Network Project
Implement
A Network Protocol from Scratch
In C

Ospf
Stp
Isis
Mpls
Ldp
RSVP
Bgp
RIP
AAA
IPSec
ICMP
IGMP
MLD
PIM
. . .

For Aspiring Core Network Dev Engineers/System Programmers ☞ Exclusively for Developers !

Level : Intermediate+

Programming language used : C

Course Objective

- The AIM of this project is to cycle you through the experience of end-to-end implementation of a typical network protocol
- You will be writing lots of C code to Implement a typical Network Protocols and its features
- Design, Document and implement new features (sub projects)
- Introduce to the world of Asynchronous programming, Timers, Packet Parsing
- Learn to use external libraries , See and understand alien code
- Focus is on solving Network Core problem – Think, design, implement and analyze the solution
- See your implementation solving real world networking problems
- This course is rehearsal of what you shall be doing as a Software Engineer at core network companies
- Same Design and implementation as a typical network Protocol is implemented in industry on a device
- Decorate your resume with an impressive project
- The project is Challenging – Sky is the limit. Expected LOCs – 20k+

Take Away

- After doing this course , you will be able to
  - Understand how Network Protocols are implemented on Network Devices
  - Understand end-to-end development of a network protocol
    - Config via CLIs, show CLIs
    - Implement new features incrementally
    - React to config changes
    - React to common admin actions such as interface shut down / IP Address change
    - Packet processing, Update protocol data structures through packets processing
    - Implement complex protocol state machines and Network Algorithms
    - Control protocol behavior based on timers
    - Debugging and troubleshoot code to resolve issues
    - Add another feather to your resume

What this Course is not

- This Course is not a :
  - Learning Programming Language or Data Structures
  - Not a Socket Programming Course
  - Not a Linux system programming learning course

Audience

- Core Developers aiming to work in Networking / Distributed Systems / System Programming side
- Not for those seeking non-development roles (This is pure Dev oriented Course)
- Not for those still struggling with basic data structures, basic C programming concepts
- Working professionals, Job Seekers, domain changers to Networking Dev, Learners, starving for knowledge
- Patience, fighters, challengers, winners

Prerequisites

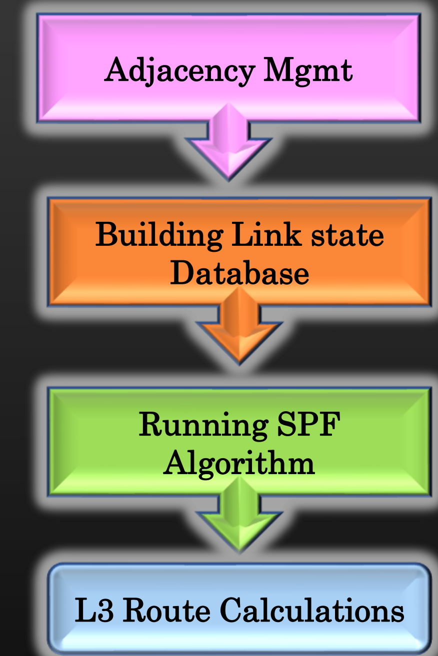
- Basic L2 and L3 Networking knowledge
- C Programming
  - Should be excellent with pointers, pointer arithmetic
  - Must understand memory manipulations in C programming
    - memcpy, memcmp, byte layout in memory
  - Type-casting, Multi-threading
  - Basic Data structures knowledge –
    - Linked list
    - Trees
  - Rest I will take Care

Good luck , lets start . . .

Agenda

- We will be going to implement a simplified Routing Protocol in this course
- Routing protocol chosen – Interior gateway protocol (IGP , ex OSPF, ISIS)
  - Don't know about it – don't worry, we shall cover theory first before any implementation
- A typical IGP (link state) protocol functionality is divided into 4 distinct parts :

1. Adjacency Management (Each device know its neighbours)
   - Sending and Receiving hello packets periodically
   - Update neighborship state machine
2. Building Link State Database (Each device internally creates a view of topology - Graph)
   - Building Link State packets
   - Flooding link state packets
   - Build a Graph – a view of network topology
3. Running SPF algorithm (Dijkstra) on LSDB
   - Process the LSDB through the algorithm
   - Compute Results and store
   - Algorithmically challenging
4. L3 Route Calculations
   - Use Results of 3 to compute final L3 routes and update Routing Table
   - Algorithmically challenging

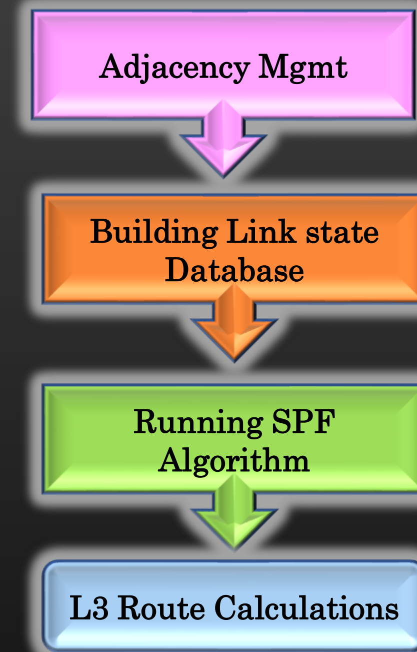


We shall be going to implement all 4 parts in this course series

Along the journey we shall implement various sub-features within the protocol

Agenda

- Implementation divided into 4 parts
- Part X is dependent on part X -1
- Additionally, our protocol must respond to other network events – link failures, config changes etc
- We must ensure concreteness of current phase of implementation of the protocol before moving to next
- It is a big project – Expected LOCs 20k+
- I will provide you library which simulate a topology of L3 routers on top of which you are required to implement your protocol and test
- Industry level Coding standard and experience
- Same Design and implementation as a typical network Protocol is implemented in industry on a device
- You will not do any socket programming !
- Sky is the limit
- Operating system Used : Linux (Ubuntu 20.04 LTS)
- Language : C (not even C++)



Challenges Ahead !

- Memory Corruptions
- Memory Leaks
- Functionality breakages
- Crashes
- Not able to Code the logic !
- Code Reading and Remembering the flows

- Capture and Analyze packets ingressing/egressing devices
- Narrow the down the problem from topology level to device level
- Analyze logs/ insert more logs as required
- Find MRE (Minimal Reproducible Example)
- Debuggers – gdb, Core-files , Valgrind
- Code Navigator Tools – Source Insight (best AFAIK)
- Deliver and test incrementally
- Code maintenance : Use github (or similar)

Project Road Map

Schooling !

Get familiar with TCP/IP Stack Library

- Building , Compiling, Running
- Packet Captures
- Build Topologies
- Config Topology –
  - Ip Address Change
  - Link up down , etc
- Collect log files
- Understand the supporting dev libraries
- Understand the library design and arch (high level)

College !

Let's Cover the theory of the Protocol we are looking to implement

- What is the protocol ?
- How complicated ?
- How does it work ?
- Development phases ?
- Complete Theory
- Get Complete Idea about protocol logistics

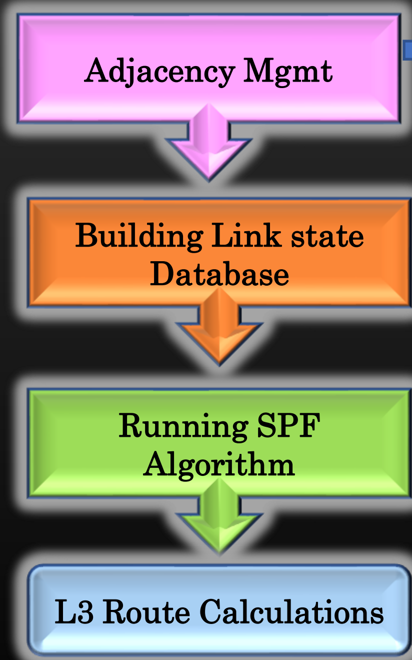
At Work !

Development Begins !!

- Adding new CLIs
- Implement Protocol Core logic
- Config protocol
- Troubleshoot
- Observe end results
- Enhancements
- Bug fixing
- Code maintenance
- Critical Analysis
- Etc . .

How to do this Course ?

- Do all necessary theory before hitting the keyboard
- Do all assignment religiously
- Use github (mandatory , do some 30 min basic tutorial – that suffice)
- Use tools for troubleshoot – gdb, valgrind, traces, printfs etc …
- Ask in Udemy QnA Or join telegram grp : telecsepracticals
- Test thoroughly before proceeding to next



- First Module (12 hr+) , Total HRs – 30+hrs
- Setup the development environment
- know the libraries to be used
- Know where to find which information
  - Cheat sheet, Code browse etc ..
- Instructor shall be writing all codes from scratch
- As we progress, students will tend to take driving seat
  - Implementing logic on their own
  - Implementing new sub-features features

Get Familiar with TCP/IP Stack Library → Download Src Code

- TCP/IP Stack Library User Guide
  https://drive.google.com/open?id=1KfSlLIeS9WSqkcJRhJyFUp9owX4MRNEp

- TCP/IP Stack library will allow us to build the Topology of L3 routers and L2 switches quickly

- Library provides the software simulation of Devices and topology, tested on linux (ubuntu 20.04)

- Download project Code :
  - Using git (Recommended)
    - Repo : https://github.com/sachinites/tcpip_stack
    - Fork my Repository into your git account and git clone your version of repository
      - git clone https://github.com/<your user name>/tcpip_stack
      - git checkout proto-dev

- Report any bugs to :
  - Open a case on git https://github.com/sachinites/tcpip_stack
  - Send email to Udemy QnA or email me sachinites@gmail.com

Setup Code Navigation Tool

Code Navigation Tool : Source Insight (Link in Resource section)

> v 3.5 with key (v 4.0 and above is paid and key is not available)

> Windows only

> Mac Users (Try something else … cscope … etc)

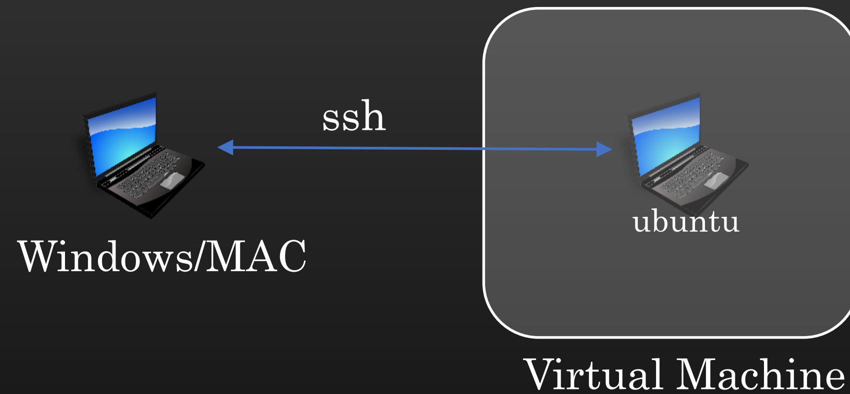
Get 30 days free Access to all our Courses :
https://csepracticals.teachable.com/p/trial-goldmine

Get Familiar with TCP/IP Stack Library → Launching a Pre-built Topology

☞ Refer to DEBUGGING section
In cheat sheet

- All topologies present in tcpip_stack/topologies.c
- main() is in testapp.c , check which topology is currently running there
- Get familiar with CLI interface
  - show topology
  - ping
  - show route and ARP tables
  - show interface stats
  - packet capture and logging
  - packet gen executable
- For this Course, don't run topology with VLANs (Limitation)

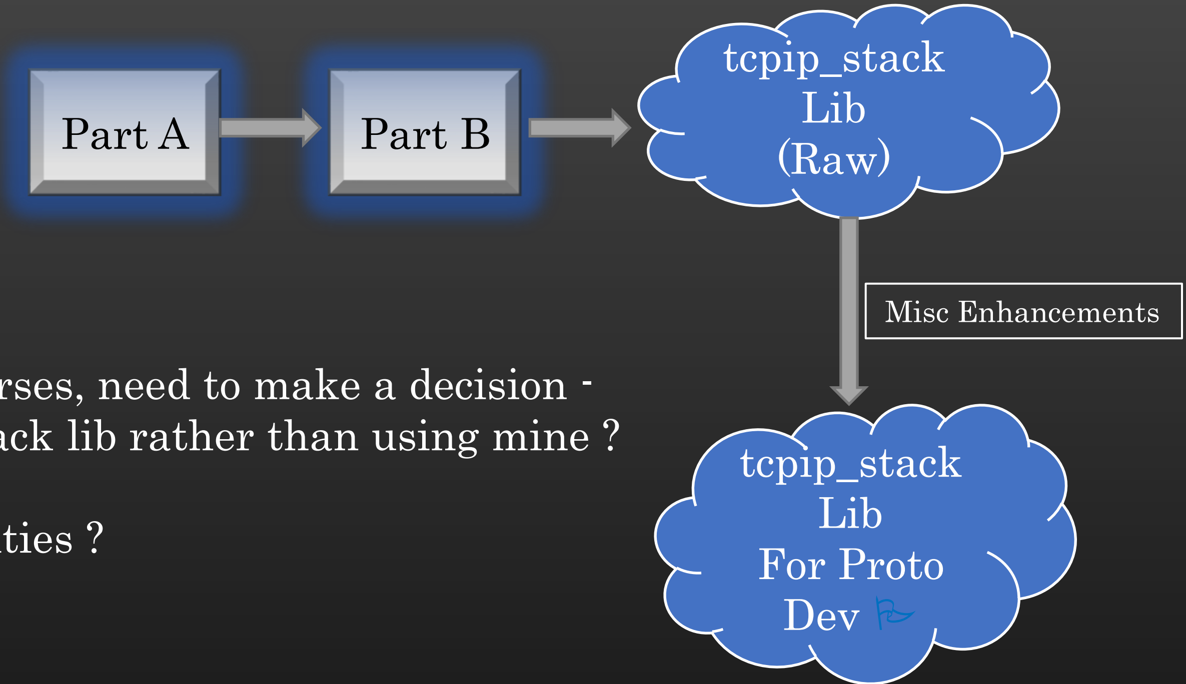
- Well , every person has his own taste for setting up his own favorite set of tools for development work
- Let me share my development environment, and probably I am very satisfied with it now
- My development environment is for those who are using windows OS/MAC as host machine, and linux machine as Guest VM Using Virtual Machine Software such as Virtual Box, VMWare etc
- Set up SSH login from windows to ubuntu (Google or goto youtube)



- Use Visual Studio code on Windows/MAC and connect VS code to guest machine and access the code dir (goto Youtube or Google for *howto* steps)
- We shall be writing and modifying all codes on windows VSCode editor , but compiling, building and running it on guest OS

- This TCP/IP Stack library is the output of my two existing udemy courses

- In this Course, we would treat this library as black box
  - No need-to-know internal implementation details
  - Use its publicly exposed APIs for protocol dev



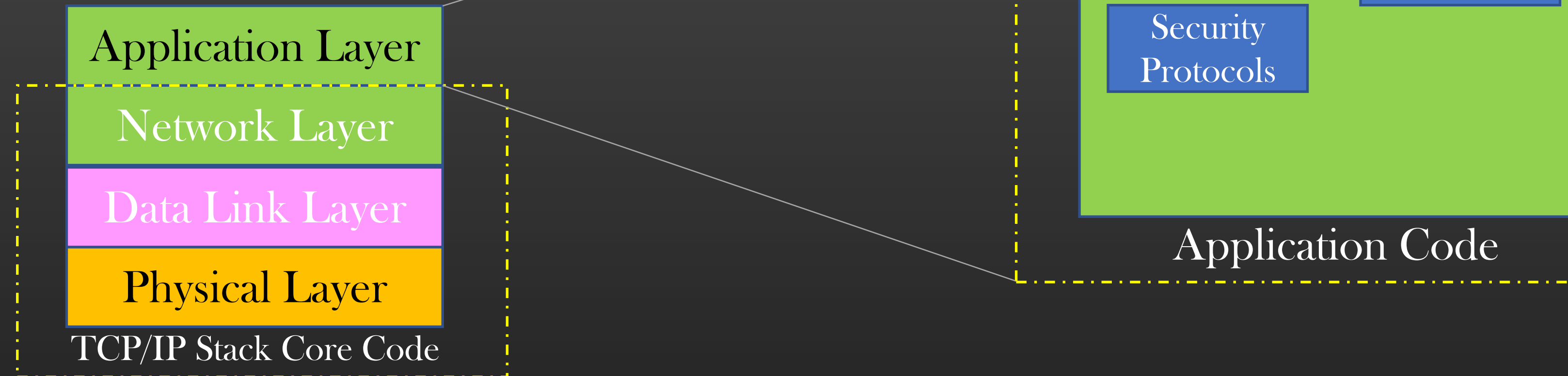
- Those who are coming after finishing existing udemy courses, need to make a decision -
  - Do you want to use your implementation of tcpip_stack lib rather than using mine ?
    - Is it completely bug free ?
    - Have you added all the features and functionalities ?
    - Thoroughly tested ?
    - Stable ?

