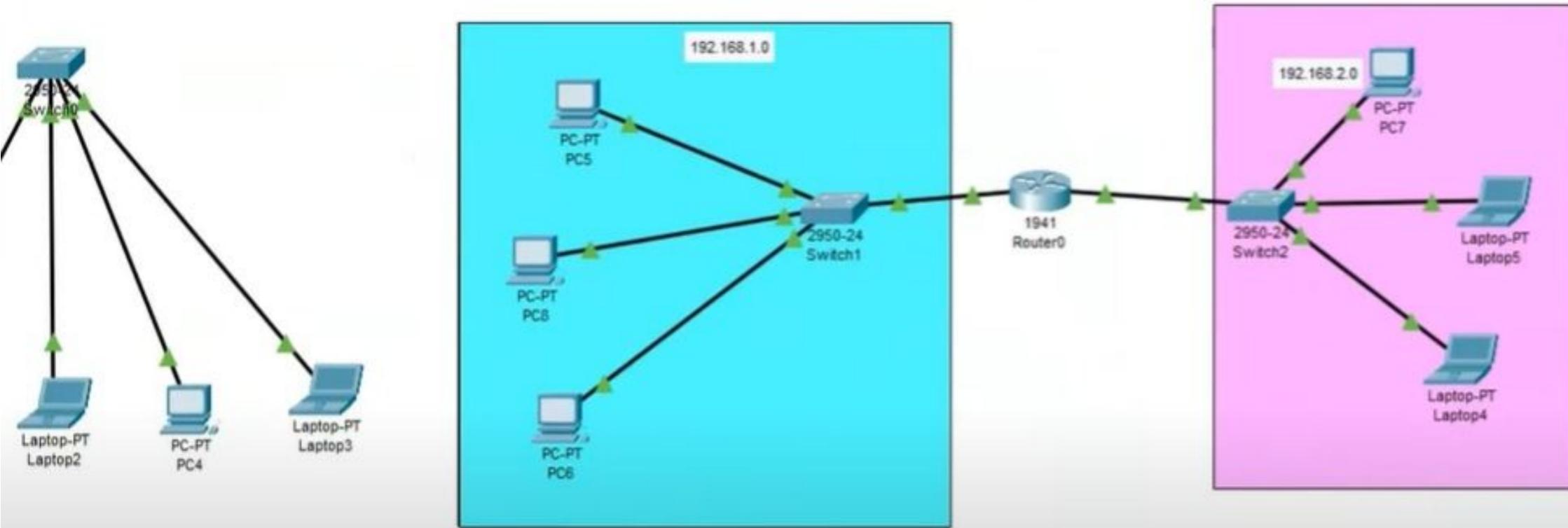




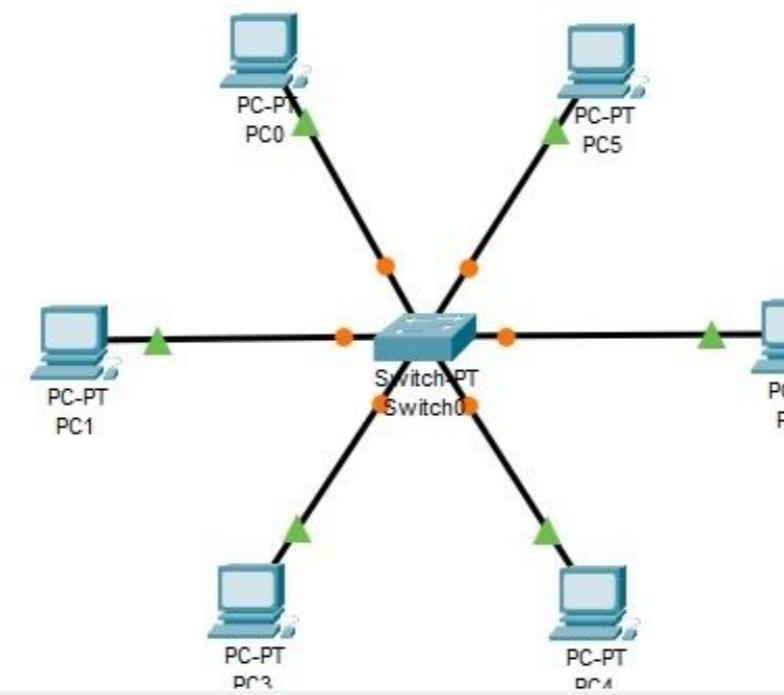
Root 08:31:00



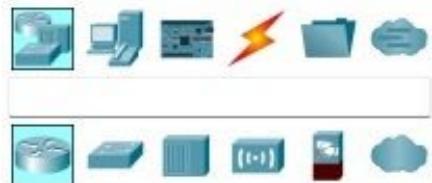


**Logical** (Physical) x: 153, y: 0  
Logical Mode (Shift+L)

Root 00:02:30



Time: 00:00:04



Scenario 0

Toggle PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic
	--	PC0	PC2	ICMP		0.000	N
	--	PC5	PC3	ICMP		0.000	N
	--	PC1	PC2	ICMP		0.000	N

