CSE 489/589
Programming Assignment 3

Outline

Part 1 – Assignment Overview

- Problem statement
- Programming environment
- Topology
- Differences from Text-Book …

Part 2 – Socket Programming

- UDP socket overview
- select(), again!
- Packet Formats

Part 1:

Assignment Overview

Problem statement

- Implement a simplified version of the Distance Vector Protocol

- Basic idea:
  - When x receives new DV estimate from neighbor, it updates its own DV using Bellman-Ford equation:

$$D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

$d_x(y)$:= cost of least-cost path from x to y

then

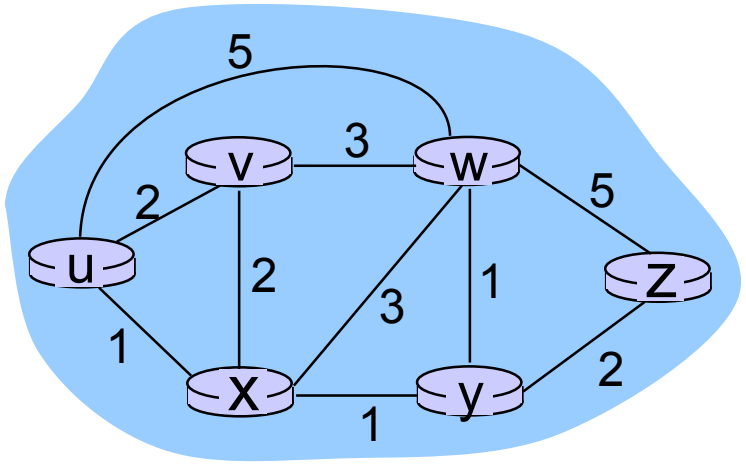
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford example



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \; c(u,x) + d_x(z), \; c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

node achieving minimum is next hop in shortest path, used in forwarding table

Distance vector algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- node x:
  - knows cost to each neighbor v: c(x,v)
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

Distance vector algorithm

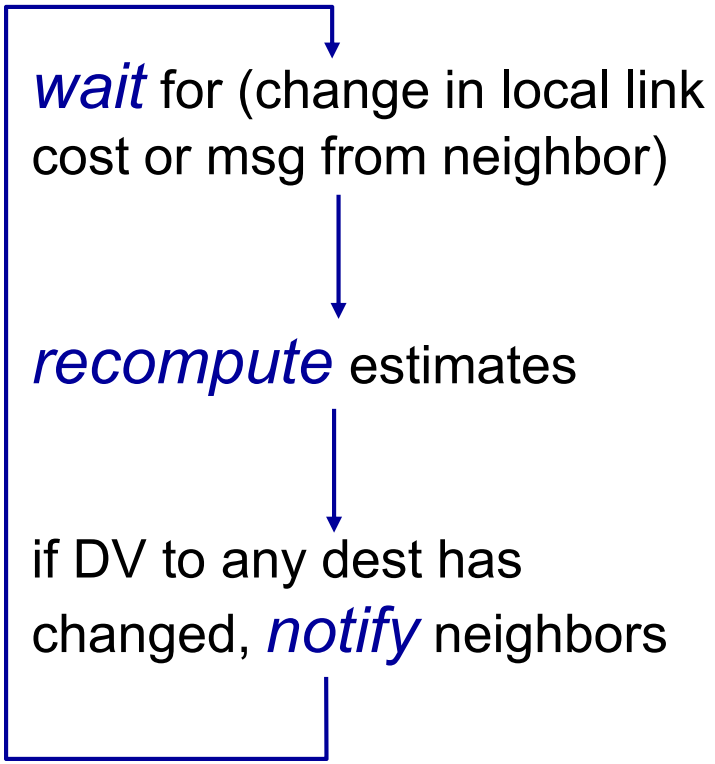
iterative, asynchronous:
each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

each node:



$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$
$= \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\}$
$= \min\{2+1\ ,\ 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

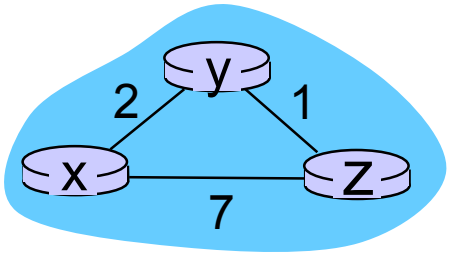
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



time

$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$
$= \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\}$
$= \min\{2+1\ ,\ 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

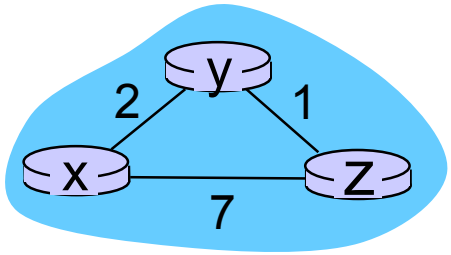
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

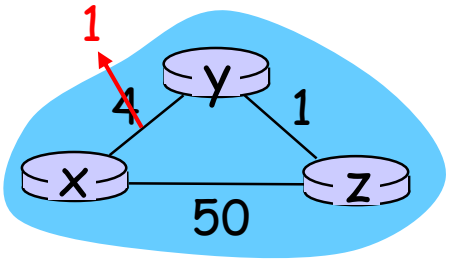


time

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

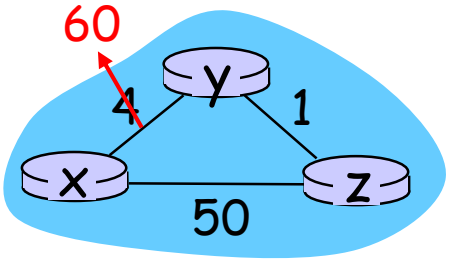
t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- *bad news travels slow* - "count to infinity" problem!
- 44 iterations before algorithm stabilizes: see text



poisoned reverse:

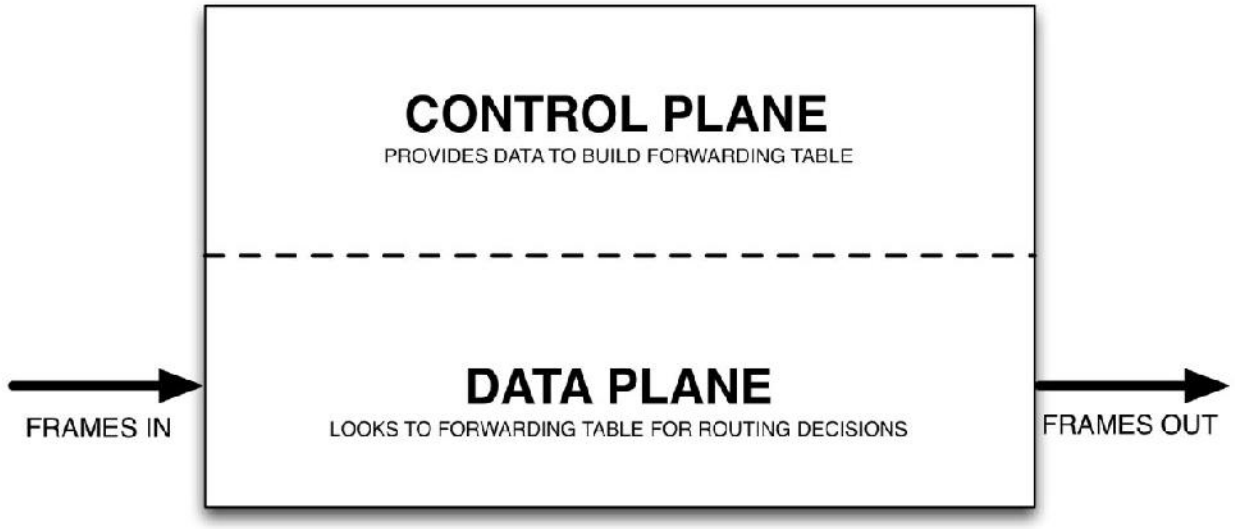
- If Z routes through Y to get to X :
  - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- will this completely solve count to infinity problem?

Programming environment

- Use 5 CSE student servers as *routers*
  - stones
  - euston
  - highgate
  - embankment
  - underground

Routers

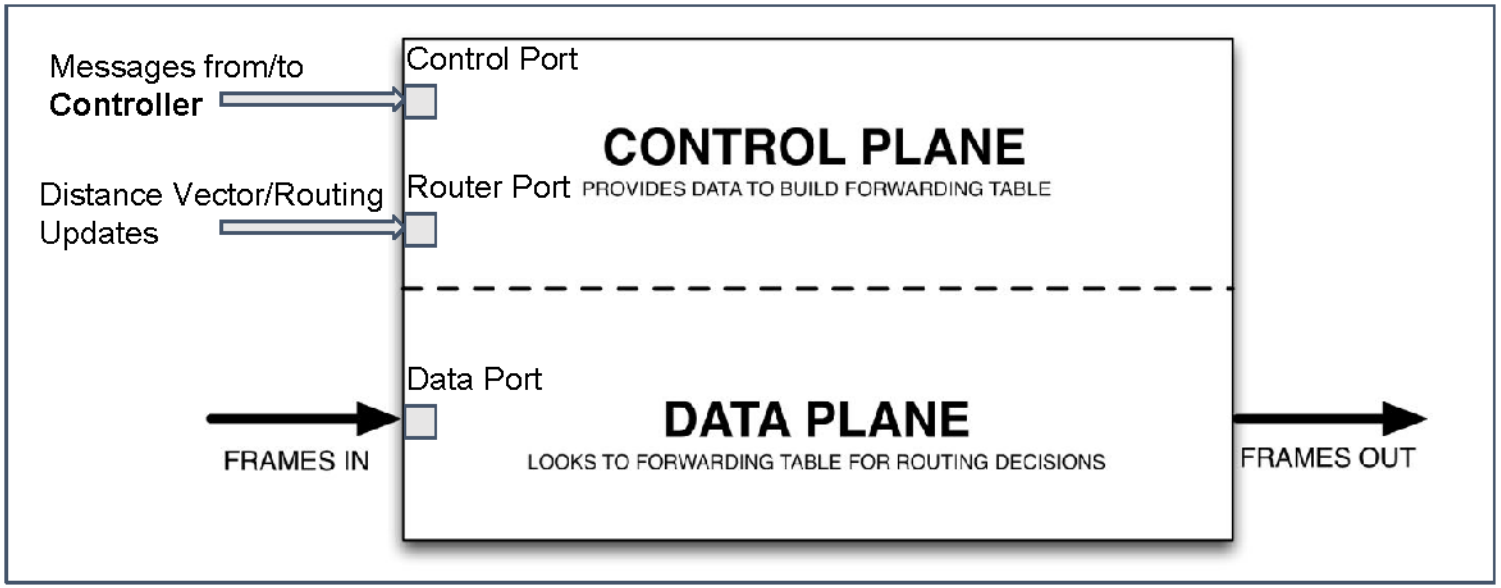
- Backbone of the internet



Graphics Source: http://blogs.salleurl.edu/data-center-solutions/2016/03/sdn-the-latest-trend-in-data-centers-e2016/

