Concept of TLVs – Type Length Value

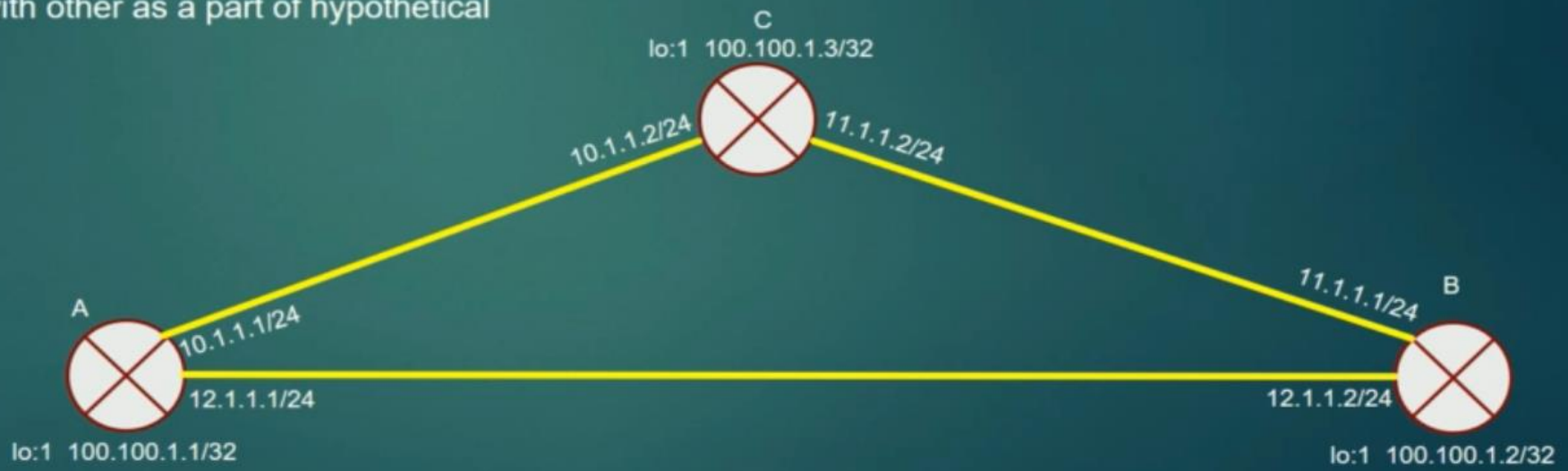
TOPIC COVERED IN THIS LECTURE

- Type Length Value: Introduction
- Need of TLVs
- Understanding TLVs
- Data Structure – STREAMS
- TLV De-Serialization using STREAMS

TLVs - Introduction

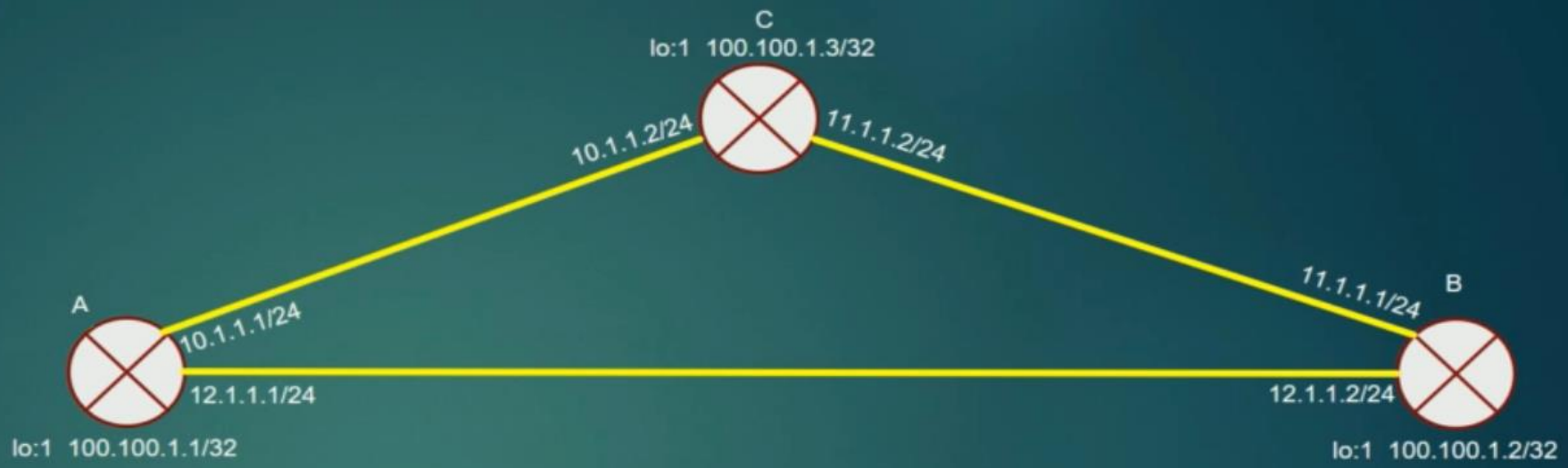
- TLV – Type Length Value
- Let us first try to understand the problem which TLV solves, and then we shall discuss What TLVs are and how they are used
- It is a very common scenario in Networking that Machines often exchange messages with each other. Many Internet routing protocols necessitate Machines to exchange various messages with each other periodically.
- For example, If you remember, Interior Gateway protocols such as OSPF exchange their Link state packets with other routers in the network for their proper functioning.
- To understand the problem, Let's say Machines A, B and C are exchanging the following msg with other as a part of hypothetical functionality P

