CSC358 A2 report

1.Simple End System

1.1 Files

1	endsys.py	mininet topolgy
2	endsys	Executable file
3	src/endsys/endsys.c	C source code

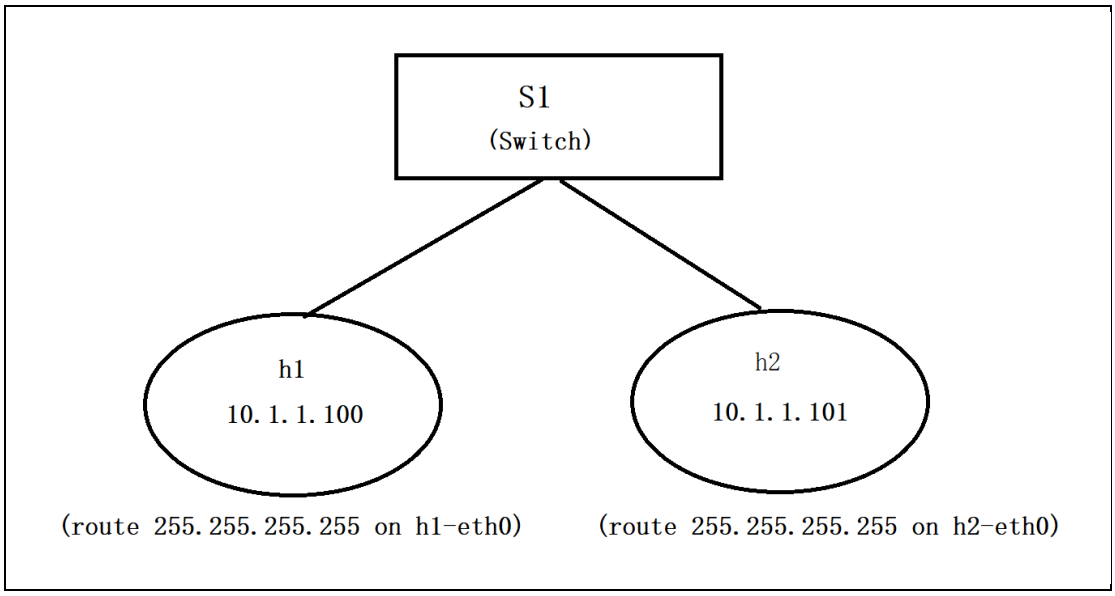
1.2 Compile

Under src/endsys ,run:

make

(It must include multithread lib.)

1.3 Mininet topolgy



1.4 Run

Two file endsys and endsys.py must be at same dir.

① Enter mininet

sudo python3 endsys.py

② Start two xterm in mininet

xterm h1 h2

③ Run endsys on h1 (power on the machine h1)

./endsys

④ Run endsys on h2 (power on the machine h2)

./endsys

You will see “10.1.1.101 is active” on h1’s xterm. (initialize). That means the machine h1 power on.

