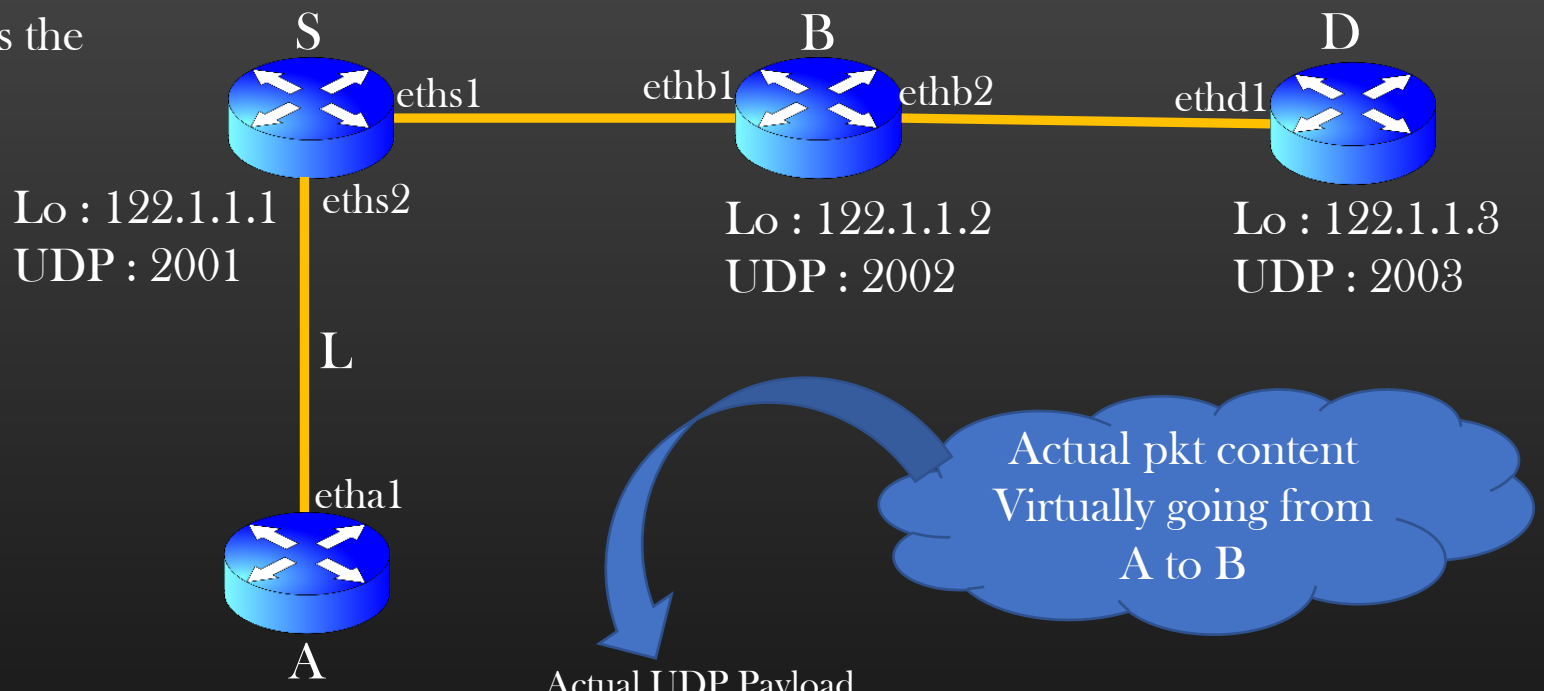


Actual TCP/IP Stack

We will be going to write our pkt generator application working on the same TCP/IP Stack pkt exchange design

TCP/IP Stack Project -> Actual and Pseudo Headers

- When you trigger run node S ping 122.1.1.3 Cmd on our TCP/IP stack prompt, how does the actual pkt look like which is pushed by send_pkt_out(..) down to the actual TCP/IP Stack running on your local machine



- sudo tcpdump -i lo -v -e udp

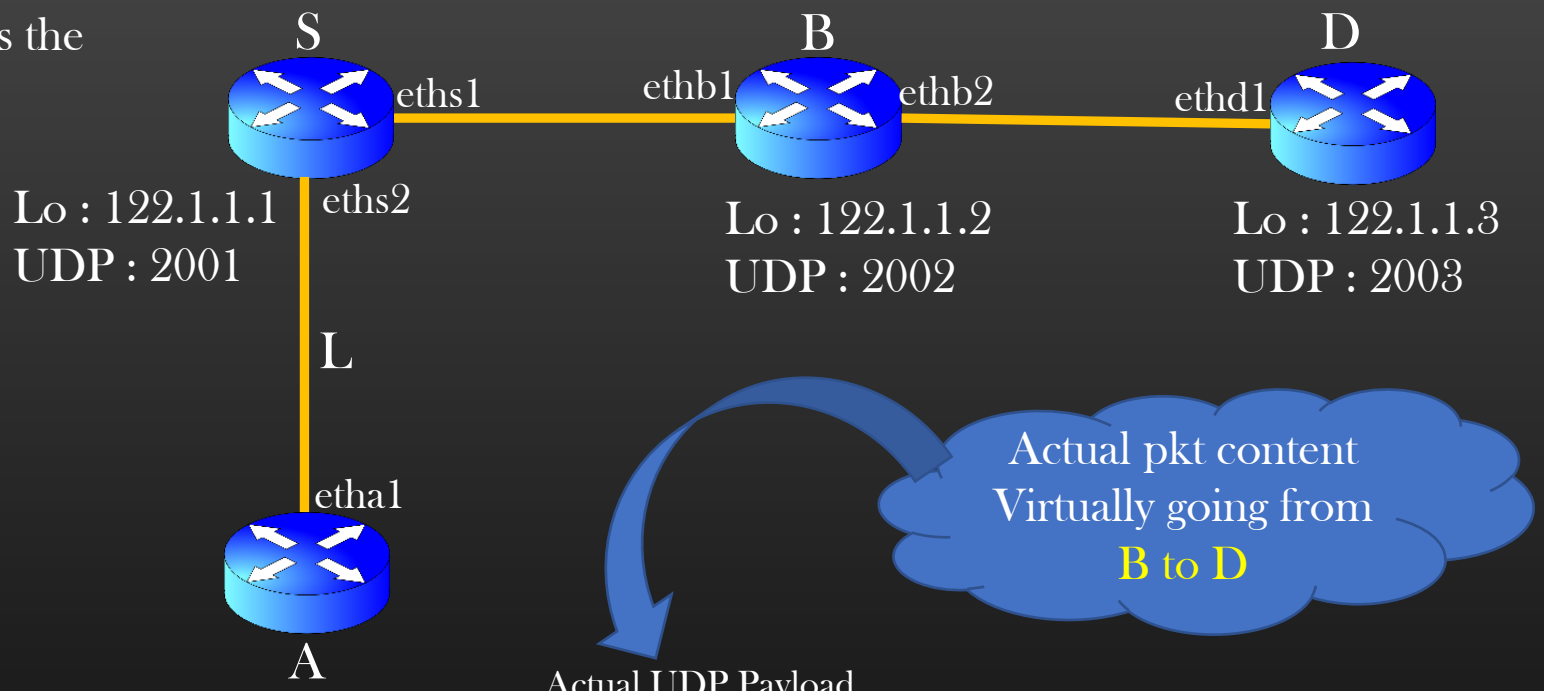
ETH Hdr	IP Hdr	UDP Hdr		Pseudo ETH Hdr	Pseudo IP Hdr		
Dst mac = 0 Src mac = 0 Type = ox0800	Proto = UDP ttl = 64 Src ip = 127.0.0.1 Dst ip = 127.0.0.1	Dst port = 2002 Src port = 2001	Aux Info : ethb1	Dst mac = mac(ethb1) Src mac = mac(eths1) Type = ox0800	Proto = ICMP ttl = 64 Src ip = 122.1.1.1 Dst ip = 122.1.1.3	FCS = 0	FCS = <some actual value>

Actual Hdrs for actual TCP/IP Stack

Pkt for our Simulated TCP/IP stack
This is just a UDP payload for actual TCP/IP Stack

TCP/IP Stack Project -> Actual and Pseudo Headers

- When you trigger run node S ping 122.1.1.3 Cmd on our TCP/IP stack prompt, how does the actual pkt look like which is pushed by send_pkt_out(..) down to the actual TCP/IP Stack running on your local machine



- sudo tcpdump -i lo -v -e udp

ETH Hdr	IP Hdr	UDP Hdr		Pseudo ETH Hdr	Pseudo IP Hdr		
Dst mac = 0 Src mac = 0 Type = ox0800	Proto = UDP ttl = 64 Src ip = 127.0.0.1 Dst ip = 127.0.0.1	Dst port = 2003 Src port = 2002	Aux Info : ethd1	Dst mac = mac(ethd1) Src mac = mac(ethb2) Type = ox0800	Proto = ICMP ttl = 63 Src ip = 122.1.1.1 Dst ip = 122.1.1.3	FCS = 0	FCS = <some actual value>

Actual Hdrs for actual TCP/IP Stack

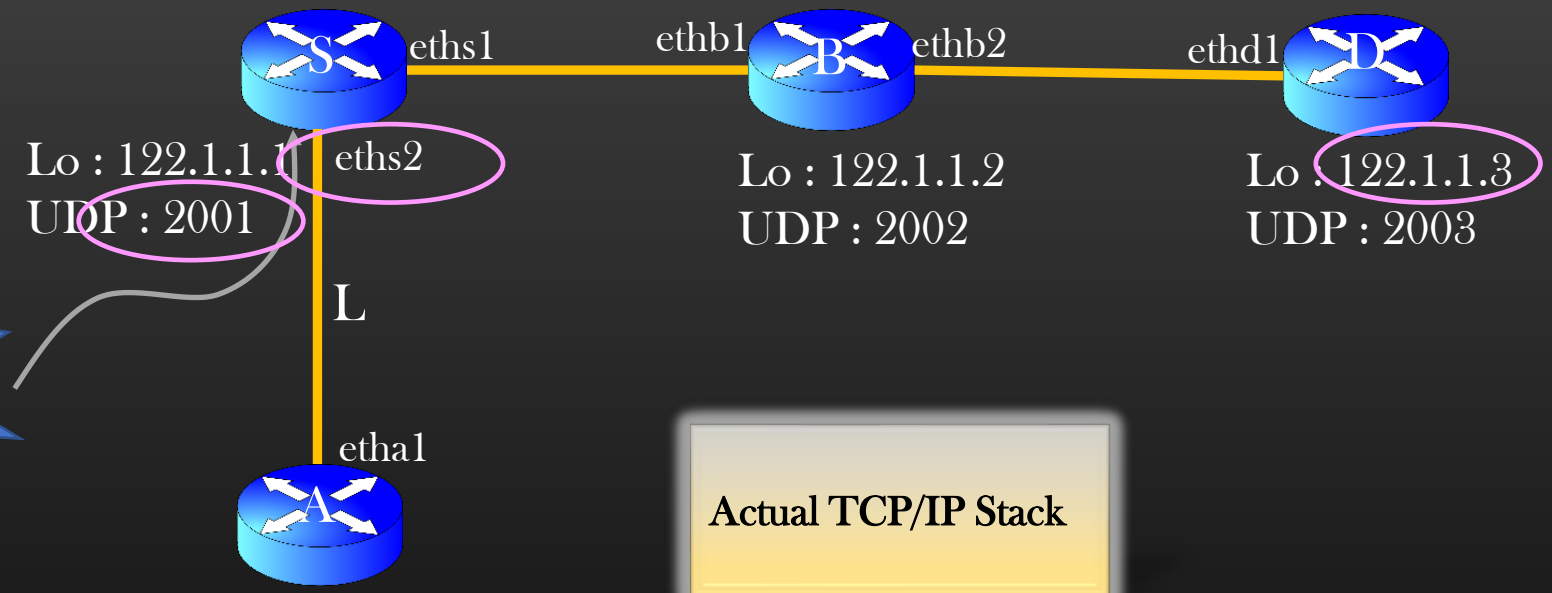
Pkt for our Simulated TCP/IP stack
This is just a UDP payload for actual TCP/IP Stack

Dst mac = 0 Src mac = 0 Type = ox0800	Proto = UDP ttl = 64 Src ip = 127.0.0.1 Dst ip = 127.0.0.1	Dst port = 2001 Src port = 0	Aux Info : eths2	Dst mac = oxFF Src mac = 0 Type = ox0800	Proto = ICMP ttl = 64 Src ip = 122.1.1.1 Dst ip = 122.1.1.3	FCS = 0	FCS = <some actual value>

☜Actual pkt

Actual Hdrs for actual TCP/IP Stack

Pkt for our Simulated TCP/IP stack



Input ?

pkt_gen.c

Actual TCP/IP Stack

Input to pkt_gen.c for Sending "our ICMP" pkts from S to D :

Dst IP : 127.0.0.1
Src ip : 127.0.0.1
Protocol : IPPROTO_UDP
Src port No : 0 (don't matter)
Dst Port No : 2001
UDP Payload :

Actual Hdrs

Aux info	Ethernet Hdr	IP Hdr	Appln Hdr
eths2	Dst mac = mac(eths2)/0xFF Src mac = mac(etha1)/0 Type = ETH_IP	Proto = ICMP ttl = 64 Src ip = 122.1.1.1 Dst ip = 122.1.1.3	Appln Payload (Not present in this case)

Pseudo Hdrs

New file : tcpip_stack/pkt_gen.c

Steps :

1. Create UDP socket

```
int udp_sock_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP );
```

2. Prepare packet

Aux info + Pseudo headers

3. Send data using socket created in 1 to 127.0.0.1:UDP Port no of Src node

(Socket layer will create Actual header for you)

```
struct sockaddr_in dest_addr;
struct hostent *host = (struct hostent *) gethostbyname("127.0.0.1");  /*Dst ip Address*/
dest_addr.sin_family = AF_INET;
dest_addr.sin_port = dst_udp_port_no;            /*UDP Port no of node S*/
dest_addr.sin_addr = *((struct in_addr *)host->h_addr);

rc = sendto(sock_fd, pkt_data, pkt_size, 0,
     (struct sockaddr *)&dest_addr, sizeof(struct sockaddr));
```

Update Project Makefile

Dynamic Construction of L3 Routing Table

TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table

- We love our TCP/IP Stack library ☺, so, let us improve its functionality
- By now, you must have realized, every time we need to install L3 routes in L3 devices manually
- It is a painful exercise and manually doing it is error prone
- In this section, We would implement an algorithm using which
  - Each L3 device in the topology computes L3 routes to every other device
  - Loop free
  - Leverage ECMP (Equal Cost Multiple Path)
  - Re-correct routes (convergence) when topology changes (such as link down/up etc..)
- Demo ... !

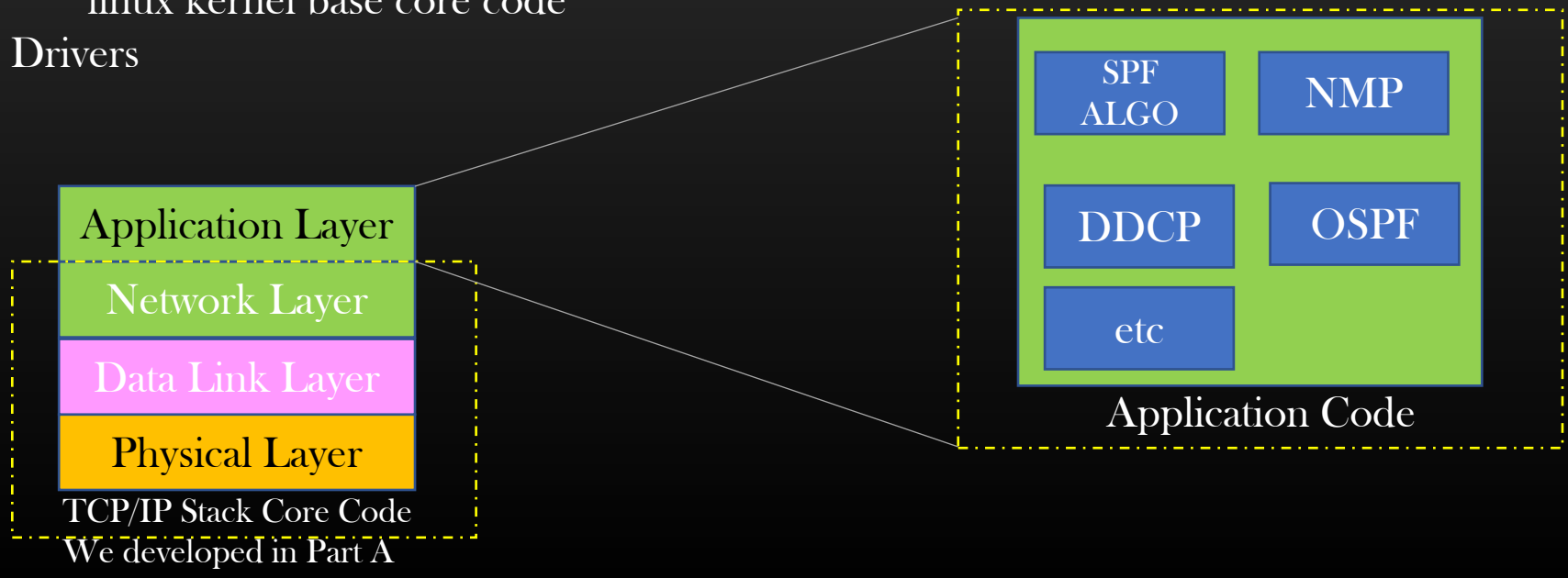
TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table

Pre-Requisites

- Pls complete Appendix section A.1 and A.2 to cover up theory behind construction of L3 routing table
- In these Appendix Section, I cover topics from the point of getting conceptual understanding, and not understanding implementation details
- Once you complete these Appendix Section, We shall be in a position to actually implement it
- From next lecture Video, I presume, you have the concept clarity on Routing Table Construction Algorithm and approach, We will discuss implementation straight away
- This will probably first time you would realize how Networking biased algorithms are implemented

TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table

- We will be going to add SPF algorithm to our TCP/IP stack library as a new application
  - It would mean, SPF algorithm implementation shall be an extension to our TCP/IP stack library and not a core part of it
  - We shall be going to develop several other applications on top of our TCP/IP stack library in plugin-play model
    - This would NOT require us to change/update TCP/IP stack lib core code in anyway
    - Real world example of plugin play model :
      - This is how we add additional functionality to Linux kernel through Linux kernel Modules without touching linux kernel base core code
      - Device Drivers



Getting Started

- Creating new Spf algo Application files and folders in the project code

  New file : Layer5/spf_algo/spf.c

  Update Project Makefile

- CLI Support (nwcli.c)

  run node <node-name> spf

  backend handler : spf_algo_handler( ) -> void compute_spf (node_t *spf_root); /*Layer5/spf_algo/spf.c*/

  show node <node-name> spf

  backend handler : spf_algo_handler( ) -> void show_spf_results(node_t *node) ; /*Layer5/spf_algo/spf.c*/