- I have resolved several bugs from library and bring it to the point through several cycles of enhancements that it is now suitable for protocol development while viewing tcpip_stack library as black box

- I recommend you – only if you could answer the above Questions in YES , then only use your library for this course
- I understand, people have the strong urge to use something which they have built with their own hands ! ☺

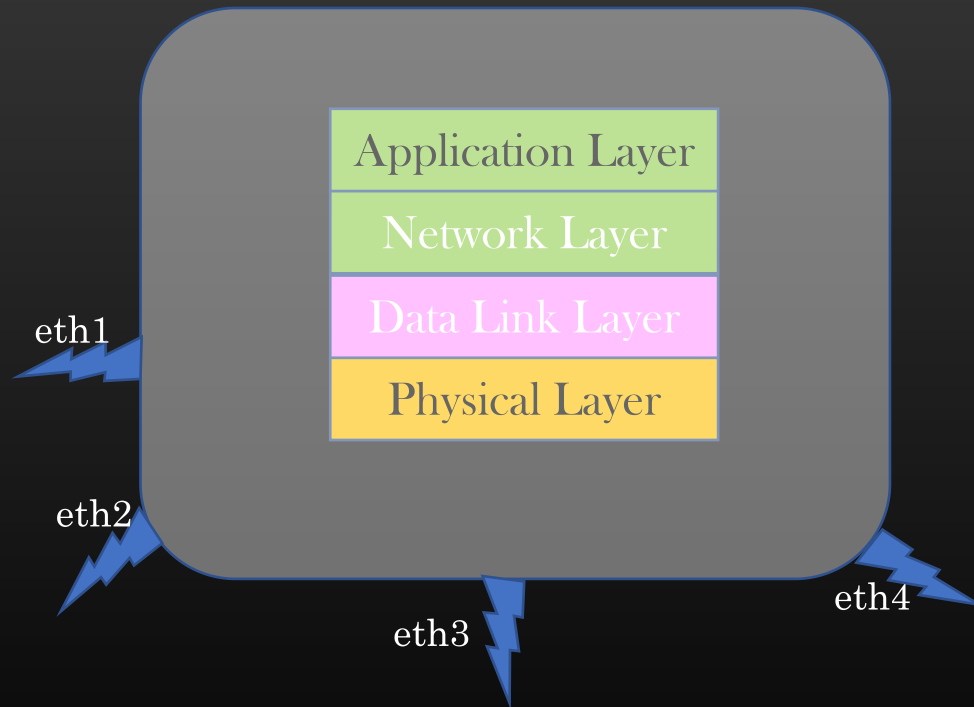
Get Familiar with TCP/IP Stack Library → Library internal design and Arch

- The library implements minimal TCP/IP stack – specifically :
  - Layer 1
    - Send and Recv packets on interfaces
  - Layer 2 :
    - Mac Learning
    - ARP resolution
    - VLAN based routing (no STP)
  - Layer 3 :
    - L3 routing
  - Layer 5 :
    - Application Layer (this Course)
    - In Application Layer, We can implement as many Applications as we want – no limit. Applications can harness the facilities Provided by lower layers
    - Application can program tcp/ip stack to express interest in pkts it want to receive. Conversely Application can also push the pkt down to the tcp/ip stack
  - No Socket Programming
    - There is no socket programming involved directly while developing Network Protocols
    - Industry usually hide socket interface/APIs behind simple high APIs to be invoked by applications



Interface Events :

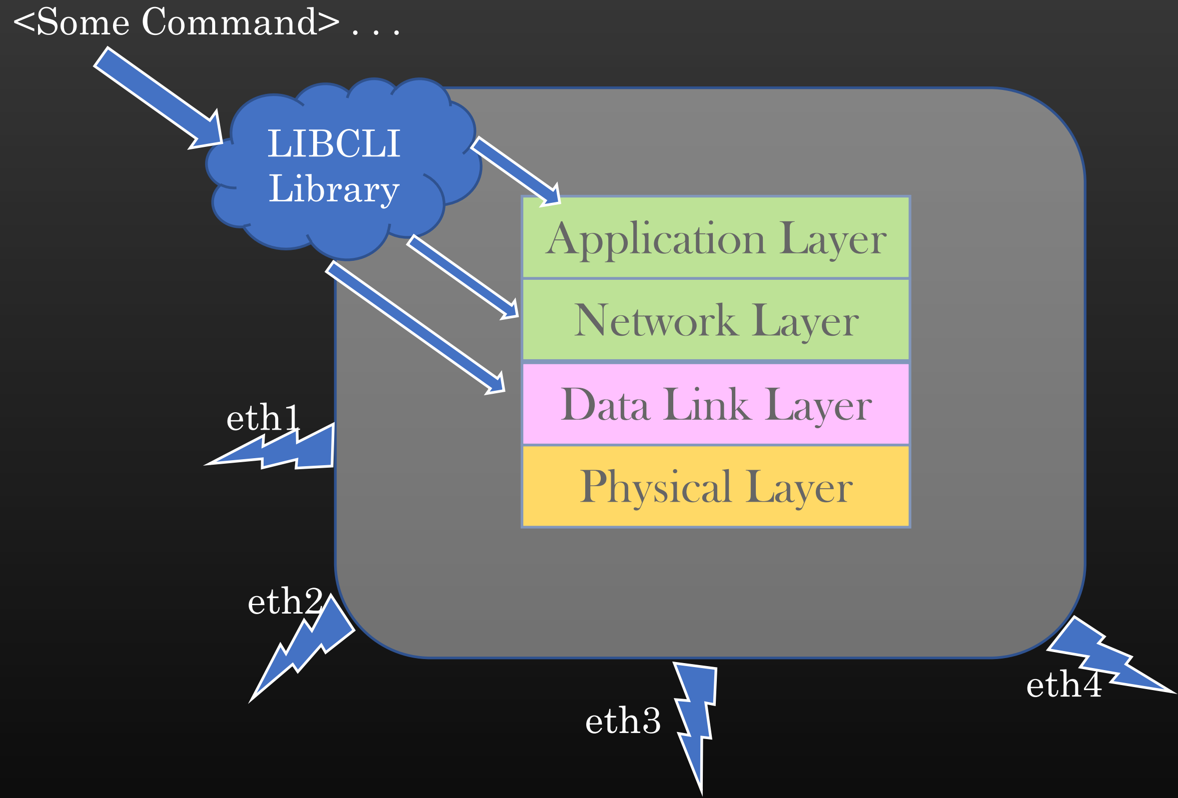
- IP Address change
- shut / not-shut
- vlan config change
- Cost change



- Applications may be interested in being notified of admin config change on an interface
- Such config change notification is sent to applications so that appln can process and react to it
- For Example, is user changes IP Address from X to Y on interface eth1, then IGP would have to change its hello pkts to advertise IP Y instead of X
- Exactly similar mechanism on real device
- Application has to register for interface events during initialization phase
- Event is notified to all registered applications

CLIs

- Config/un-config
- Show / clear
- Debugging



- User interacts with each node of the topology using CLI interface
- LIBCLI library parse the CLI tokens, validate CLI format and invoke the appropriate backend handler for processing in backend code
- The backend handler could be fn in L2 Or L3 or L5 or anywhere ..
- Throughout the course, we would be going to develop bunch of custom CLIs to control and config our baby – our protocol
- We shall go through brief training in which we will learn how to develop and add custom CLIs
- This is not a Quick do and Move on to next Course ! Take your time ..
- Along the journey, We shall be using various libraries :
  - Linked-list
  - Trees
  - Timers
  - CLIs
  - Scheduler
  - TLV encoders
  - BIT manipulation macros

☜This is what you are supposed to do

☜This is the skill you need to develop

☜ Only interface change, fundamentals do not

Getting Started with the Routing Protocol Development → Complete Theory

- We will be going to implement a working prototype of the IGP – ISIS
- ISIS and OSPF are two popular IGP protocols being used in industry for decades
- OSPF has been more popular until recently, when ISIS is being shown more interest and preferred over ospf more by Network Admins
- First, we need to get familiar how IGP protocols work – at high level. It works in 4 phases :

1. Adjacency Management (Each device know its neighbours)
   - Sending and Receiving hello packets periodically
   - Update neighborship state machine
2. Building Link State Database (Each device internally creates a view of topology - Graph)
   - Building Link State packets
   - Flooding link state packets
   - Build a Graph – a view of network topology
3. Running SPF algorithm (Dijkstra) on LSDB
   - Process the LSDB through the algorithm
   - Compute Results and store
   - Algorithmically challenging
4. L3 Route Calculations
   - Use Results of 3 to compute final L3 routes and update Routing Table
   - Algorithmically challenging

☞ Videos following this lecture Videos Covers all required theory to get to know how IGP works

☞ Once we complete this section, we would have fair idea how IGP (ISIS) works end to end

☞ Pls spend ~60 minutes to learn the theory

- So far, we have completed the theory regarding protocol internal functioning
- Now , we shall work towards seeing this theory in action
- Let's begin protocol development starting from :
  - Integrating the protocol as an application with TCP/IP Stack library ecosystem
  - Implement Protocol phase wise

Congratulations for Finishing the College Professional

- So let us bring our protocol to life first

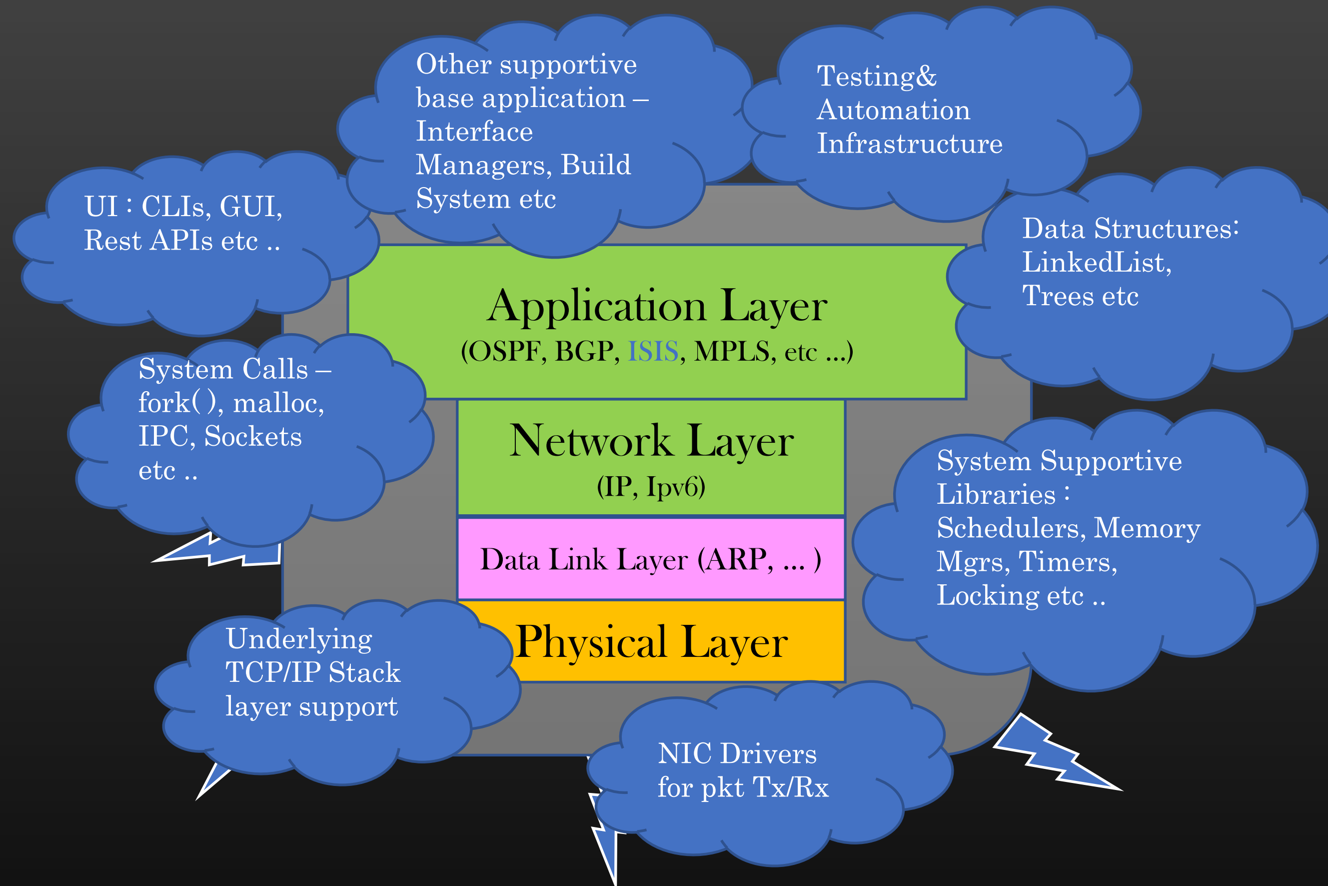
- Let's do some preliminary steps to develop minimal barely breathing ISIS as a new application running in appln layer of TCP/IP stack library

- I would help you to achieve this, we need to follow some fixed steps. Once you are familiar with the steps, then this process become quite mundane

✰ *In production also, we need to follow some definite steps to setup new application. You just don't go and write xyz_main.c in whatever way you want !! Production environment have their own infrastructure/ecosystem of Whole code base and application needs to be written in harmony with ecosystem*

- Protocol application Setup Steps :
  - Register with TCP/IP stack for interested ISIS pkts
  - Develop first appln CLI
  - Some more steps – later . . .