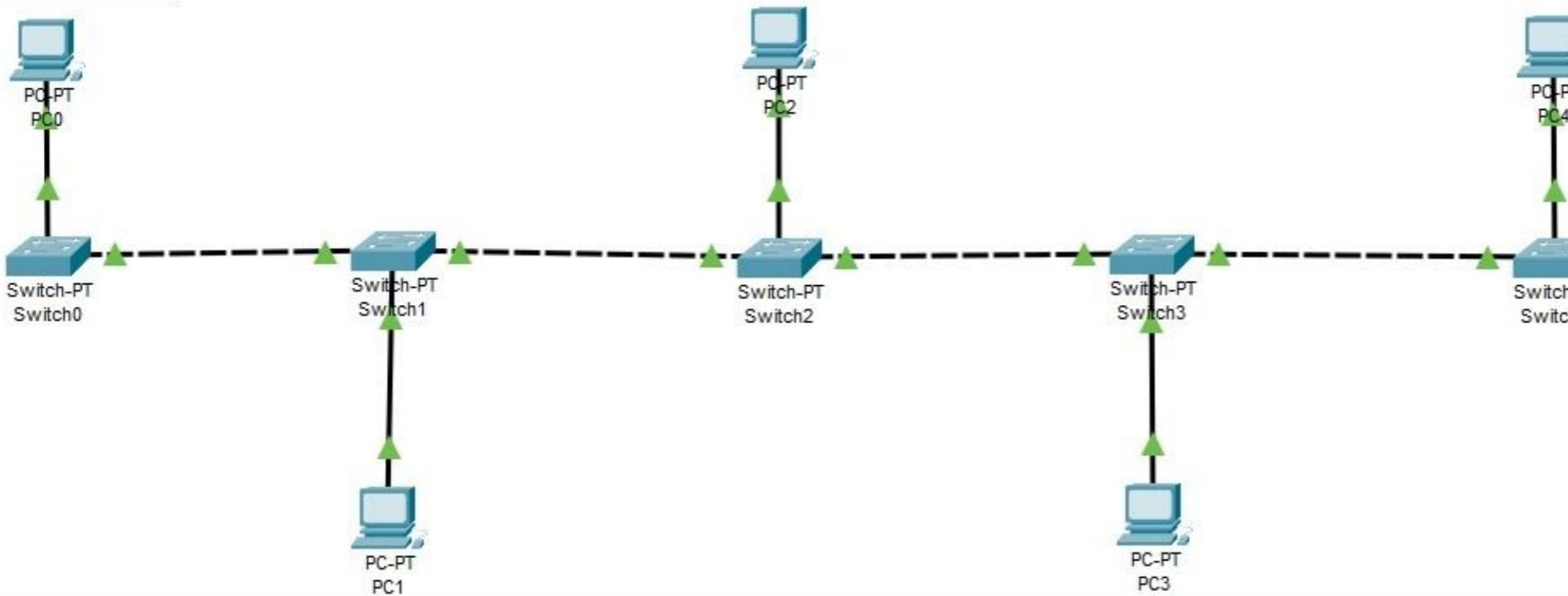
(Select a Device to Drag and Drop to the Workspace)



Logical Physical x: 6, y: 86

Root ↺ ⚡ ⚡ ⚡ ⚡ ⚡ 08:03:30

Logical Mode (Shift+L)