Preparation

- Before we write any code to implement SPF algo, we need to do some groundwork :
  - Understand New Data Structures
  - Helper APIs used
  - Operations with Priority Queue

- Every node would have a data structure which would store all information related to spf algorithm
- We need to define Two Data structures to implement spf algorithm

```
typedef struct node_{
        . . .
        . . .
        spf_data_t *spf_data;
        . . .
} node_t;
```

1

```
typedef struct spf_data_{
        node_t *node;
        glthread_t spf_result_head;   // Valid only for spf root
        /*Temp fields used for calculations*/
        uint32_t spf_metric;          // Valid for other nodes
        glthread_t priority_thread_glue;
        nexthop_t *nexthops[MAX_NXT_HOPS];
} spf_data_t;
```

2

```
typedef struct spf_result_{
        node_t *node;
        uint32_t spf_metric;
        nexthop_t *nexthops[MAX_NXT_HOPS];
        glthread_t spf_res_glue;
} spf_result_t;
```

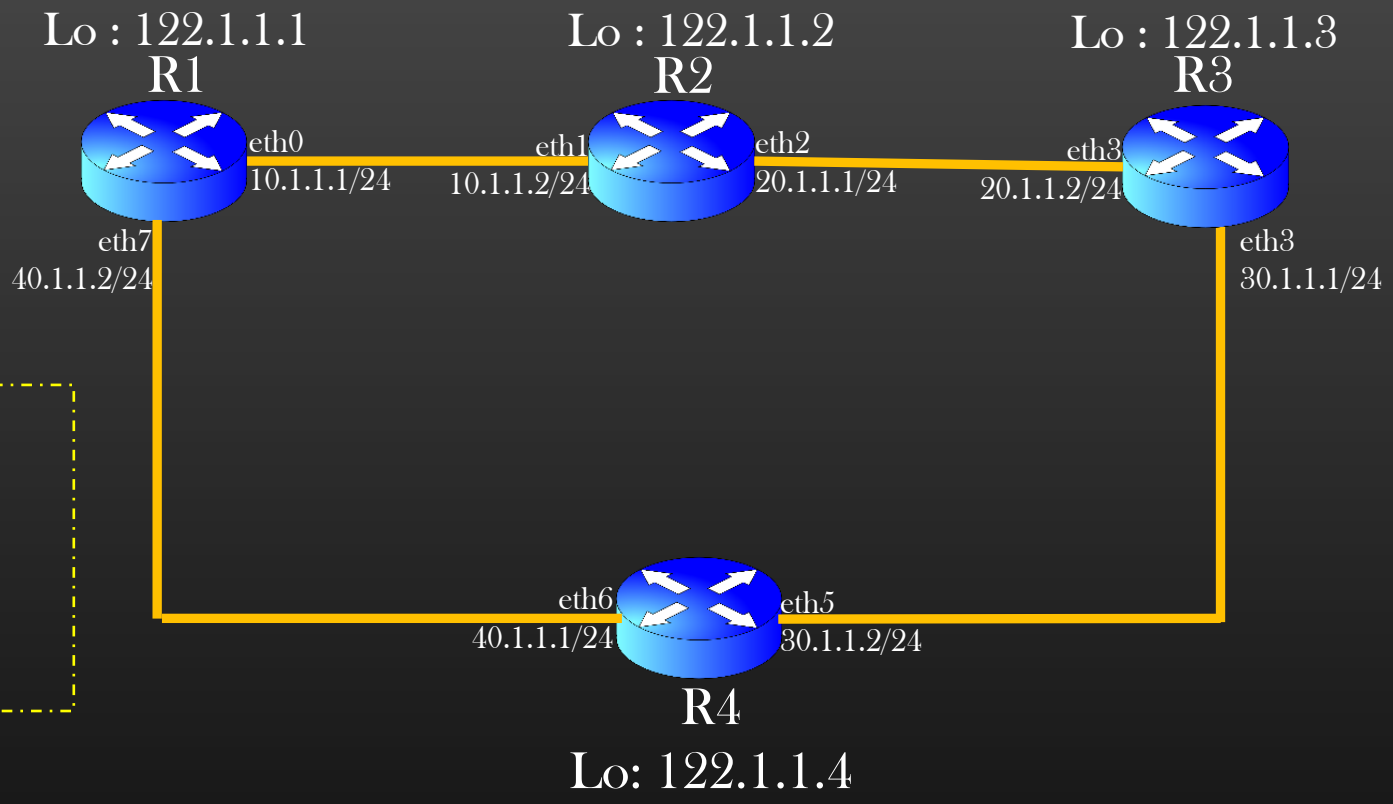
TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Data Structures

➢ Example Snapshot :

```
typedef struct nexthop_{

    char gw_ip[16];
    interface_t *oif;
    uint32_t ref_count;
} nexthop_t;
```

NH1 = 10.1.1.2 R1-eth0 0

NH2 = 40.1.1.1 R1-eth7 0



R1 as spf root node->spf_data-> spf_result_head

R2	R4	R3
10	10	20
NH1	NH2	NH1, NH2

Result of compute_spf(R1)

TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Helper APIs

- We need to write some Helper APIs (arnd 10) which would make it easier to implement Spf algo
- These Helper APIs shall be invoked from compute_spf(..) fn
- You must thoroughly test these APIs before actually start using them .. Otherwise Nightmare !
- You can write a small driver program to test the APIs you will write Or Cross check against partB soln code base

API 1 :

net.h/.c

```
bool_t
is_interface_l3_bidirectional (interface_t *interface);   /*ptr to R1-eth0*/
```

R1-eth0 is L3 bidirectional if and only if :

- R1-eth0 and Nbr interface R2-eth1 are UP &&
- R1-eth0 and R2-eth1 are configured with IP addresses &&
- IP Addresses are in same subnet &&
- None of the interfaces are operating in L2 mode (ACCESS or TRUNK)

API 2 :

spf.c

```
void
spf_flush_nexthops(nexthop_t * *nexthop);
```

Eg : spf_flush_nexthops (node->spf_data->nexthops);
spf_flush_nexthops (spf_result->nexthops);

```
void
spf_flush_nexthops (nexthop_t * *nexthop){

    int i = 0;

    if(!nexthop) return;

    for( ; i < MAX_NXT_HOPS; i++){

        if(nexthop[i]){
            assert(nexthop[i]->ref_count);
            nexthop[i]->ref_count - = 1;
            if (nexthop[i]->ref_count == 0){
                free(nexthop[i]);
            }
            nexthop[i] = NULL;
        }
    }
}
```

API 3 :
spf.c

```
static inline void
free_spf_result(spf_result_t *spf_result) {

    spf_flush_nexthops(spf_result->nexthops);
    remove_glthread(&spf_result->spf_res_glue);
    free(spf_result);
}
```

Reference :

```
typedef struct spf_result_{
        node_t *node;
        uint32_t spf_metric;
        nexthop_t *nexthops[MAX_NXT_HOPS];
        glthread_t spf_res_glue;
} spf_result_t;
```

API 4 :

spf.c

```
static nexthop_t *
create_new_nexthop (interface_t *oif) { /*ptr to S-eth0*/

    nexthop_t *nexthop = calloc(1, sizeof(nexthop_t));
    nexthop->oif = oif;
    interface_t *other_intf = &oif->link->intf1 == oif ?    \
        &oif->link->intf2 : &oif->link->intf1;
    if(!other_intf){
        free(nexthop);
        return NULL;
    }
    strncpy(nexthop->gw_ip, IF_IP(other_intf), 16);
    nexthop->ref_count = 0;
    return nexthop;
}
```

```
typedef struct nexthop_{

    char gw_ip[16];
    interface_t *oif;
    uint32_t ref_count;
} nexthop_t;
```

Nh = { 10.1.1.2 , ptr to S-eth0, 0 }

API 5 :

spf.c

```
static bool_t
spf_insert_new_nexthop(nexthop_t **nexthop_arry,
                       nexthop_t *nxthop) {

 int i = 0;

   for( ; i < MAX_NXT_HOPS; i++){
      if(nexthop_arry[i]) continue;
      nexthop_arry[i] = nxthop;
      nexthop_arry[i]->ref_count++;
      return TRUE;
   }
   return FALSE;
}
```

API6 :

spf.c

```
static bool_t
spf_is_nexthop_exist(nexthop_t **nexthop_array,
                                nexthop_t *nxthop){

 int i = 0;
   for( ; i < MAX_NXT_HOPS; i++){

      if (!nexthop_array[i])
         return FALSE;

      if (nexthop_array[i]->oif == nxthop->oif)
         return TRUE;
   }
   return FALSE;
}
```

API 7 :

spf.c

```
/* Copy all nexthops of src to dst, do not copy which are already
 * present*/
static int
spf_union_nexthops_arrays(nexthop_t **src, nexthop_t **dst);
```

- Increase the ref count of nexthops copied
- Do not copy which are already present in dst
- Use spf_is_nexthop_exist(..) to verify if Nexthop NH already present in dst

API returns the number of Nexthops copied

API 8 :

spf.c

API to compare two spf_data_t objects

```
/*Return -1 , 0 or 1 */
static int
spf_comparison_fn(void *data1, void *data2){

    spf_data_t *spf_data_1 = (spf_data_t *)data1;
    spf_data_t *spf_data_2 = (spf_data_t *)data2;

    if(spf_data_1->spf_metric < spf_data_2->spf_metric)
        return -1;
    if(spf_data_1->spf_metric > spf_data_2->spf_metric)
        return 1;
    return 0;
}
```

- This API is used as a fn pointer to PQ API
- PQ will use this function to insert a new spf_data object in PQ as per the priority (spf_metric)

Reference :

```
typedef struct spf_data_{
        . . .
        /*Temp fields used for calculations*/
        uint32_t spf_metric;
        . . .
} spf_data_t;
```

API 9 :

spf.c

API to look up spf_result_t object from spf result list of spf_root using node ptr as lookup key

```
static spf_result_t *
spf_lookup_spf_result_by_node(node_t *spf_root, node_t *node){

    glthread_t *curr;
    spf_result_t *spf_result;
    spf_data_t *curr_spf_data;

    ITERATE_GLTHREAD_BEGIN(&spf_root->spf_data->spf_result_head, curr){

        spf_result = spf_res_glue_to_spf_result(curr);
        if(spf_result->node == node)
            return spf_result;
    } ITERATE_GLTHREAD_END(&spf_root->spf_data->spf_result_head, curr);
    return NULL;
}
```

API 10 :
spf.c

Short-hand Macros :

```
#define INFINITE_METRIC     0xFFFFFFFF


#define spf_data_offset_from_priority_thread_glue \
    ((size_t)&(((spf_data_t *)0)->priority_thread_glue))


#define SPF_METRIC(nodeptr) (nodeptr->spf_data->spf_metric)
```

```
typedef struct spf_data_{
    node_t *node;
    glthread_t spf_result_head;
    uint32_t spf_metric;
    glthread_t priority_thread_glue;
    nexthop_t *nexthops[MAX_NXT_HOPS];
} spf_data_t;
```

Priority Queue Operations

- SPF Algorithm uses PQ for its implementation
  - We insert node->spf_data object into PQ on the basis of node->spf_data->spf_metric
  - Initialization
  - Insertion
  - Deletion
  - Dequeue

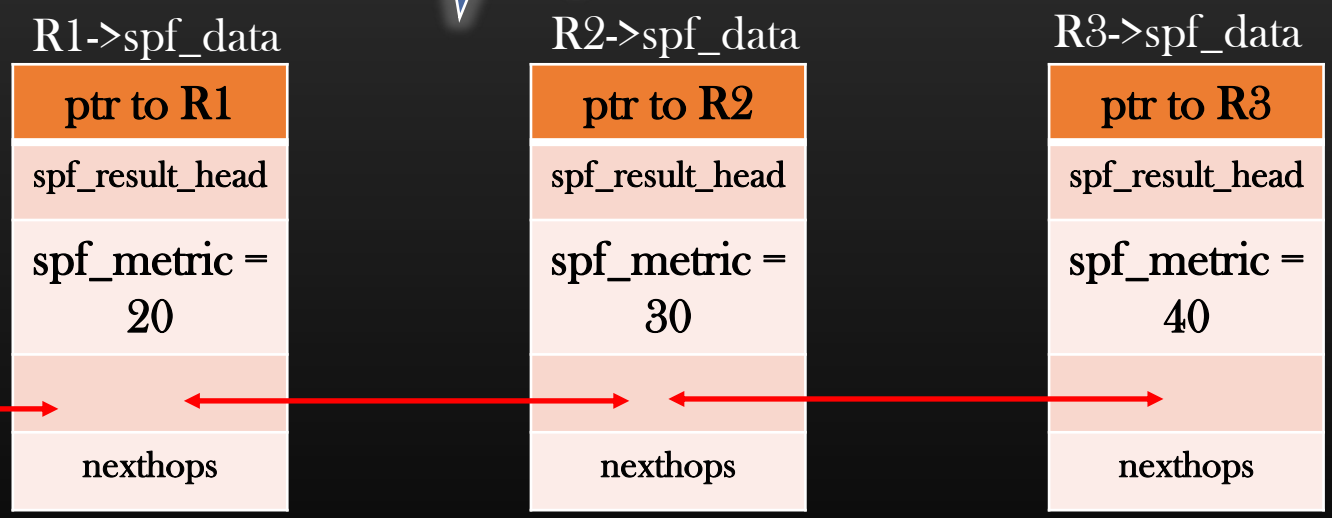
- Operations with Priority Queue

Reference :

```
typedef struct spf_data_{
        node_t *node;
        glthread_t spf_result_head;
        uint32_t spf_metric;
        glthread_t priority_thread_glue;
        nexthop_t *nexthops[MAX_NXT_HOPS];
} spf_data_t;
```

PQ of spf_root S maintained
in increasing order of spf_metric ☞☞

glthread_t priority_lst;



- PQ stores the spf_data objects of nodes of the topology in the increasing order of *node->spf_data->spf_metric*

- Operations with Priority Queue

Reference :

```
typedef struct spf_data_{
        node_t *node;
        glthread_t spf_result_head;
        uint32_t spf_metric;
        glthread_t priority_thread_glue;
        nexthop_t *nexthops[MAX_NXT_HOPS];
} spf_data_t;
```

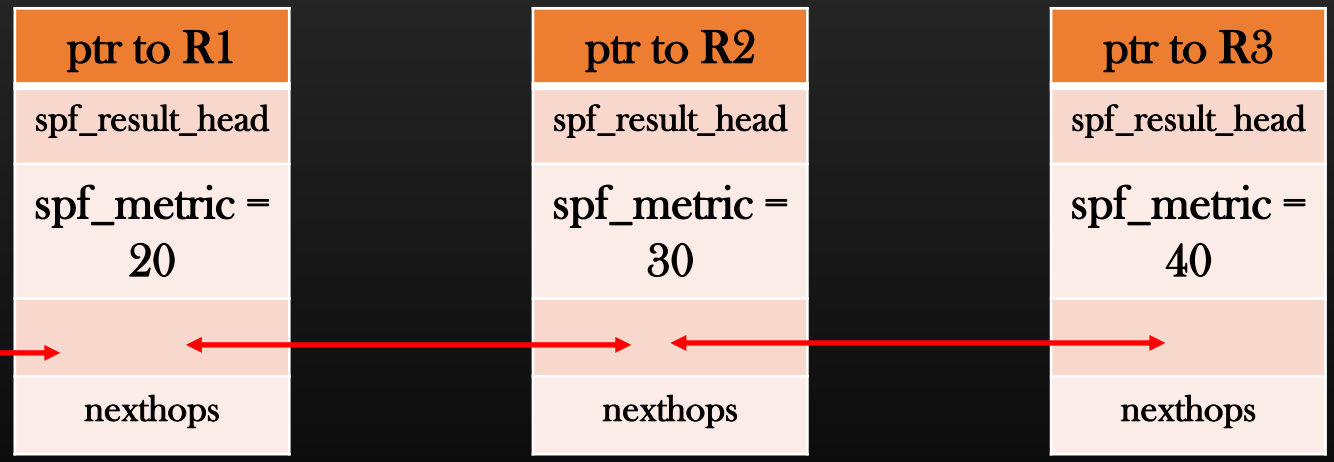
Checking if PQ Is Empty :

IS_GLTHREAD_LIST_EMPTY (&priority_lst)

Note : End of the Day, PQ is just a glthread (Doubly linked list)

PQ of spf_root S maintained
in increasing order of spf_metric ☞☞

glthread_t priority_lst;



- Operations with Priority Queue

Reference :

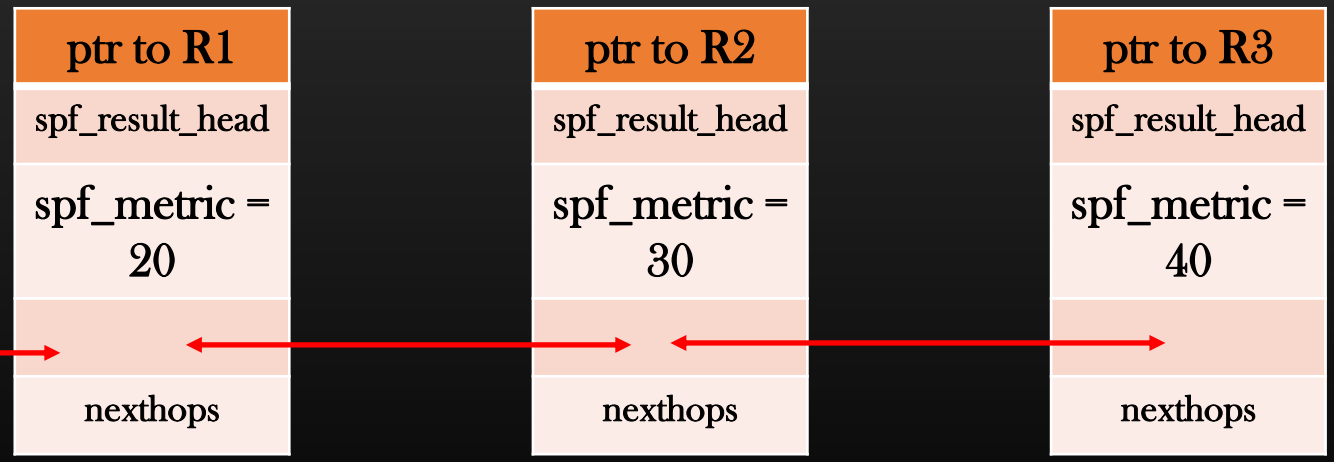
```
typedef struct spf_data_{
        node_t *node;
        glthread_t spf_result_head;
        uint32_t spf_metric;
        glthread_t priority_thread_glue;
        nexthop_t *nexthops[MAX_NXT_HOPS];
} spf_data_t;
```

Insertion into PQ :

```
glthread_priority_insert (&priority_lst,
        &node->spf_data->priority_thread_glue,
        spf_comparison_fn,
        spf_data_offset_from_priority_thread_glue);
```

PQ of spf_root S maintained
in increasing order of spf_metric ☞☞

glthread_t priority_lst;