⑤Input the destination IP on h1

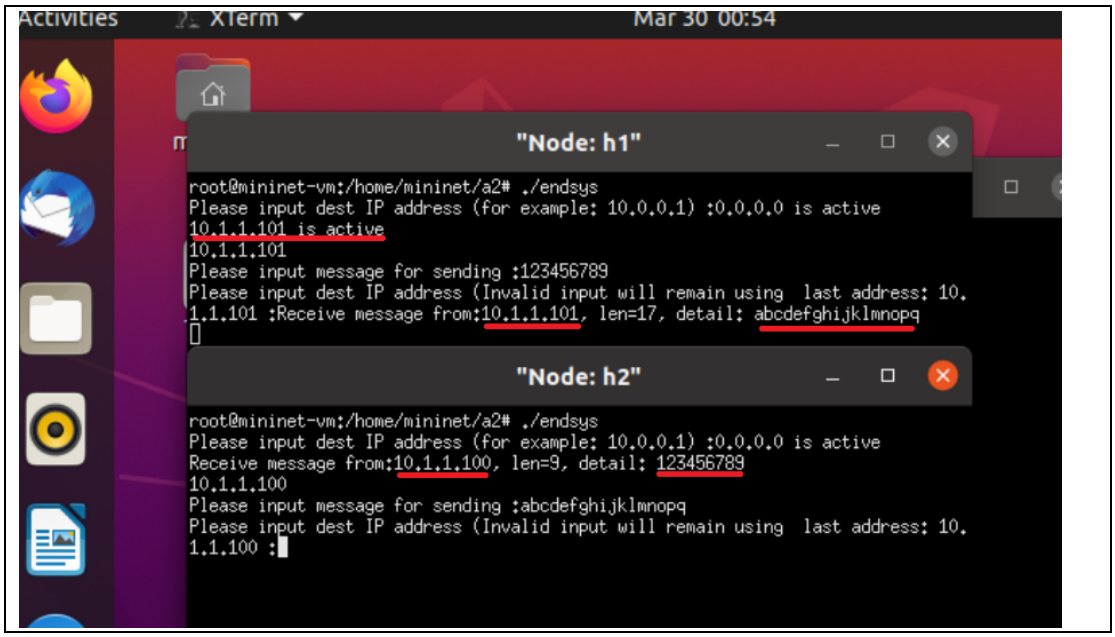
10.1.1.101

If it’s the first time, must input valid destination IP address. Later, if you input an invalid IP, the endsys will keep last valid IP. But you must input at least one character, you cannot just input RERURN, otherwise, endsys will keep waiting input.

⑥Input the message

If you enter RETURN you will see the message you input on h2’s xterm

⑦ If you want send message from h2 to h1 , you input destination IP:10.1.1.100



1.5 Implementation

① Use UDP for sending and receiving

② Two threads: main thread for sending, the other child thread for receving

③ send_mesg(int first_time): process sending, if first_time=1, broadcast message.

④void* receive_mesg(void* arg) :process receiving.

2. Simple Router

2.1 Files

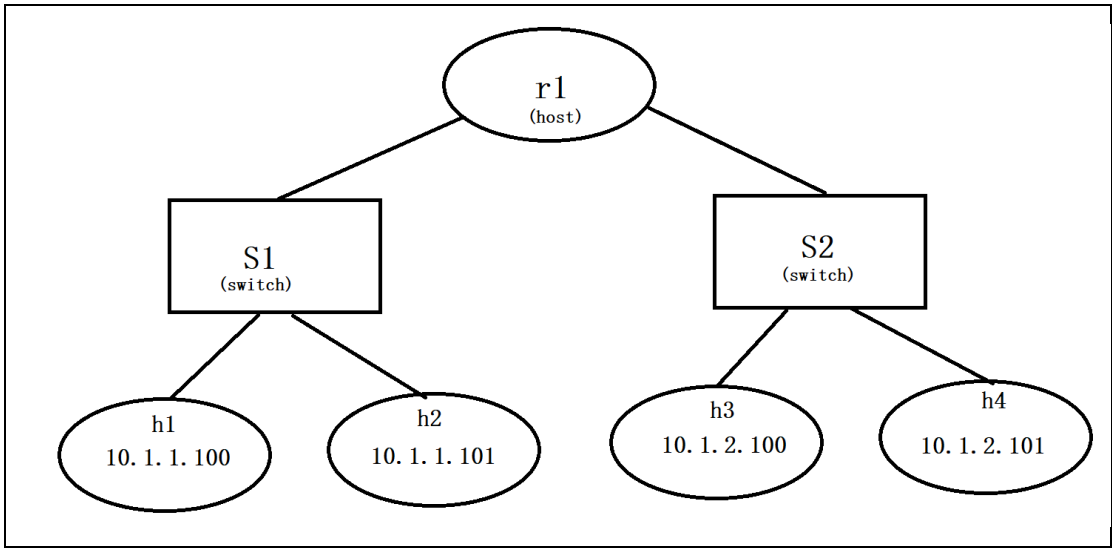
1	simplerouter.py	mininet topolgy
2	sr	Executable file, runs on r1
3	endsys	Executable file, runs on h1 and h3(or on h2 and h4)
4	arp.c arp.h	Under ~/a2/src/sr
5	interface.c interface.h	Under ~/a2/src/sr
6	ip_link.c ip_link.h	Under ~/a2/src/sr
7	main.c main.h	Under ~/a2/src/sr
8	Makefile	Under ~/a2/src/sr

2.2 Compile

Under src/sr, run:

make

2.3 Mininet topolgy



I use host for router r1.