➢ One just cannot start writing protocol from int main ( ) straightaway, an ecosystem is required to facilitate network protocol development



- Quality of the end product largely depends on quality of technology stack
- Companies have teams managing different components of ecosystem (Infrastructure teams)
- In this Course, we would need to harness the ecosystem to build our protocol
- Project will rehearse you how to pick up reusable software packages and use them for software dev
- We shall be incrementally integrating our protocol (appn) one by one with each of the supplementary component/library

Our TCP/IP Stack library exactly tend to provide (Partially) Simulated ecosystem to facilitate pseudo protocol development from scratch

This Ecosystem is also called as Technology stack

Getting Started with the Routing Protocol Development → Develop Ist CLI

- Network Equipment generally do not have GUI, and we operate them using CLIs only
  - FAST
  - Automatable
  - Don't need heavy resources
  - Runnable on system with no GUI (embedded)

- TCP/IP Stack library comes with pre-integrated CLI library
- How to use Quick Tutorial : Appendix A (Part 1 and Par 2) Sections

- Recommended to go through Section Appendix A
  - How to use the CLI library
  - Developing some custom example command
  - Move to next lecture Video after finishing this tutorial
  - Finish tutorial fast, do not bang head
  - In the next lecture video, we will develop our first CLI

Getting Started with the Routing Protocol Development → Develop Ist CLI

- Create a new files : isis_cli.c & isis_cmdcodes.h
- Whenever create new .c file, always update tcpip_stack/Makefile

- Let us add CLI to enable/disable protocol on a particular device
  - Adding a CLI
  - Code the backend handler

config node <node-name> [no] protocol isis

- ☆ *Every company provide their own infrastructure to build/add new CLIs, There is no standard way of doing this.*
- ☆ *Learn and Unlearn*

1

```
int
isis_config_cli_tree(param_t *param) {

        {
                // write protocol CLI hiérarchie here
                static param_t isis_proto;
                init_param(&isis_proto, CMD, "isis", isis_config_handler, 0, INVALID, 0, "isis protocol");
            libcli_register_param(param, &isis_proto);
                set_param_cmd_code(&isis_proto, ISIS_CONFIG_NODE_ENABLE);
        }
        return 0;
}
```

3 Prototype in tcpip_stack/Layer5/app_handlers.h

2

```
static int
isis_config_handler(param_t *param,
                    ser_buff_t *tlv_buf,
                    op_mode enable_or_disable){
                    printf(« blah blah ... »)
                    return 0;
}
```

4 Register isis_config_cli_tree( )in tcpip_stack/nwcli.c

5 # define in isis_cmdcodes.h

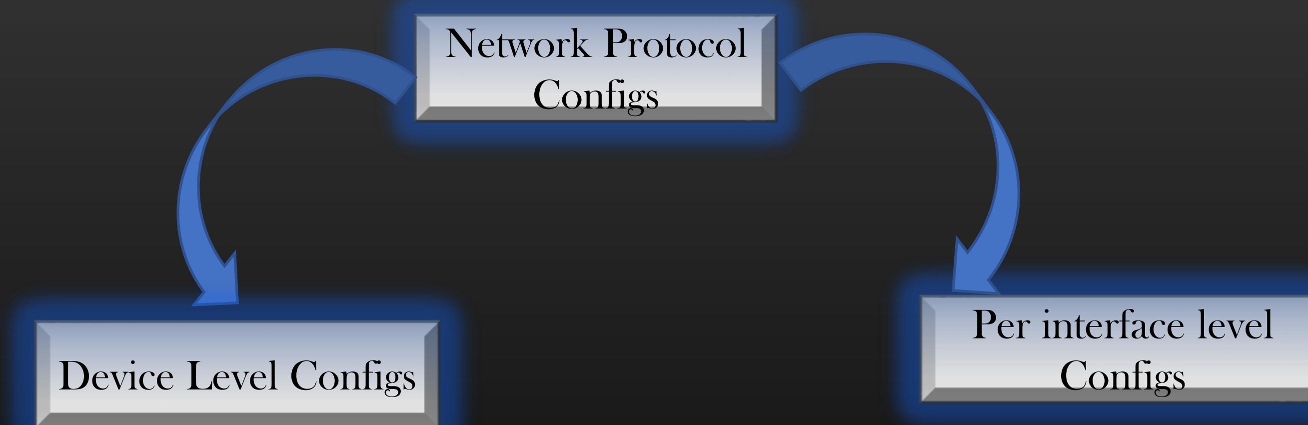
6 Rebuild project, run and press . (dot)
See your new CLIs must appear

```
static int
isis_config_handler(param_t *param,
                    ser_buff_t *tlv_buf,
                    op_mode enable_or_disable){


    int cmdcode = -1;
    char *node_name = NULL;
    node_t *node = NULL;
    tlv_struct_t *tlv = NULL;

     cmdcode = EXTRACT_CMD_CODE(tlv_buf);

     TLV_LOOP_BEGIN(tlv_buf, tlv){

        if (strncmp(tlv->leaf_id, "node-name", strlen("node-name")) ==0)
            node_name = tlv->value;
        else
            assert(0);
    } TLV_LOOP_END;

    node = node_get_node_by_name(topo, node_name);

    switch(cmdcode) {
     // . . .
    }
    return 0;
}
```

```
switch(cmdcode) {
    case ISIS_CONFIG_NODE_ENABLE:

        switch(enable_or_disable) {
            case CONFIG_ENABLE:
                isis_init(node);
                break;
            case CONFIG_DISABLE:
                isis_de_init(node);
                break;
            default: ;
        }
    break;
    default: ;
}
```

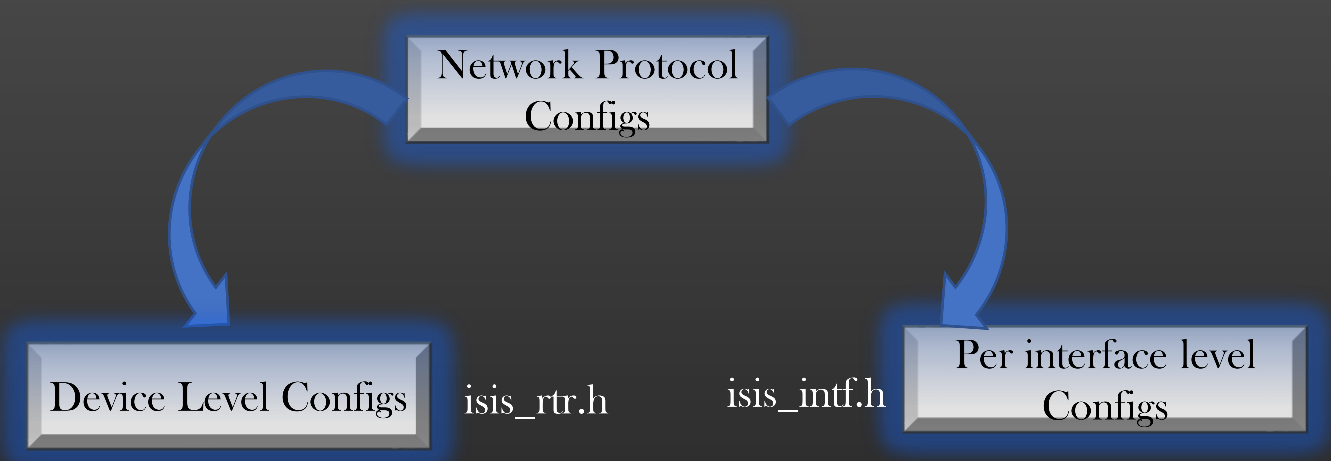
- We need Data structures which can hold proto configs :
  - At device level (one instance is required)
  - Per interface level (one instance per interface)

- Next : Steps to add protocol specific node and per-interface level data configuration holders (data objects)



- Impact protocol behavior on a device
- For ex :
  - Stop generating LSP packets
  - Advertise information X in LSP pkts
  - If protocol is disabled at device level, protocol is automatically disabled on all interfaces

- Impact protocol behavior per interface
- For ex :
  - Stop sending LSP pkts out of eth0
  - Start sending hellos out of eth1 at 5 sec/hello pkt
  - Enable/disable protocol on eth4




```
typedef struct isis_node_info_ {

  . . .
} isis_node_info_t;

node_t-> node_nw_prop->isis_node_info; // type void *
```

```
typedef struct isis_intf_info_ {

  . . .
} isis_intf_info_t;

interface_t->intf_nw_props->isis_intf_info; // type void *
```

- Use existing files : isis_cli.c & isis_cmdcodes.h

- Let us add CLI to see protocol status on a particular device
  - Adding a CLI
  - Code the backend handler

show node <node-name> protocol isis

```
tcp-ip-stack> $ show node R0 pro isis
Parse Success.
ISIS Protocol : Disabled
CLI returned
```

2

```
static int
isis_show_handler(param_t *param,
                  ser_buff_t *tlv_buf,
                  op_mode enable_or_disable){
                  printf(« blah blah ... »)
                  return 0;
}
```

3

```
int
isis_show_cli_tree(param_t *param) {
```

Prototype in tcpip_stack/Layer5/app_handlers.h

1

```
        {
                // write protocol CLI hiérarchie here
                static param_t isis_proto;
                init_param(&isis_proto, CMD, "isis", isis_show_handler, 0, INVALID, 0, "isis protocol");
            libcli_register_param(param, &isis_proto);
                set_param_cmd_code(&isis_proto, CMDCODE_SHOW_NODE_ISIS_PROTOCOL);
        }
        return 0;
}
```

4

Register isis_config_cli_tree( )in tcpip_stack/nwcli.c

5

define in isis_cmdcodes.h

6

Rebuild project, run and press . (dot)
See your new CLIs must appear

Network Protocol Development → Do not use Printf for show command output

- Until now we have learnt how to configure and unconfigure the protocol which is a device level configuration

- Let's us implement a similar config CLI to enable disable protocol on an interface of a device i.e Example of interface level configuration

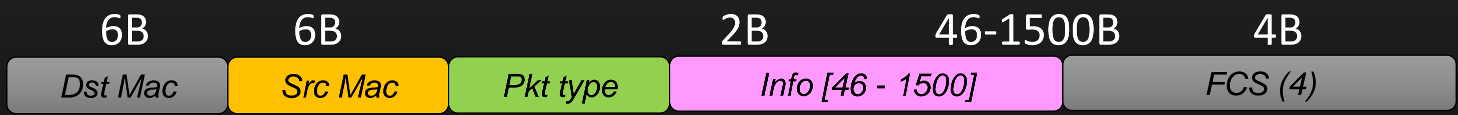
config node <node-name> [no] protocol isis interface all
- Must enable/disable protocol (alloc/free isis_intf_info) on all interfaces of a device

config node <node-name> [no] protocol isis interface <if-name>
- Must enable/disable protocol (alloc/free isis_intf_info) on a specified interface only

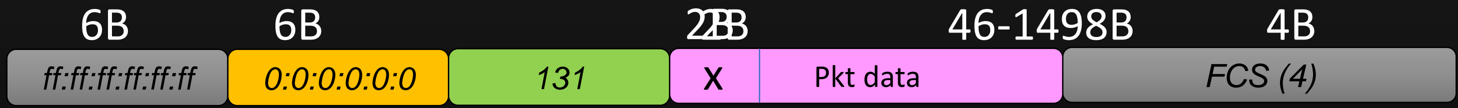
Note : Disabling the protocol on a device must disable protocol on all interfaces also
Enabling the protocol on a device must not enable protocol at interface level, user should do it manually through above new CLIs

➢ Let's design our own packet formats for the sake of keeping protocol implementation not go over complicated and not invest time in things not worth enough

ethernet_hdr_t ->



ethernet_hdr_t ->

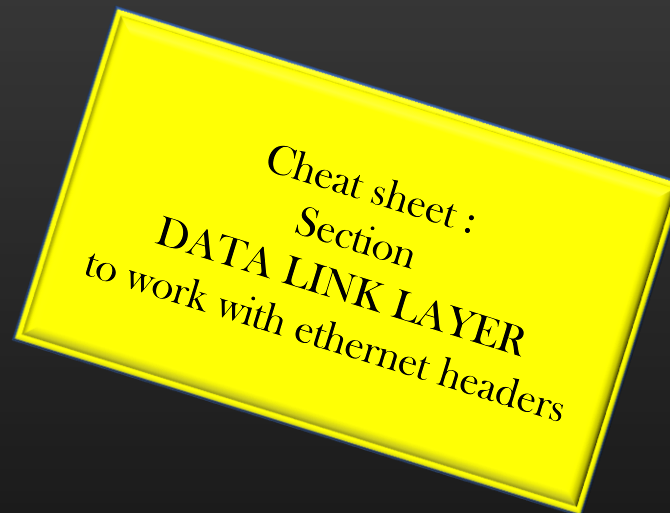



```
#define ISIS_ETH_PKT_TYPE          131 // ( Randomly chosen, no logic)
X values :
#define ISIS_PTP_HELLO_PKT_TYPE 17  // as per standard
#define ISIS_LSP_PKT_TYPE          18  // as per standard
```

New File : tcpip_stack/Layer5/isis/isis_const.h

- Now that we know that ISIS both pkt types are ethernet pkts, we need to learn to work with APIs Provided by TCP/IP stack library to work with ethernet pkts

  > Malloc a new pkt memory space
  > updating MAC addresses
  > Getting payload pointer
  > Updating FCS (= 0)
  > Freeing the pkt memory



- ISIS protocol runs as application process in application layer, so it has to tell Data link layer that –
  "Hey, if you ever recv ISIS Hello or LSP pkt, handover it to me, they are mine, it is only me who Understand them" .