- Operations with Priority Queue

Reference :

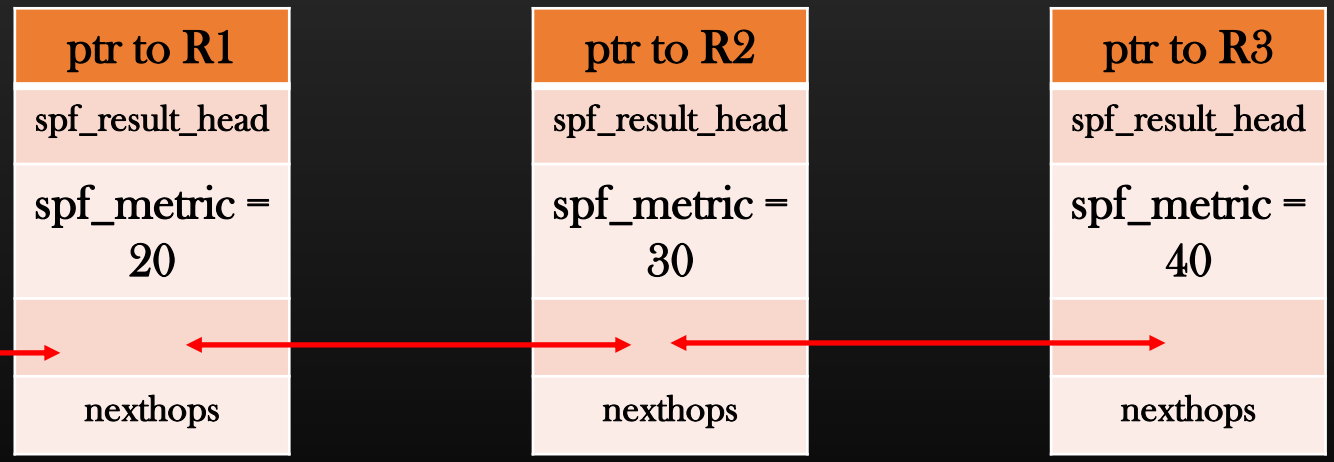
```
typedef struct spf_data_{
        node_t *node;
        glthread_t spf_result_head;
        uint32_t spf_metric;
        glthread_t priority_thread_glue;
        nexthop_t *nexthops[MAX_NXT_HOPS];
} spf_data_t;
```

Removal from PQ :

```
remove_glthread(&spf_data->priority_thread_glue);
```

PQ of spf_root S maintained
in increasing order of spf_metric ☞☞

glthread_t priority_lst;



- Operations with Priority Queue

Reference :

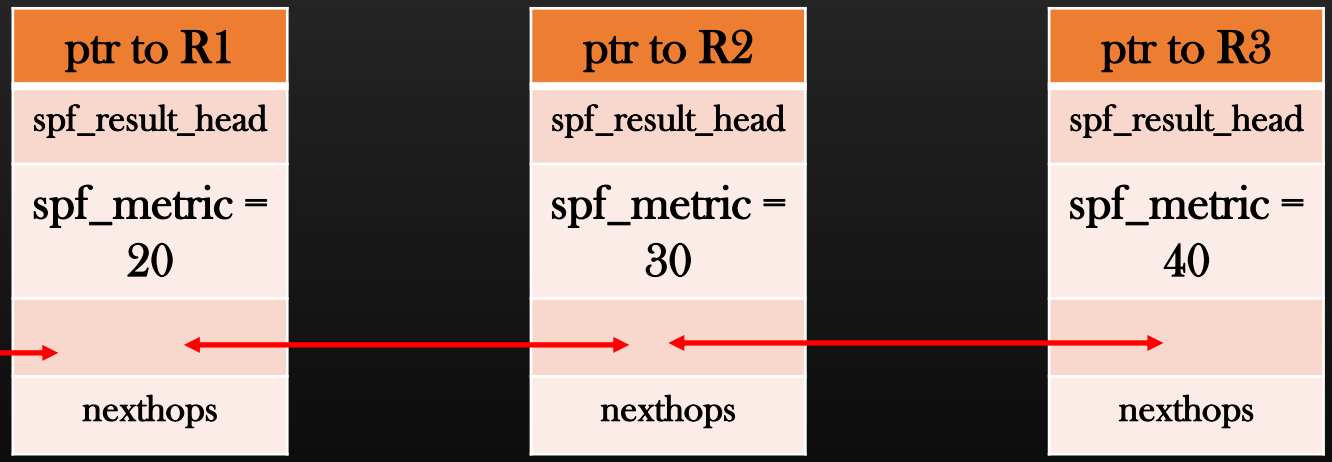
```
typedef struct spf_data_{
        node_t *node;
        glthread_t spf_result_head;
        uint32_t spf_metric;
        glthread_t priority_thread_glue;
        nexthop_t *nexthops[MAX_NXT_HOPS];
} spf_data_t;
```

Dequeue from PQ :

```
glthread_t *curr;
 curr = dequeue_glthread_first (&priority_lst);
 spf_data_t *curr_spf_data = priority_thread_glue_to_spf_data (curr);
```

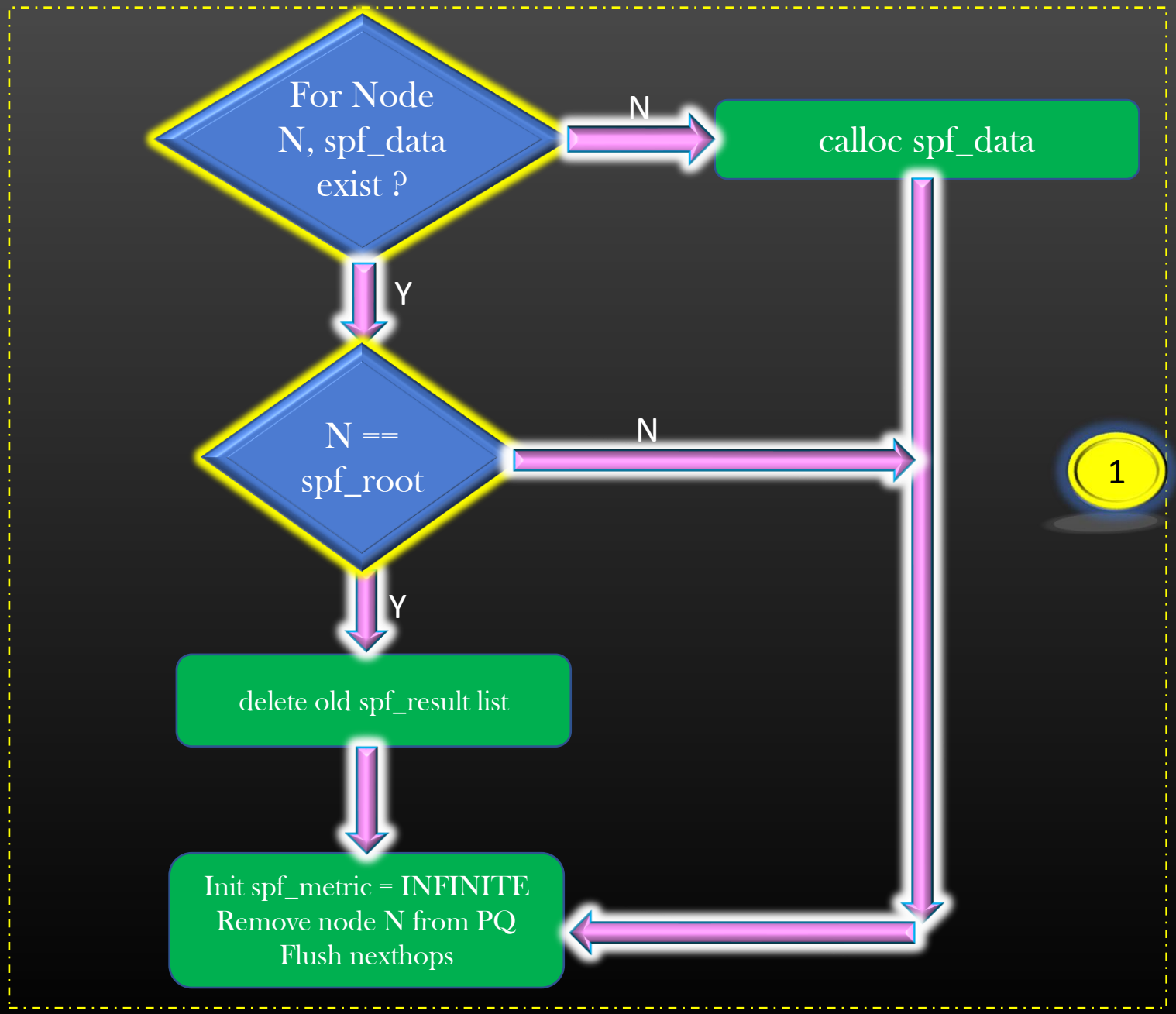
PQ of spf_root S maintained
in increasing order of spf_metric ☞☞

glthread_t priority_lst;



- **We will now implement SPF Algorithm Step by Step**
  - Initialization Phase 1
    - Step 1, 2 and 3
  - Execution Phase 2
    - Step 4, 5 and 6
  - ECMP support in l3_route_t
  - Route Calculation Phase 3 (Step 7)

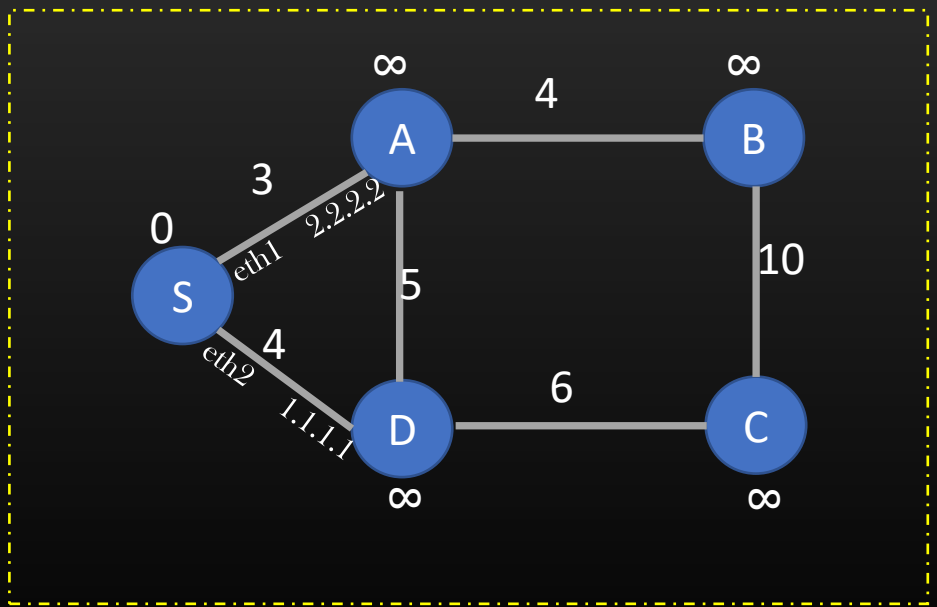
TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Initialization Phase



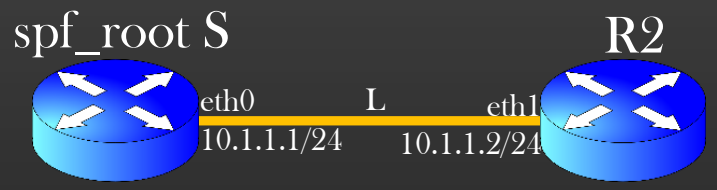
Initialization : Part 1

- Initialize Metrics
- delete old spf results if any
- Remove Nodes from PQ
- Flush Nexthops if any

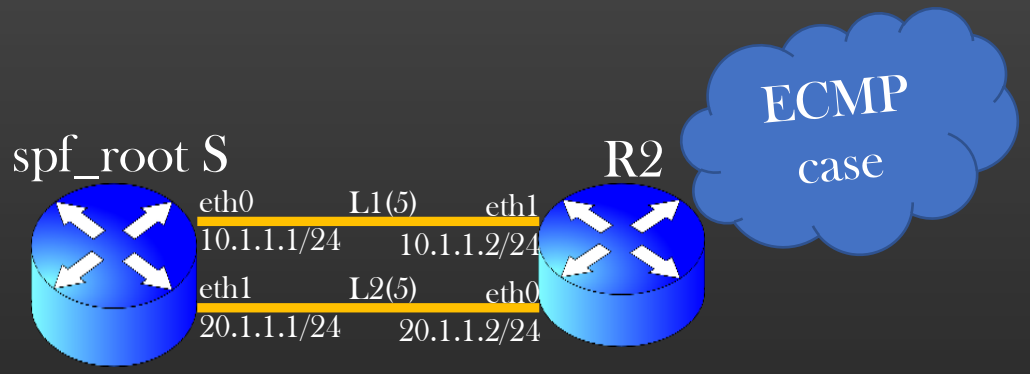
```
void
init_node_spf_data (node_t *node,
bool_t delete_spf_result);
```



Next-hop Calculation

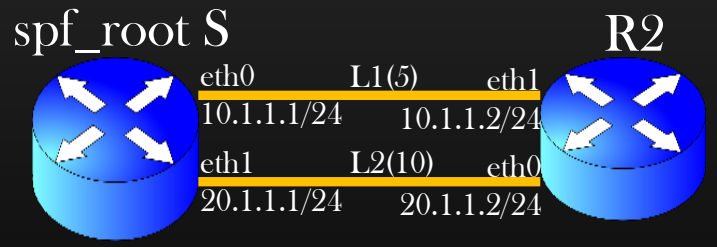


Nh = { 10.1.1.2 , ptr to S-eth0, 0 }



Nh1 = { 10.1.1.2 , ptr to S-eth0, 0 }

Nh2 = { 20.1.1.2 , ptr to S-eth1, 0 }



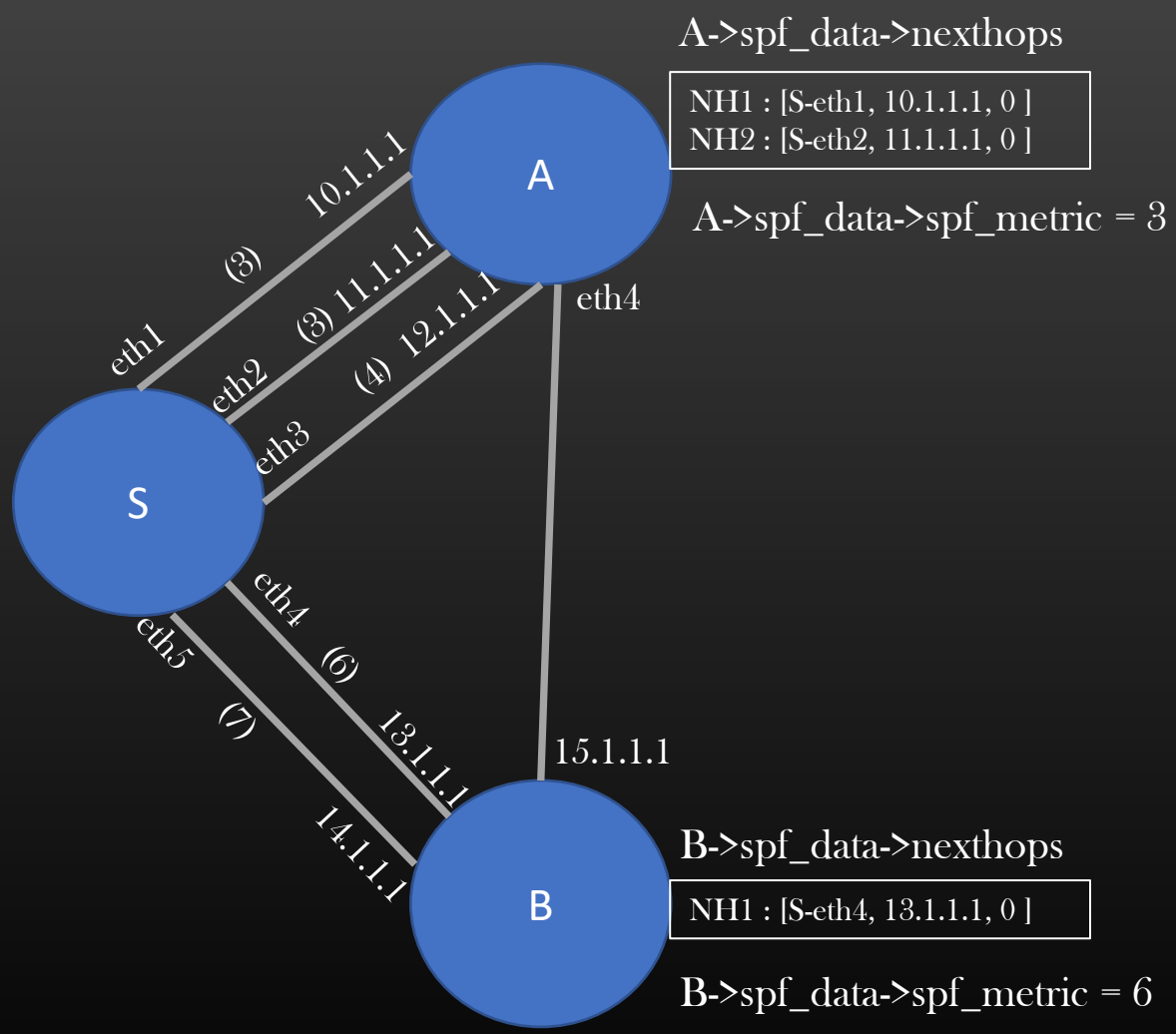
Nh = { 20.1.1.2 , ptr to S-eth1, 0 }

☞ The nexthops computed by spf root are stored in Nbr's spf_data->nexthops[ ] array