Routers: PA3

- Backbone of the internet



ROUTER APPLICATION

$ ./router <control port>

Controller

- A separate application
  - Generates *control* messages for routers
  - Expects response within a specified time limit

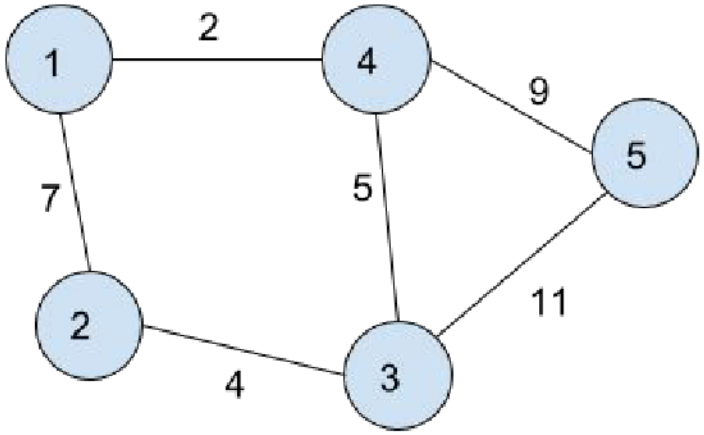
- Included in the template. No need to implement

- Router application needs to act and generate response packets to control messages

Topology

- Established by controller using INIT messages
- Reads a *topology file*
- Topology file
  - Snapshot of the network
  - Number of routers in the network and ID
  - IP address and all three port numbers
  - Links and their cost

Topology File: Example



Line #	Line entry
1	5
2	1 128.205.36.8 4091 3452 2344
3	2 128.205.35.24 4094 4562 2345
4	3 128.205.36.24 4096 8356 1635
5	4 128.205.36.4 7091 4573 1678
6	5 128.205.36.25 7864 3456 1946
7	1 2 7
8	4 5 9
9	1 4 2
10	3 4 5
11	3 2 4
12	3 5 11

<ID> <IP address> <control port> <router port> <data port>

<router ID 1> <router ID 2> <cost>

Differences from Text-Book ...

- Routing Updates: Periodic exchange Only
  - Not on DV changes

- To solve count-to-infinity?
  - No!

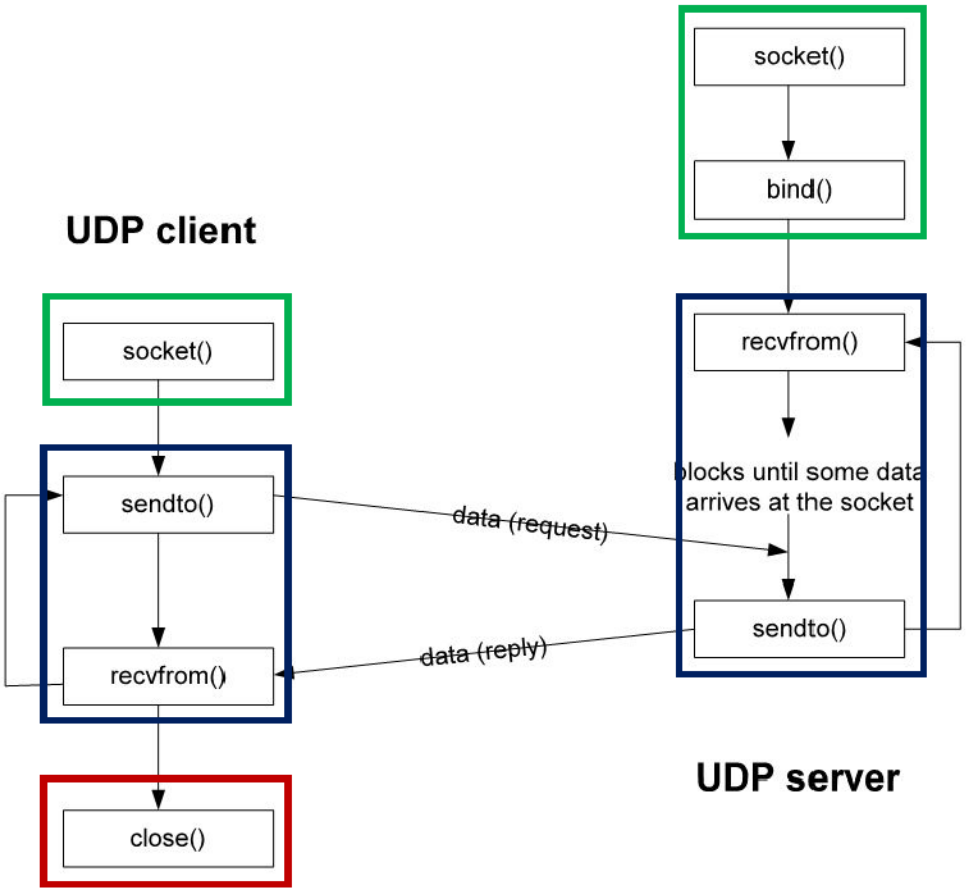
Part 2:

Socket Programming

UDP socket overview

- Client Init.(socket)
- Server Init. (socket, bind)
  - No connection establishment
- Data transfer
  - Client sendto - Server recvfrom
  - Server sendto - Client recvfrom

UDP socket overview



UDP server socket overview

```
server_socket = socket(AF_INET, SOCK_DGRAM, 0);
if(server_socket < 0)
    return err_msg_ERR("Cannot create socket");
```

```
bzero(&server_addr, sizeof(server_addr));

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(port);

if(bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0 )
    return err_msg_ERR("Bind failed");
```

PA3: Multiplexing

- Control Port
  - **TCP**
  - Controller connections/messages
- Router Port
  - **UDP**
  - Distance vector updates
- Data Port
  - **TCP**
  - Actual data packets

select() again!