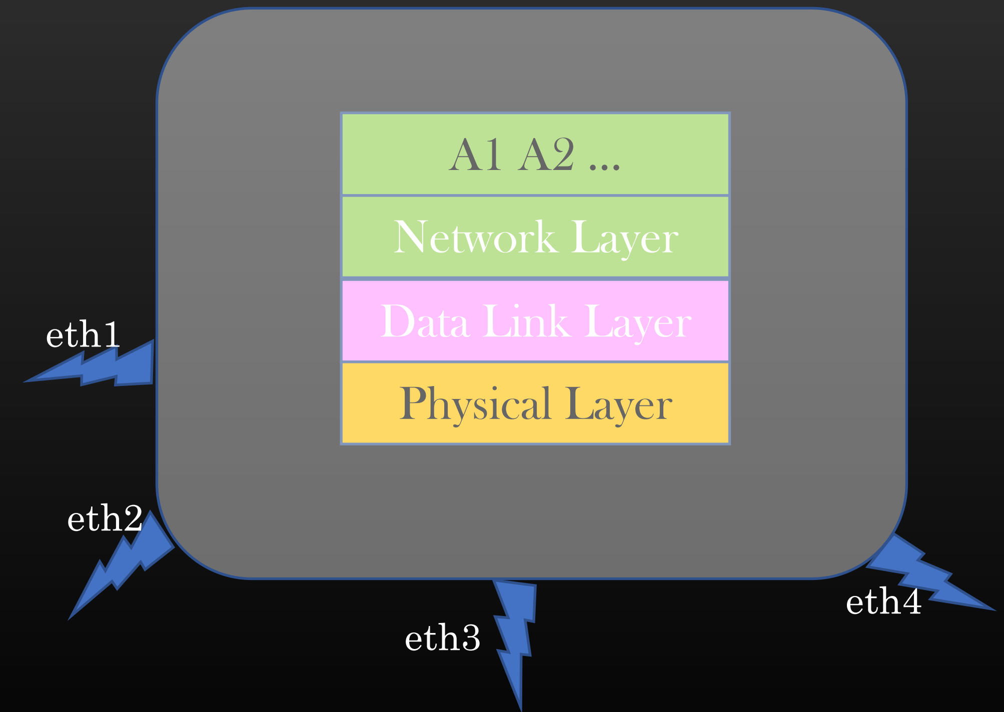
This is called protocol registration for control packets. We usually use sockets to achieve this. In production, an API is provided which wrap all socket complexities and all appln need to do is to invoke the API to express interest in specific packet types.

- The TCP/IP Stack whenever recvs some packets, it checks if there is any application interested in processing these packets

Two Step Process :

1. Pkt classification rule

2. Install the rule in L2 Or L3



- ISIS protocol runs as application in application layer, so it has to tell Data link layer that –
  "Hey, if you ever recv ISIS Hello or LSP pkt, handover it to me, they are mine, it is only me who Understand them".

  This is called protocol registration for control packets. We usually use sockets to achieve this. In production, an API is provided which wrap all socket complexities and all appln need to do is to invoke the API to express interest in specific packet types.

  - The TCP/IP Stack whenever recvs some packets, it checks if there is any application interested in processing these packets

New File : tcpip_stack/Layer5/isis/isis_pkt.h | .c

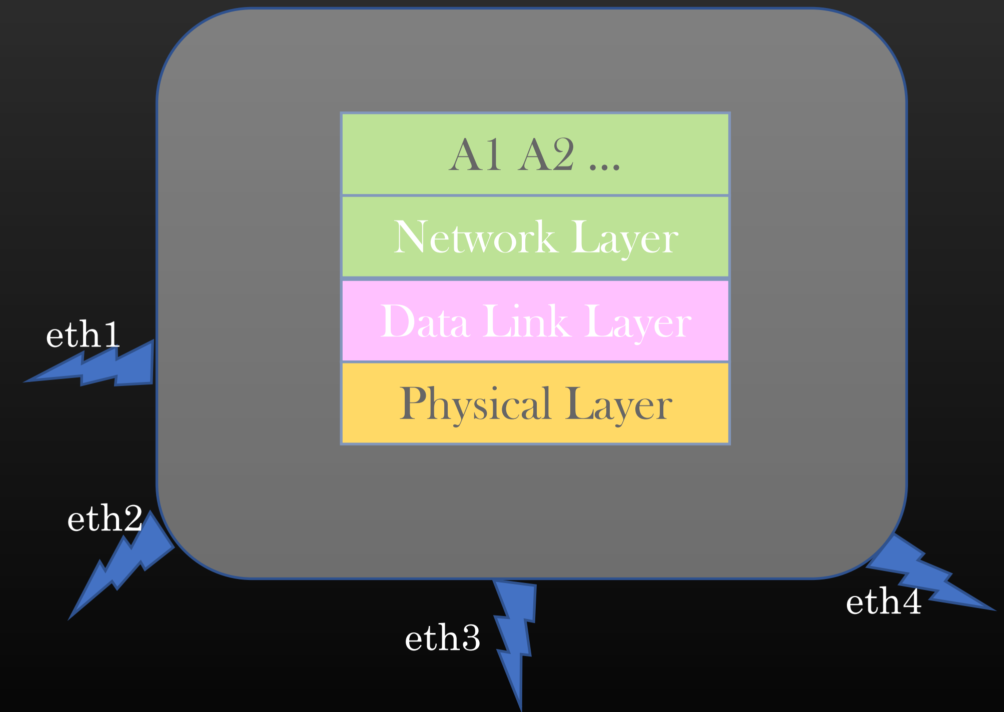
Write an API in isis_pkt.c :

```
bool
isis_pkt_trap_rule (char *pkt, size_t pkt_size) ;
```

Where : pkt - pointer to start of ethernet hdr
pkt_size – size of pkt (ethernet hdr + ethernet payload + FCS)

The fn returns true if this is ISIS pkt else false.

Update Makefile and compile the project



Step 2

Install the Rule in underlying TCP/IP Stack Subsystem

isis_pkt.c / isis_pkt.h

```
void
isis_pkt_recieve(void *arg, size_t arg_size) {

}

/* Register for interested pkts */
tcp_stack_register_l2_pkt_trap_rule(
        node, isis_pkt_trap_rule, isis_pkt_recieve);


/* De-Register for interested pkts */
tcp_stack_de_register_l2_pkt_trap_rule(
        node, isis_pkt_trap_rule, isis_pkt_recieve);
```

- Call when protocol is enabled on device
- Call when protocol is disabled on device

Logging Internal Traces

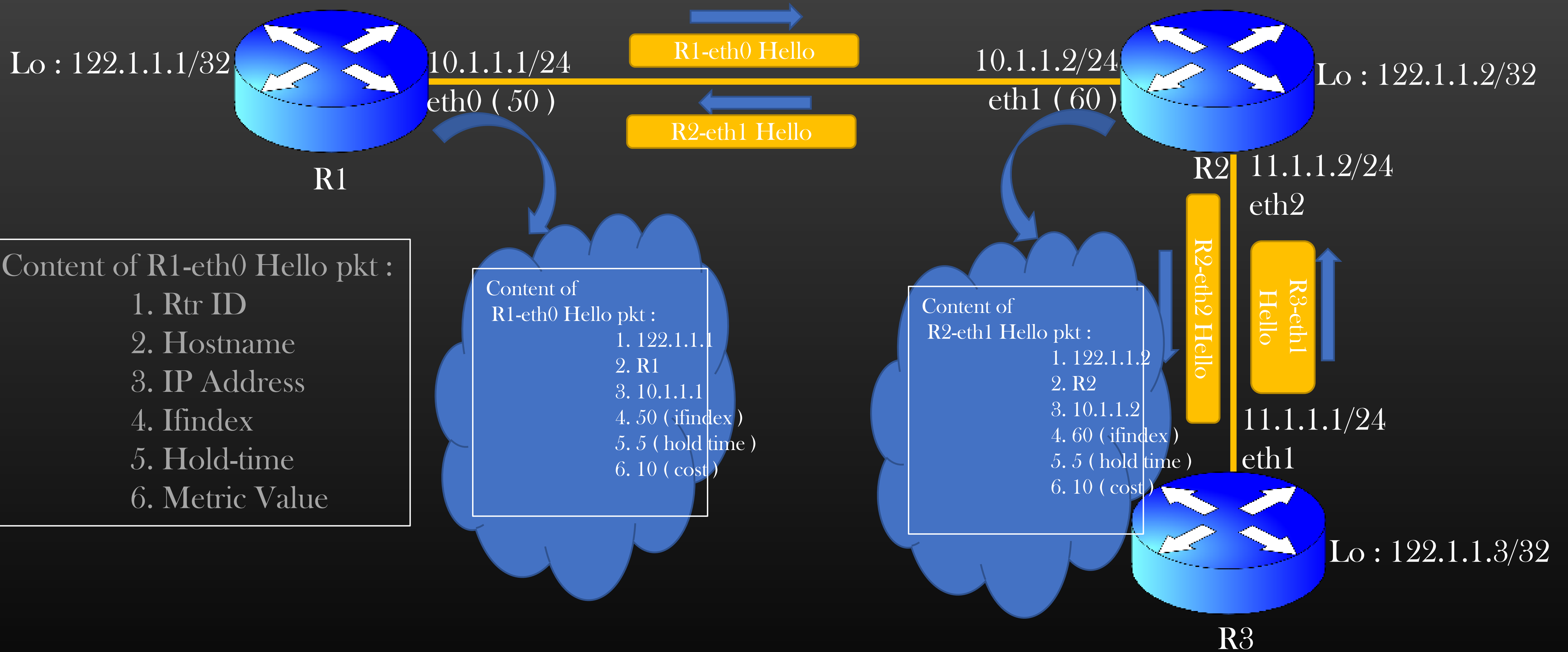
- Inserting printfs are fundamental to debugging, replacement still not found !
- TCP/IP Stack library provides facility to insert traces in the code to ease debugging
- During development, you are encouraged to insert as many traces as you feel necessary
- Traces will be your best friend to help troubleshoot the issues !!
- Generally - Do not insert traces in the packet recv or sent path, if you do then enable/disable them using CLI
- For Error Msg, you can directly write to console using printf, no need to trace error msgs

- So far, we have accomplished :
  - How to configure the protocol using config CLI
  - How to display protocol state using show CLI
  - How a protocol can do pkt (un)registration
- Until now we were trying to setup the protocol as per the TCP/IP stack library ecosystem
- Now , we are in a position that we can start the core development of core protocol internal logic
- Notification for Interface events is still pending which we shall pick up in the midst of protocol development along the way ..
- From Next section, let start the development of Ist phase of the protocol – *Adjacency Management*

1. Adjacency Management (Each device know its neighbours)
   - Sending and Receiving hello packets periodically
   - Update neighborship state machine
2. Building Link State Database (Each device internally creates a view of topology - Graph)
   - Building Link State packets
   - Flooding link state packets
   - Build a Graph – a view of network topology
3. Running SPF algorithm (Dijkstra) on LSDB
   - Process the LSDB through the algorithm
   - Compute Results and store
   - Algorithmically challenging
4. L3 Route Calculations
   - Use Results of 3 to compute final L3 routes and update Routing Table
   - Algorithmically challenging

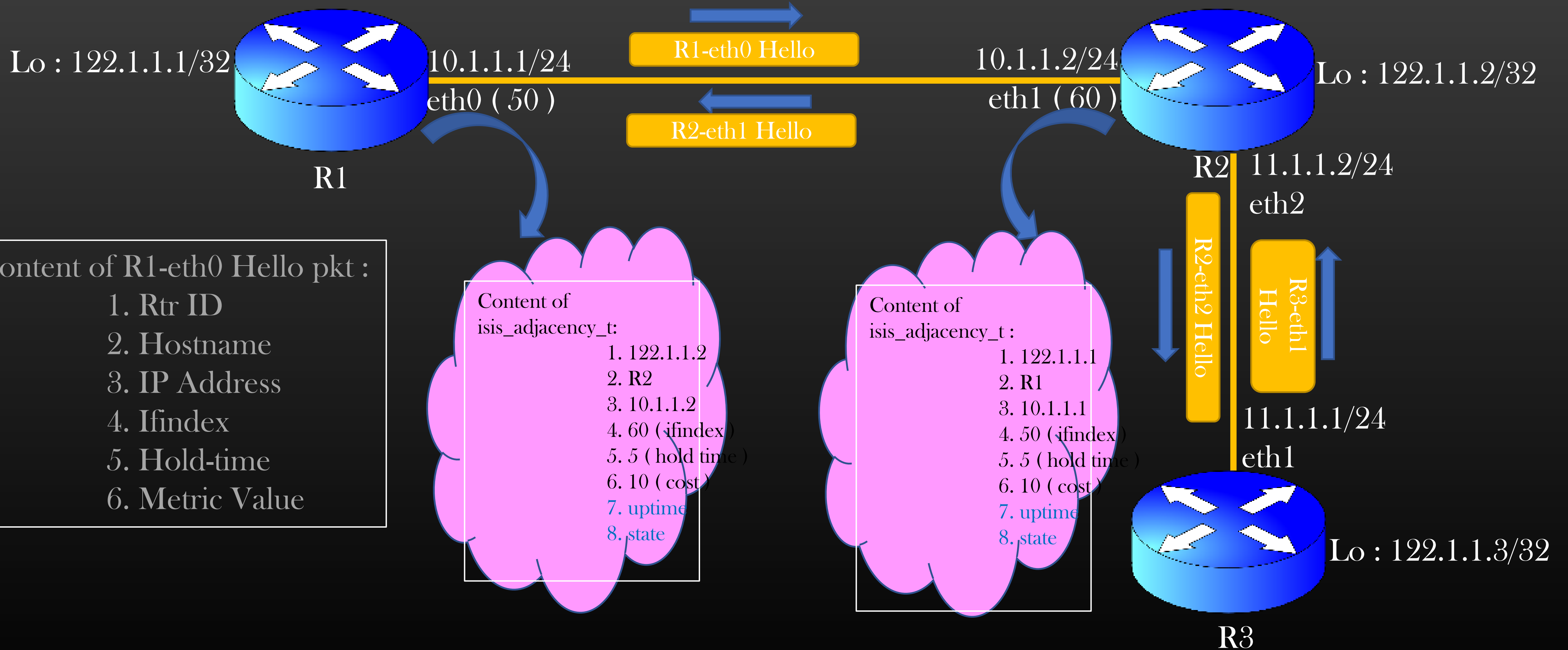
- Let's start with the implementation of phase 1 of the protocol – Adjacency Mgmt
- Adjacency Means Neighborship
- You know your neighbors only when you start exchanging some greeting words with them, talk to them
- ISIS protocol need to send periodic hellos packets to its nbr device
  (Many Network protocols does this – OSPF, BGP, PIM etc)
- ISIS Hello packets contain information about the self-device and interface out of which the pkt is egressing
- Adjacency Mgmt works is a three-step process :
  - Send out Hello packet
  - Process Hello Packet
  - Extract information from Hello packet and store/update nbrs information