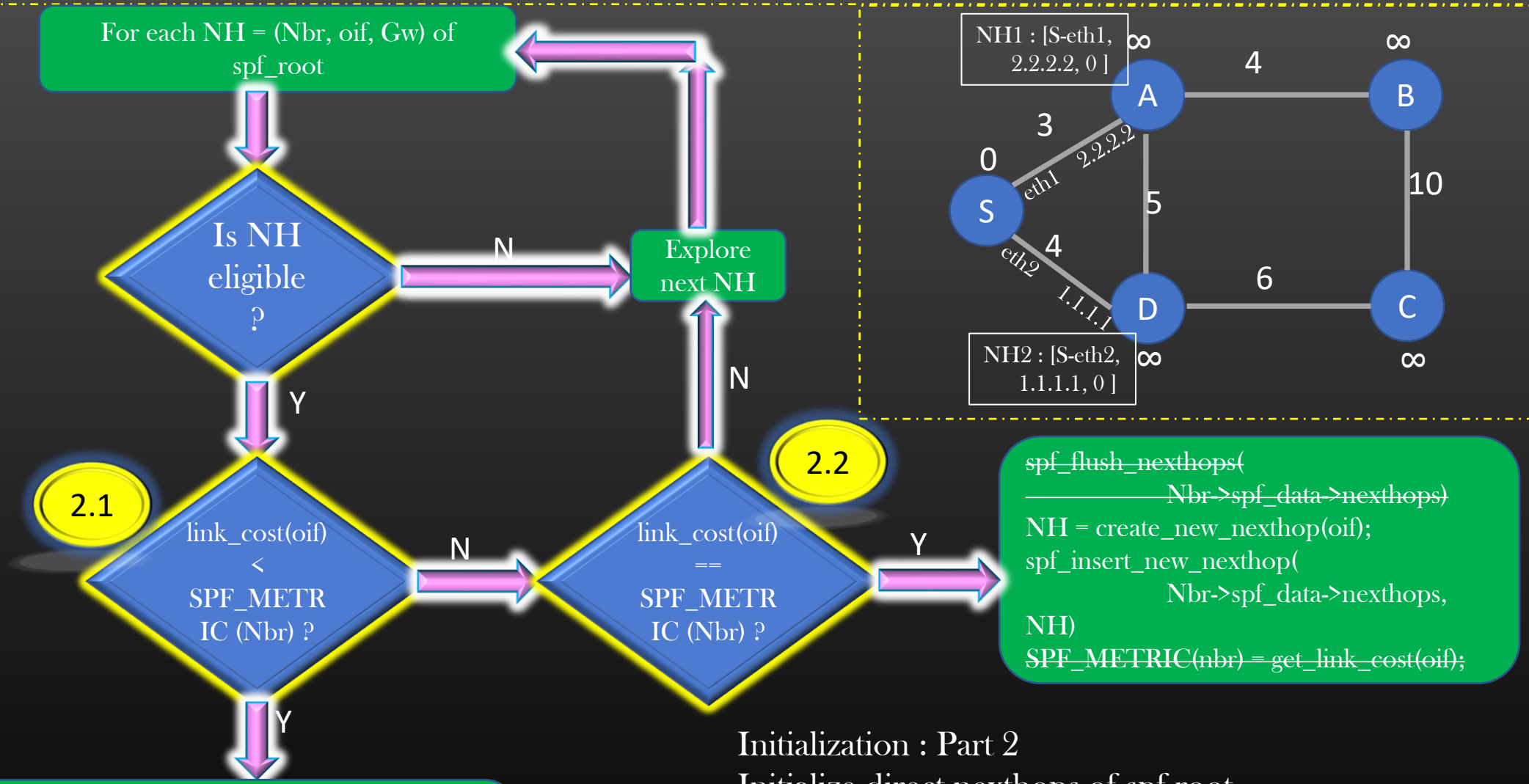
☞ Nexthop Eligibility
is_interface_l3_bidirectional( )

TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Initialization Phase

Example



TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Initialization Phase



Initialization : Part 2
Initialize direct nexthops of spf root

void
initialize_direct_nbrs (node_t *spf_root);

TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Initialization Phase

Looping Macro

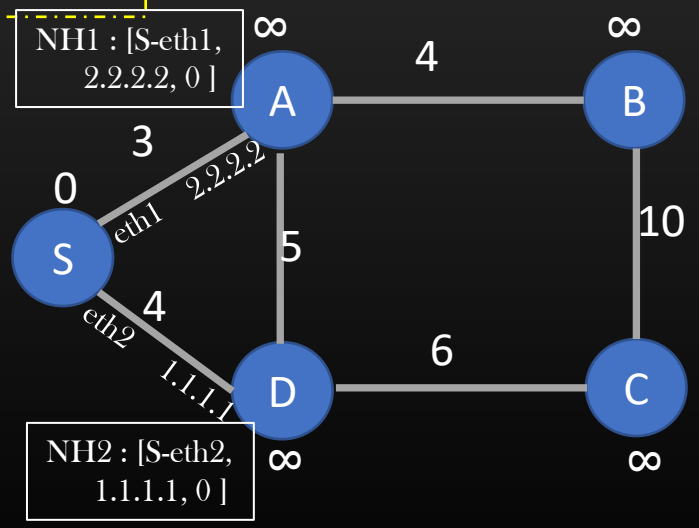
```
glthread_t priority_lst;
init_glthread(&priority_lst);
```

3

```
glthread_priority_insert (&priority_lst,
        &spf_root->spf_data->priority_thread_glue,
         spf_comparison_fn,
         spf_data_offset_from_priority_thread_glue);
```

Initialization: Part 3
Initialize Priority Queue
Add Spf_root in PQ

PQ = { S }



TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> CLI Requirements

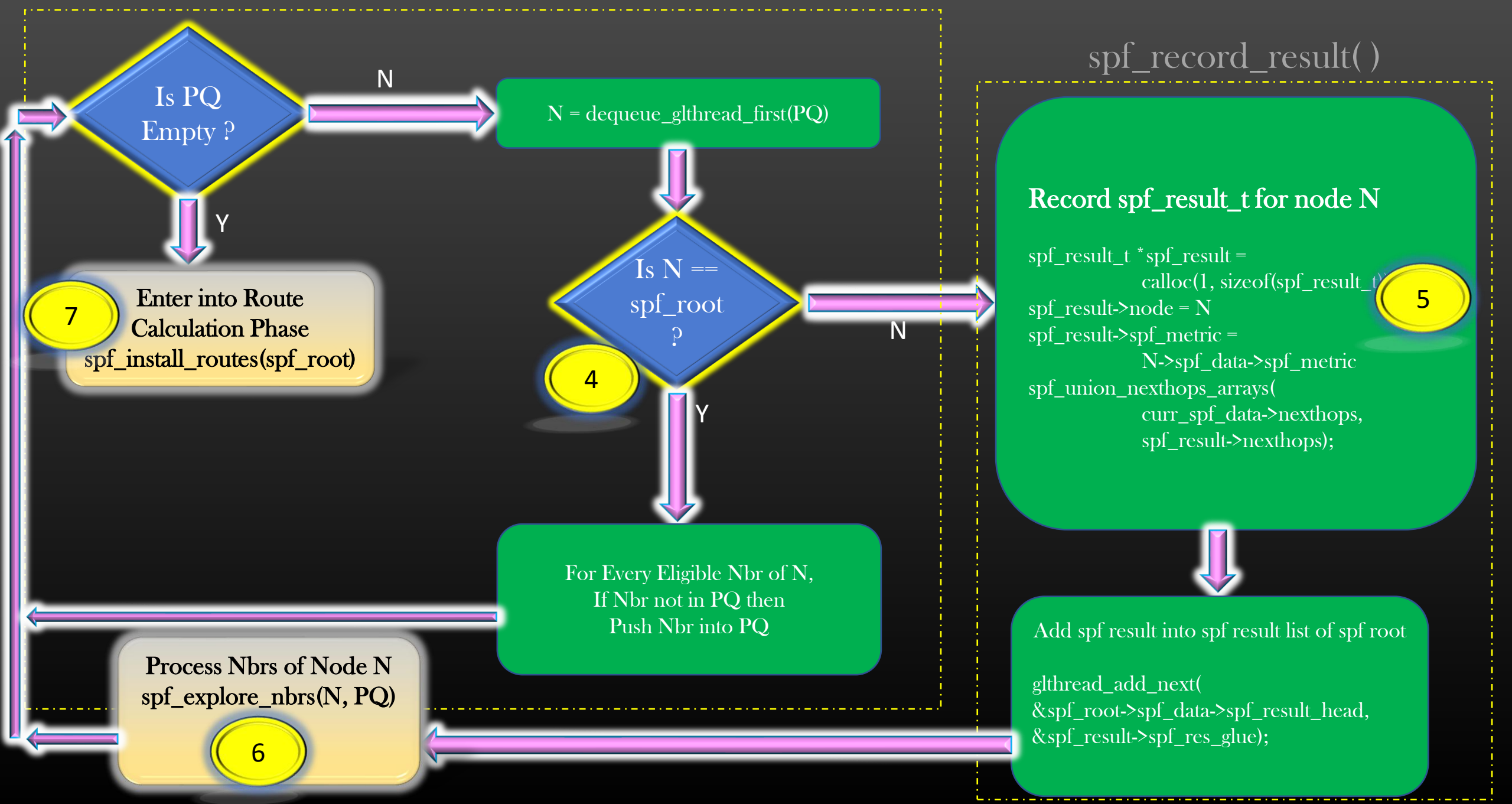
- Now that we have finished implementation of Core SPF Algorithm, Let us test it before proceeding forward
- File : spf.c

- Use CLI : run node <node-name> spf
  - To Trigger SPF on a particular node
  - spf_algo_handler( ) - -> compute_spf(node_t *node)

- Use CLI : show node <node-name> spf
  - To Check SPF result list of a particular node
  - spf_algo_handler( ) - -> show_spf_results(node_t *node)

- Use CLI : run spf all
  - Running SPF on all nodes of the Topology
  - spf_algo_handler( ) - -> compute_spf_all_routers(graph_t *topo)

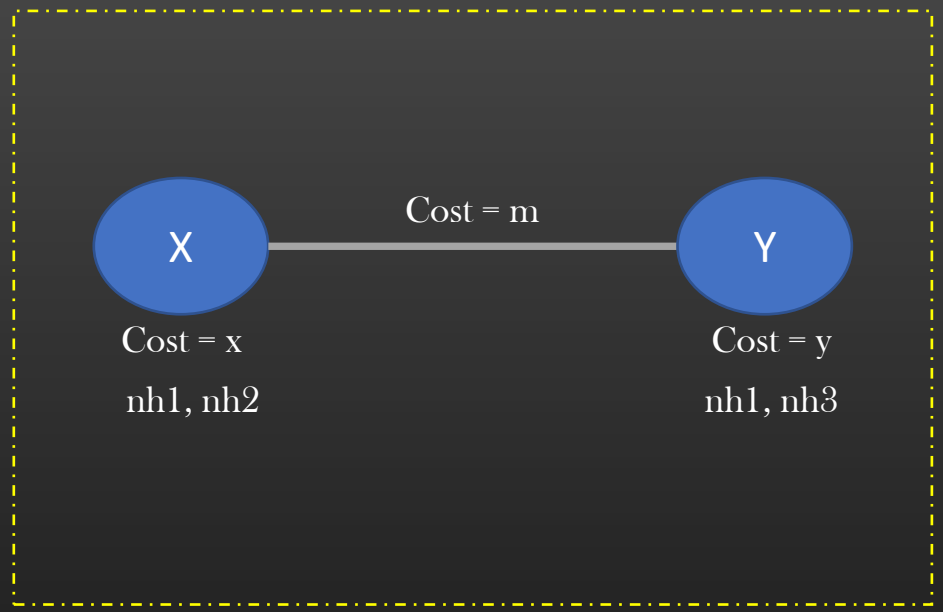
TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Execution Phase



Algorithm Rules When a new node is explored

void spf_explore_nbrs()

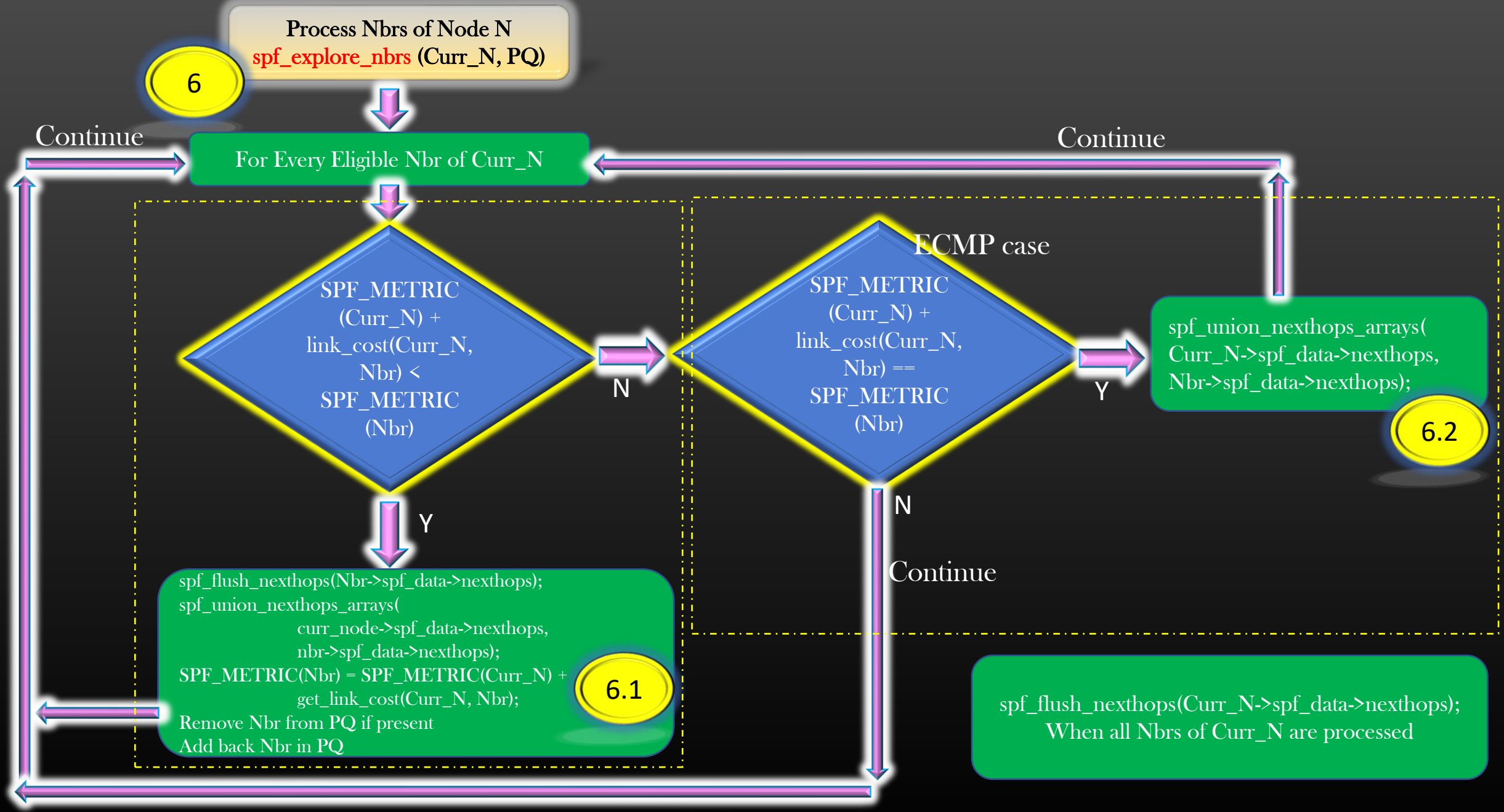
- If node Y is explored from predecessor node X with a better metric
  i.e. x + m < y
  - Remove nexthops of Y
  - Copy nexthops of X into Y
  - Remove Y from PQ if present, and add it back to PQ
  - Cost(y) = Cost(x) + m



- If node Y is explored from predecessor node X with a same metric
  i.e. x + m = y (ECMP case)
  - Keep existing nexthops of Y
  - Copy nexthops of X into Y
  - Remove Duplicates in next-hop list of Y, if any

- If node Y is explored from predecessor node X with higher metric
  i.e. x + m > y
  - Ignore Y through this link

TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Execution Phase



TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Testing

- It is important that we test our implementation of SPF algorithm on various different topologies and verify the correctness of the output before proceeding forward

- You need to run :
  - run node <node-name> spf << to trigger spf on a particular node
  - run spf all << to trigger spf on all nodes of topology
  - show node <node-name> spf << see the output of spf run

- In addition, change topologies using various existing CLIs as below, and trigger run spf all and verify all nodes update their spf results
  - conf node <node-name> interface <if-name> metric <metric-val>
  - conf node <node-name> interface <if-name> [up | down]

- Attached is the document which shows the topologies you need to be build and testing procedure
- Pls follow the instructions against each *Test* in the document

- Proceed to next section only when you have verified that your SPF implementation reveals correct output in all cases
- Fix the bugs/anomalies if any

- We will use the output produced by SPF algorithm for final Routing table calculation of SPF root node

- We need to implement fn :

```
int spf_install_routes (node_t *spf_root)   /*Phase 3/3, Step 7 */
```

- But before that, we need to make some minor code changes to make our Routing Table and L3 routes ECMP aware

TCP/IP Stack Project -> ECMP Support

- The L3 route having more than 1 nexthops is an ECMP aware route
- Assumption : cost of each link is 1

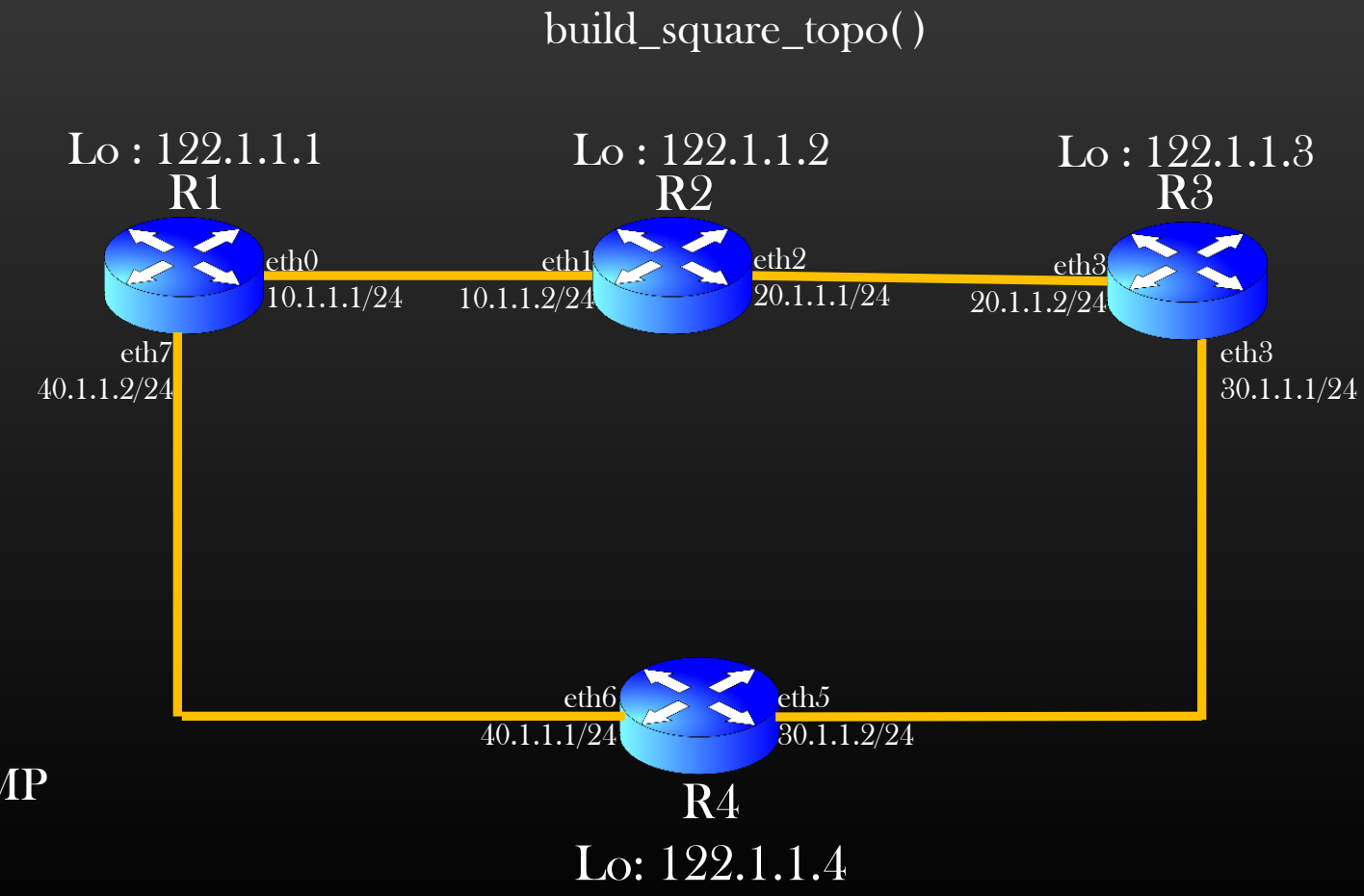
R1's RT entry

122.1.1.3/32	10.1.1.2 40.1.1.1	eth0 eth7	2

RT Table / L3 route data structure must allow a provision to allow multiple nexthops associated with an L3 route