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        uint link1_bw;
        uint link2_bw;
}
```



TLVs - Introduction

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        uint link1_bw;
        uint link2_bw;
}
```




```
struct xmsg{
        100.100.1.1
        A
        10.1.1.1
        12.1.1.1
        100
        200
}
```

A

TLVs - Introduction

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        uint link1_bw;
        uint link2_bw;
}
```


```
struct xmsg{
        100.100.1.1
        A
        10.1.1.1
        12.1.1.1
        100
        200
}
```

A

```
struct xmsg{
        100.100.1.2
        B
        11.1.1.1
        12.1.1.2
        110
        220
}
```

B

```
struct xmsg{
        100.100.1.3
        C
        10.1.1.2
        11.1.1.2
        90
        190
}
```

C

TLVs - Introduction

- The problem in such exchange of messages arises due to heterogeneity of communicating machines
- Heterogeneity reasons could be mannnyyy ….
  - Different manufacturing vendors
  - Using different Hardware and Technologies
  - Using Different C compilers
  - And so on . . .

- We cannot ask all the vendors around the world to manufacture their network equipment's using Identical technologies and hardware !

TLVs - Introduction

- So let us try to understand the technical glitches that arises due to heterogeneity of the communicating machines in the network

We will discuss two scenarios :

- When machines are distinct and incompatible
- When selective machines in the network are upgraded

Why We Need TLVs

Ok, before going forward , let us revise our C knowledge a bit …

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        uint link1_bw;
        uint link2_bw;
}
```

loopbck_ip
router_name
if_addr1
if_addr2
link1_bw
link2_bw

On a 32 bit system

Fields	Size	offset
loopbck_ip	4	0
router_name	32	4
if_addr1	4	36
if_addr2	4	40
link1_bw	4	44
link2_bw	4	48

Struct xmsg *ptr;
ptr->if_addr2 -- reading/writing 4 bytes @40th byte from starting address

Why We Need TLVs

- Ok, now let us see the real problem
- Let us Say Machine A is a 32 bit machine, and machine B is a 64 bit machine.
- It means, sizeof(uint) on A is 4 bytes, whereas it is 8 bytes on B
- Now, let see the xmsg layout on wire when they are generated by machine A and B respectively.


```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        uint link1_bw;
        uint link2_bw;
}
```

A

100.100.1.1	A	10.1.1.1	12.1.1.1	100	200
4	32	4	4	4	4

B

100.100.1.2	B	11.1.1.1	12.1.1.2	110	220
8	32	8	8	8	8

Why We Need TLVs

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        uint link1_bw;
        uint link2_bw;
}
```

When A receives xmsg from B, A will typecast the msg according to its belief of definition of xmsg:

- So, When machine A receive the xmsg from machine B over the network, A will type cast the msg according to its own definition of struct xmsg :

```
struct xmsg *recv_msg = (struct xmsg *)buffer;
recv_msg->uint_loopback_ip;    /*Instead of reading 8 bytes, A will read only 4 bytes*/
recv_msg->router_name;         /*From B's perspective, it is 8th byte from start of msg, from A's perspective it is 4th byte from start of msg*/
recv_msg->if_addr1;
recv_msg->if_addr2;
recv_msg->link1_bw;
recv_msg->link2_bw;
```

Why We Need TLVs – Another Scenario

- Lets see the same problem due to other reason.
- Let us Say Machine A's software is upgraded such that it introduces a new member in the structure.

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        char if_mac1[6];
        char if_mac2[6];
        uint link1_bw;
        uint link2_bw;
}
```