Hello Packet Content

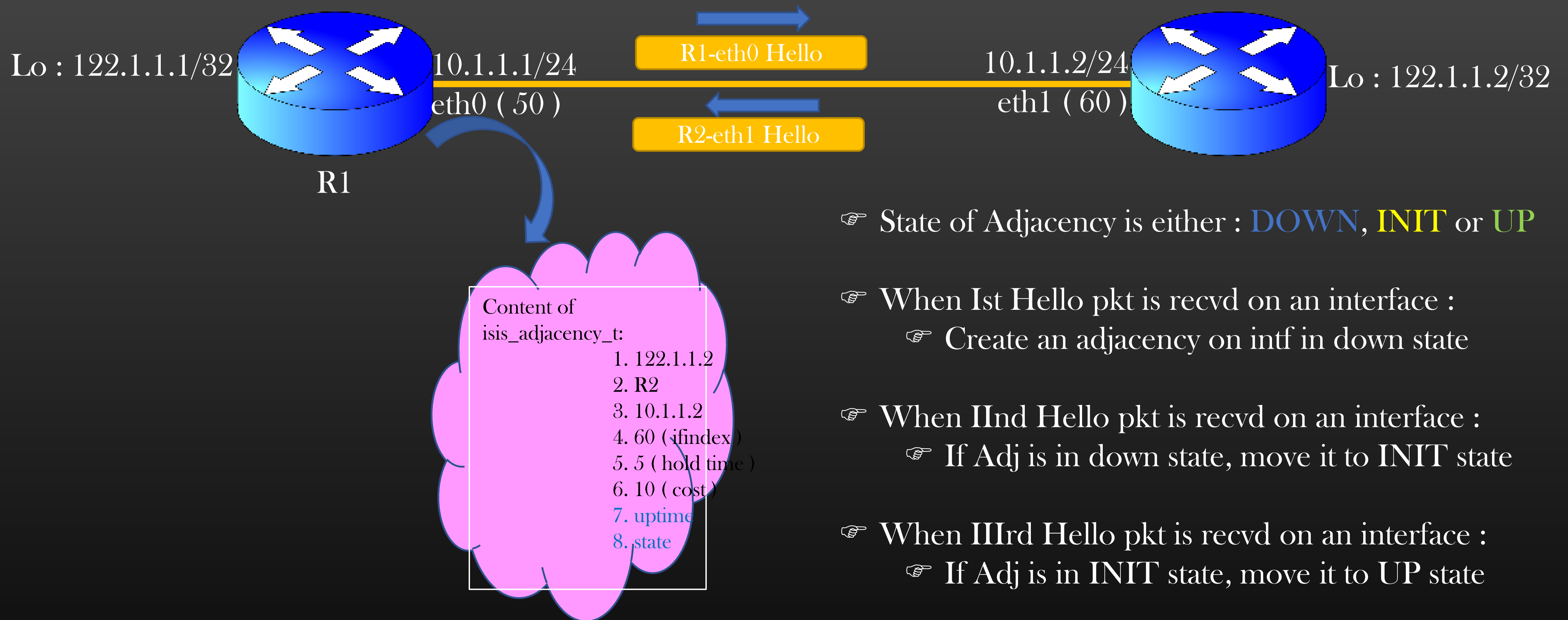


Adjacency Formation

Adjacency is a data structure maintained per interface which stores nbrs Information reachable through that interface



Adjacency State



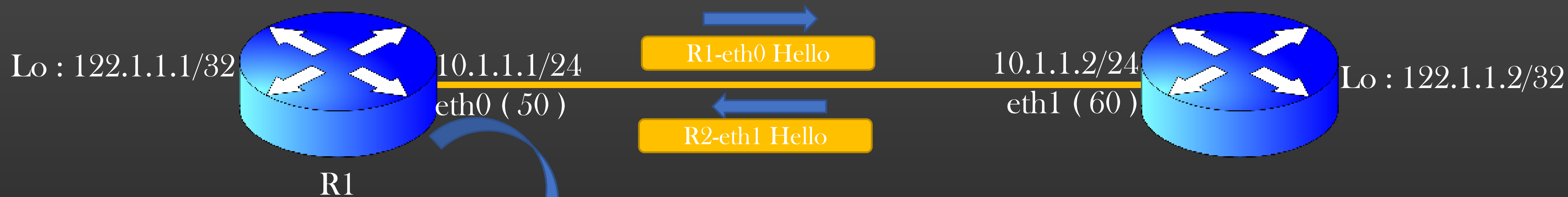
- State of Adjacency is either : DOWN, INIT or UP
- When Ist Hello pkt is recvd on an interface :
  - Create an adjacency on intf in down state
- When IInd Hello pkt is recvd on an interface :
  - If Adj is in down state, move it to INIT state
- When IIIrd Hello pkt is recvd on an interface :
  - If Adj is in INIT state, move it to UP state
- When no hello pkt is recvd for hold_time sec on intf
  - If Adj is in UP state, move it to down state and delete it after some time if hellos still do not come
  - If Adj is in down state already, delete it

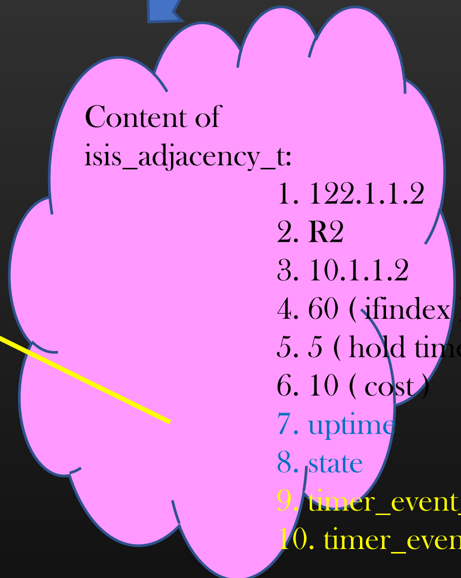
In order to implement Adjacency mgmt Logic, we need to first design the Adjacency State Machine

Timers In General

- Timers, help in scheduling the events to be performed in future after some definite period of time
- Timers are used extensively in Networking Development
- Protocol uses Timers for various purposes :
  - Deleting Stale Data structures
  - Fire Algorithm/Computation
  - Send packets periodically
- TCP/IP Stack library provides us TIMER sub library to be used for our development
- Using Timers is somewhat a standard procedure, API signatures differs from company to company but underlying concept/requirement is same
- We shall discuss Timer APIs when we shall be using Timers in our code

Adjacency Timers





- Adjacency object is managed by Two timers :
  - Delete timer
    - = 5 sec
    - Delete the Adjacency permanently
  - Expiry timer
    - = hold time
    - Change the Adj state from UP/INIT to DOWN state
    - Start the delete timer

Uptime : time elapsed since Adjacency moved from init To UP State
In other words, it is a time since Adjacency has been in UP state

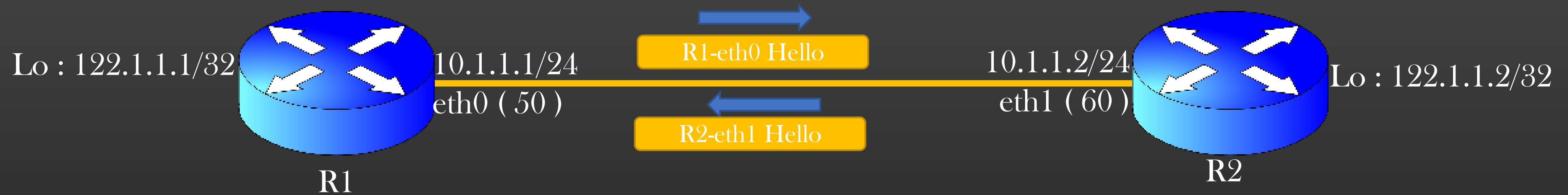
You don't have to run any timer for this :

1. Store the time stamp when adj move to UP state
2. Subtract current system time with stored time stamp

Implementation Steps in this section

1. Hello packet format
2. Cook up hello packets from Node & Interface data
3. Send out hello packets periodically out of protocol enabled interfaces
4. Stop hello packets when protocol is disabled on interface
5. Receiving Hello packets by recipient node
6. Processing Hello packets
7. Creating Adjacency Objects from Hello pkt processing
8. Enhancing show command to display Adjacency
9. Timer managing the Adjacency Object
10. Updating Hellos as a result of generic intf config change by admin
11. Adjacency State Machine (Separate Section After above)

Hello Pkt format



ISIS Protocol packets are formatted as TLVs (Type Length Value)

Refer to Appendix D to learn about TLV

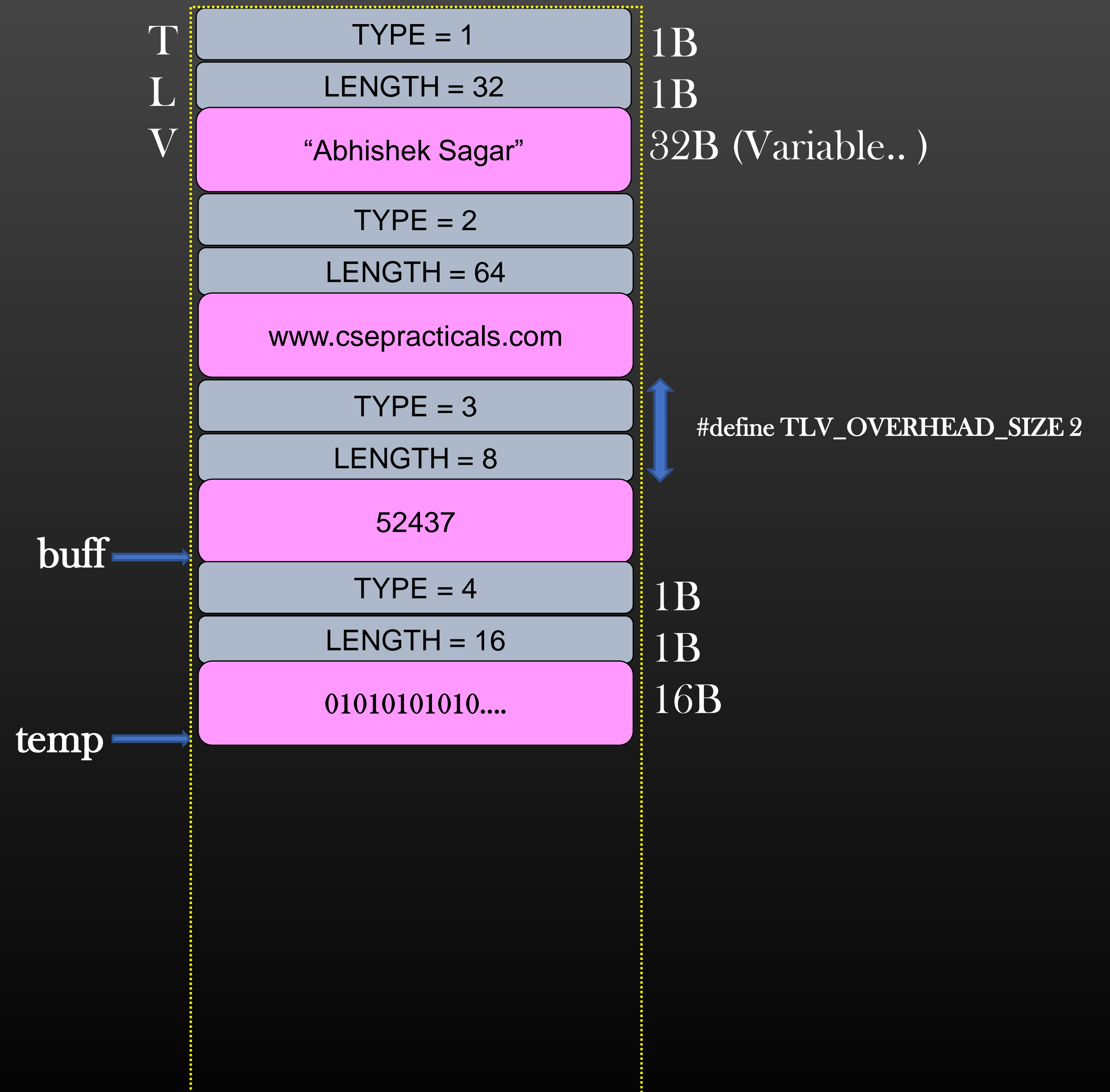
- What is TLV ?
- What are benefits
- APIs to work with TLVs
- Interview Ques
- Cheat Sheet . . .

TLV APIs – Inserting a new TLV

```
byte *temp = tlv_buffer_insert_tlv(
                                    byte *buff,
                                    uint8_t tlv_no,
                    uint8_t data_len,
                         byte *data);
```

```
byte *temp = tlv_buffer_insert_tlv(
                                    buff,
                                    4,
                    16,
                    01010101010....);
```

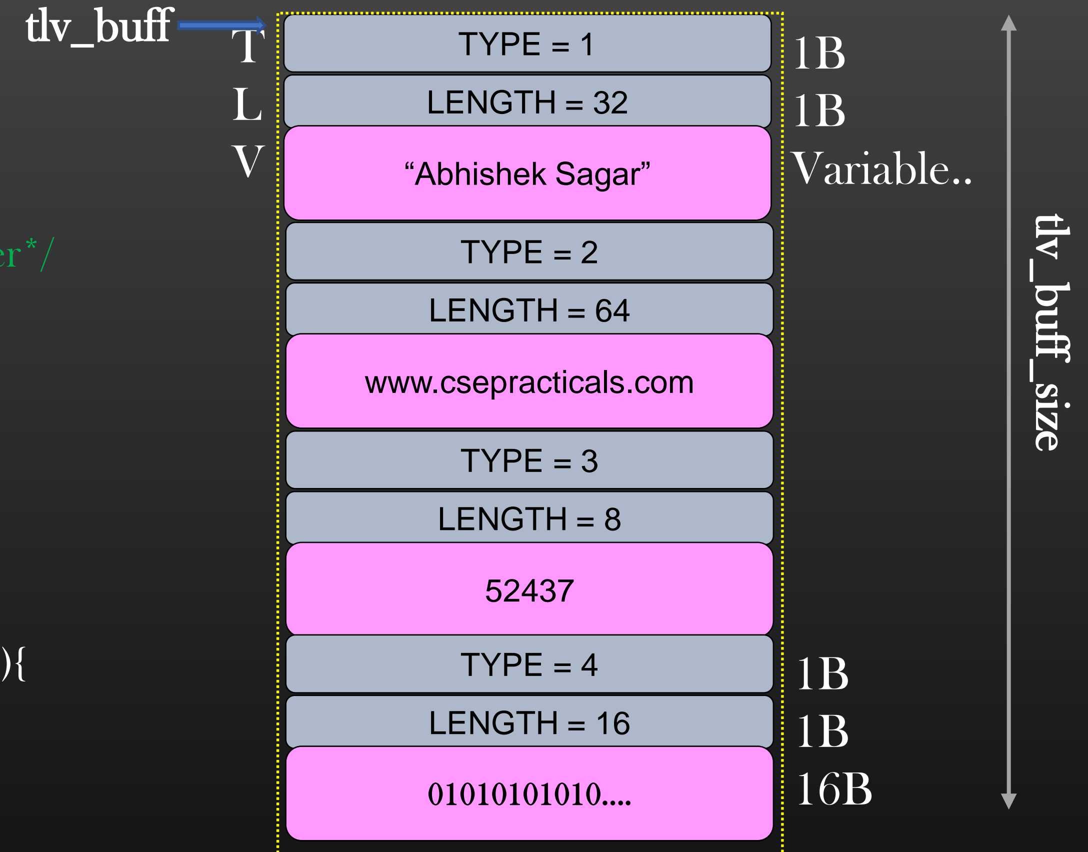
```
byte *
tlv_buffer_insert_tlv(byte *buff, uint8_t tlv_no,
            uint8_t data_len, byte *data){

    *buff = tlv_no;
    *(buff+1) = data_len;
    memcpy(buff + TLV_OVERHEAD_SIZE, data, data_len);
    return buff + TLV_OVERHEAD_SIZE + data_len;
}
```