- I/O multiplexing with select
  - Control, router and data ports
  - Data on any connections established on these ports
- Timeout
  - To implement periodic distance vector broadcast

```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

  - Multiple timers needed!
    - Each router may be on a different schedule

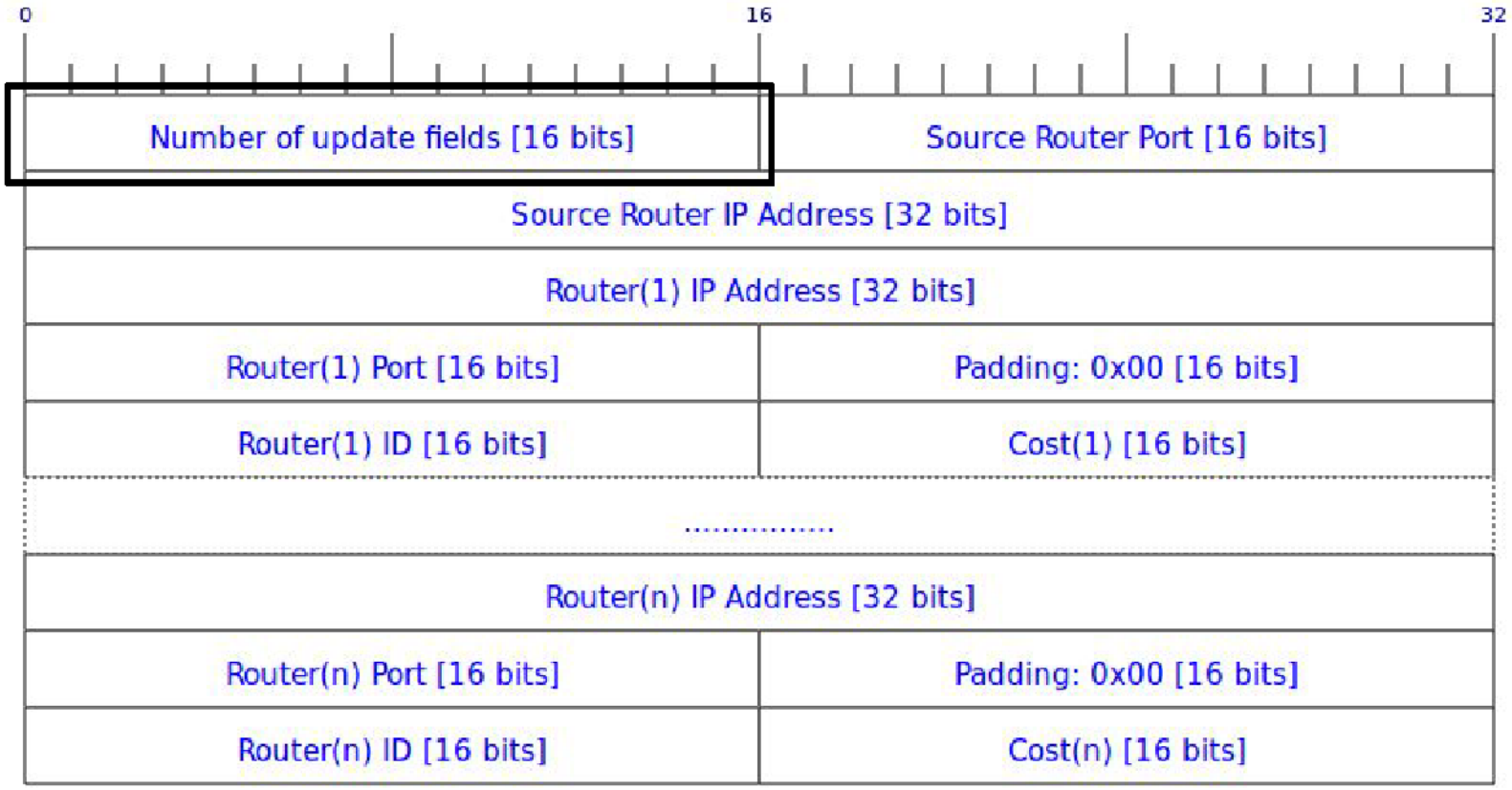
Packet Formats

- Multiple packet formats
  - Routing updates
  - Control
  - Control-Response
  - Data

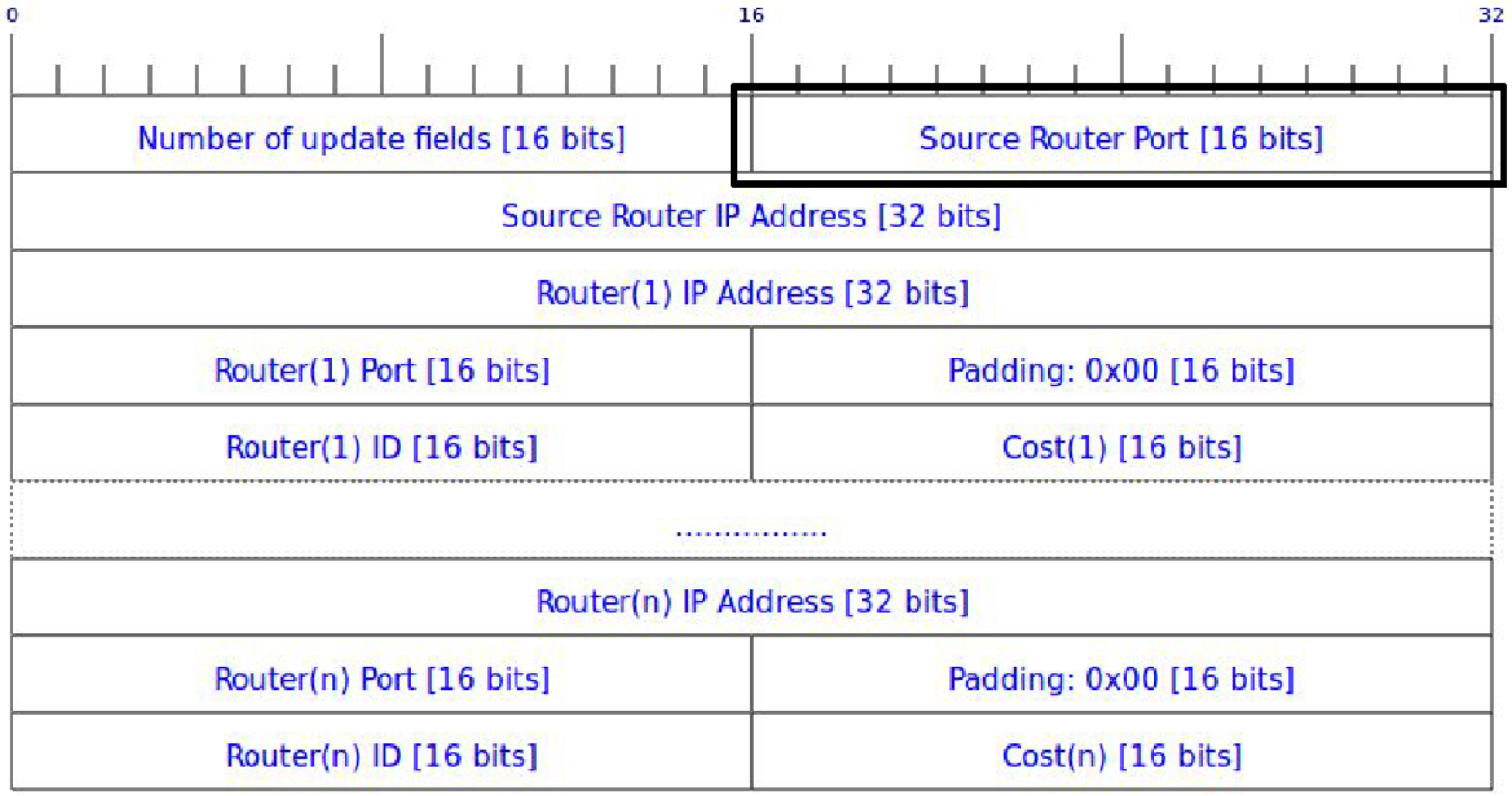
- Need to be followed exactly!

- Example: Routing Update Packet

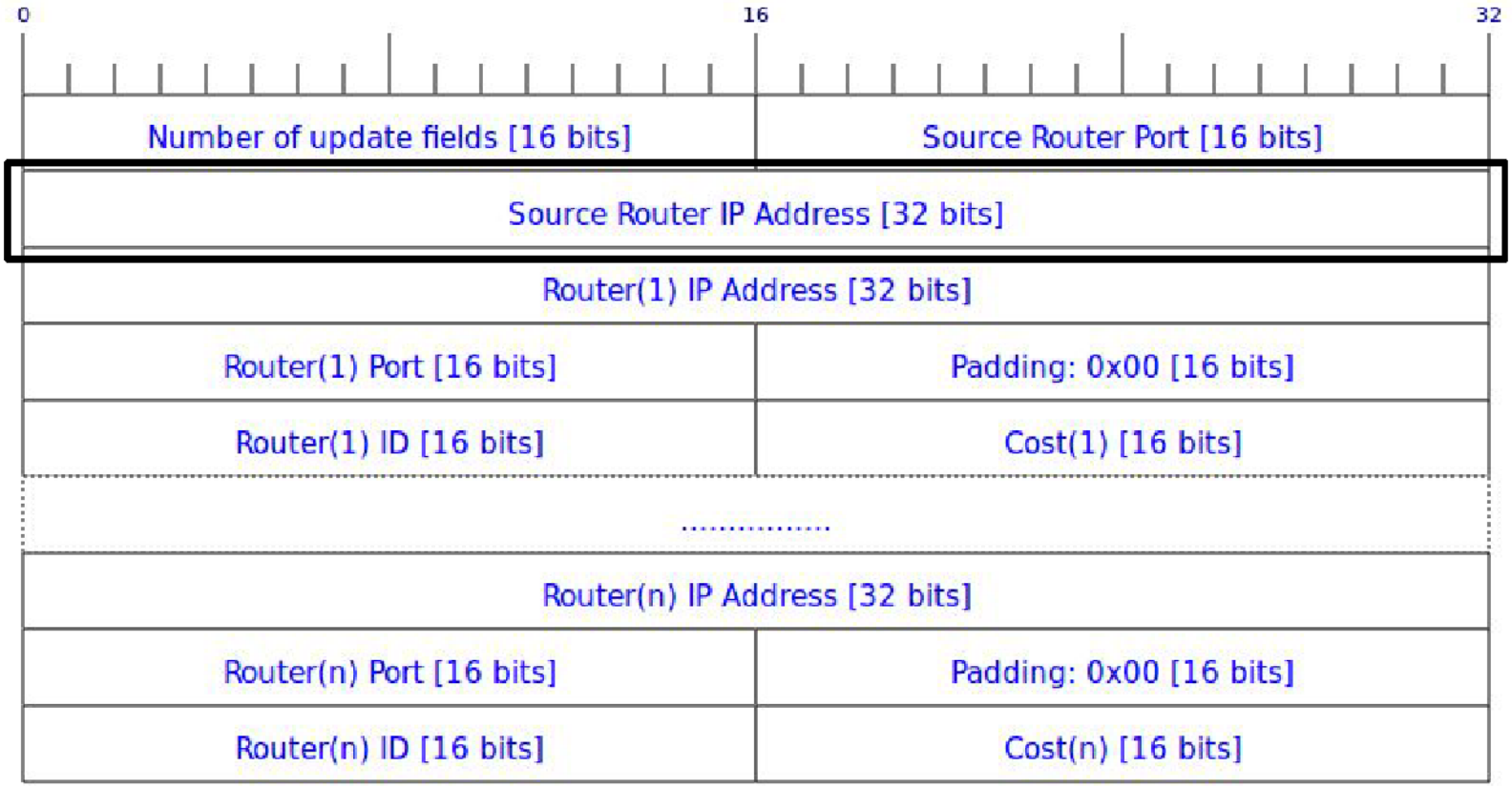
Packet Formats: Example