How ECMP help ??
Ans : Distribute traffic to utilize Network available Bandwidths/links
Protection against link/node failures

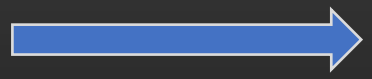
Next : Let us discuss Code changes to implement ECMP



Implementing ECMP

Current :

```
typedef struct l3_route_{
        char dest[16]; /*key*/
        char mask;     /*key*/
        bool_t is_direct;
        char gw_ip[16];
        char oif[IF_NAME_SIZE];
        glthread_t rt_glue;
} l3_route_t;
```



New :

```
typedef struct l3_route_{
        char dest[16]; /*key*/
        char mask;     /*key*/
        bool_t is_direct;
        nexthop_t *nexthops[MAX_NXT_HOPS];
        uint32_t spf_metric;
        int nxthop_idx;
        glthread_t rt_glue;
} l3_route_t;
```

Re-write from scratch : rt_table_add_route( ) to accommodate this change.
User should be able to install MAX_NXT_HOPS nexthops for a given route

Update clear_rt_table( ) to free the nexthops first before freeing the route, else it will cause mem leak

TCP/IP Stack Project -> Dynamic Construction of L3 Routing Table -> Spf Algorithm -> Route Calculation

Consider an L3 route with 4 nexthops :
nh1...nh4

New :

```
typedef struct l3_route_{
        char dest[16]; /*key*/
        char mask;     /*key*/
        bool_t is_direct;
        nexthop_t *nexthops[MAX_NXT_HOPS];
        uint32_t spf_metric;
        int nxthop_idx;
        glthread_t rt_glue;
} l3_route_t;
```

layer3.c/.h

```
nexthop_t *
l3_route_get_active_nexthop(l3_route_t *l3_route);
```

- nxthop_idx - points to the nexthop which will be used by the route to forward the next pkt
- Every time the route forwards the pkt, nxthop_idx is incremented to point to the next nexthop in array
- Route forwards the pkt P1,P2,P3,P4,P5... using nexthops NH3,NH4,NH1,NH2,NH3...
- Update Functions : layer3_ip_pkt_recv_from_layer2( ) & demote_packet_to_layer3 ( ) to use new API to retrieve the forwarding nexthop of the route

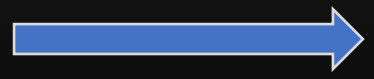
```
int
spf_install_routes(node_t *spf_root);
```

- Use spf_root->spf_data->spf_result_head list to compute routes to loopback address of all other L3 devices of the topology
- Returns number of routes installed

Spf_root S
SPF Algo Result

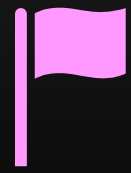
Dest Node	Oif	Gateway	cost	Nexthop
A	eth1	2.2.2.2	3	A
D	eth2	1.1.1.1	4	D
B	eth1	2.2.2.2	7	A
C	eth2	1.1.1.1	10	D

Replace Dest Nodes With their loop-back Address



Final Routing Table of Spf Root S

Dest Address	Oif	Gateway	cost	Nexthop
122.1.1.1/32	eth1	2.2.2.2	3	A
122.1.1.3/32	eth2	1.1.1.1	4	D
122.1.1.2/32	eth1	2.2.2.2	7	A
122.1.1.4/32	eth2	1.1.1.1	10	D



Initializing the TCP/IP stack on Start up

- We may need to do some one-time initialization on TCP/IP stack library start up

- For example:
  - We want L3 Routing Tables get populated automatically as soon as we start our project
  - User should not be compelled to run “run spf all” on project start up to populate RTs

- Write a new file : tcp_stack_init.c
  - Write a fn : init_tcp_ip_stack( )
  - Call this fn from main( )

Logging Infra

Let TCP/IP Stack tell you what's going on !

TCP/IP Stack Project -> Logging Infra -> Problem Statement

- Nodes of our Emulated topology forward and Recv Packets
- Our TCP/IP Stack library could be used to implement more new Network protocols which would generate their own Network packets (Application headers)
- The common way of debugging and troubleshooting networks is to analyse what packets are being sent or recvd by node(s) of the topology
- On real devices, we have tools to capture and analyze packet contents flowing across the network
  - Tcpdump
  - Wireshark
- For our Emulated TCP/IP stack library, standard tools would not likely to work and we would like to develop the way our TCP/IP stack library could emit out what packets are in flows across the network
- This would help in developing and debugging Network application immensely
- Lets start . .

TCP/IP Stack Project -> Logging Infra -> Working Demo

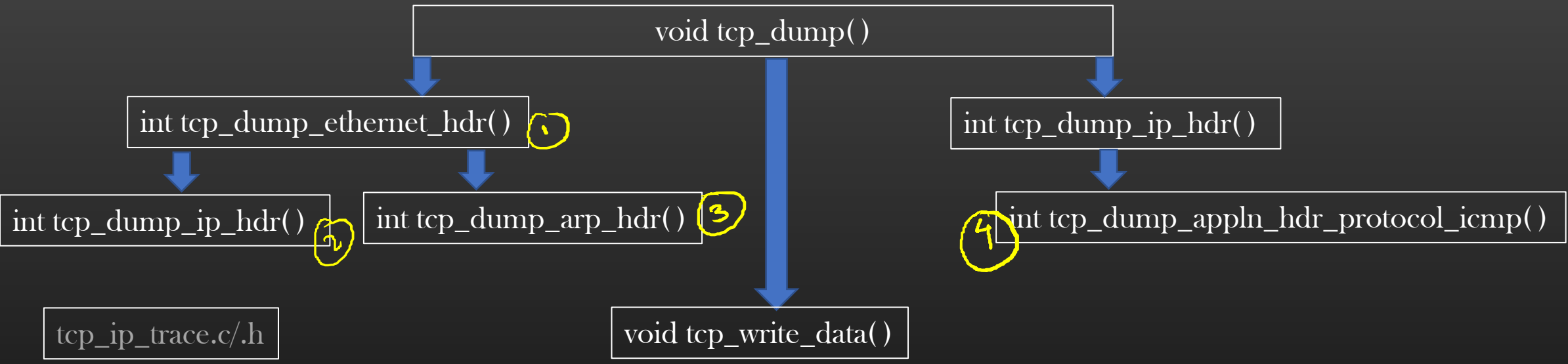
- Let us see what we are getting into ...
- Let us see how logging is expected to work ..

Goals :

- User should be able to enable or disable logging at will using CLIs
- User should have a choice whether he wants to enable logging for only sent or only recv pkts or both
- Extra : User should have a choice whether he wants to enable logging for some selective interfaces of a node
- New Files :
  tcpip_stack/tcp_ip_trace.c
  tcpip_stack/tcp_ip_trace.h
- Explore use of sprint/snprintf from internet. We would be using this fn extensively for logging
- Let us first brainlessly write all required APIs first, then we shall see how to use the APIs for logging and control logging using CLIs

TCP/IP Stack Project -> Logging Infra -> logging APIs hierarchy

- First, we need set of APIs to format the headers of various types into buffer




```
int
tcp_dump_ethernet_hdr(
                char *out_buff,
                ethernet_hdr_t
*eth_hdr,
        uint32_t pkt_size);
```

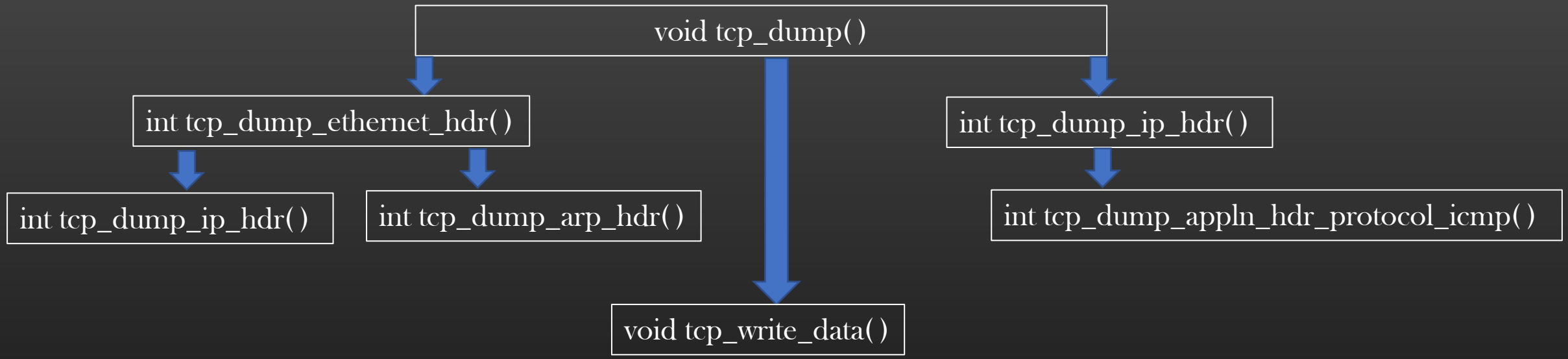
```
int
tcp_dump_ip_hdr(
                char *out_buff,
                ip_hdr_t *ip_hdr,
        uint32_t pkt_size);
```

```
int
tcp_dump_arp_hdr(
                char *out_buff,
                arp_hdr_t *arp_hdr,
        uint32_t pkt_size);
```

```
int
tcp_dump_appln_hdr_protocol_icmp(
                char *out_buff,
                char *appln_data,
                uint32_t pkt_size);
```

- APIs returns number of bytes written to the output buffer

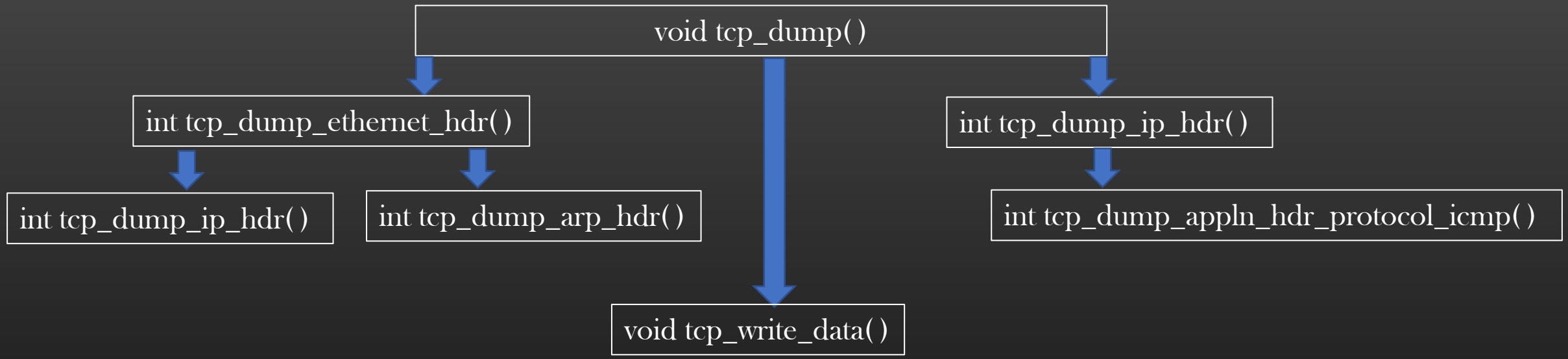
TCP/IP Stack Project -> Logging Infra -> logging hierarchy



Fn to parse the pkt and prepare formatted output in buffer memory

```
static void
tcp_dump (int sock_fd,                  << if 0 then  formatted pkt contents shall be written to console, -1 otherwise
          FILE *log_file1,                    << Node level Log file ptr into where formatted pkt contents shall be w
          FILE *log_file2,                    << Per interface level Log file ptr into where formatted pkt contents shall be writ
          char *pkt,                          << Pkt pointer to parse
          uint32_t pkt_size,                  << pkt size
          hdr_type_t hdr_type,          << Starting hdr type of the pkt
          char *out_buff,                     << output buffer into which the formatted pkt content would b
          uint32_t write_offset,        << starting position in output buffer to write formatted pkt content
          uint32_t out_buff_size);       << size of output buffer in bytes
```

TCP/IP Stack Project -> Logging Infra -> logging hierarchy



Fn to write the formatted pkt content to output source(s)
: Console
: log file(s)

```
static void
tcp_write_data (int sock_fd,          << 0 to emit on console, -1 otherwise
                FILE *log_file1,      << Node's log file to write formatted pkt content
                     FILE *log_file2,      << NULL
                char *out_buff,       << Buffer which contains formatted output
                     uint32_t buff_size)      << Size of contents in the buffer in bytes
```

TCP/IP Stack Project -> Logging Infra -> CLIs for logging

Added As Assignment

Notification Chains

Notify the Subscribers about the events !

Notification Chains

- Notification Chains is an architectural concept used to notify multiple subscribers interested in the particular event
- A party which generates an Event is called Publisher, and parties which are interested in being notified of the event are called subscribers
- There is one publisher and multiple subscriber
- Once the Event is generated/produced by the Publisher, the Event is pushed to Subscriber(s)
- Subscriber can register and de-register for the event at their will
- Publisher/Subscribers could be any entities :
  - Multiple threads of the same process
  - Multiple processes running on same system
  - Multiple processes running on different systems
  - Different components of the same big software system

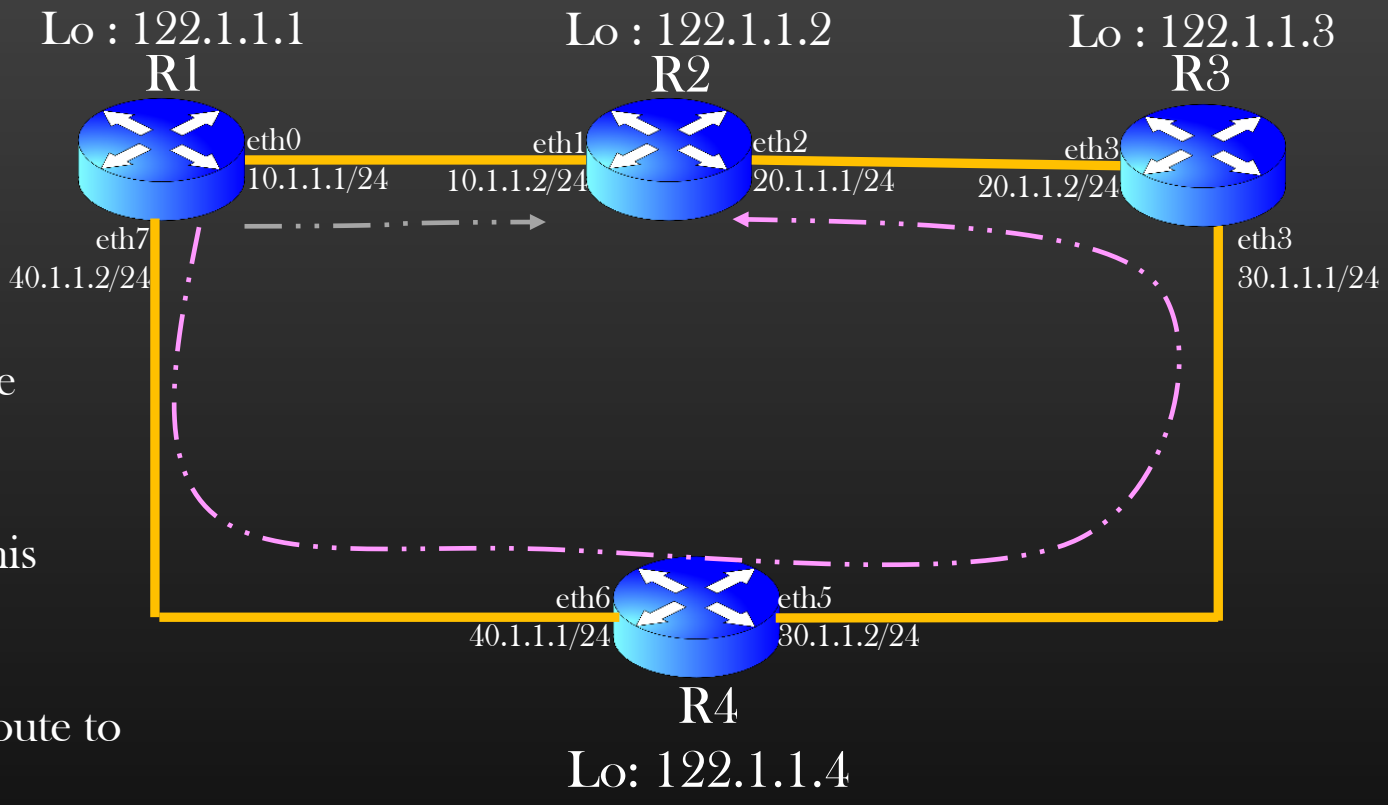
Notification Chains

- We implemented an Application – Spf Algorithm
- An Application (= spf algo as an example) must react to certain common configuration changes or events happening on a networking device
  - Example : Interface level config change by admin
    - IP address change
    - MTU change
    - Enable/disable IP address
    - VLAN config change
    - etc
- We need a mechanism that whenever an admin change such common interface level config, then all interested application must be notified of this change so that they can take appropriate action
- Let me illustrate more ...