2.4 Run

Two files: sr and sr.py must be at same dir.

First part, we don not start our router.

① Enter mininet

sudo python3 sr.py

② Start three xterm in mininet

xterm h1 h3 r1

③ Run endsys on machine h1

./endsys

④ Run endsys on machine h3

./endsys

⑤Input the destination IP on h1

10.1.2.100

⑥Input the message

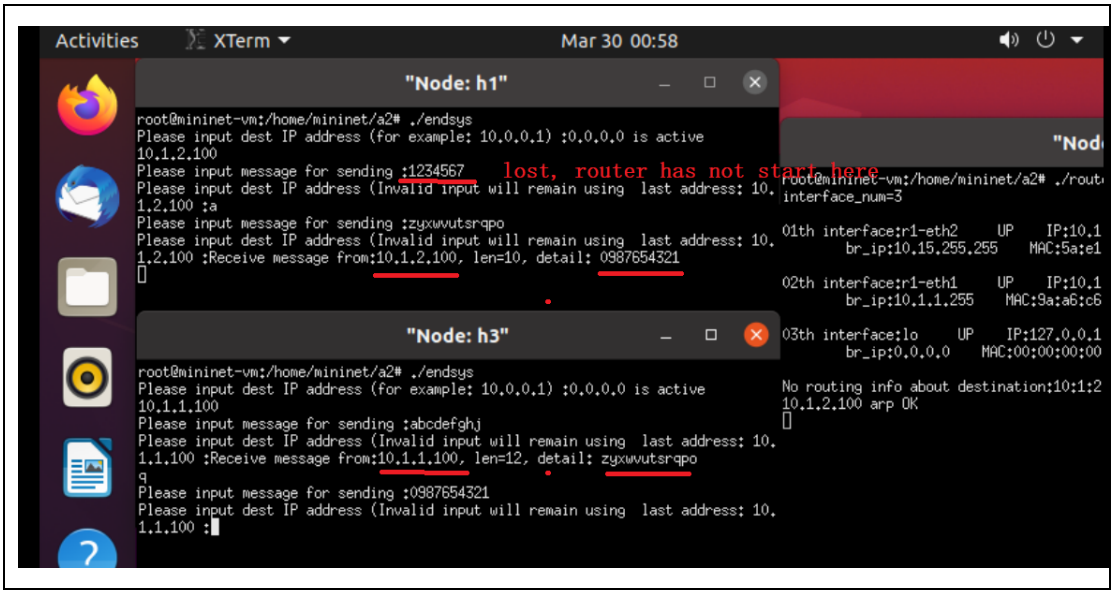
If you enter RETURN you cannot see anything you input on h2's xterm. Same as on h1.

If it's the first time, you must input valid destination IP address. Later, if you input an invalid IP, the endsys will keep last valid IP. But you must input at least one character, you cannot just input RERURN, otherwise, endsys will keep waiting input.

Second part. We start ./sr on r1.

Then when you repeat ⑤ and ⑥ on h1 and h3 , you will see what you input in one term on the other host.

⑦ If you want send message from h2 to h1 , you input destination IP:10.1.1.100



2.5 Implementation

① main() :

Call init_interface() getting interfaces info on r1.

Create raw socket.

Enter while loop :receive packets and then process.

② init_interface():

Call ioctl() get all information of all interfaces.

Save these information in global variable net_interface[], include: IP, MAC, netmask, state, broadcast address.

③ while loop:

Call recvfrom() receive packets

Judge if the packet is ARP, if yes, create new thread, call arp_packet_process() save route info into global variable arp_head.

If the packet is an IP, create a new thread, and call transfer_data() processing .

④ transfer_data():

First judge if the packet is endsys's initialize one. If it is, save it's IP and MAC.

If the packet is broadcast package, discard.

Call find_arp_from_ip(), find if no corresponding routing info, send ARP REQUST.

If there is corresponding routing info, transfer the packet using the routing info.

3.Multi Network Router

I only developed RIP.

I have tested this RIP on my old laptop. It can run on a network with 17 level routers.