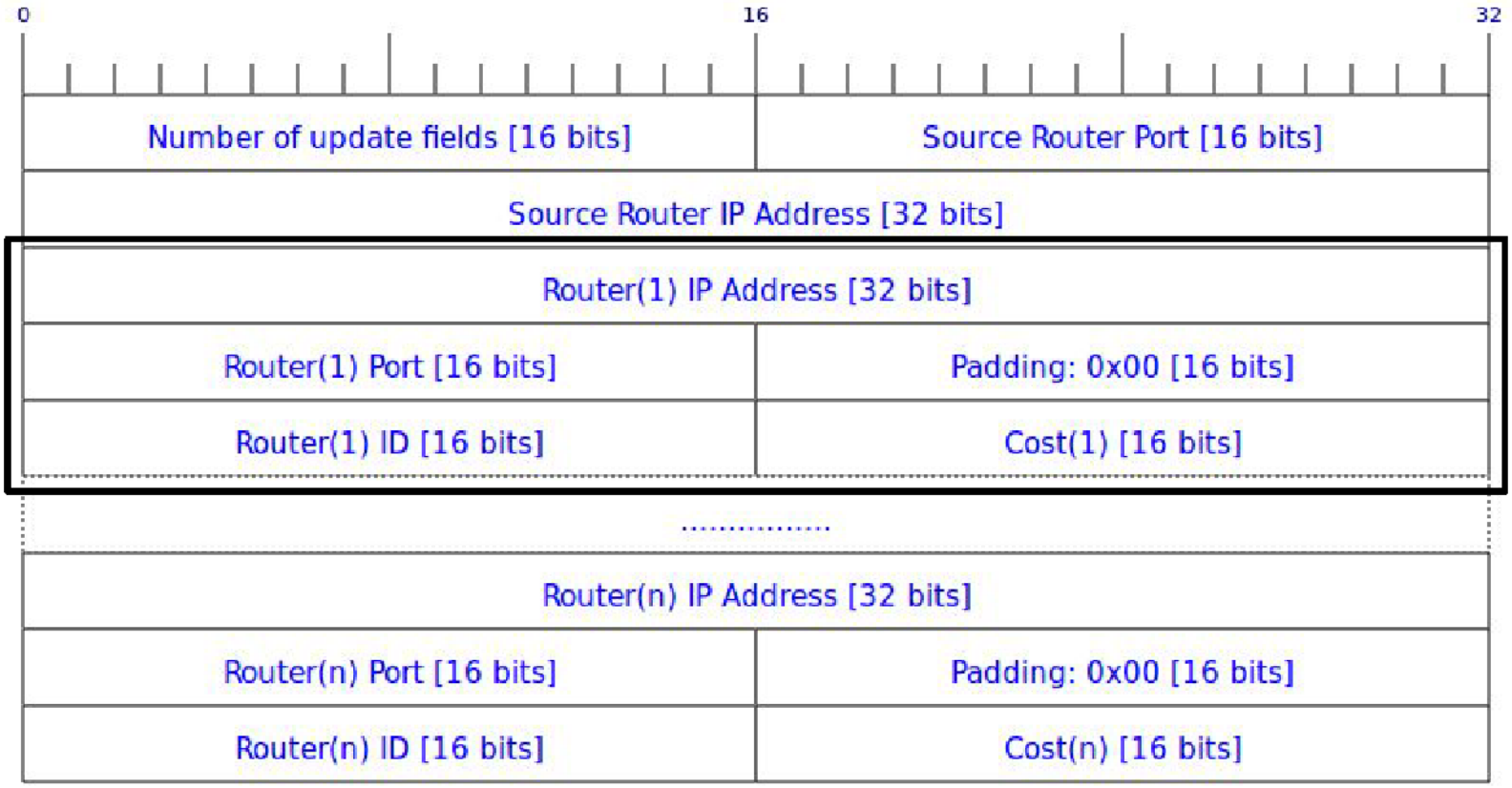
Packet Formats: Example



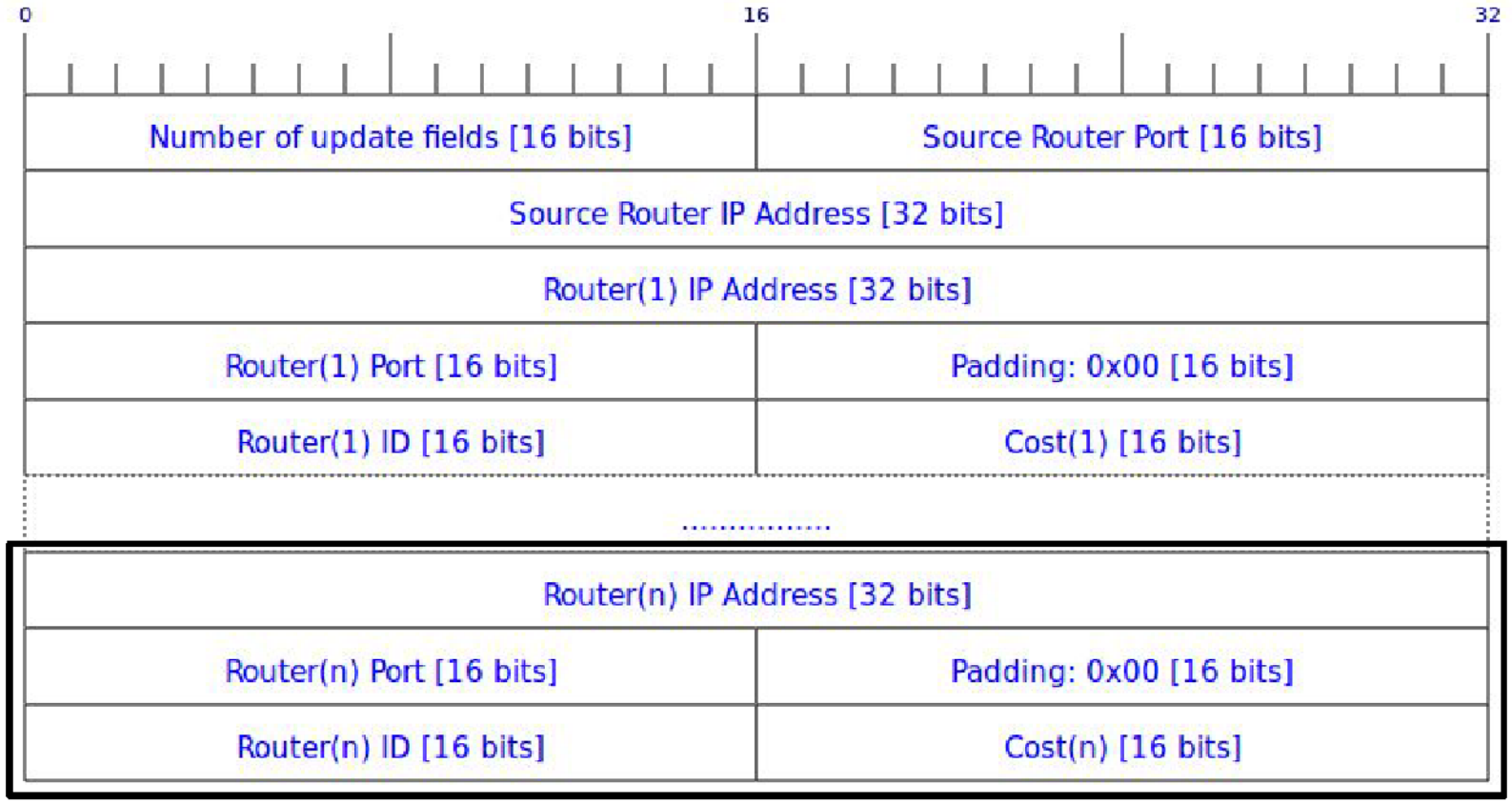
Packet Formats: Example



Packet Formats: Example



Packet Formats: Example



Numeric Types

- All numeric fields, unless mentioned otherwise, are represented/encoded as their unsigned versions

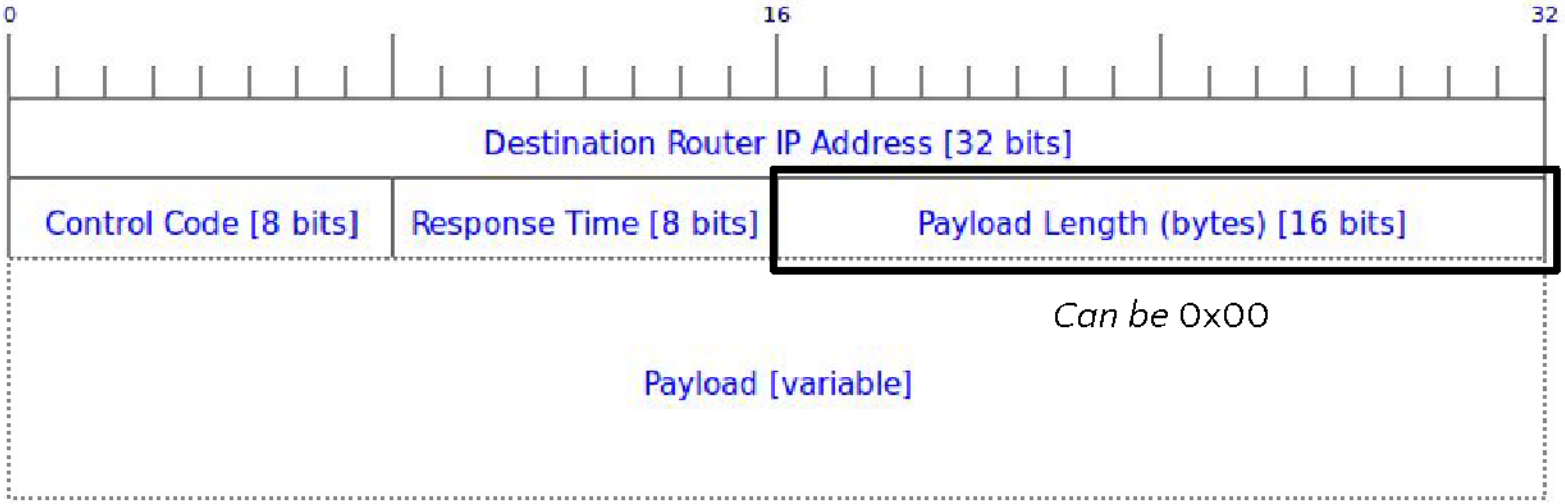
- How to represent infinity (INF)?
  - Largest 2-byte unsigned integer (65535)

How do I verify my packets?

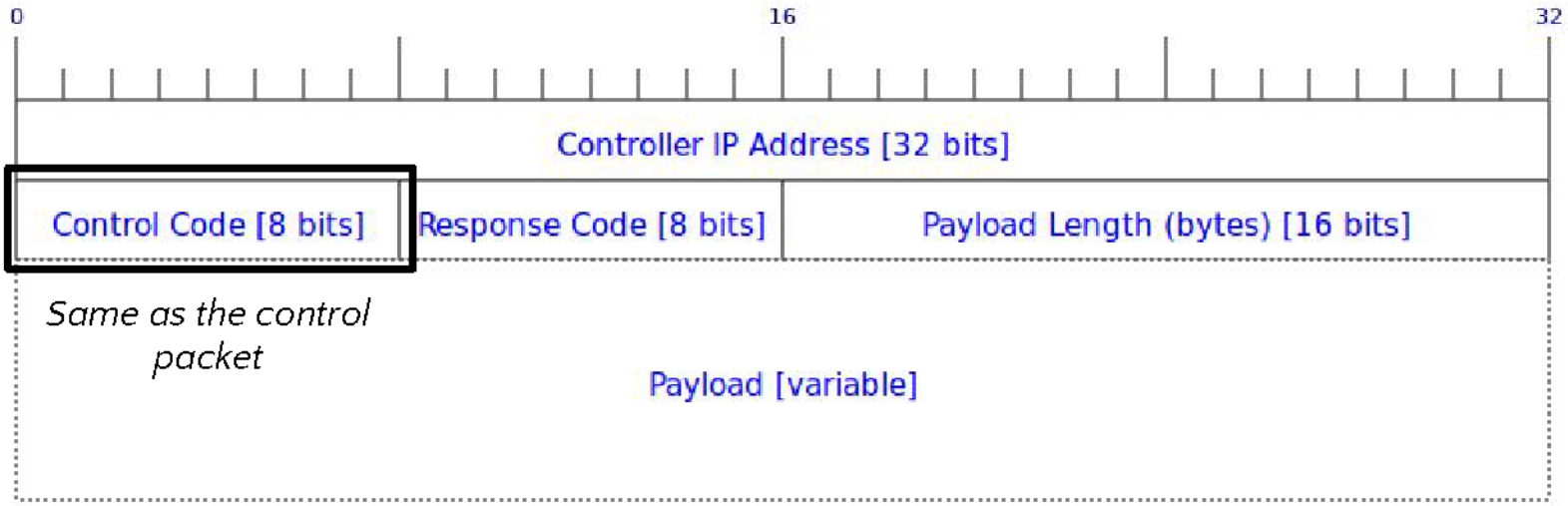
- Look at each bit manually!

- Can use tools like: hexdump
  - Byte-wise (hexadecimal)
    $ hexdump -C packet.file
  - Byte-wise (decimal)
    - Use the -e option with %u