Notification Chains

- Consider a topology

build_square_topo( )



- R1's route to R2

122.1.1.2/32	10.1.1.2	eth0	1

- Let us assume, that admin has disabled the interface R1-eth0
- Application Running on R1 must be informed of this event and take corrective measures
- SPF Algo on R1 must re-trigger and compute the route to R2 should be :

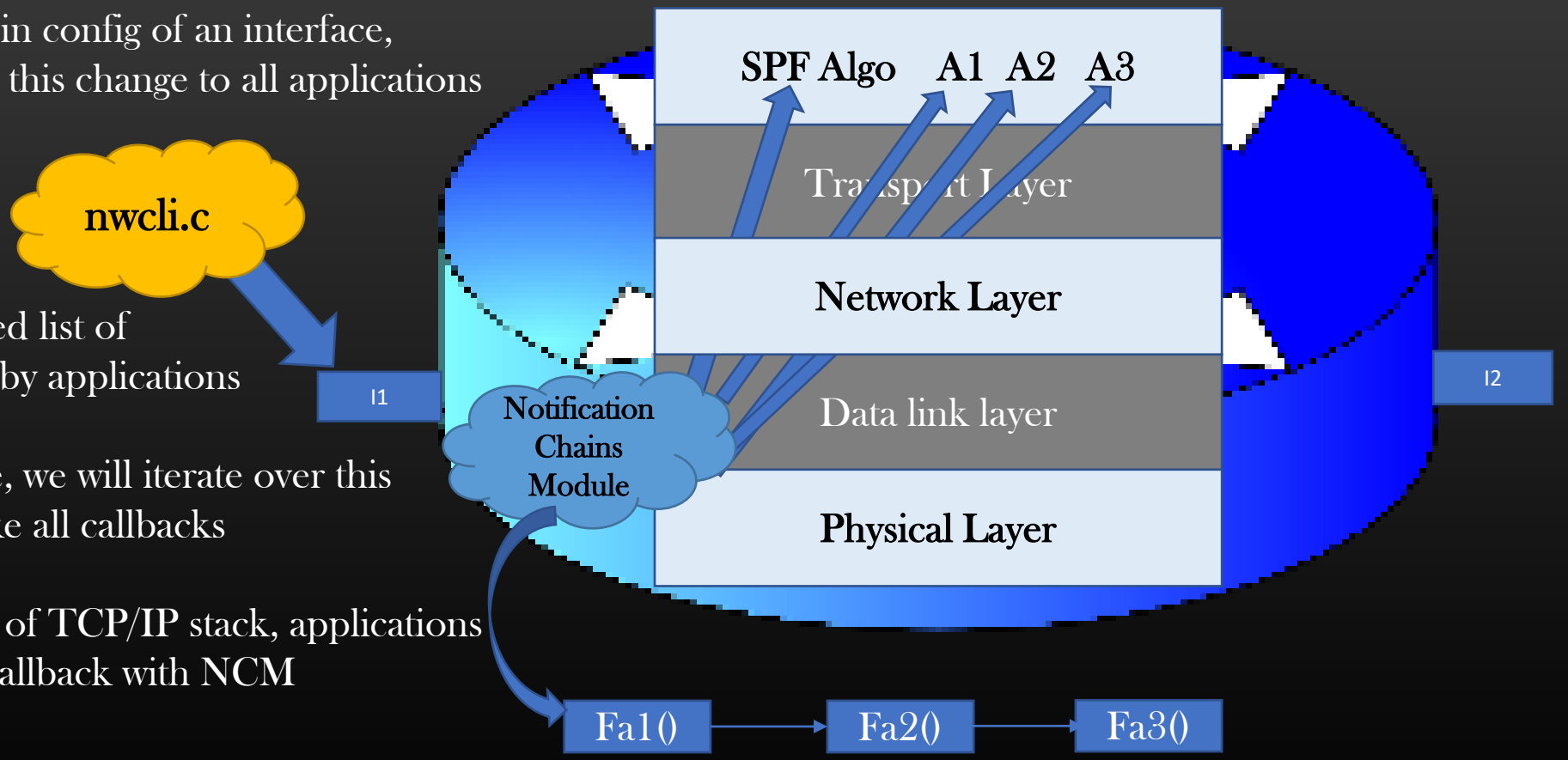
122.1.1.2/32	40.1.1.1	eth7	3

Conclusion : Individual Applications must be notified of generic config change done by admin, We would achieve this through notification chains, a mechanism to distribute events

TCP/IP Stack Project -> Notification Chains

Plan :

- We will implement NCM as a mini-lib
- Whenever user do change in config of an interface,
  NCM will distribute this change to all applications
- Design :
- NCM is nothing but a linked list of
  callbacks registered by applications
- On Interface config change, we will iterate over this
  linked list and invoke all callbacks
- At the time of initialization of TCP/IP stack, applications
  need to register its callback with NCM



TCP/IP Stack Project -> Notification Chains -> Another Working Example

Subscribers

App1
Fa1()

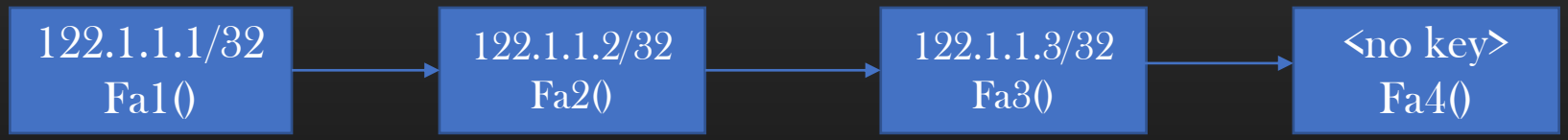
App2
Fa2()

App3
Fa3()

App4
Fa4()

Publisher

122.1.1.1/32	10.1.1.2	eth0	1
122.1.1.2/32	10.1.1.3	eth1	1
122.1.1.2/32	10.1.1.3	eth1	1
122.1.1.2/32	10.1.1.3	eth1	1



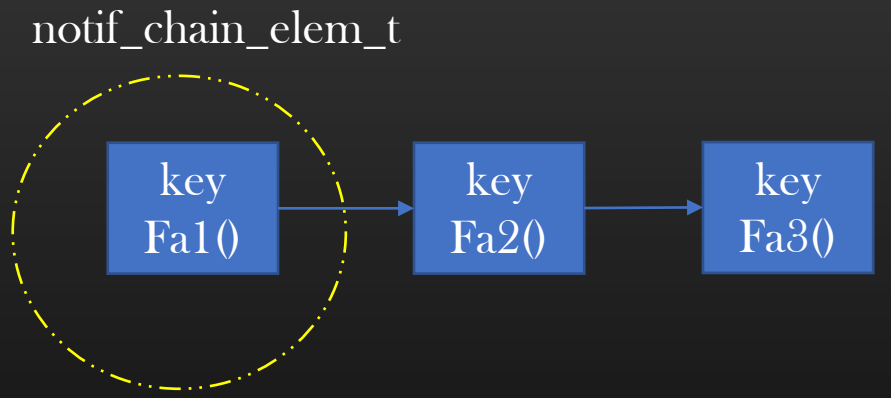
- Let us say, Publisher is the owner of Routing Table (data source)
- Subscribers have registered their callbacks with the publishers against the entries which they are interested in
- Whenever publisher update the entry in its routing table, it iterates over NFC and invoke callbacks of Subscribers matching the entry
- Subscribers who have registered without key will be notified for all updates

We will implement a generic NFC infrastructure in files :

tcpip_stack/notif.c
tcpip_stack/notif.h

- NFC is a linked list of callbacks (function pointers)

```
typedef struct notif_chain_ {

    char nfc_name[64];
    glthread_t notif_chain_head;    << head of linked list
} notif_chain_t;
```




```
typedef struct notif_chain_elem_{

    char key[MAX_NOTIF_KEY_SIZE];
    size_t key_size;
    bool_t is_key_set;
    nfc_app_cb app_cb;
    glthread_t glue;
} notif_chain_elem_t;
```

```
typedef void (*nfc_app_cb)(void *, size_t);
```

We will implement a generic NFC infrastructure in files :

tcpip_stack/notif.c
tcpip_stack/notif.h

- NFC is a linked list of callbacks (function pointers)

```
typedef struct notif_chain_ {

    char nfc_name[64];
    glthread_t notif_chain_head;    << head of linked list
} notif_chain_t;

typedef struct notif_chain_elem_{

    char key[MAX_NOTIF_KEY_SIZE];
    size_t key_size;
    bool_t is_key_set;
    nfc_app_cb app_cb;
    glthread_t glue;
} notif_chain_elem_t;
```

Subscription Request

```
void
nfc_register_notif_chain (notif_chain_t *nfc,
                          notif_chain_elem_t *nfce);
```

Invoke Request

```
void
nfc_invoke_notif_chain (notif_chain_t *nfc,
                        void *arg, size_t arg_size,
                        char *key, size_t key_size);
```

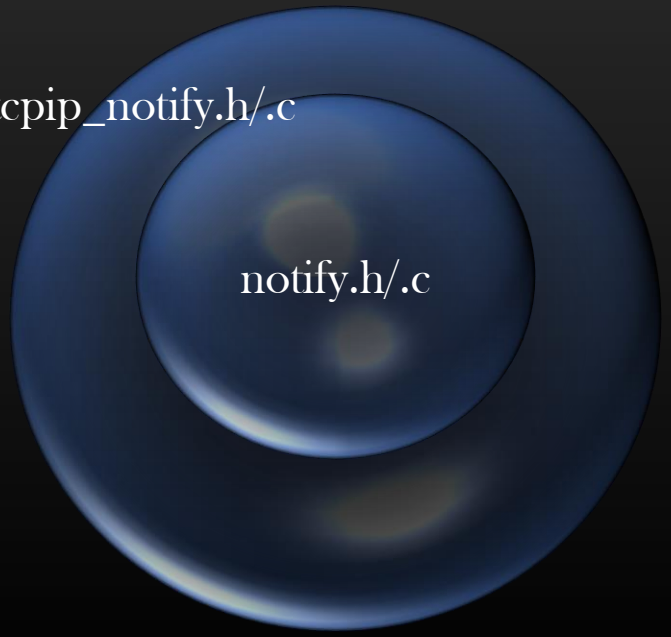
```
typedef void (*nfc_app_cb)(void *, size_t);
```

TCP/IP Stack Project -> Notification Chains -> Interface Events Relay

- Now that we have implemented a generic NFC infra, Now we will implement how any change in interface config done by by the admin can be notified to applications (spf algo) using NFC

- Once you understand this pattern of communication, We shall implement several other notification chains in our project to implement different features
  Eg : how application can tell TCP/IP Stack dynamically that in which protocol packets it is interested in

- We will write two new files which would represent notification Chains specific to our TCP/IP Stack. These two files would be wrapper over generic notif.c/notif.h



To illustrate, let me first implement NFC concept for Interface config change notifications

Once we implement a complete mechanism, then you would Clearly understand and look at the bigger picture regarding how NFC works

Step 1 : Define registration by the Subscriber

Step 2 : Generating notification

Step 3 : Processing notification (by the Subscriber)

Step 1 : Define registration by the Subscriber

```
void
nfc_intf_register_for_events(nfc_app_cb app_cb);
```

- To be invoked by subscriber during init
- For interface notification, subscriber don't have to specify key (the specific intf)
- It means, whenever there will be interface config change, subscriber will be notified irrespective of which interface it is
- It is subscriber responsibility to discard the notif if interface is of no interest
- Eg : For SPF algo, change in interface vlan membership is of no interest to SPF algo since spf algo appln do no work on L2 Interfaces (vlan enabled interfaces)

Step 2 : Generating notification

```
void
nfc_intf_invoke_notification_to_sbscribers(
    interface_t *intf,
    intf_nw_props_t *old_intf_nw_props,
    uint32_t change_flags);
```

- To be invoked by NFC to notify subscriber about change in interface config
- Based on *change_flags* value, subscriber can find out *what is changed*
- This routine packs all data into a structure intf_notif_data_t to be passed as argument to s Callback fn

- Notif need to be generated whenever there is config change on an interface by the user
- Publisher must pack all interested data into a single container (intf_notif_data_t) to be notified to subscriber

Step 3 : Processing notification (by the Subscriber)

- Subscriber's Callback fn will be invoked with notif data as an argument
- Appln must de-wrap the structure intf_notif_data_t and find the data of interest in its callback fn
- Appln can now process the notified data

Timers

Let’s add a dynamic flavor to our Project

TCP/IP Stack Project -> Using Timers

Relevance

Video Already Uploaded

TCP/IP Stack Project -> Using Timers

Plan

- We shall integrate an external timer library to our project
- This timer library is based on POSIX threads, so specific to Unix family Operating systems only
- We shall first go through the steps for library integration, and then we walk over the timer APIs and how to use them
- Timer would give our TCP/IP Stack a dynamic flavor
- Timers are used to implement various Networking based problem statements and their solutions
- We shall then add a minor Timer based functionality to our project

TCP/IP Stack Project -> Using Timers -> Lib Integration

Timer Library Integration Steps

- Download these two files :
  - https://github.com/sachinites/tcpip_stack/blob/partB/WheelTimer/WheelTimer.c
  - https://github.com/sachinites/tcpip_stack/blob/partB/WheelTimer/WheelTimer.h

- Place in directory :
  - tcpip_stack/WheelTimer

- Update Makefile

  - Add to OBJS list : WheelTimer/WheelTimer.o

  - Add compilation Rule :

```
WheelTimer/WheelTimer.o:WheelTimer/WheelTimer.c
${CC} ${CFLAGS} -c -I gluethread -I WheelTimer WheelTimer/WheelTimer.c -o WheelTimer/WheelTimer.o
```

  > Add clean and cleanall rule

```
rm -f WheelTimer/WheelTimer.o
```

TCP/IP Stack Project -> Dynamic ARP Tables

Dynamic ARP Tables

- Properties :
  - Currently ARP table entry is static – meaning once installed it shall be there for forever in ARP Table
  - ARP Table entry my go obsolete as topology changes, and therefore, ARP table entries which are useless should be deleted from ARP table automatically
  - Install ARP Table entry with the Expiration time of 30 sec
  - If ARP Table entry is not used for traffic forwarding for 30 sec, delete it
  - If ARP table entry is used for traffic forwarding, refresh its expiration timer back to 30 sec again
  - Enhance show node <node-name> arp to show expiration time arp entry as an additional field

Get Familiar with Timer Library

- Go through Section Appendix B
- Go to next video only after you have completed Appendix B
- Next we shall proceed to implement dynamic ARP tables

Dynamic ARP Tables

- Solution Steps :
  - Each Node in the topology, must have its own Wheel-Timer instance running
  - A Node will schedule all its events using its own Wheel-Timer instance

- Now that each ARP table entry is associated with the timer, we need to add below member :

```
struct arp_entry_{
        . . .
        . . .
        wheel_timer_elem_t *exp_timer_wt_elem;  /* This represents that this ARP entry is being tracked by Timer now */
};
```

Dynamic ARP Tables

New APIs to be written in layer2.h/layer2.c to handle ARP entry's timer

1
```
wheel_timer_elem_t *
arp_entry_create_expiration_timer(
        node_t *node,
        arp_entry_t *arp_entry,
        uint16_t exp_time);
```

- Should be called when ARP entry is installed in routing table for the first time

2
```
void
arp_entry_delete_expiration_timer(
        arp_entry_t *arp_entry);
```

- Should be called when ARP entry is deleted from ARP Table

3
```
void
arp_entry_refresh_expiration_timer(
        arp_entry_t *arp_entry);
```

- Should be called whenever ARP entry is referenced for traffic forwarding Or
- Whenever the ARP entry status changed from Sane to Resolved

4
```
uint16_t
arp_entry_get_exp_time_left(arp_entry_t *arp_entry);
```

- Should be used while displaying the arp entry and showing how many seconds remaining for it to expire

Sequel Course Part C (Final)

Developing TCP/IP Stack Part C

- How to Implement a new Network Application in Application Layer
  - Dynamic Protocol Registration

- Sample Network Applications We shall Develop :
  - NMP Protocol (Nbrship Mgmt Protocol)
    - Extensive use of Timers
    - Election Algorithm

  - DDCP (Distributed Data Collection Protocol)
    - Concept of TLVs
    - Packet flooding
    - Gathering Replies from different nodes

Pre-Requisite :

Must have completed Part A and B

Dynamic Protocol Registration

Notification Chains

TCP/IP Stack Project -> Dynamic Protocol Registration

- We shall be going to implement a mechanism using which an application can express interest in the packets it is interested in receiving from underlying TCP/IP stack

- The Application must be able to :
  - Tell our TCP/IP Stack that in which packets (ip_hdr->protocol for L3 pkts & ethernet_hdr->type for L2 pkts) the application is interested in
  - Our TCP/IP Stack should handover the pkt to the application based on protocol (&type) fields in IP Hdr (& ethernet hdr)
  - I will show the implementation for ip_hdr->protocol field, you need to complete it for ethernet_hdr->type field as a part of assignment
  - You should be able to realize that it implementation should make use of Notification Chains as it is another typical case of publisher (Layer3/Layer2) and Subscribers (Applications)
  - Let us understand the problem statement pictorially

TCP/IP Stack Project -> Dynamic Protocol Registration

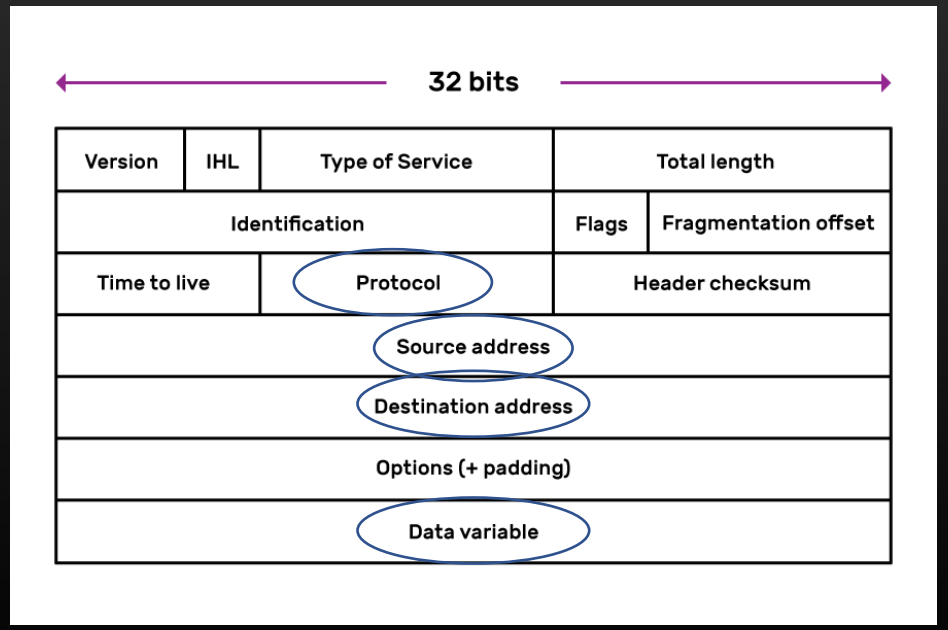
- Let us say, we have some dummy appln - ddcp (Distributed Data Collection) running in application layer
- This application works with two types of packets :
  - DDCP_MSG_TYPE_FLOOD_QUERY
    - This is L2 packet, meaning the pkt is encapsulated within ethernet hdr

Dst MAC	Src MAC	Type	Payload	FCS

    - Dst Mac : ff:ff:ff:ff:ff:ff
    - Src Mac : 00:00:00:00:00:00
    - Type = DDCP_MSG_TYPE_FLOOD_QUERY
    - Payload : DDCP Application Data
    - FCS = 0
  - DDCP_MSG_TYPE_UCAST_REPLY
    - This is L3 pkt , meaning the pkt is encapsulated within IP hdr