3.1 Files

1	m17.py	mininet topology(17 levels)
2	M6.py	mininet topology(6 levels)
3	rip	Executable file, runs on r1,r2,…,r17
4	endsys	Executable file, runs on h1,h2,…,h17
5	arp.c arp.h	Under ~/a2/src/rip
6	interface.c interface.h	Under ~/a2/src/rip
7	ip_link.c ip_link.h	Under ~/a2/src/ rip
8	main.c main.h	Under ~/a2/src/ rip
9	rip.c rip.h	Under ~/a2/src/ rip
10	Makefile	Under ~/a2/src/ rip

3.2 Compile

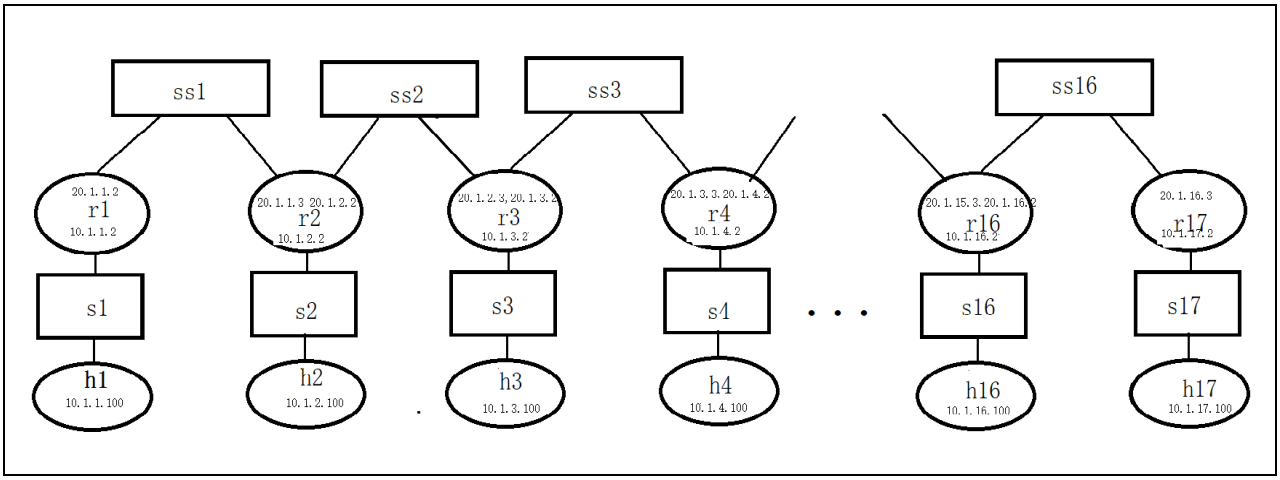
Under src/rip, run:

make

3.3 Mininet topolgy

S1,s2,s3…,s17, ss1,ss2,…,ss16: Switch.

H1,h2,h3,…,h17,r1,r2,…,r17: Host.



Both r1 and r17 has 2 interfaces. R2 has 3 interface, one connects to s2, one connects to ss1, the third connected to ss2. R3,r4,…,r16 have 3 interfaces each.

3.4 Run

Three files: rip,m6.py and m17.py must be at the same dir.