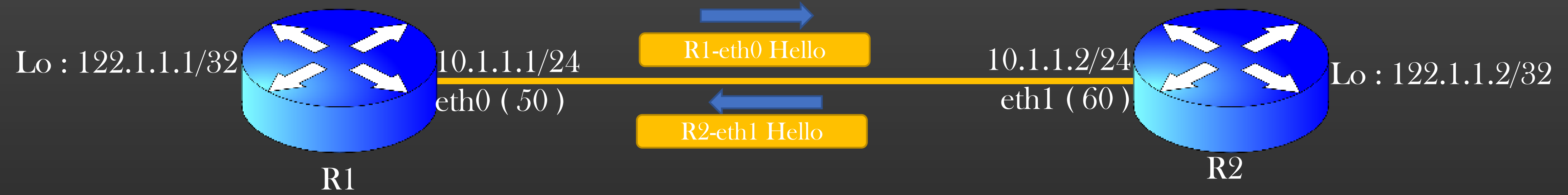
TLV APIs – Searching a TLV & Iterating TLV Buffer

- We may need to search a particular TLV in a TLV buffer

```
byte*
tlv_buffer_get_particular_tlv(
                        byte*tlv_buff,              /*Input TLV Buffer*/
              uint32_t tlv_buff_size,      /*Input TLV Buffer Total Size*/
              uint8_t tlv_no,              /*Input TLV Number*/
              uint8_t *tlv_data_len) {     /*Output TLV Data len*/

  byte tlv_type, tlv_len, *tlv_value = NULL;

  ITERATE_TLV_BEGIN(tlv_buff, tlv_type,
                                  tlv_len, tlv_value, tlv_buff_size){

     if(tlv_type != tlv_no) continue;
     *tlv_data_len = tlv_len;
     return tlv_value;
  } ITERATE_TLV_END(tlv_buff, tlv_type,
                                  tlv_len, tlv_value, tlv_buff_size);

  *tlv_data_len = 0;
  return NULL;
}
```



ISIS Pkt Hdr Format



Hello Pkt Fmt :

Ethernet payload

isis_pkt.h

```
#pragma pack (push,1)

typedef struct isis_pkt_hdr_{

    isis_pkt_type_t isis_pkt_type;
    uint32_t seq_no;
    uint32_t rtr_id;
    isis_pkt_hdr_flags_t flags;
} isis_pkt_hdr_t;
#pragma pack(pop)
```

Field	Size
isis_pkt_type_t 17	2
uint32_t seq_no	4
uint32_t rtr_id	4
isis_pkt_hdr_t flags	1

Hello Pkt Preparation

isis_pkt.h/.c

byte *
isis_prepare_hello_pkt (interface_t *intf, size_t *hello_pkt_size);

Supporting APIs :

1. TLV_OVERHEAD_SIZE - TLV Overhead
2. NODE_NAME_SIZE – Max name length of node name
3. ETH_HDR_SIZE_EXCL_PAYLOAD – Sum of length of all fields of Eth hdr except payload (= 18 B)
4. tcp_ip_get_new_pkt_buffer ( ) – malloc a new pkt buffer
5. tcp_ip_free_pkt_buffer( ) – free a pkt buffer
6. layer2_fill_with_broadcast_mac( ) – Fill mac address array with ff:ff:ff:ff:ff:ff
7. GET_ETHERNET_HDR_PAYLOAD – get ptr to start of ethernet payload from ethernet hdr ptr
8. NODE_LO_ADDR(node_ptr) – Get Node's Loopback address/Rtr ID
9. tcp_ip_covert_ip_p_to_n( ) – Convert IP Address from A.B.C.D to integer
10. tcp_ip_covert_ip_n_to_p( ) – Convert IP Address from Integer to A.B.C.D format
11. tlv_buffer_insert_tlv( ) – Insert a new TLV into TLV buffer
12. SET_COMMON_ETH_FCS( ) – Update FCS value in FCS field of eth hdr

- ☞ *By the end of Phase 1, we Would be knowing all type of APIs the library provides us to work with*
- ☞ *In development env, it takes several months to get familiar with all libraries, infrastructures, tools etc for an engineer*
- ☞ *You change a job and you need to redo it all again. But as you grow experienced, you get quicker to adjust to new environment*

Implementation Steps in this section

1. Hello packet format
2. Cook up hello packets from Node & Interface data
3. Send out hello packets periodically out of protocol enabled interfaces
4. Stop hello packets when protocol is disabled on interface
5. Receiving Hello packets by recipient node
6. Processing Hello packets
7. Creating Adjacency Objects from Hello pkt processing
8. Enhancing show command to display Adjacency
9. Timer managing the Adjacency Object
10. Updating Hellos as a result of generic intf config change by admin
11. Adjacency State Machine (Separate Section After above)

Timers

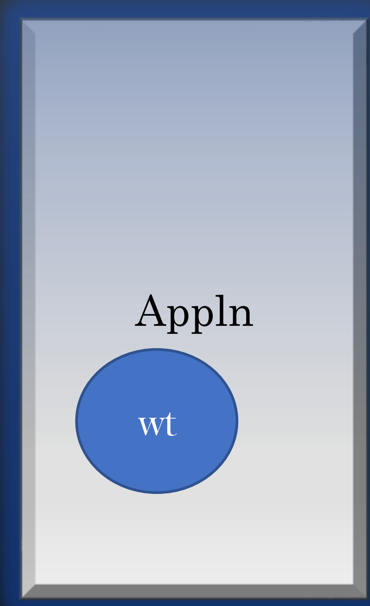
Appendix E

Timers

- Timers allow us to Fire events (computation) in future after some definite period of time
- Repeated or Just once
- Timers runs as separate threads
- When Timer Expires, Events is fired (Computation is triggered)
- Example :
  - Sending out periodic packets
  - Delete Stale Data structures
  - Perform computation after scheduled time

Timers

How Timer Works ?



Starting a Timer Clock (separate thread)

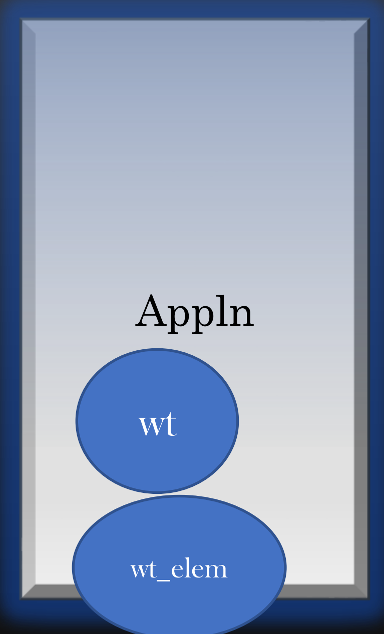
```
wheel_timer_t *wt =
        init_wheel_timer(60, 1, TIMER_SECONDS);
start_wheel_timer(wt);
```

Timers

How Timer Works ?

Suppose the appln wants to trigger fn foo( ) with
arg mem whose size is mem_size after n seconds from now

ticking



Event Registration

```
wheel_timer_elem_t *wt_elem=
            timer_register_app_event (
                        wt,
                        foo,
                        (void *)mem,
                        mem_size,
                        n * 1000, // in millisec, multiple of 1000
                        0);         // 1 for repeat
```

When clock hits the slot,
foo(mem) is invoked by timer

foo()
mem
mem_size

Common Error : Appln must not free mem while the event is scheduled with timer On mem (Memory Corruption)

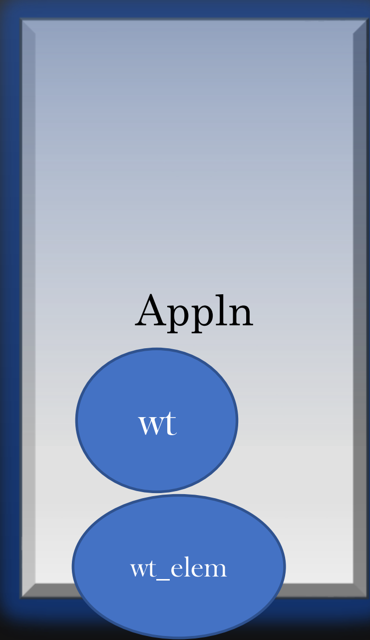
foo() must be of prototype

```
typedef void (*app_call_back)(void *arg, uint32_t sizeof_arg);
```

Timers

How Timer Works ?

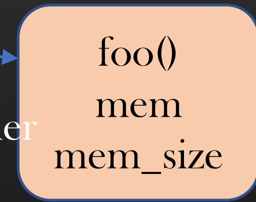
Suppose the appln wants to cancel the already scheduled event with timer



Event De-Registration

```
void *mem =
              wt_elem_get_and_set_app_data(wt_elem, 0);

timer_de_register_app_event(wt_elem);

free(mem); // if appln wishes to
```

When clock hits the slot,
foo(mem) is invoked by timer



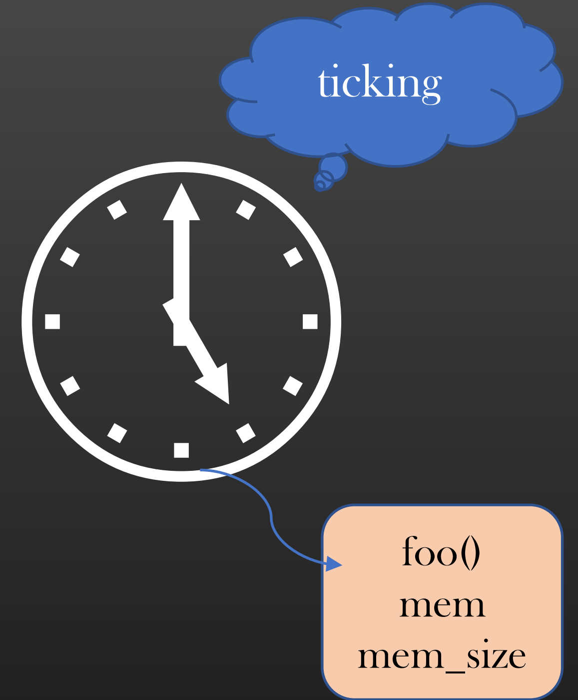
Timers

How Timer Works ?

Demo !

Timers

Other Helpful Timer APIs




```
void
timer_reschedule(wheel_timer_elem_t *wt_elem,
                 int new_time_interval);
```

```
int
wt_get_remaining_time(wheel_timer_elem_t *wt_elem);
```

```
char*
hrs_min_sec_format(unsigned int seconds);
```

```
void
isis_start_sending_hellos(interface_t *intf);
```

- Setup a timer to send Hello packets out of the Interface periodically (hello_interval)
- wheel_timer_t *wt = node_get_timer_instance(node);
- byte *hello_pkt = isis_prepare_hello_pkt(intf, &hello_pkt_size);
- Use timer_register_app_event( ) to schedule periodic hellos

```
void
isis_stop_sending_hellos(interface_t *intf);
```

- Do opposite !
- Let's code straightaway !

To Send out the pkt out of interface :

```
int
send_pkt_out (char *pkt, uint32_t pkt_size,
        interface_t *intf);
```

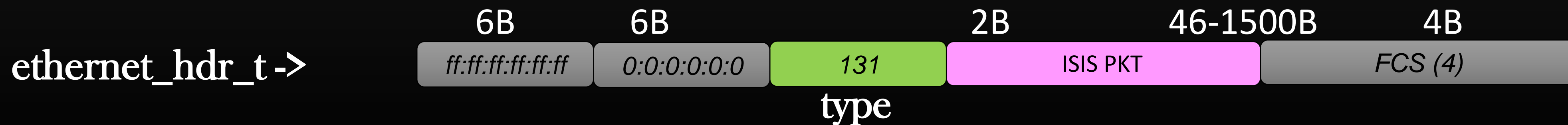
return -1 if unsuccessful
return pkt_size if success

- TCP/IP Stack library has a tcpdump equivalent utility which can print the pkts ingressing/egressing on an interface of a device
- Let's call this utility as packet capture utility (pcap)
- This pcap utility works at just above physical layer of tcp/ip stack implementation
- We demonstrated this utility in *Packet Capture and Debugging* ***Lecture*** under section *Schooling – Get familiar with TCP/IP Stack library.*
- Pls recap if required
- Currently, pcap utility do not know how to read and print ISIS Hello packets.
- Application (isis protocol) has to convey the debugging infra of tcpip/stack library how to read and print ISIS protocol packets
- Steps involved :

TCP/IP Stack Library debugging infra will use isis_print_hello_pkt( ) fn to log the packets in log files.
I hope you completed the assignment !
API to use :

nfc_register_for_pkt_tracing (ISIS_ETH_PKT_TYPE, isis_print_pkt);
where isis_print_pkt ( ) is of type : typedef void (*nfc_app_cb)(void *, size_t);

ethernet_hdr_t ->



Implementation Steps in this section

1. Hello packet format
2. Cook up hello packets from Node & Interface data
3. Send out hello packets periodically out of protocol enabled interfaces
4. Stop hello packets when protocol is disabled on interface
5. Receiving Hello packets by recipient node
6. Processing Hello packets
7. Creating Adjacency Objects from Hello pkt processing
8. Enhancing show command to display Adjacency
9. Timer managing the Adjacency Object
10. Updating Hellos as a result of generic intf config change by admin
11. Adjacency State Machine

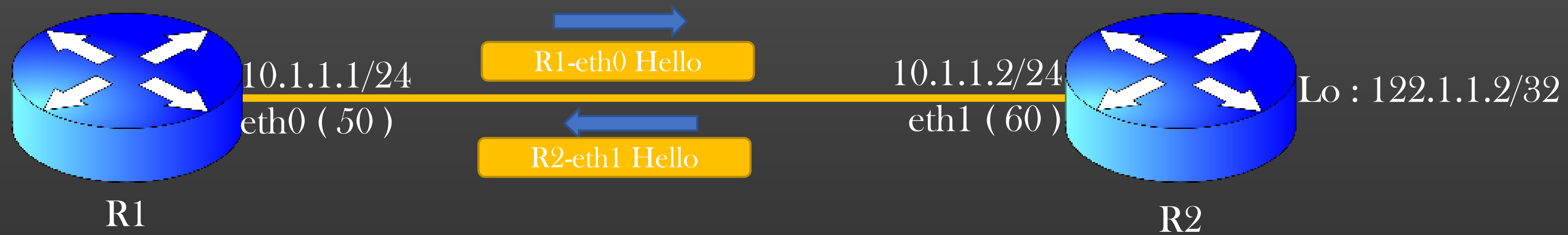
- ISIS protocol has installed the trap rule with TCP/IP Stack library
- All packets in which ethernet frame encapsulates protocol 131 shall be trapped to ISIS appln

```
void
isis_pkt_recieve(void *arg, size_t arg_size) {

        /* Receive Hello / LSP Packets */
}
```

```
isis_process_hello_pkt(node_t *node,
                       interface_t *iif,
                       ethernet_hdr_t *hello_eth_hdr,
                       size_t pkt_size);
```

```
isis_process_lsp_pkt(node_t *node,
                     interface_t *iif,
                     ethernet_hdr_t *lsp_eth_hdr,
                     size_t pkt_size);
```