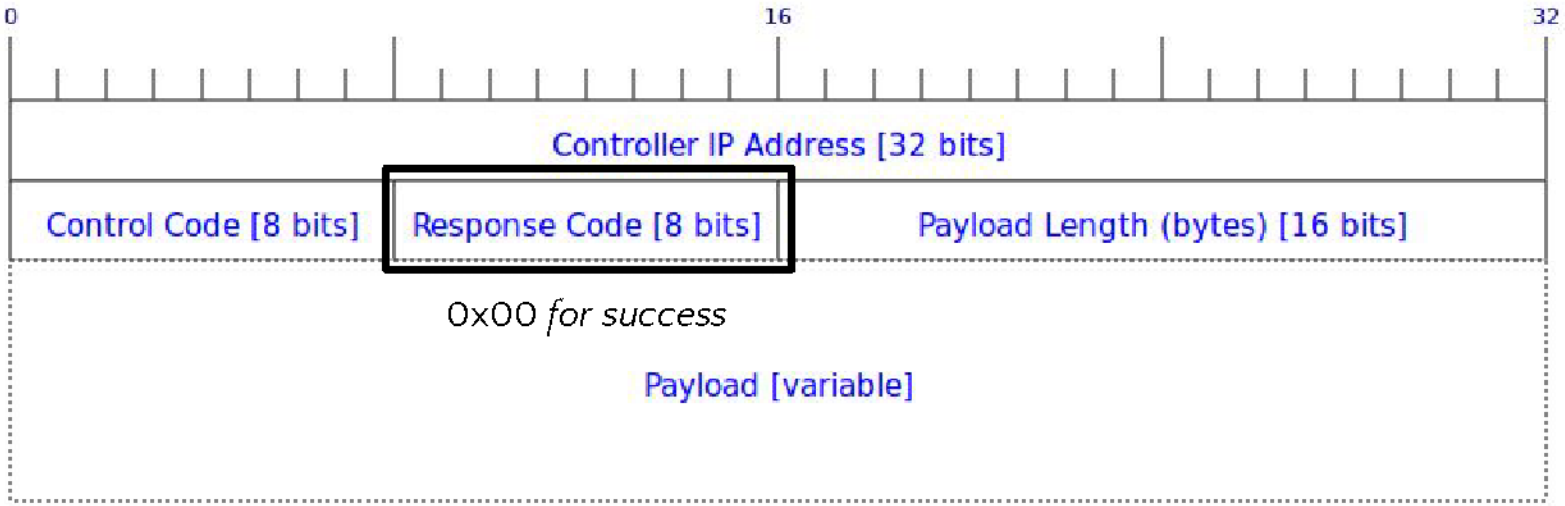
Control Header



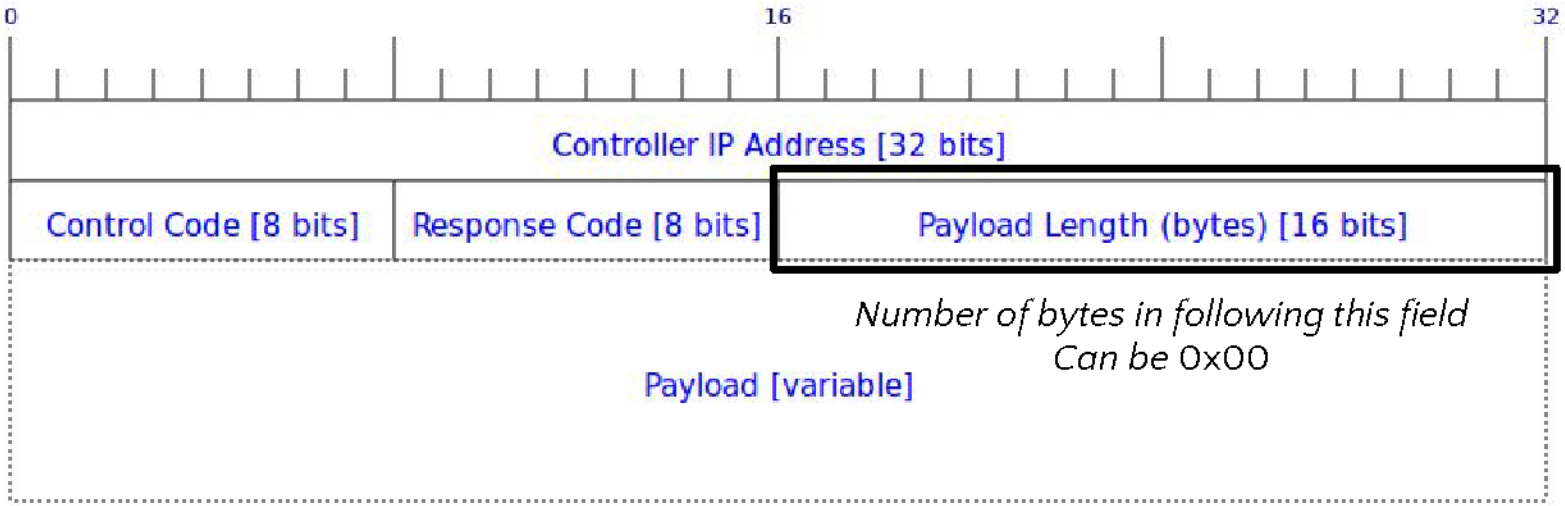
Control-Response Header



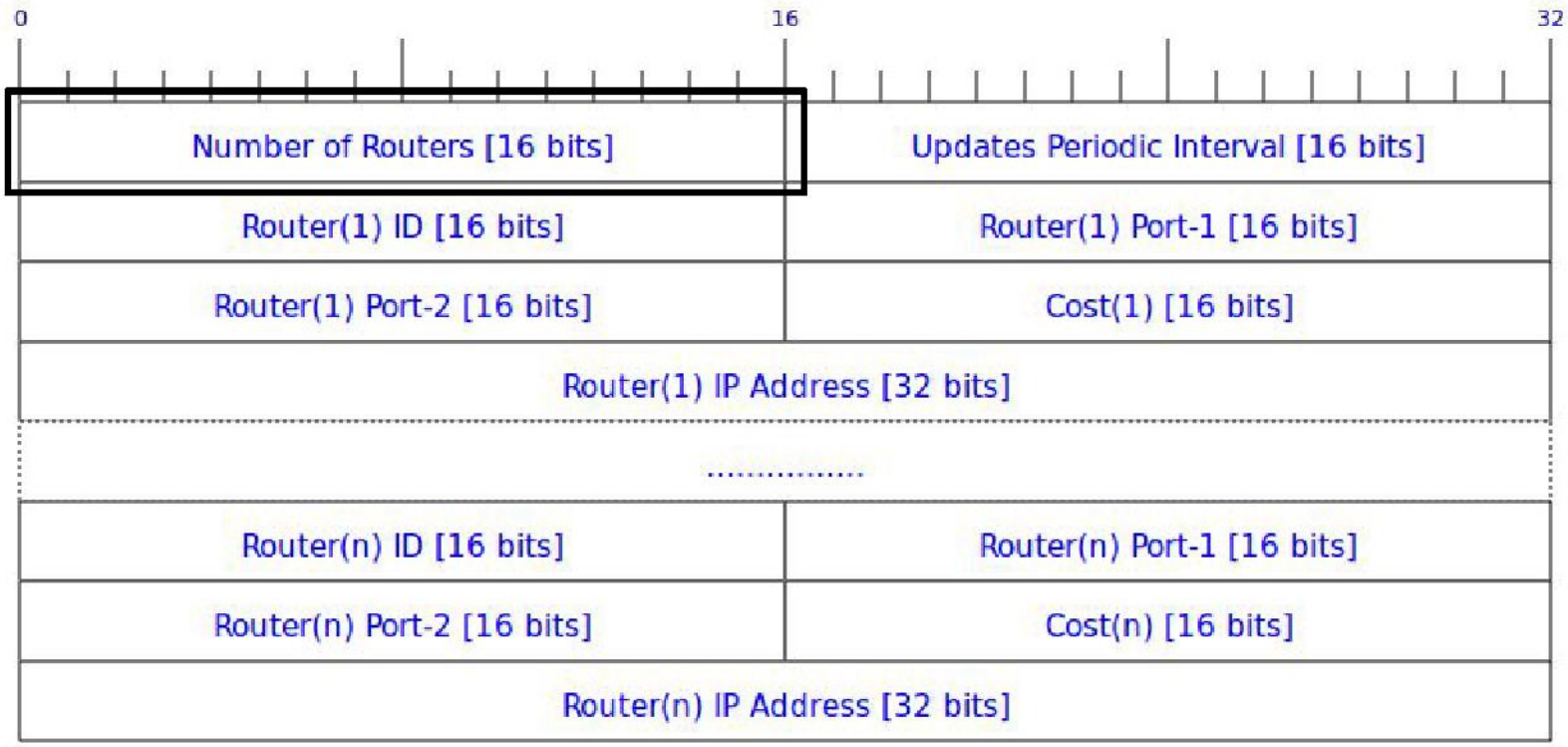
Control-Response Header



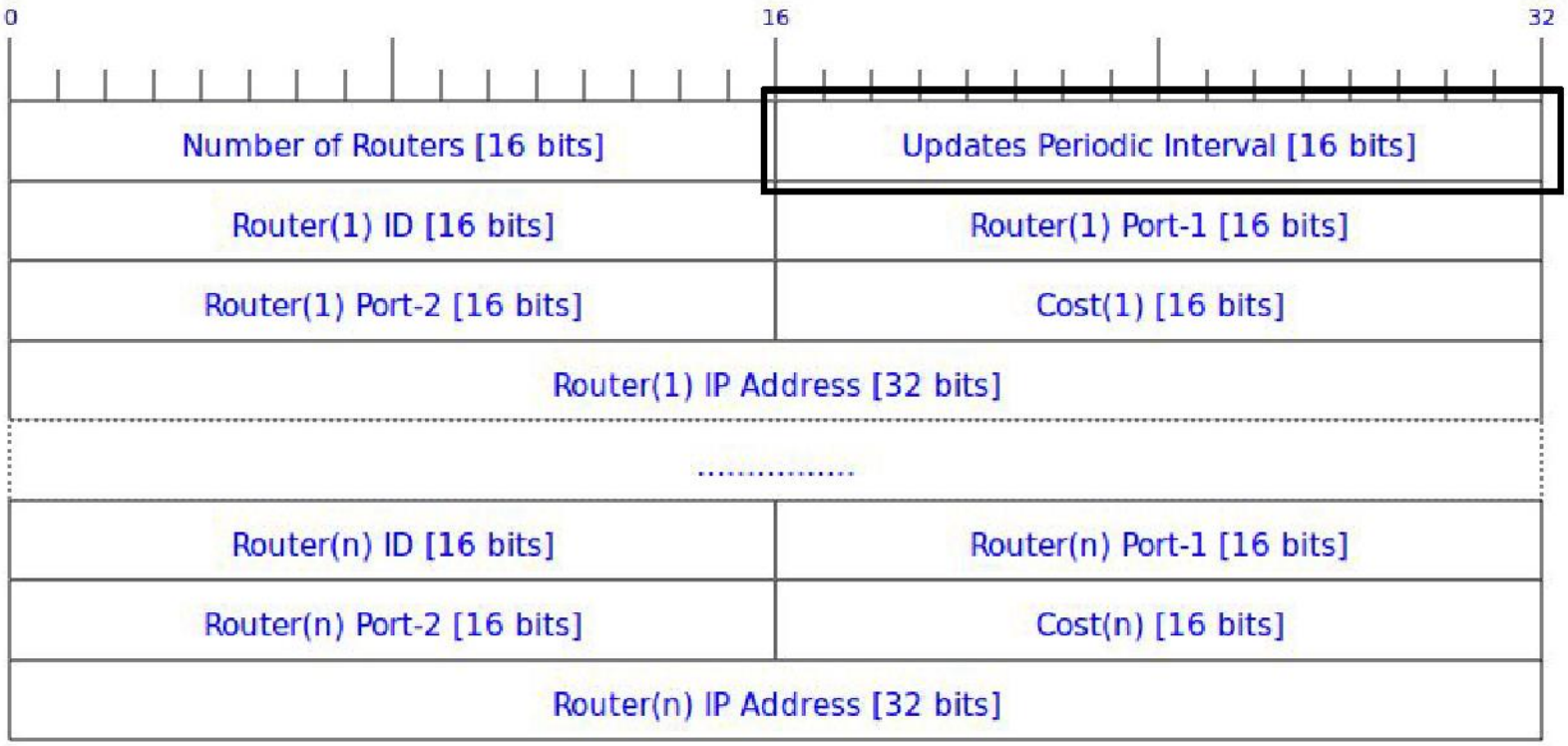
Control-Response Header



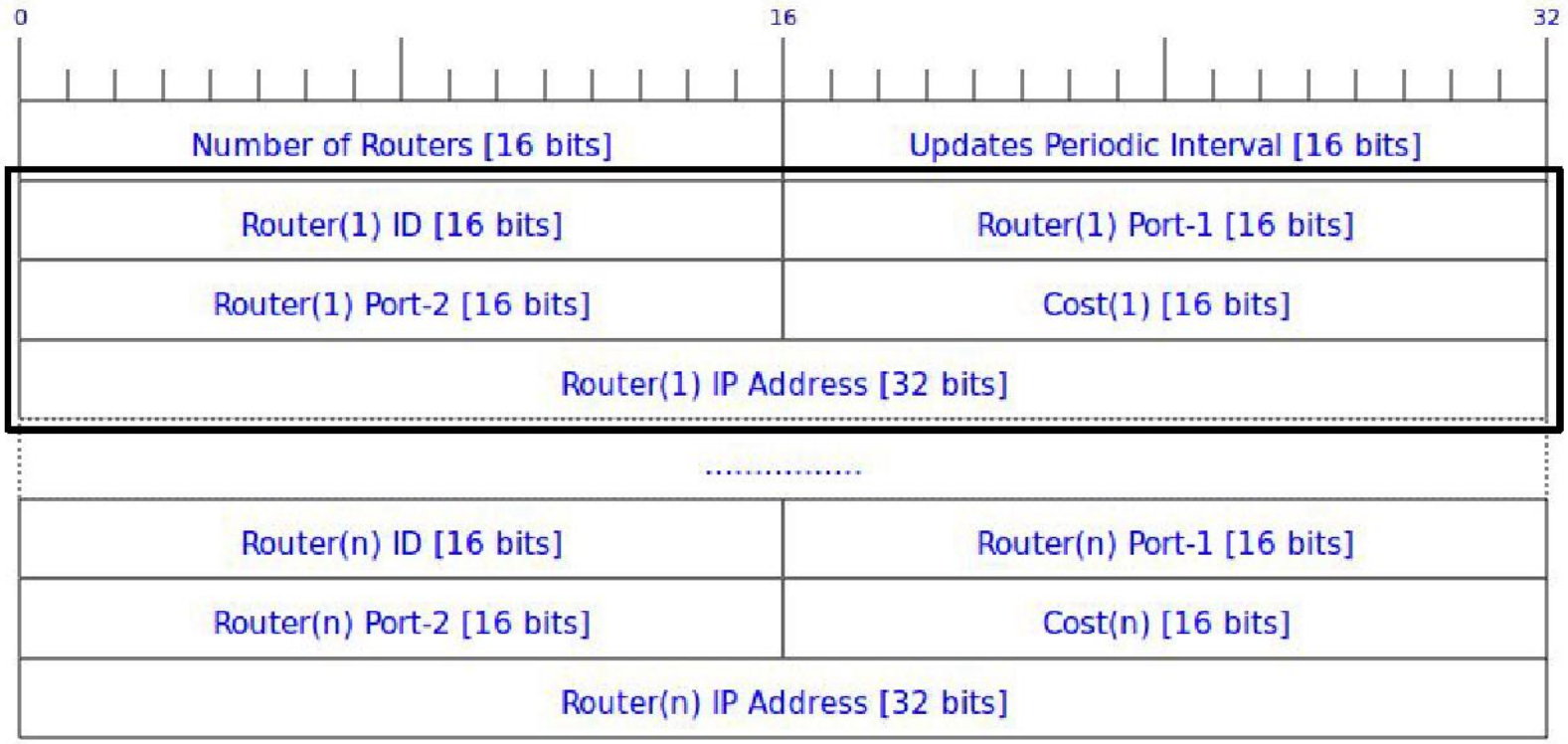
INIT (Control)



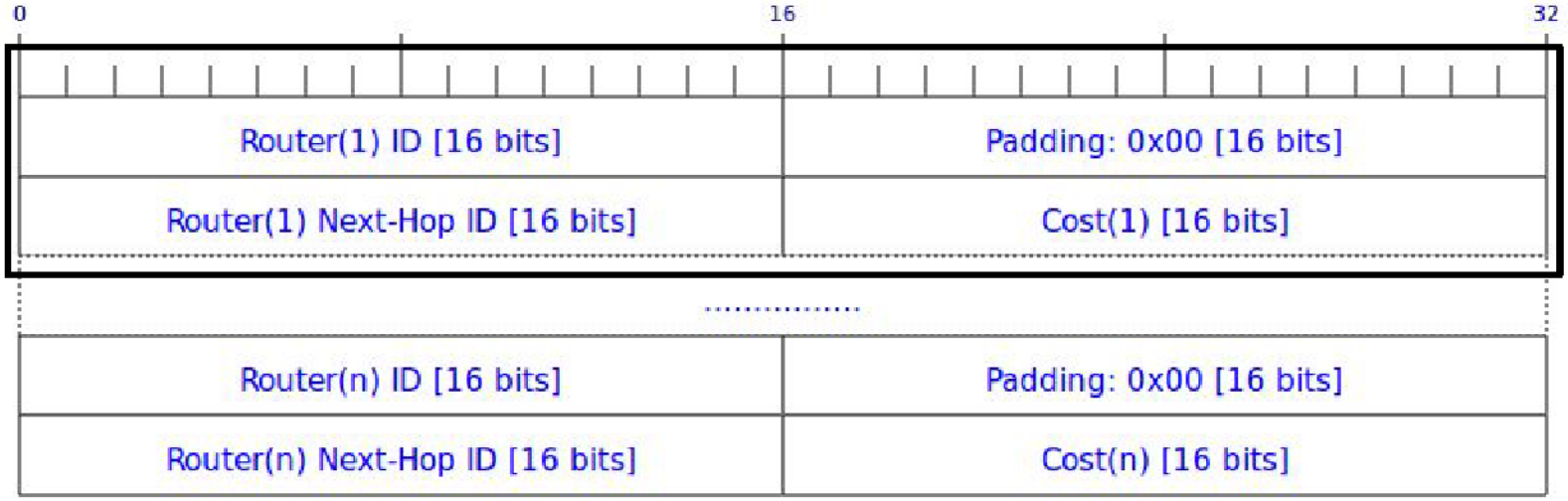
INIT (Control)



INIT (Control)

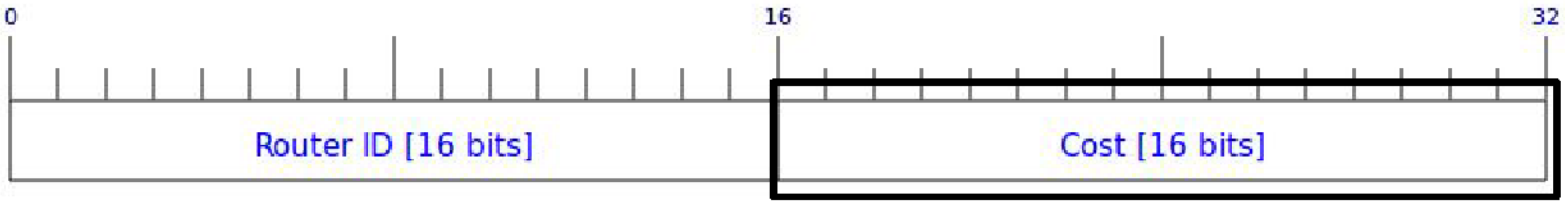


ROUTING-TABLE (Control-response)



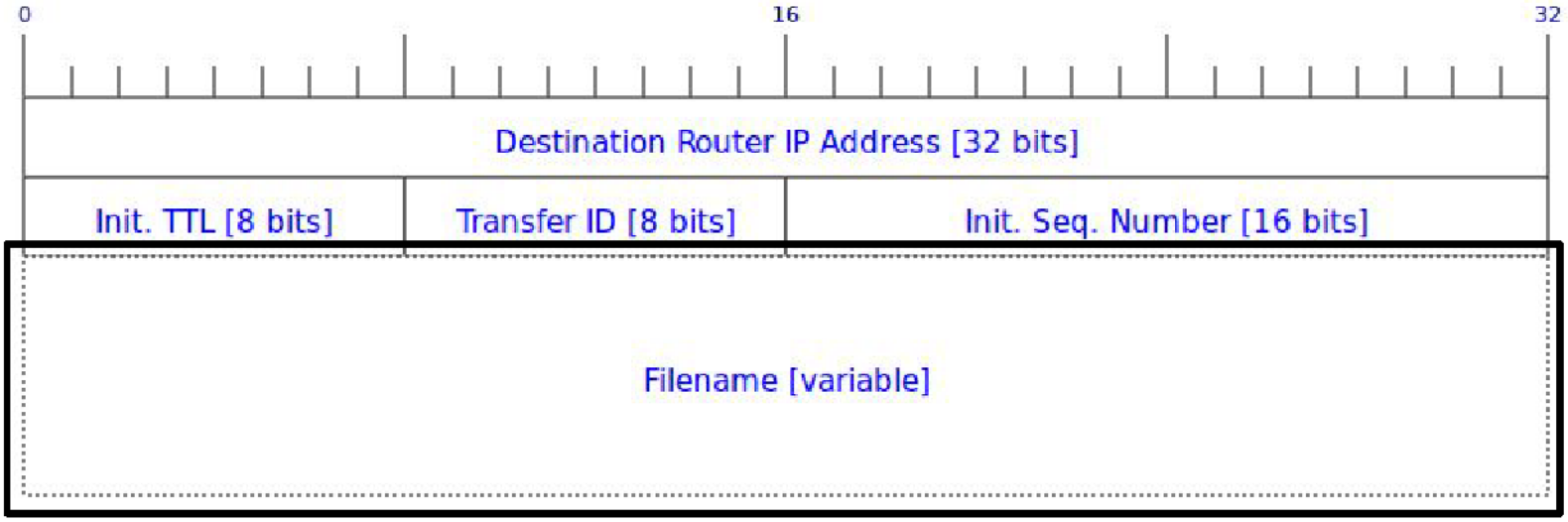
If you do not implement the ROUTING-TABLE response correctly, most automated tests will fail.

UPDATE (Control)



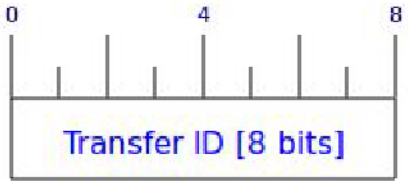