① Enter mininet

sudo python3 m17.py

② Start three xterm in mininet

xterm h1 r1 r2 r3 … r17

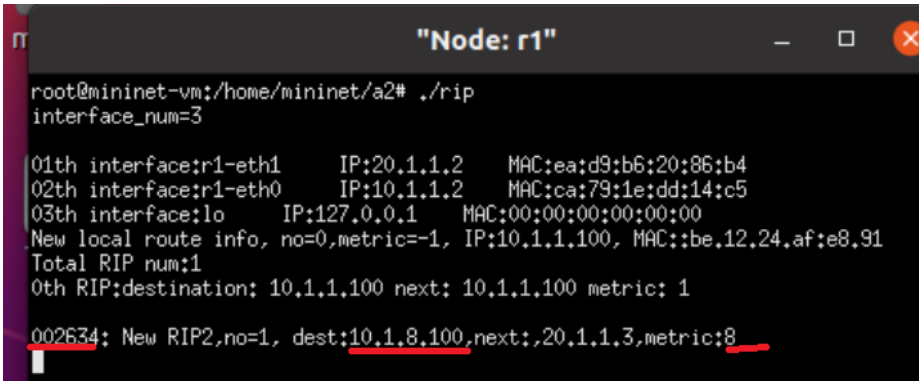
③ Run rip on router r1,r2,…,r17

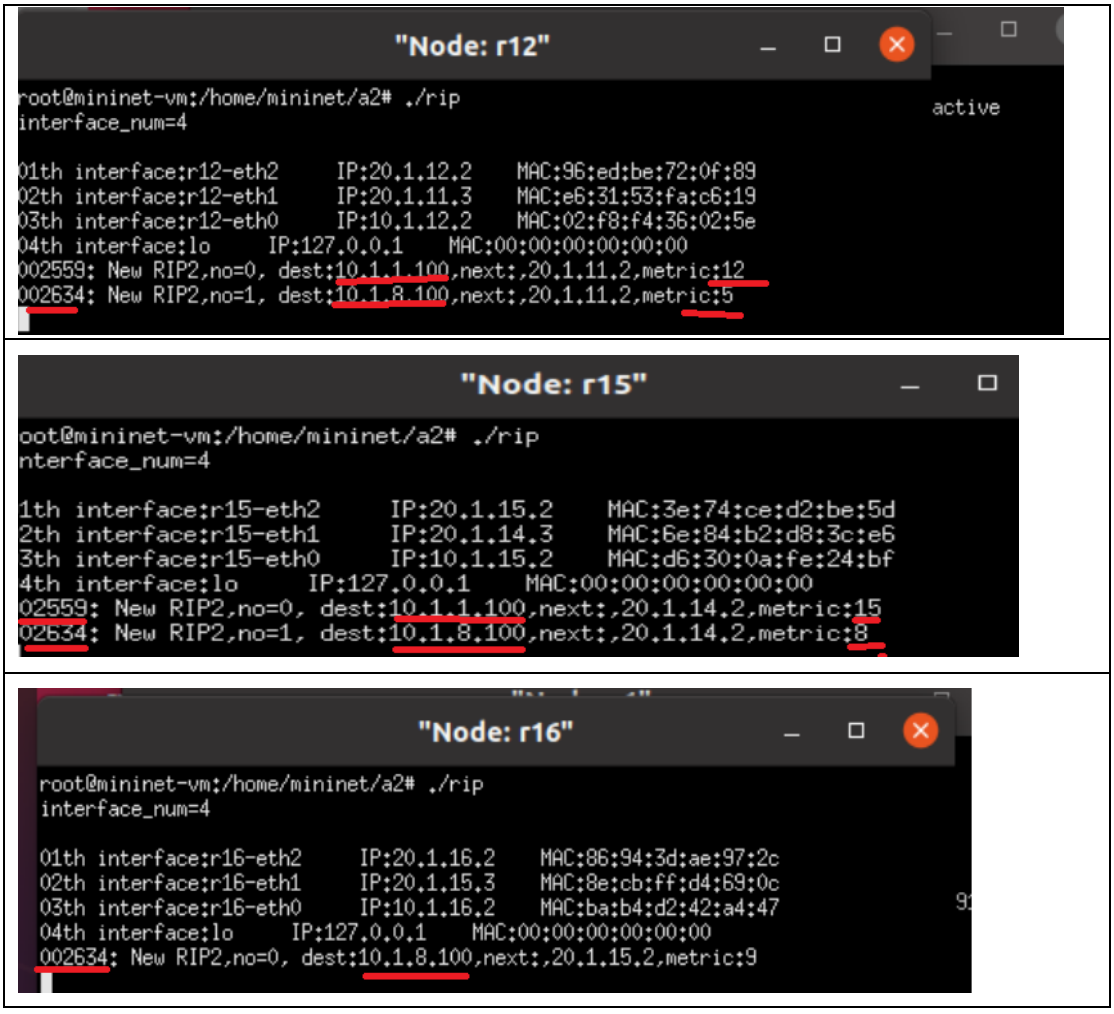
./rip

④Run endsys on machine h1

./endsys

Then you will see on RIP info on r2,r3…r17.