TCP/IP Stack Project -> Dynamic Protocol Registration

- Let us say, we have some dummy appln - ddcp (Distributed Data Collection) running in application layer
- This application works with two types of packets :
  - DDCP_MSG_TYPE_FLOOD_QUERY
    - This is L2 packet, meaning the pkt is encapsulated within ethernet hdr
  - DDCP_MSG_TYPE_UCAST_REPLY
    - This is L3 pkt , meaning the pkt is encapsulated within IP hdr



- Protocol : DDCP_MSG_TYPE_UCAST_REPLY
- Src Addr : Lo address of sender
- Recv Addr : lo addr of Receiver

Using Timers

Learn Integrate Timer Library in 30 Minutes

Using Timers

- Goal of this Course :

  - To quickly take a look how we can integrate a ready-made timer library with our application and start using timers straightaway on linux systems

    > No discussion on internal Implementation

Using Timers

Src Code

git clone https://github.com/sachinites/WheelTimer
Dir : WheelTimer/WheelTimer/
Files : WheelTimer.c, WheelTimer.h
gluethread/glthread.h, gluethread/glthread.c
Demo : main.c
Compile : gcc -g main.c gluethread/glthread.c WheelTimer.c -o exe -lpthread

Timer Basics and Terminologies

- Suppose at t = 0 , you start a timer of 10 sec to send out a packet to remote machine
  - At t = 0 , timer is started
  - At t = 10, timer is fired Or timer is expired
  - Timer was running from t = 0 to t = 10

Using Timers

Working with Timers

Application

wt = init_wheel_timer( )
start_wheel_timer(wt);

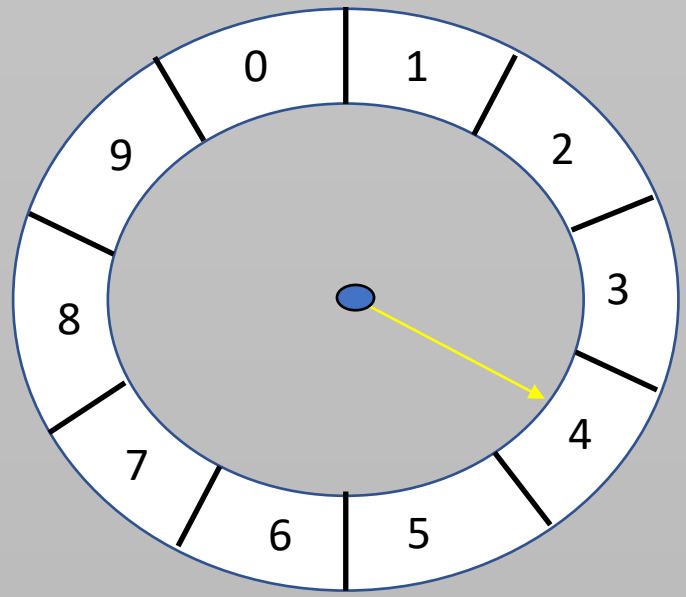
wt_elem = register_app_event(wt, . . .)

A handle which application has to cache to perform future operations on this event

de_register_app_event (wt, wt_elem);

wt_elem = NULL;

TimerLibrary (separate thread)



Using Timers

Application Callback

- The Timer library does the application's work by invoking the Application fn
- This Application fn is called 'timer callback'

Prototype : void (*fn)(void *, int);

```
Eg :   void print_string(void *arg, int arg_size) {

                    char *data = (char *)arg;
                    int string_len = arg_size;
                    printf("String = %s\n", data);
       }
```

- Application should write the fn of this prototype to be register with the timer
- Now let us see timer in Action..

Using Timers

Using Timer Library

1
```
/* Creating a new timer */

wheel_timer_t*
init_wheel_timer(int wheel_size, int clock_tic_interval);
```

2
```
/* Starting the new timer */

void
start_wheel_timer(wheel_timer_t *wt);
```

3
```
/*Scheduling event with the timer */

wheel_timer_elem_t *
register_app_event(wheel_timer_t *wt,
        app_call_back call_back,
        void *arg,
        int arg_size,
        int time_interval,
        char is_recursive);
```

4
```
/* De-register the already scheduled event */

void
de_register_app_event (wheel_timer_t *wt,
                       wheel_timer_elem_t *wt_elem);
```

```
/* Reschedule event again */
```

5
```
void
wt_elem_reschedule (wheel_timer_t *wt,
                    wheel_timer_elem_t *wt_elem,
                    int new_time_interval);
```

```
/* Get Remaining time left for the event to fire */
```

6
```
int
wt_get_remaining_time(wheel_timer_t *wt,
                      wheel_timer_elem_t *wt_elem);
```

```
/* Stop the timer */
```

7
```
void
cancel_wheel_timer (wheel_timer_t *wt);
```

Using Timers

Conclusion

- We Quickly (~30 min) toured the API to make use of timers
- Quickly Integrate the timer library with your projects and start using !
- Assignment
  - Src Code : https://github.com/sachinites/WheelTimer/WheelTimer/Assignment/

Using Timers

Solution Steps

1. Create a Wheel Timer instance for a Routing Table

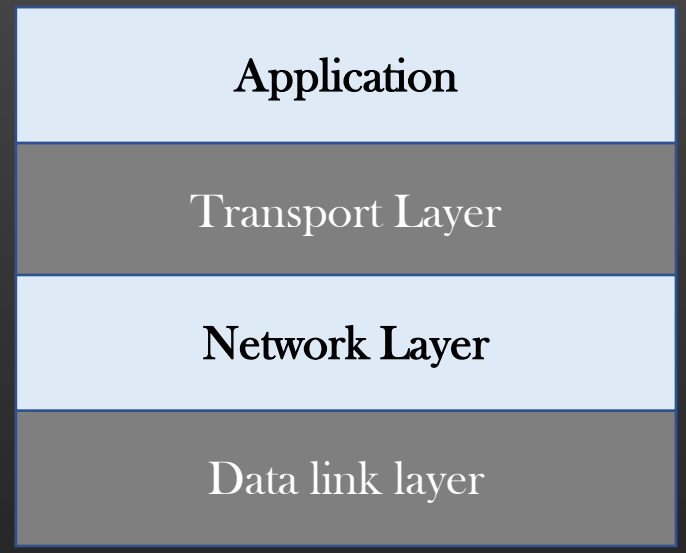
2. Add a wheel_timer_elem_t member to the rt entry. This member keeps track of the exp timer of the entry

3. Add the time of installing the rt entry into routing Table, schedule the expiration of the rt entry with the Wheel timer

4. Write a new timer callback function. This fn shall be invoked by WheelTimer on timer expiry. This fn should delete the rt entry after un-scheduling the rt entry with the WheelTimer

Agenda

Developing TCP/IP Stack Part B



1. Interface Management & statistics
2. Dynamic L3 Route Calculation
   No More Manual installation of L3 routes
3. Making TCP/IP stack dynamic
   Dynamic ARP table
4. Sample L2 Layer Application & Working with Timers
5. Programmable TCP/IP Stack
6. Develop Logging Infra
   Packet Captures

Thank You !

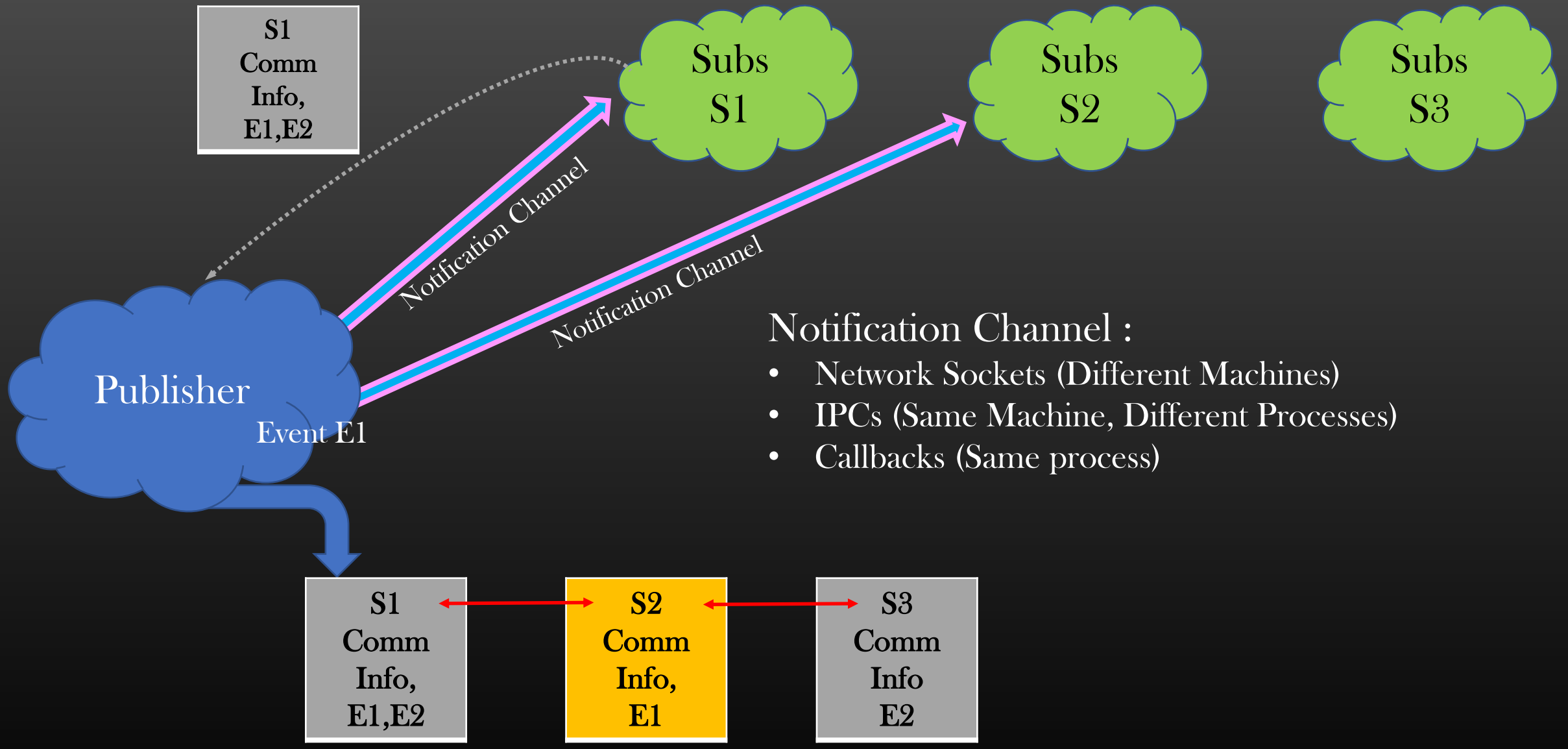
Notification Chains

Notify the Subscribers about the events !

Notification Chains

- Notification Chains is an architectural concept used to notify multiple subscribers interested in the particular event
- A party which generates an Event is called Publisher, and parties which are interested in being notified of the event are called subscribers
- There is one publisher and multiple subscriber
- Once the Event is generated/produced by the Publisher, the Event is pushed to Subscriber(s)
- Subscriber can register and de-register for the event at their will
- Publisher/Subscribers could be any entities :
  - Multiple threads of the same process
  - Multiple processes running on same system
  - Multiple processes running on different systems
  - Different components of the same big software system

Notification Chains -> Pictorial Representation

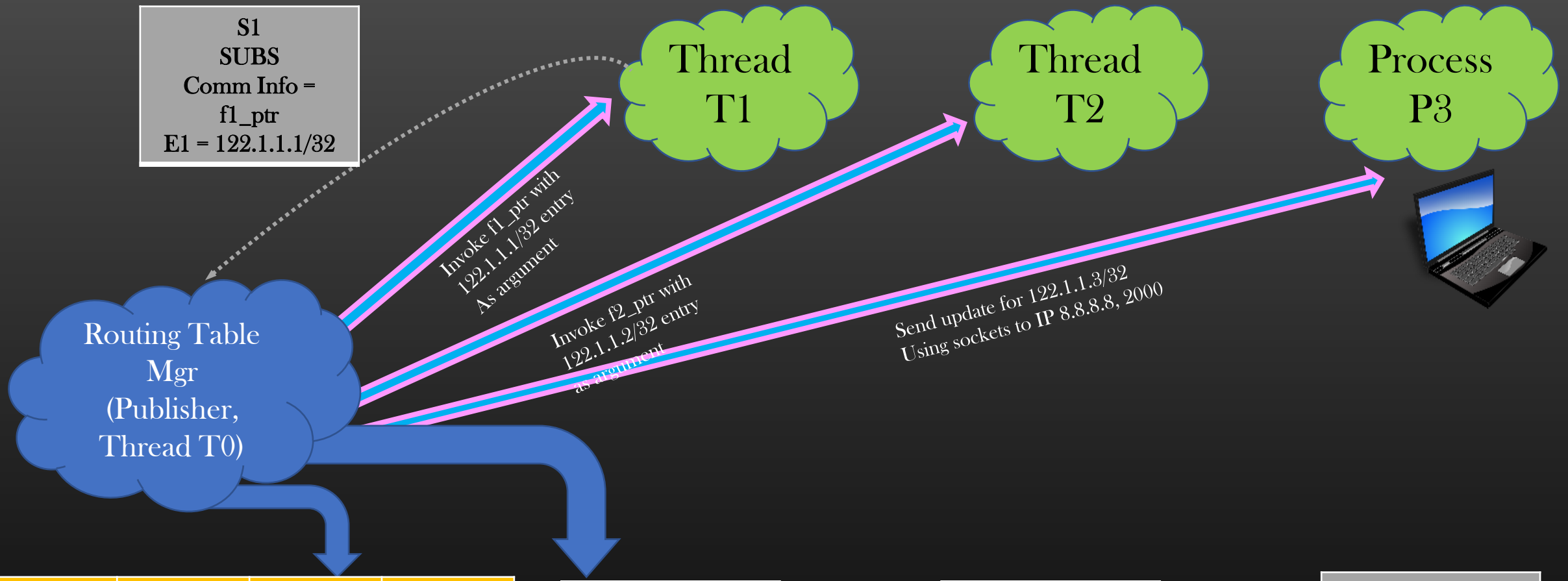


Notification Channel :

- Network Sockets (Different Machines)
- IPCs (Same Machine, Different Processes)
- Callbacks (Same process)

Let us discuss one realistic Example !

Notification Chains -> Pictorial Representation



Dest	Mask	OIF	Gw
122.1.1.1	32	Eth0	10.1.1.1
122.1.1.2	32	Eth1	10.1.1.2
122.1.1.3	32	Eth2	10.1.1.3

S1
SUBS
Comm Info =
f1_ptr
E1 = 122.1.1.1/32

S2
SUBS
Comm Info =
f2_ptr
E1 = 122.1.1.2/32

S3
SUBS
Comm Info =
“8.8.8.8, 2000”
E1 = 122.1.1.3/32

How Subscribers Subscribe and unsubscribe with publisher ?

Notification Chains -> Project Pre-requisites

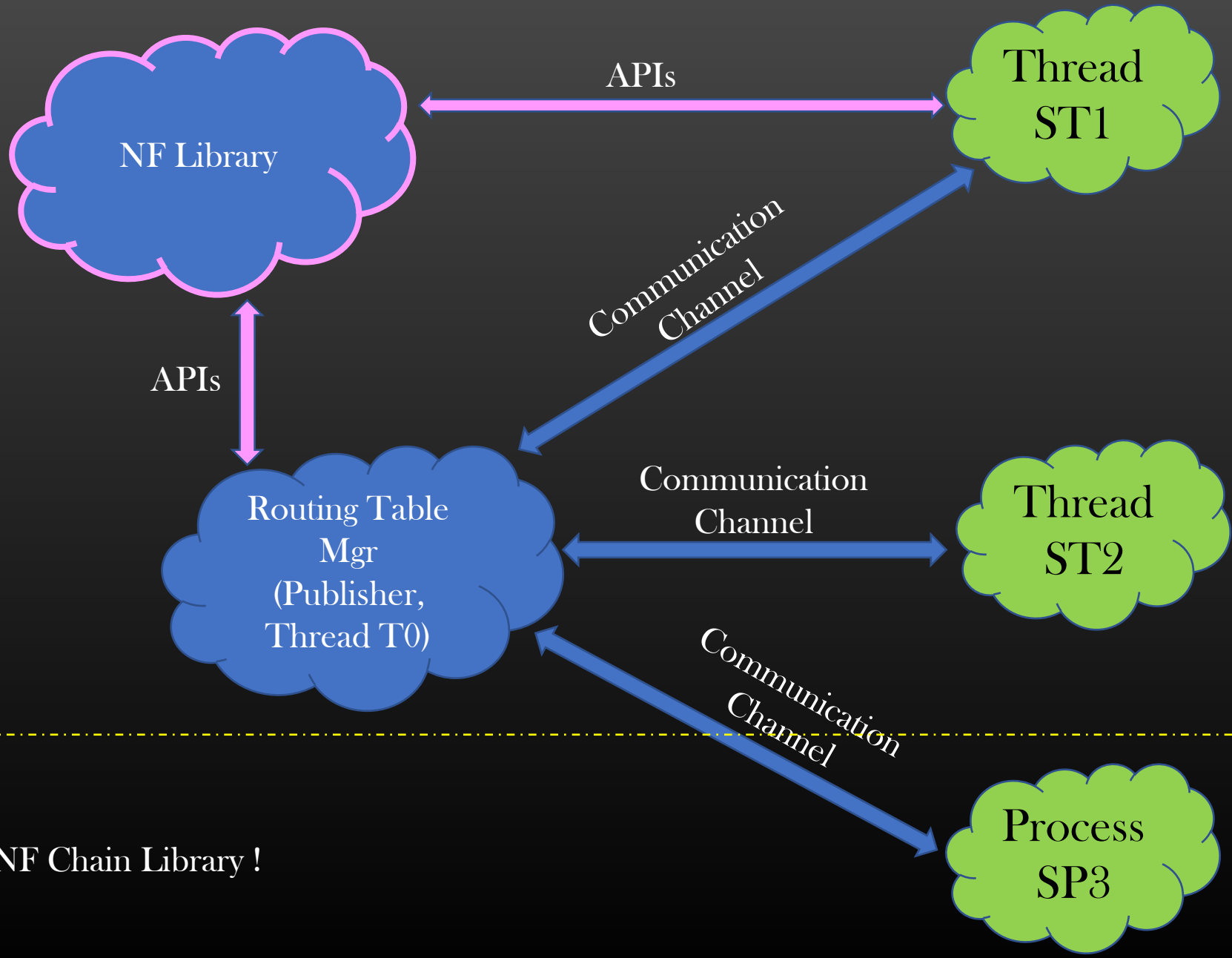
- We will be going to implement a Library which implements Notification Chains functionality
  - Pre-requisites
    - IPC
      - Socket Programming
      - Function Pointers
      - Good Command on C/C++
      - MsgQs, Unix Domain Sockets (optional)
  - Expected LOCs
    - 1500 – 2000 (if you implement Fully)
    - Level of Difficulty : Medium
  - It's a concept – So you are free to implement in your fav Programming language, But I will use C for Demo on Linux
  - NF is used all over the industry (Publisher Subs Model is very common Design pattern of communication)
  - Create your github account if not already, Do all your coding on github
  - Along the way you will learn more new programming concepts
    - TLVs, Timers, KeepAlive msgs

Notification Chains -> Project Components

APIs to be provided by the NF Lib:
Publisher Use :
Create NF chains
Send Notification to Subs

Subscriber Use:
Register/Unregister for Events

Common:
Data Serialization/Deserialization



One Single Process

The End product of this Course is a NF Chain Library !