- ISIS as an application has to process its control packets – LSPs or Hellos




```
static void
isis_process_hello_pkt(node_t *node,
                       interface_t *iif,
                       ethernet_hdr_t *hello_eth_hdr,
                       size_t pkt_size);
```

```
void
isis_update_interface_adjacency_from_hello(
        interface_t *iif,
        byte *hello_tlv_buffer,
        size_t tlv_buff_size);
```

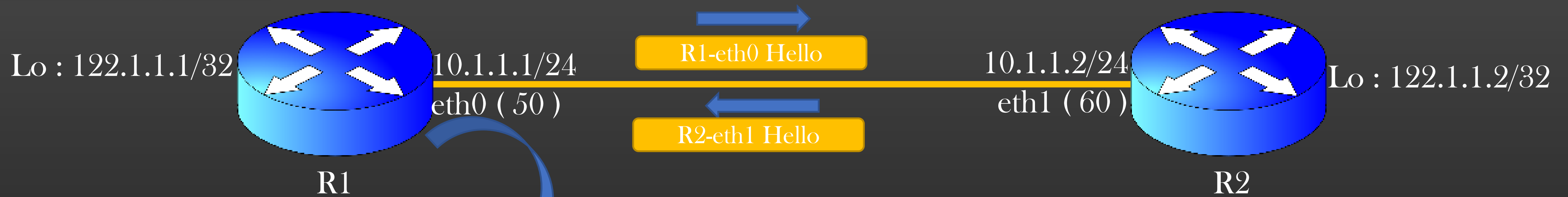
Implementation Steps in this section

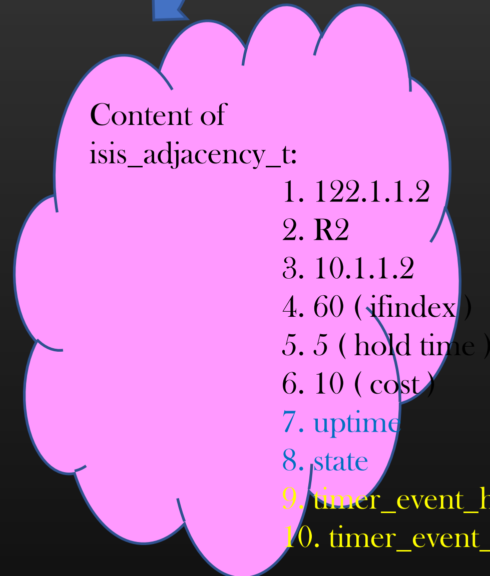
1. Hello packet format
2. Cook up hello packets from Node & Interface data
3. Send out hello packets periodically out of protocol enabled interfaces
4. Stop hello packets when protocol is disabled on interface
5. Receiving Hello packets by recipient node
6. Processing Hello packets
7. Creating Adjacency Objects from Hello pkt processing
8. Enhancing show command to display Adjacency
9. Timer managing the Adjacency Object
10. Updating Hellos as a result of generic intf config change by admin
11. Adjacency State Machine

Implementation Steps in this section

1. Hello packet format
2. Cook up hello packets from Node & Interface data
3. Send out hello packets periodically out of protocol enabled interfaces
4. Stop hello packets when protocol is disabled on interface
5. Receiving Hello packets by recipient node
6. Processing Hello packets
7. Creating Adjacency Objects from Hello pkt processing
8. Enhancing show command to display Adjacency
9. **Timer managing the Adjacency Object**
10. Updating Hellos as a result of generic intf config change by admin
11. Adjacency State Machine

Adjacency Timers

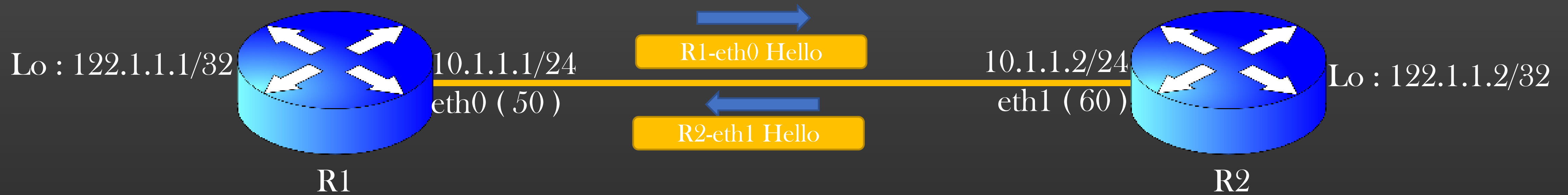




- Adjacency object is managed by Two timers :
  - Delete timer
    - When Started : Started when Adj move to DOWN state
    - Duration : = 5 sec
    - Action : Delete the Adjacency permanently
    - Cancelled : when Adj move out of DOWN state Or Deleted by Admin Action (proto disabled on intf)
    - Refreshed : Never
  - Expiry timer
    - When Started : When Adj move to INIT/UP state
    - Duration : = hold time
    - Action : Move the Adj to DOWN state and start delete timer
    - Cancelled : When Adj is Deleted by Admin Action (proto Disabled on intf)
    - Refreshed : When “good” hello pkt arrives

Note : Expiry and Delete Timers are mutually exclusive, only one runs at any given point of time

Adjacency Deletion



Ques : Suppose Adjacency exist on interface eth0 of Device R1 in up state. Now, Suppose the protocol running on device R2 crashed. How is the Adj on eth0 on Device R1 is deleted ?

Ans :

1. Adj R1-eth0 stays in UP state for *hold-time* sec.
2. Expiry timer of Adj fires when *hold-time* sec elapsed
3. Adj is moved to DOWN state, delete timer is triggered
4. After 5 sec, Delete timer fires and delete the Adj

Adjacency Timers

isis_adjacency.c

- Adjacency object is managed by Two timers :
  - Delete timer
    - When Started : Started when Adj move to DOWN state
    - Duration : = 5 sec
    - Action : Delete the Adjacency permanently
    - Cancelled : when Adj move out of DOWN state Or Deleted by Admin Action (proto disabled on intf)
    - Refreshed : Never
  - Expiry timer
    - When Started : When Adj move to INIT/UP state
    - Duration : = hold time
    - Action : Move the Adj to DOWN state and start delete timer
    - Cancelled : When Adj is Deleted by Admin Action (proto Disabled on intf)
    - Refreshed : When “good” hello pkt arrives

Note : Timers are mutually exclusive, only one runs at any given point of time

Once We implement these APIs , we shall be in position to start the Implementation of adjacency state transition

APIs :

```
static void
isis_adjacency_start_delete_timer(
        isis_adjacency_t *adjacency);

static void
isis_adjacency_stop_delete_timer(
        isis_adjacency_t *adjacency);

static void
isis_adjacency_start_expiry_timer(
        isis_adjacency_t *adjacency)

static void
isis_adjacency_stop_expiry_timer(
        isis_adjacency_t *adjacency);

static void
isis_adjacency_refresh_expiry_timer(
        isis_adjacency_t *adjacency);
```

Show command Enhancement :

Enhance *show node <node-name> protocol* isis **to display Adjacencies** Expiry & Delete timers**, along with** Uptime **in** hrs:min:sec format

```
Adjacencies :
    Nbr : R2(122.1.1.2)
    Nbr intf ip : 20.1.1.2  ifindex : 106
    Nbr Mac Addr : 50:6d:e6:e5:00:00
    State : Up   HT : 6 sec   Cost : 10
    Expiry Timer Remaining : 5000 msec
    Delete Timer : Nil
    Up Time : 0::0::8
```

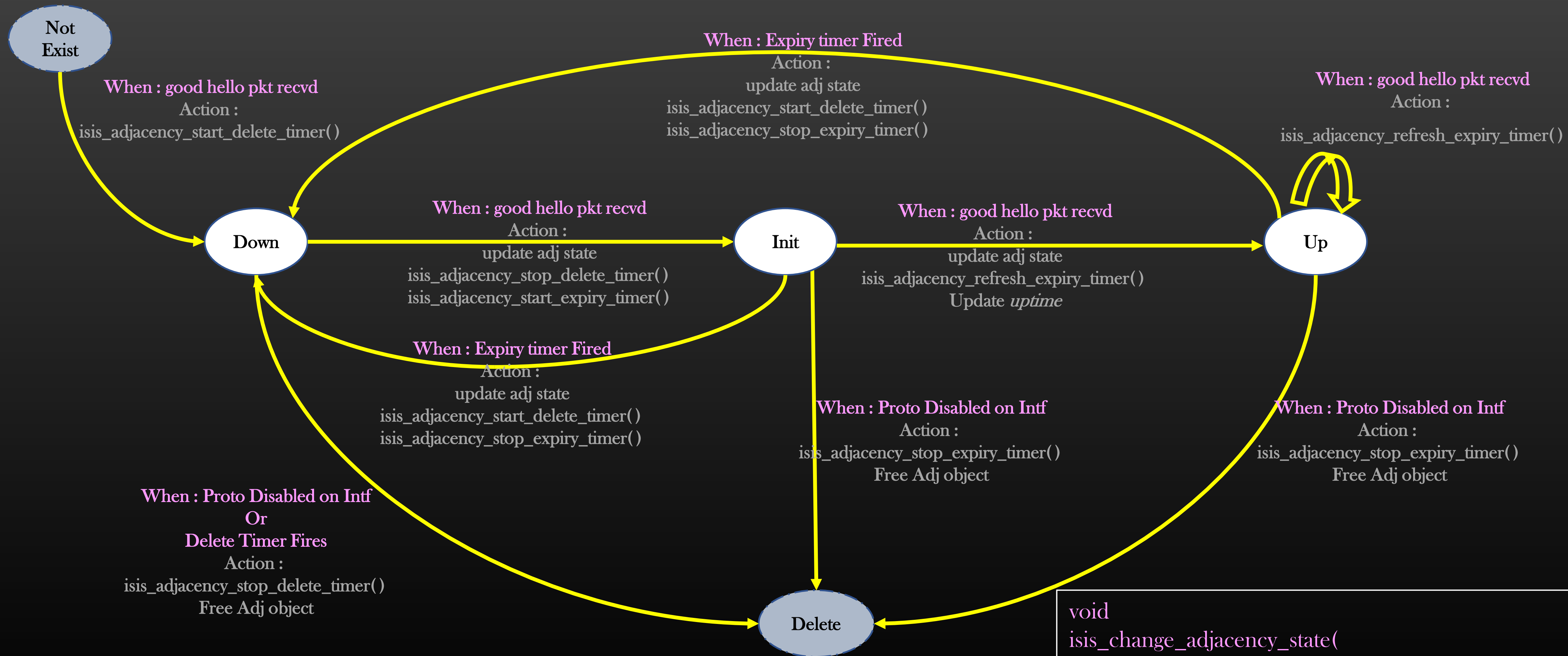
```
Adjacencies :
    Nbr : R2(122.1.1.2)
    Nbr intf ip : 20.1.1.2  ifindex : 106
    Nbr Mac Addr : 50:6d:e6:e5:00:00
    State : Down   HT : 6 sec   Cost : 10
    Expiry Timer : Nil
    Delete Timer Remaining : 5000 msec
```

APIs to be used :

wt_get_remaining_time()
hrs_min_sec_format()

isis_show_adjacency() ← to be enhanced

Adjacency State Transition Diagram

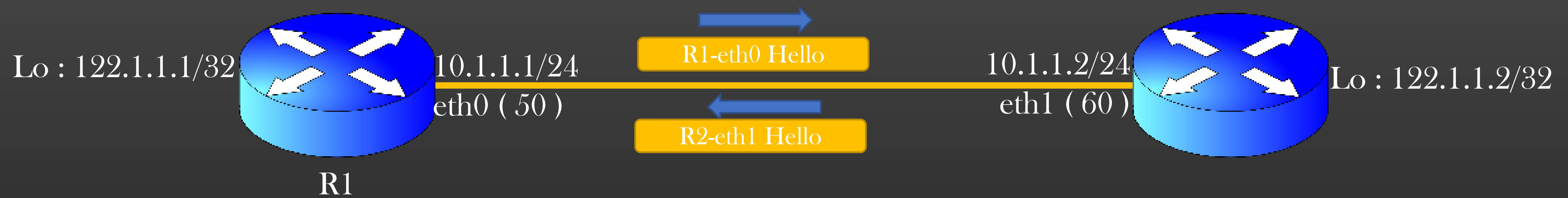



```
void
isis_change_adjacency_state(
        isis_adjacency_t *adjacency,
        isis_adj_state_t new_adj_state);
```

Implementation Steps in this section

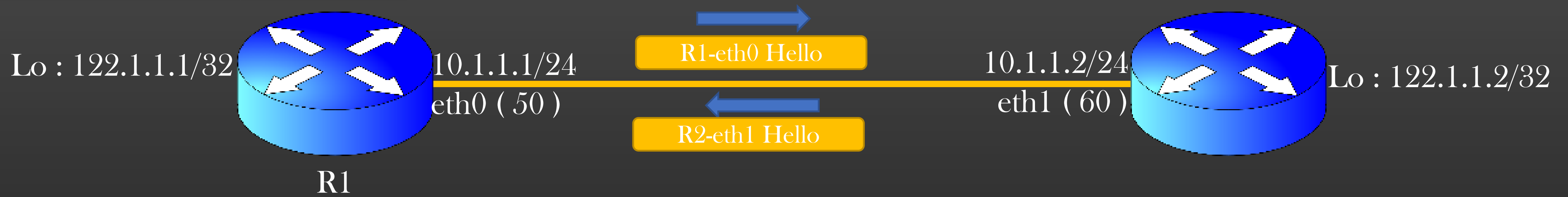
1. Hello packet format
2. Cook up hello packets from Node & Interface data
3. Send out hello packets periodically out of protocol enabled interfaces
4. Stop hello packets when protocol is disabled on interface
5. Receiving Hello packets by recipient node
6. Processing Hello packets
7. Creating Adjacency Objects from Hello pkt processing
8. Enhancing show command to display Adjacency
9. Timer managing the Adjacency Object
10. Adjacency State Machine
11. Updating Hellos as a result of generic intf config change by admin