002559: hour=00, minute=257, second=59. When machine h1 powers on, r2,r3,…,r15 receive this RIP almost at same time, but r16 cannot receive this RIP for matric exceeding 16.

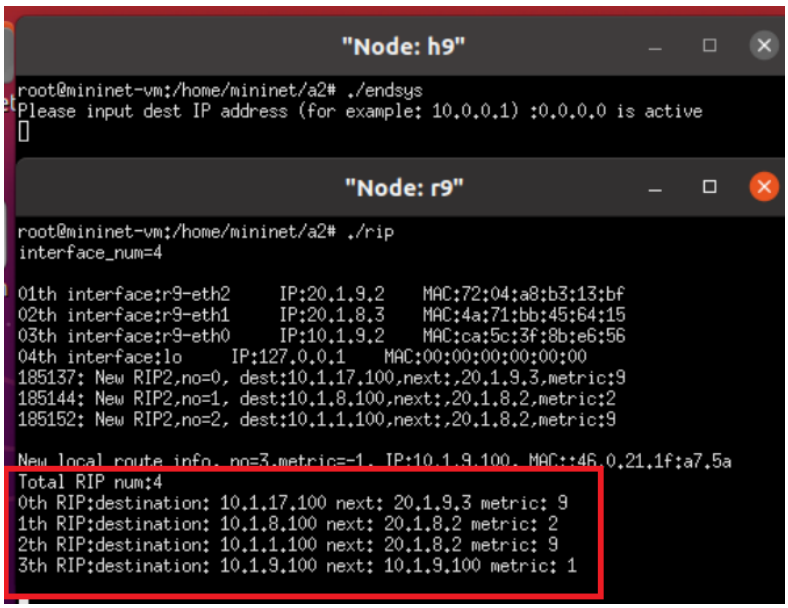
And when machine h8 powers on, r1-r17 receive this RIP almost at same time: 00h26m34s.

⑤If you want to know all RIP info of one route, just restart endsys on the corresponding host.

For example, if you want to know RIP on r9, just start endsys on h9,

./endsys

On r9 you will see all RIP info (see the info in red box of following firgure):



3.5 Implementation

① main() :

First part: initianization

Call init_routing_table() initianizing routing table.

Call rip_init() initianizing rip info.

Call init_interface() get interface info.

Create raw socket and set option.

Second part: main loop using while.

Call recvfrom() receive packet.

If this packet is ARP, create a new thread and call arp_packet_process() processing it.

If this packet is IP packet, first call judge_and_process_rip_packet() judge if it is RIP (request, response. If yes, process in this function. Else call transfer_data() processing all other packet types(there are some problems here).

② init_routing_table():

Just set route_table[i].metric =-1 for all route entry. metric =-1 means this entry is not in use.

③ rip_init():

Create two socket for sending RIP info: gsock and broadcast_sock . Both are global variable.

Initializing global variable current_rip_packet, who is used for sending RIP request.

④ init_interface():

Call ioctl() get all information of all interfaces.

Save these information in global variable net_interface[], include: IP, MAC, netmask, state, broadcast address.

⑤ while loop:

Call recvfrom() receive packets

Judge if the packet is ARP. If yes, create new thread, call arp_packet_process() save route info into global variable arp_head.

If the packet is an IP, create a new thread, first call judge_and_process_rip_packet() judge if it is RIP (request, response. If yes, process in this function. Else call transfer_data() processing all other packet types(there are some problems here).

⑥ judge_and_process_rip_packet():

If the packet is RIP(request, response) (not form local machine), call process_rip_packet() processing it.

If the packet is RIP_REQUEST, call process_rip_request(). If it is RIP_RESPONSE:,call process_rip_response().

⑦ process_rip_request():

Send all RIP in local machine(saving in global variable route_table) to the machine who requesting. There are at most 25 RIP every packet.

⑧ process_rip_response():

Insert every RIP info from packet into global variable route_table.

For every RIP info, if there are no corresponding info, just save it into one empty entry of route_table, with metric increasing 1. If this RIP has already existed in route_table, and new metric is less than the old one, replace it, otherwise, discard it.

At the end of this function, broadcast new RIP to all neighbor routers.