Notification Chains -> Project Code Access

- All Codes Accessible at
  - https://github.com/sachinites/NotificationChains
    - /CompleteProject (Complete solved Project, use it for reference)
    - /PubSubModelDemo (Simplified, we develop in this course)

Let's Start ... ! ☺

Notification Chains -> Development Phases

Phase 1 : Writing a Sample Publisher Code which is in-charge of sample data source

Phase 2 : Notification Chain Data Structures

Phase 3 : Writing Sample Subscriber with Communication Channel as Network Sockets

Phase 4 : (De)-Registration Procedure

Phase 5 : Sending Event notification from Publisher to Subscriber

Phase 6 : Detecting Subscriber Liveness

Phase 7 : Serialized Communication using TLVs

Phase 8 : What if Publisher Dies ?

Notification Chains -> Phase 1 -> Publisher

- A Publisher is an entity which is in-charge/owner of the data source
- In our Project, We will create a Publisher – Routing Table Manager who is in-charge of Routing table, a Data source

Dest	Mask	OIF	Gw
122.1.1.1	32	Eth0	10.1.1.1
122.1.1.2	32	Eth1	10.1.1.2
122.1.1.3	32	Eth2	10.1.1.3

- A publisher can modify the data source at its will
- Publisher must notify the update of its data source to all subscribers who have subscribed for the update
- Publisher can be in-charge of multiple data sources

Notification Chains -> Phase 1 -> Publisher

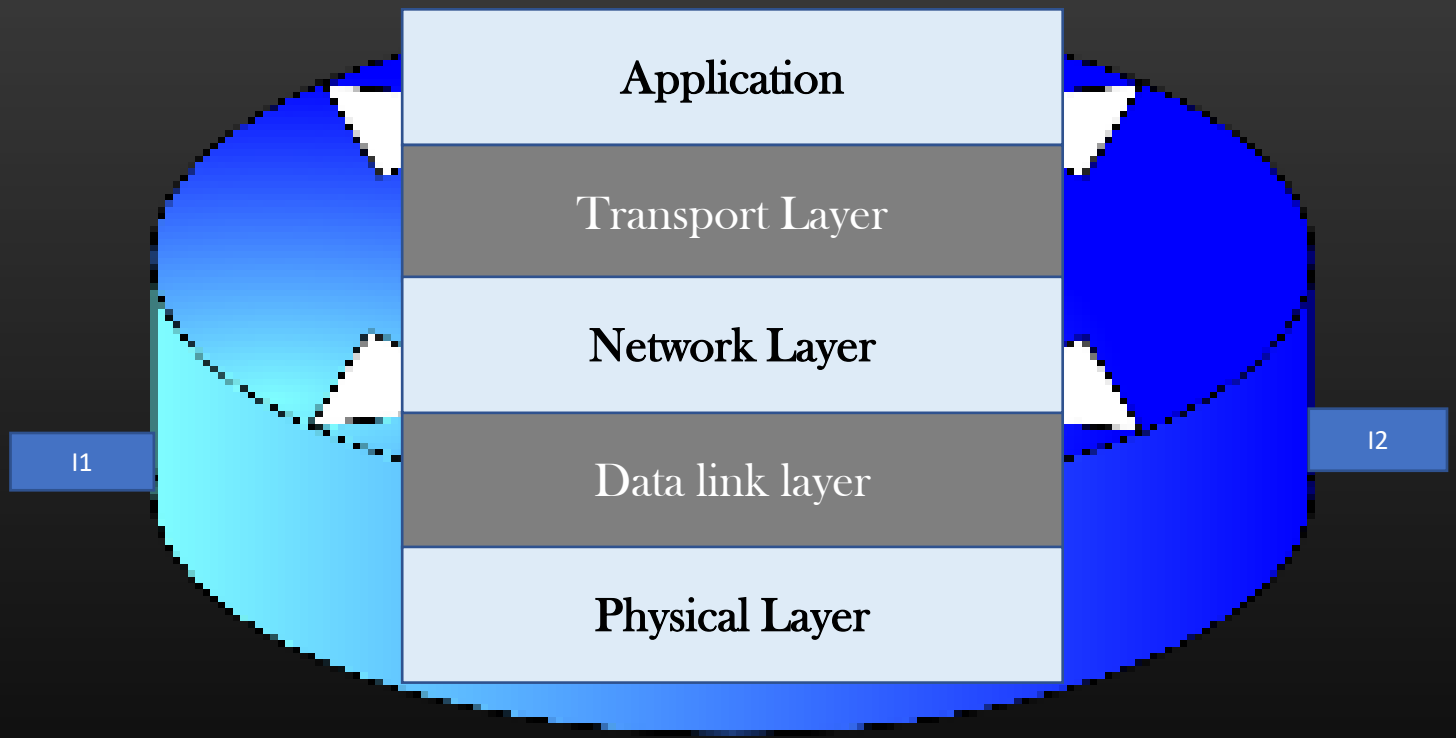
Data Source : rt.h , rt.c

```
typedef struct rt_entry_keys_{

    char dest[16];
    char mask;
} rt_entry_keys_t;

typedef struct rt_entry_{

   rt_entry_keys_t rt_entry_keys;

    char gw_ip[16];
    char oif[32];
    struct rt_entry_  *prev;
    struct rt_entry_  *next;
} rt_entry_t;

typedef struct rt_table_{

    rt_entry_t *head;
} rt_table_t;
```

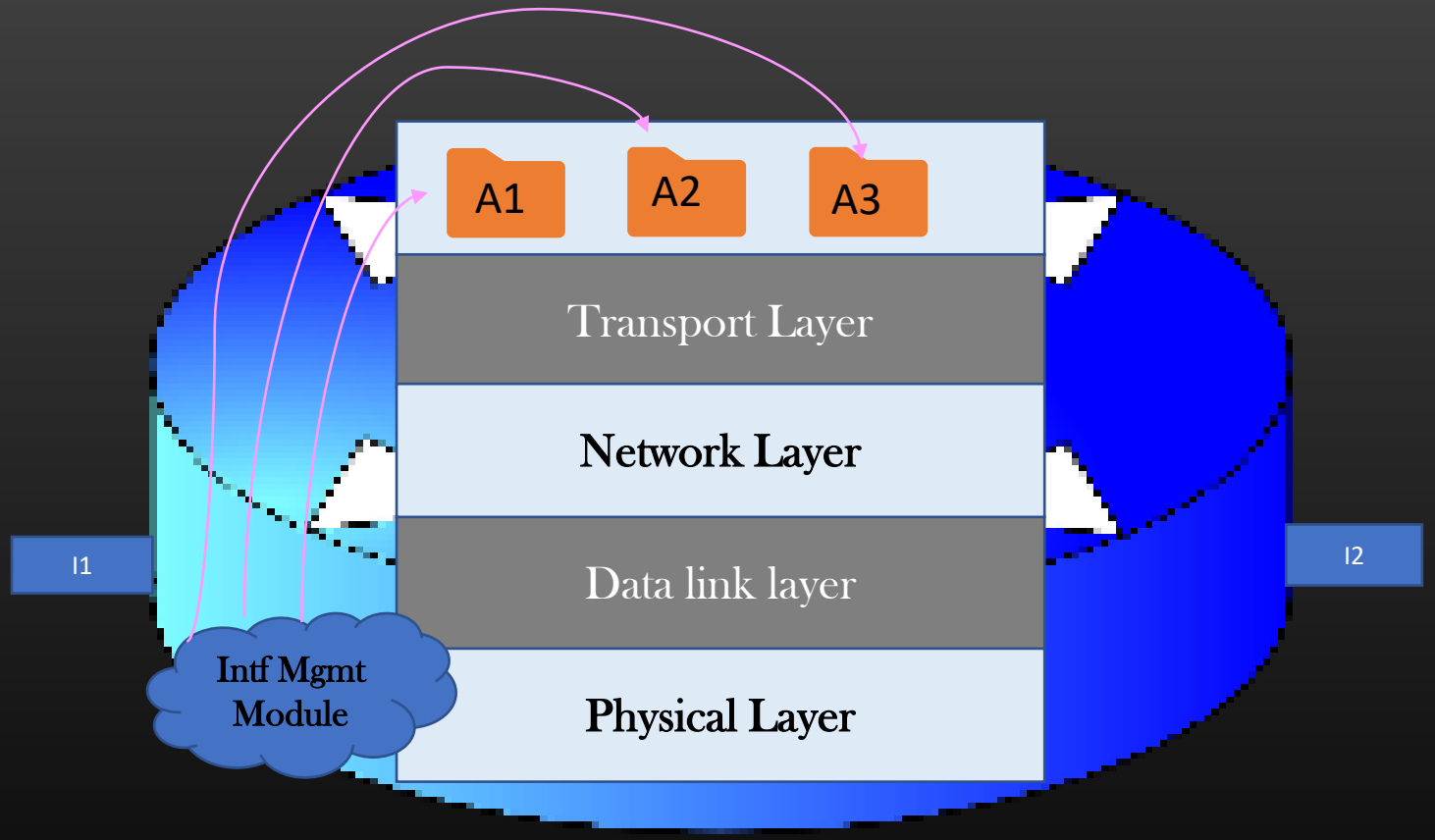
Dest	Mask	OIF	Gw
122.1.1.1	32	Eth0	10.1.1.1
122.1.1.2	32	Eth1	10.1.1.2
122.1.1.3	32	Eth2	10.1.1.3

APIs : CRUD

Notification Chains -> Realistic Example



Notification Chains -> Realistic Example



When Admin Shut down an interface I1:

- A1, A2 may want to stop sending hello pkts
- A3 may want to trigger some algorithm

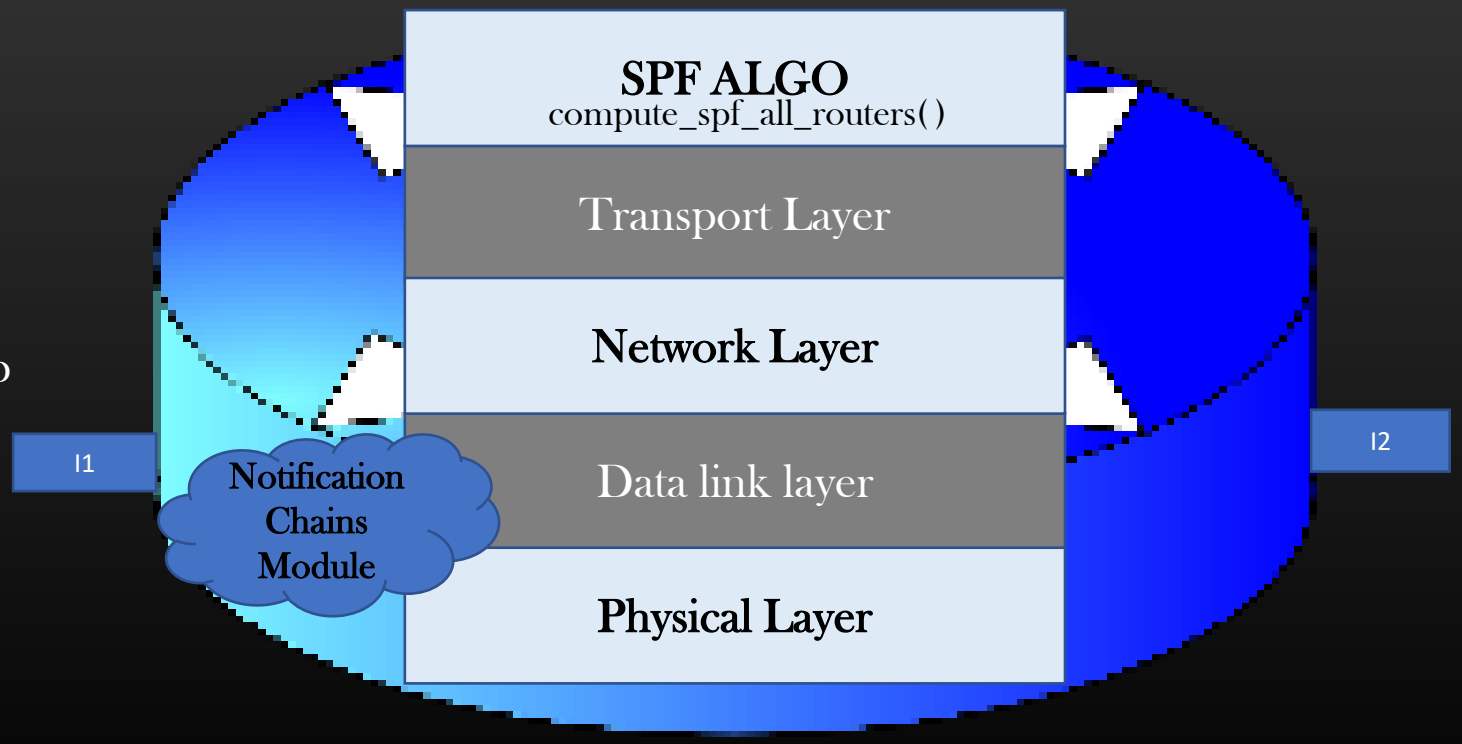
Publisher : Intf mgmt Module (Kernel space)
Subscribers : Application (User Space)
Communication Channel :
USA < - - > KS IPC techniques

TCP/IP Stack Project -> Notification Chains

- In the prev section, we noticed that every time the user changes the physical structure of the topology, user need to manually trigger spf algo using “run spf all” to update the Routing tables

- Our Goal is :
  - Whenever user does anything which changes the physical topology and require update of routes, then SPF algorithm must trigger automatically
    - Events :
      - Intf up/down
      - Intf IP Address Change
      - Link metric change
      - Etc ..

- We will implement NC in our TCP/IP stack lib
  - Publisher : Interface Backend handlers
  - Subscriber : Applns(spf.c)
  - Communication Channel :
    - Appln callbacks

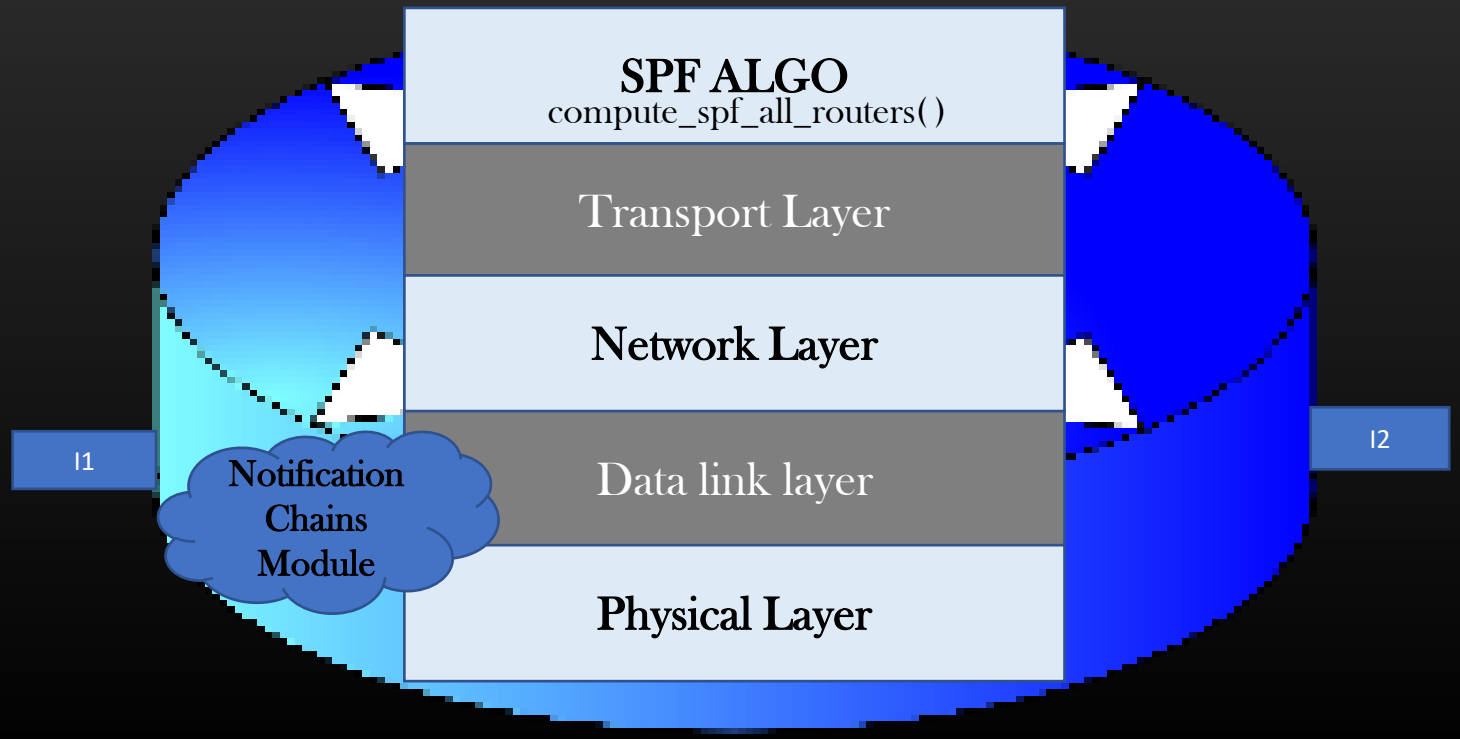


TCP/IP Stack Project -> Notification Chains

- Todo :
  - How SPF algorithm (any application) Express its interest with NCM in specifically interface up/down events ?
    - SPF algo may have no interest if user changes the MTU or bandwidth of the interface
  - How Application can unregister or register for more new events dynamically with NCM ?
  - How NCM notifies the exact event to Spf Algo appln ?
    - Is it IP address changes Or link metric Change Or interface enable/disable ?

Plan :

1. First we will implement NCM as a mini-lib
2. Test it with dummy publisher/subscriber
3. Will integrate this library with TCP/IP stack lib



TCP/IP Stack Project -> Notification Chains

Notification Chains -> Design and Implementation

- We Create one instance of notification Chain per Event type
- For example :
  - Publisher generates Events E1, E2 and E3
  - There could be different set of subscribers interested in each events
  - All Subscribers interested in particular event are grouped together under one notification chain
  - Though subscribers subscribed for same Event E, may choose different communication channels to be notified

Notification Chains

Thank you