A

100.100.1.1	A	10.1.1.1	12.1.1.1	100	200
4	32	4	4	4	4

A.1

100.100.1.2	B	11.1.1.1	12.1.1.2	If1_MAC	If2_MAC	100	200
8	32	8	8	6	6	4	4

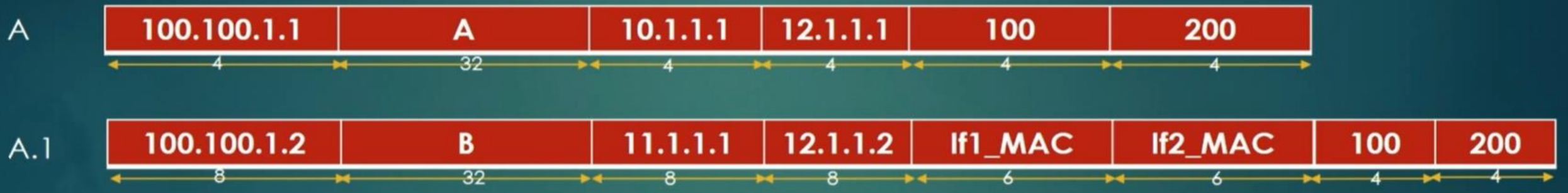
Why We Need TLVs – Another Scenario

A.1

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        char if_mac1[6];
        char if_mac2[6];
        uint link1_bw;
        uint link2_bw;
}
```

B and C

```
struct xmsg{
        uint loopbck_ip;
        char router_name[32];
        uint if_addr1;
        uint if_addr2;
        uint link1_bw;
        uint link2_bw;
}
```



Machines B and C, when receives the new msg A.1 generated by machine A, they will try to read the msg according to their own definition of struct xmsg. Again Data corruption !

So many problems !! Vendor manufacturer has invented his new patented technology but he cannot upgrade his software with new technology because other machines in the network wont work with his new version of Software ! Competitors are happy ! Funny !!

Why We Need TLVs – Another Scenario

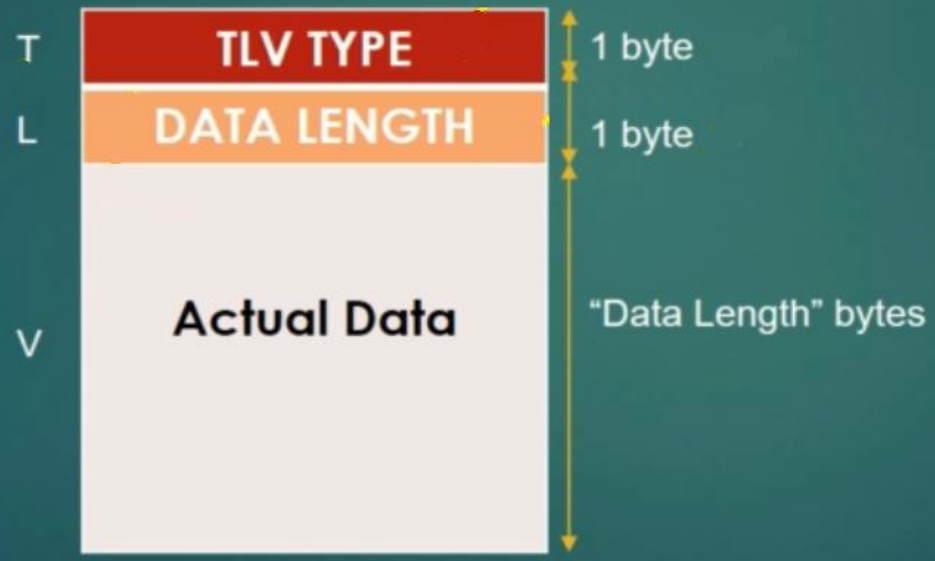
- Networking is a field where various network equipment's being manufactured by various vendors, need to work In complete cooperation and harmony with each other for the network protocol to work.

- Machines need to comply with each other for the network functionality to work correctly, yet at the same time Network Vendors should be free to innovate/upgrade/update their software without breaking the existing compliance with the other Machines deployed in the network.

The concept of TLVs Solves these problems very easily

Understanding the concept of TLVs

- TLV (Type length value)
  - Is a mechanism to encode the data in the format that is independent of
    - Machine Architecture
    - Underlying Operating system
    - Compiler
    - Programming language
  - TLVs has three components :



TLV TYPE – TLV UNIQUE IDENTIFER (0-255)

TLV LENGTH – The length of the actual data

Understanding the concept of TLVs

- Example :
  - Suppose Machine A wants to send machine B, the set of all IP addresses which is configured on all its interfaces

  - We can take any number as TLV type. Let's take it as 132.
  - Next we need to define the definition of TLV 132 :
    - 4 byte integer number (which is ip address)
    - 1 byte mask value
    - This is called *TLV definition*
  - Any machine which is suppose to process this TLV when received, us suppose to be aware of TLV definition.

132
20
201392385
24
234946817
24
218169601
24
184615169
24

Understanding the concept of TLVs

132
20
201392385
24
234946817
24
218169601
24
184615169
24

- When receiving machine receives this TLV :
  - It reads first byte
  - Now it knows that it is TLV 132
  - It means, its unit data size is 5 bytes,
    4 bytes of ip address followed by 1 byte of mask value
  - Read next 1 byte which is 20
  - Divide 20 by 5 = 4
  - Now, machine knows it has four occurrence of unit data type
  - Iterate over rest of the data and read all units of data

Thank You