Can be INF
Can be 0x00

SENDFILE (Control)

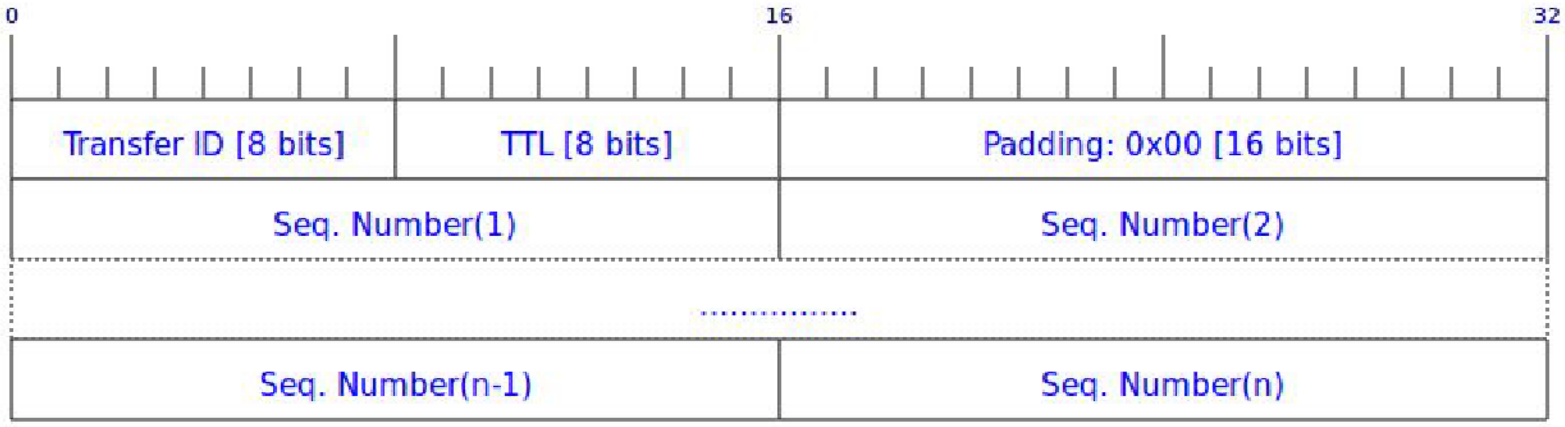


ASCII encoded

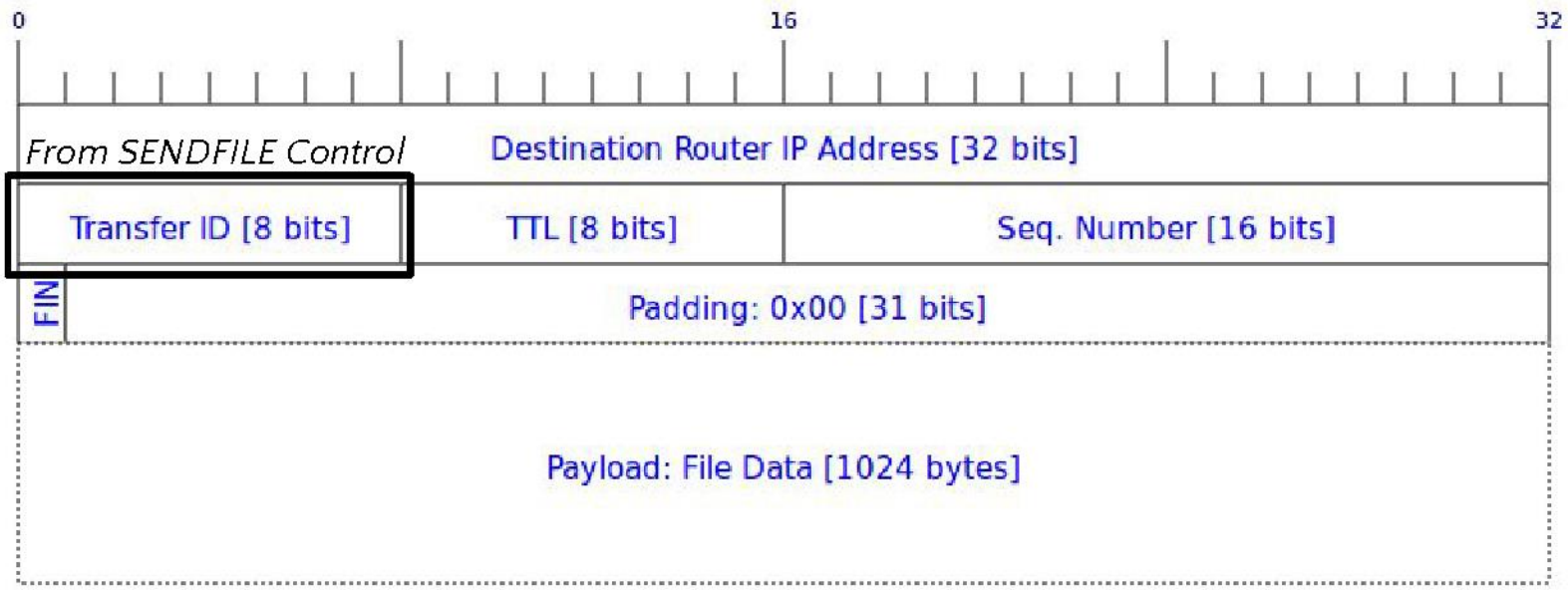
SENDFILE-STATS (Control)



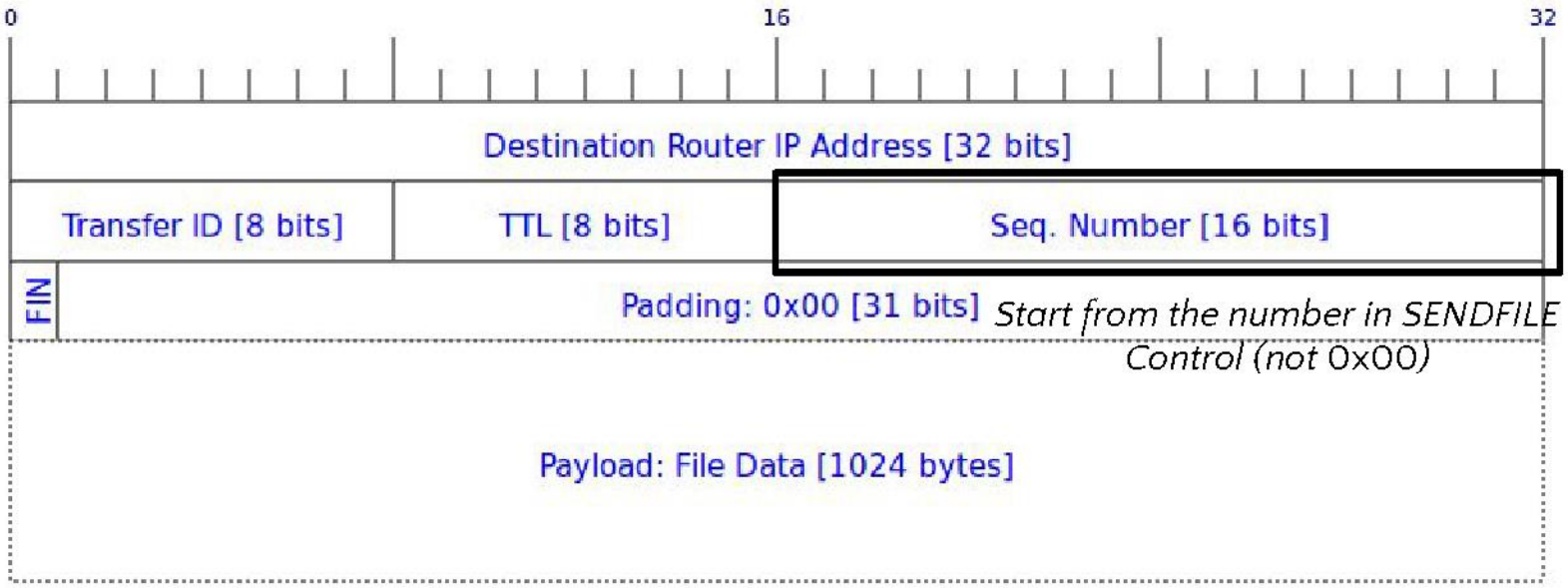
SENDFILE-STATS (Control-response)



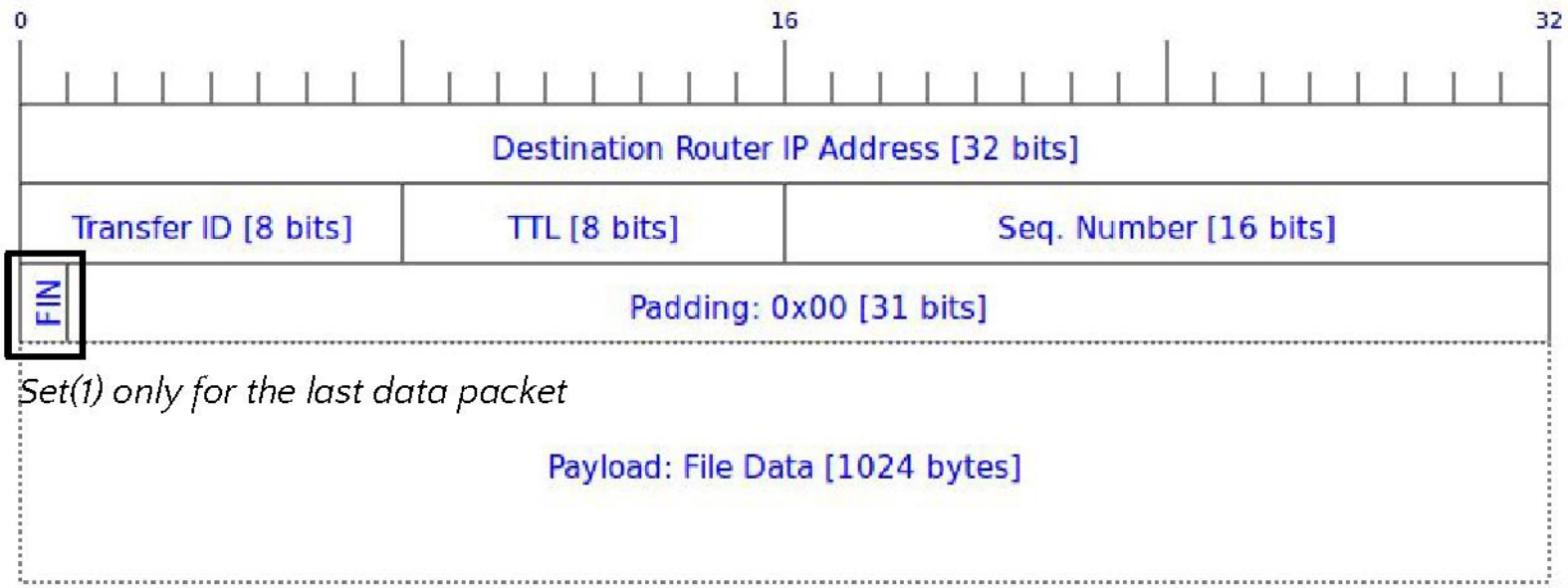
Data Packet


0
16
32
From SENDFILE Control
Destination Router IP Address [32 bits]
Transfer ID [8 bits]
TTL [8 bits]
Seq. Number [16 bits]
FIN
Padding: 0x00 [31 bits]
Payload: File Data [1024 bytes]

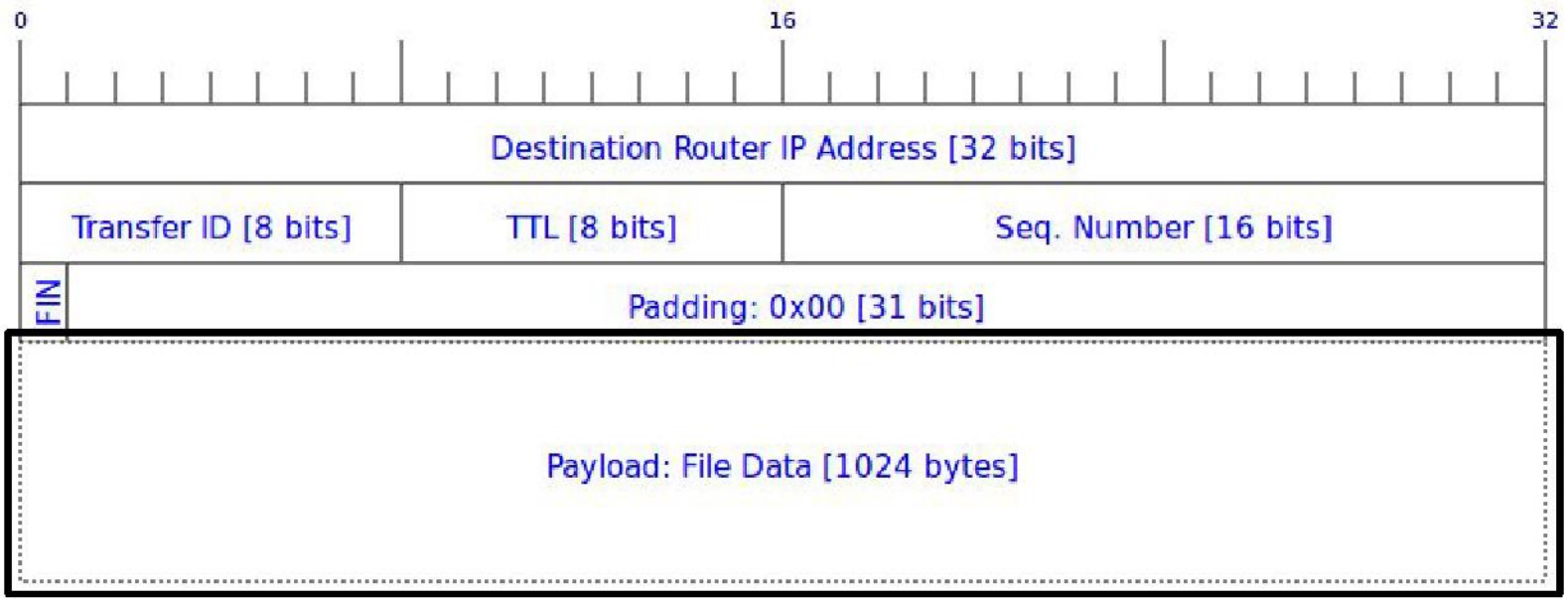
Data Packet


0
16
32
Destination Router IP Address [32 bits]
Transfer ID [8 bits]
TTL [8 bits]
Seq. Number [16 bits]