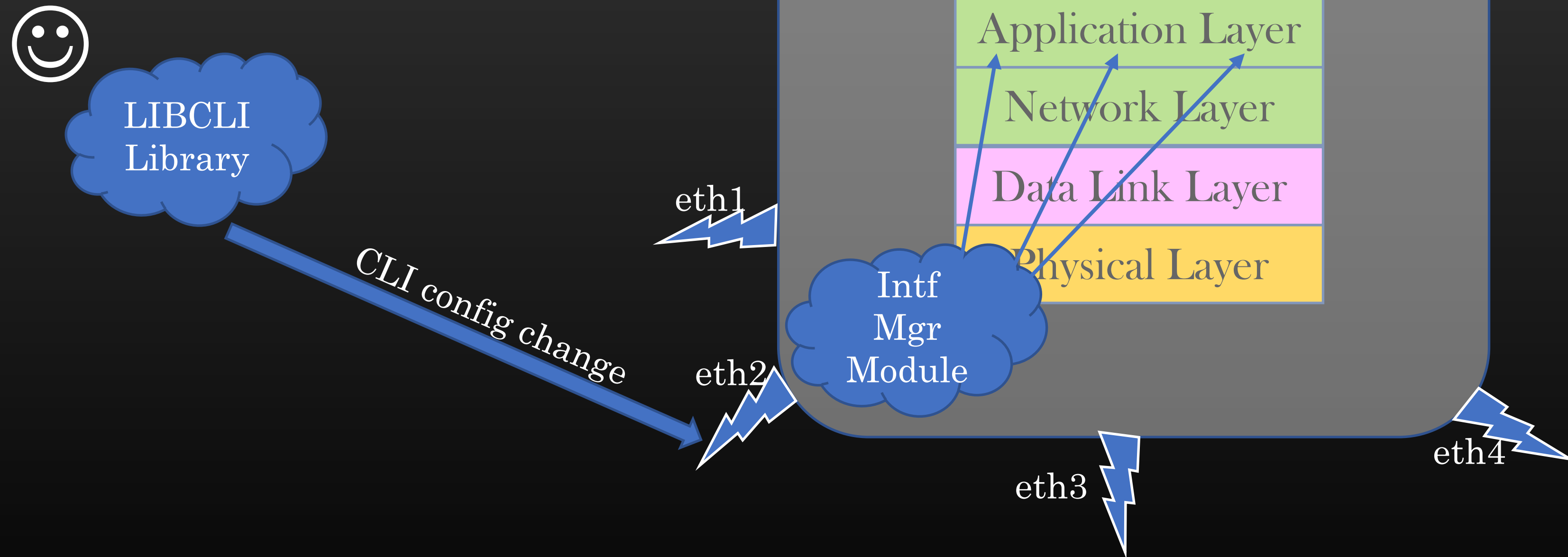
Interface Config Processing



- Let R1-th0 Hello packet contains a value of TLV ISIS_TLV_IF_IP as 10.1.1.1
- Now admin changes the config on the device R1 :
  config node R1 interface eth0 ip-address 10.1.1.5 24
- Appln protocol (ISIS) must react to this config change and update hello packet content
- Soln : Protocols must listen to interface generic config change events from lower layers of TCP/IP stack
- Companies built a separate process which manages interface generic configurations. It could be user space appln or Kernel module. Such a process pushes the config change to all user space applications when a config change event occurs

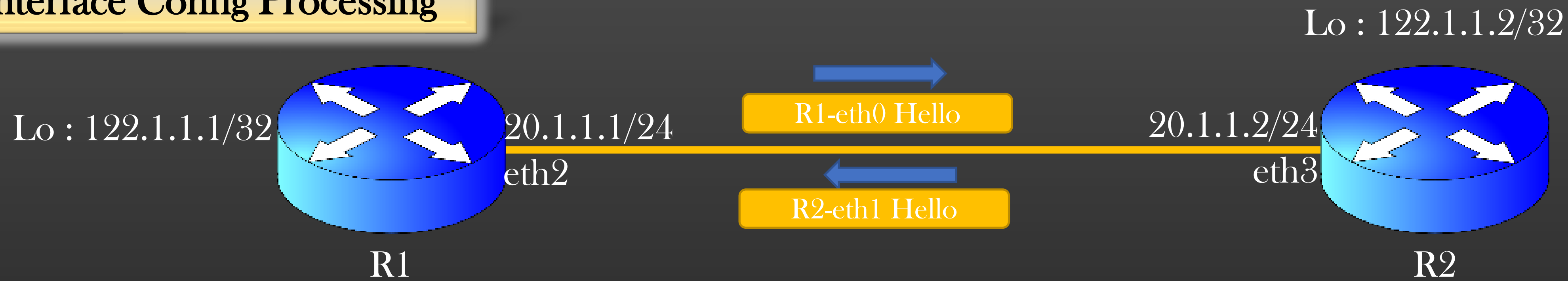
Interface Config Processing





- Let us see how an appln protocol can register with TCP/IP stack library for interface events
- In production also, All applications need to do similar registration with Intf Mgr Module during initialization

Interface Config Processing



➢ Complete Sequence of Events :

1. Adj is in UP state at both ends
2. User changes ip address on R1-eth2 to 20.1.1.5/24
3. R1 updates new IP Address in its Hello packet

4. R2 sees a hello pkt but with IP = 20.1.1.5 which is different from what it had in its adjacency object (= 20.1.1.1) . This is IP-mismatch

5. On IP-mismatch R2 does the following :
   5.1 update adjacency object with new IP 20.1.1.5
   5.2 transition adj state from UP/INIT to DOWN Immediately
   5.3 Rely on subsequent hello packets from R1 to recycle adjacency state back to INIT to UP

config node <node-name> no protocol isis

- Must disable the protocol completely on a node
  - All ISIS Data Structures must be freed
    - Node level
      - interface level
        - Free Adjacency
        - Free Periodic Hello Timer
          - check_and_delete isis_intf_info
    - check_and_delete isis_node_info

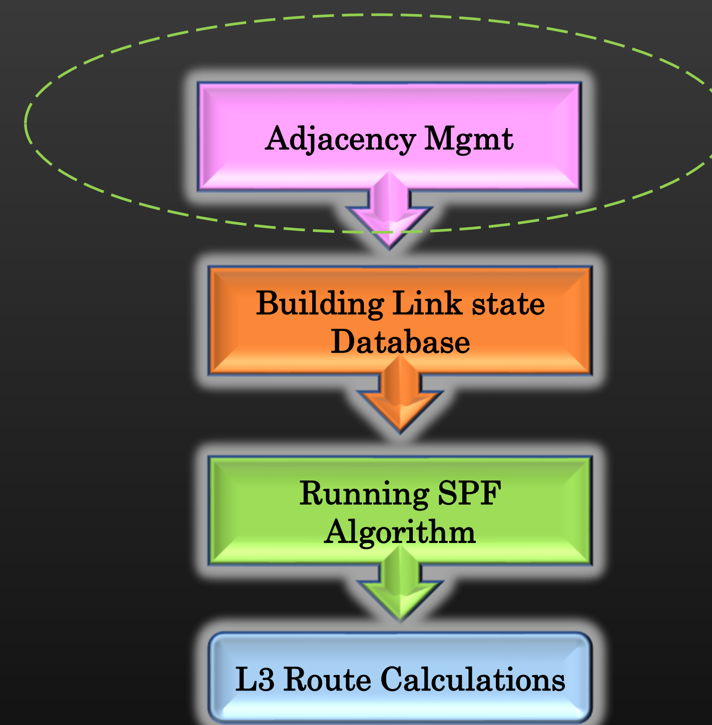
- Guidelines for shutting down the protocol :
  - Delete the child objects first
  - Then delete the parent objects
  - Free parent objects using check_delete() APIs
  - Follow Bottom-Up approach for object deletion

- Enhance isis_de_init(node_t *node) function
- Keep Revisiting isis_de_init API to shut-down ISIS features which we shall be implementing in future

Where to go from here ?

- We will be going to implement a simplified Routing Protocol in this course
- Routing protocol chosen – Interior gateway protocol (IGP , ex OSPF, ISIS)
  - Don't know about it – don't worry, we shall cover theory first before any implementation
- A typical IGP (link state) protocol functionality is divided into 4 distinct parts :

1. Adjacency Management (Each device know its neighbours)
   - Sending and Receiving hello packets periodically
   - Update neighborship state machine
2. Building Link State Database (Each device internally creates a view of topology - Graph)
   - Building Link State packets
   - Flooding link state packets
   - Build a Graph – a view of network topology
3. Running SPF algorithm (Dijkstra) on LSDB
   - Process the LSDB through the algorithm
   - Compute Results and store
   - Algorithmically challenging
4. L3 Route Calculations
   - Use Results of 3 to compute final L3 routes and update Routing Table
   - Algorithmically challenging



We shall be going to implement all 4 parts in this course series

Along the journey we shall implement various sub-features within the protocol

Problem Statement

R1's ARP Table

IP Address	MAC	Intf

R1's Routing Table

IP Address	Gateway Ip	OIF
100.1.1.1/32	10.1.1.2	eth0

Soln : Let R1 installs ARP entries for all its neighbors in its ARP table even when there is no traffic

Implement a Feature Layer 2 Mapping

- Prerequisite : Understand ARP
- ARP learning is triggered by the traffic (ARP Resolution)
- Traffic, at high ingress rate may get dropped when ARP resolution is in progress
- Solution : We can have ARP resolved already for each interface of a device using ISIS protocol

Content of isis_adjacency_t:

1. 122.1.1.2
2. R2
3. 10.1.1.2
4. <mac>
4. 60 (ifindex)
5. 5 (hold time)
6. 10 (cost)
7. uptime
8. state
9. timer_event_handle *delete_timer;
10. timer_event_handle *expiry_timer;

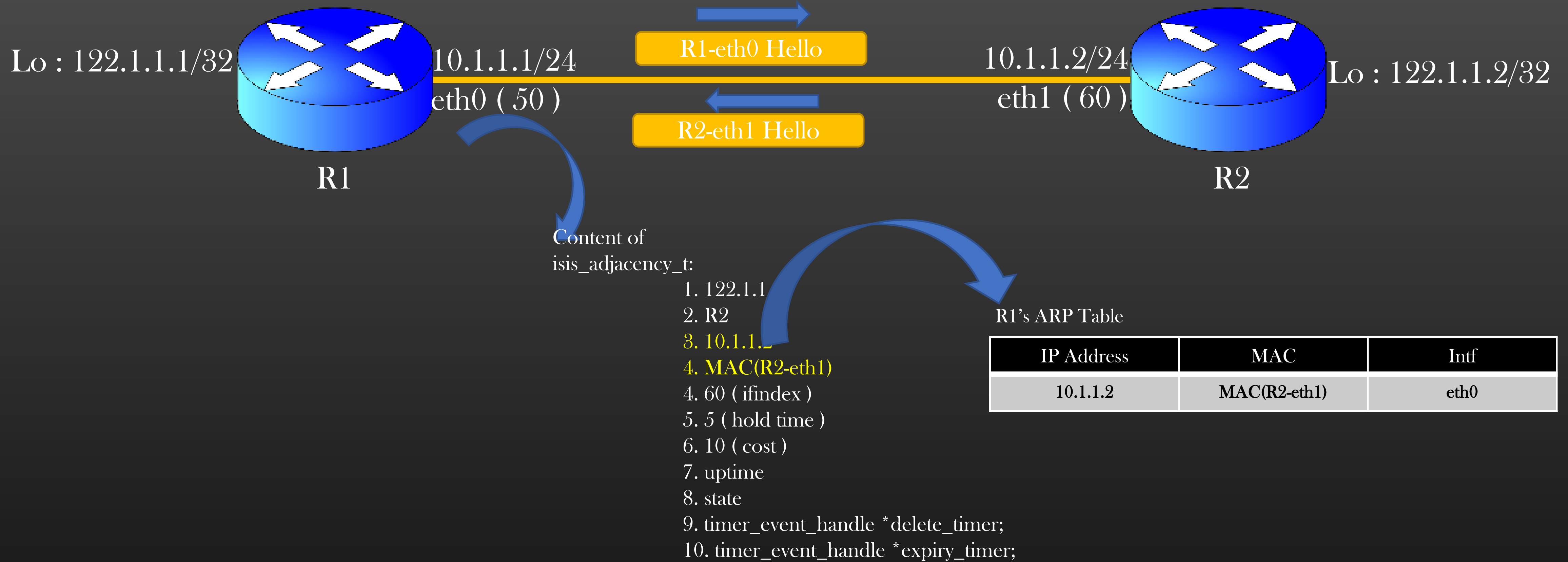
- Procedure :
  - As soon as Adjacency Goes up on an interface I:
    - Fetch Mac and intf IP from Adjacency object on an interface I
    - Populate ARP entry using below API

```
bool arp_entry_add ( node_t *node, char *ip_addr,
                     mac_add_t mac, interface_t *oif, uint16_t proto);
```

  - As soon as Adjacency goes down :
    - Delete the ARP entry for the Interface IP Address present in this Adjacency object using below API :

```
void arp_entry_delete (node_t *node, char *ip_addr, uint16_t proto);
```

Network Protocol Development → Phase 1 : Mini Project



- Result :
  - ARP entry is inserted when ISIS Adjacency goes UP on an interface
  - ARP entry is deleted when Adjacency goes DOWN on the interface
  - ARP entry should get updated when IP Address on an interface is changed by admin
  - Corresponding ARP entry should get deleted if protocol is disabled on an interface I
  - Behavior should be enabled by default
    - conf node <node-name> no protocol isis layer2-map

isis_layer2map.h / isis_layer2map.c
APIs to be written

```
typedef struct isis_node_info_ {
...

/* Layer 2 Mapping */
    bool layer2_mapping;

...
} isis_node_info_t;
```

```
/* Return True if layer 2 mapping is enabled, else return false */
bool
isis_is_layer2_mapping_enabled (node_t *node);

/* Enable layer2 mapping config, return 0 on success, -1 on failure */
int
isis_config_layer2_map (node_t *node);

/* Disable layer2 mapping config, return 0 on success, -1 on failure */
int
isis_un_config_layer2_map (node_t *node);

/* install ARP entry in ARP table when Adjacency goes up, return true on success else false*/
bool
isis_update_layer2_mapping_on_adjacency_up (isis_adjacency_t *adjacency);

/*Remove arp entry from ARP table when Adj goes down, return true on success else false */
bool
isis_update_layer2_mapping_on_adjacency_down (isis_adjacency_t *adjacency);
```

TestCases :

1. Should be enabled by default
2. All Layer 2 mapping on a node should get deleted on disabling protocol on a node
3. All Layer 2 mapping on a node should get re-added back on re-enabling the disabled protocol
4. A Layer 2 mapping on a node should get deleted when adj on an interface goes down Or deleted
5. A Layer 2 mapping on a node should get re-added back when adj on an interface goes up
6. *clear node <node-name> protocol isis adjacency* should delete all ISIS Adjacencies + All Layer 2 mapping, but as adjacencies are re-formed again, layer 2 mapping must install back to ARP table
7. Layer 2 mapping on local node should get deleted when
   1. Local interface is shut down (instantly)
   2. Remote peer interface is shutdown (due to Adj time out)
8. Local Layer 2 mapping must get updated when nbr intf ip address is changed