FIN
Padding: 0x00 [31 bits]
Start from the number in SENDFILE Control (not 0x00)
Payload: File Data [1024 bytes]

Data Packet


0
16
32
Destination Router IP Address [32 bits]
Transfer ID [8 bits]
TTL [8 bits]
Seq. Number [16 bits]
FIN
Padding: 0x00 [31 bits]
Set(1) only for the last data packet
Payload: File Data [1024 bytes]

Data Packet



Should always be 1024 bytes. Assume filesize to be a multiple of 1024 bytes

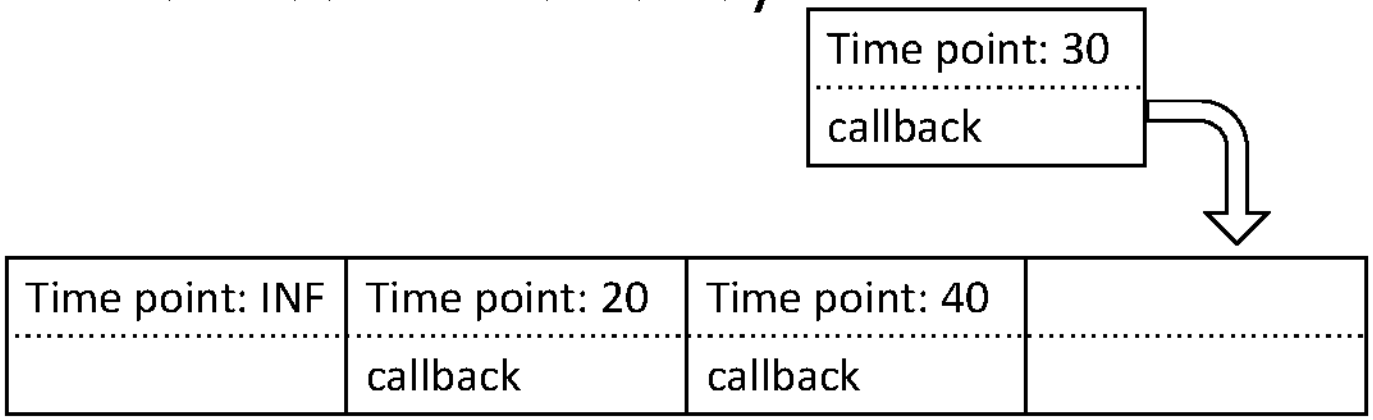
Timer: Basic idea

- Timeout
  - To implement periodic distance vector broadcast
  - To detect the absence of three consecutive routing updates
- Multiple timing tasks
  - Each router may be on a different schedule
- Single timer

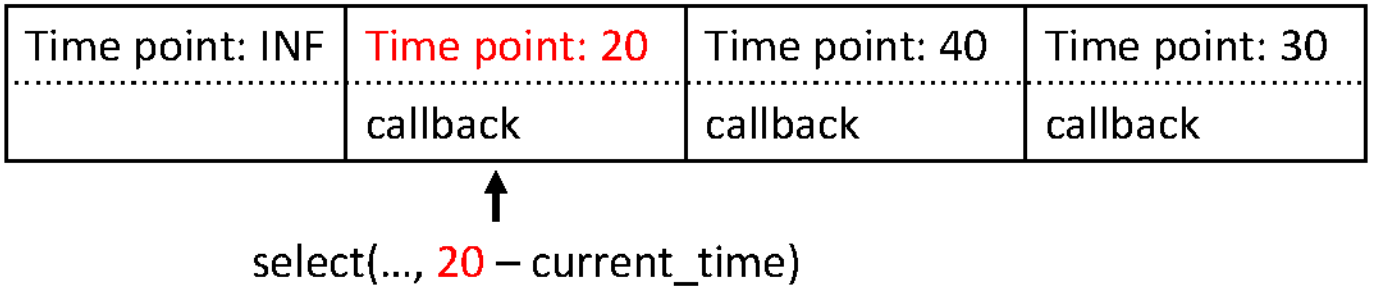
```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
```

Timer: Start a timer

- Add new task into an array



- Time for the task with earliest starting time



Timer: Restart/Stop a timer

- Interruption & restart timer
  - Notice: select() can be interrupted by incoming messages as well as timeout!
  - You have to restart timer after interruption!

- Execute callback & remove the task
  - Execute callback ONLY after time out
  - Remove the task from array after executing callback

Time point: INF	Time point: 40	Time point: 30
	callback	callback

Questions?