Time: 00:15:55 ⏪ ⏴

Realtime Simulation



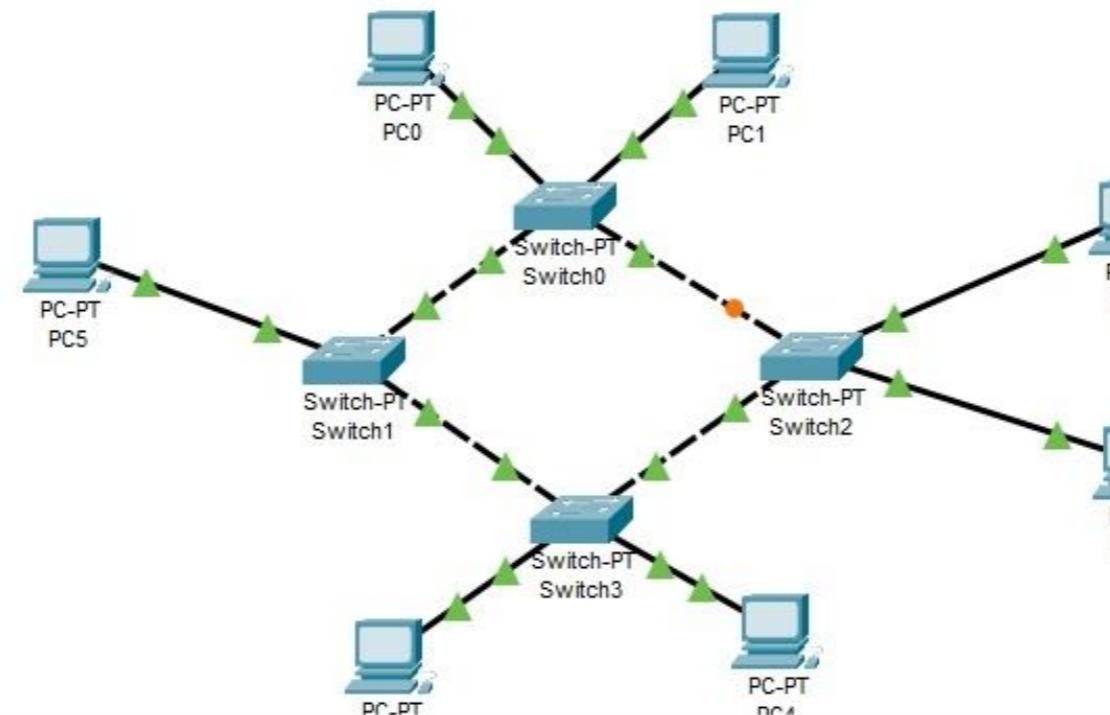
Scenario 0  
New Delete  
Toggle PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Peri
●	--	PC0	PC1	ICMP	purple	0.000	
●	--	PC1	PC2	ICMP	teal	0.000	
●	--	PC2	PC3	ICMP	magenta	0.000	
●	--	PC3	PC4	ICMP	green	0.000	



Logical Physical x: 1013, y: 403

Root 00:17:00



Time: 00:00:33

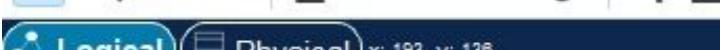
Realtime Simulation



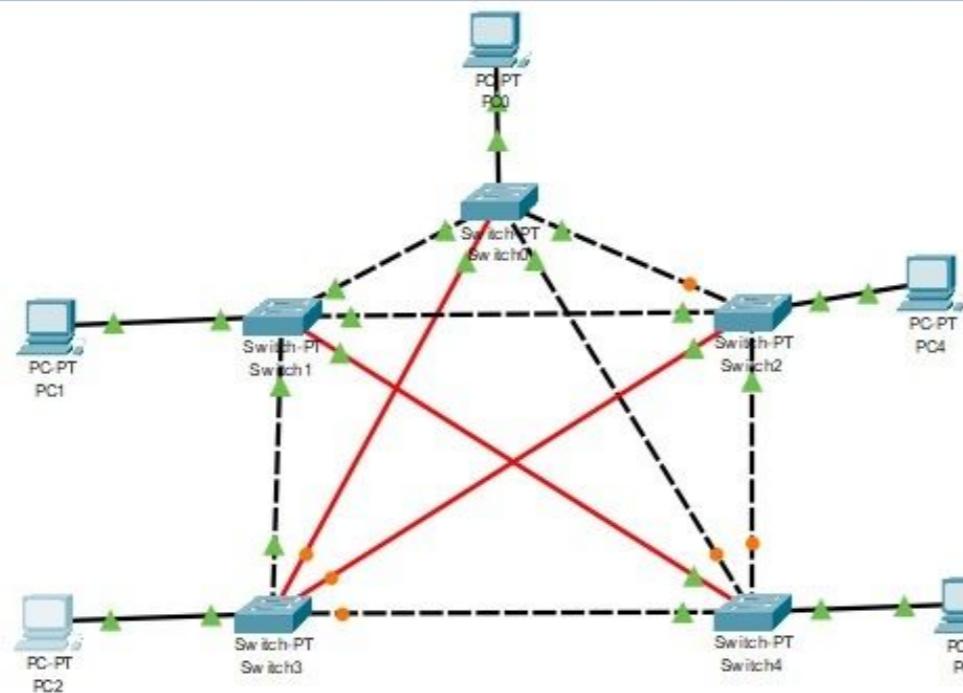
Scenario 0

Toggle PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic
	--	PC0	PC3	ICMP		0.000	N
	--	PC1	PC4	ICMP		0.000	N
	--	PC2	PC4	ICMP		0.000	N



Root 00:20:00



Time: 00:00:38

Realtime Simulation

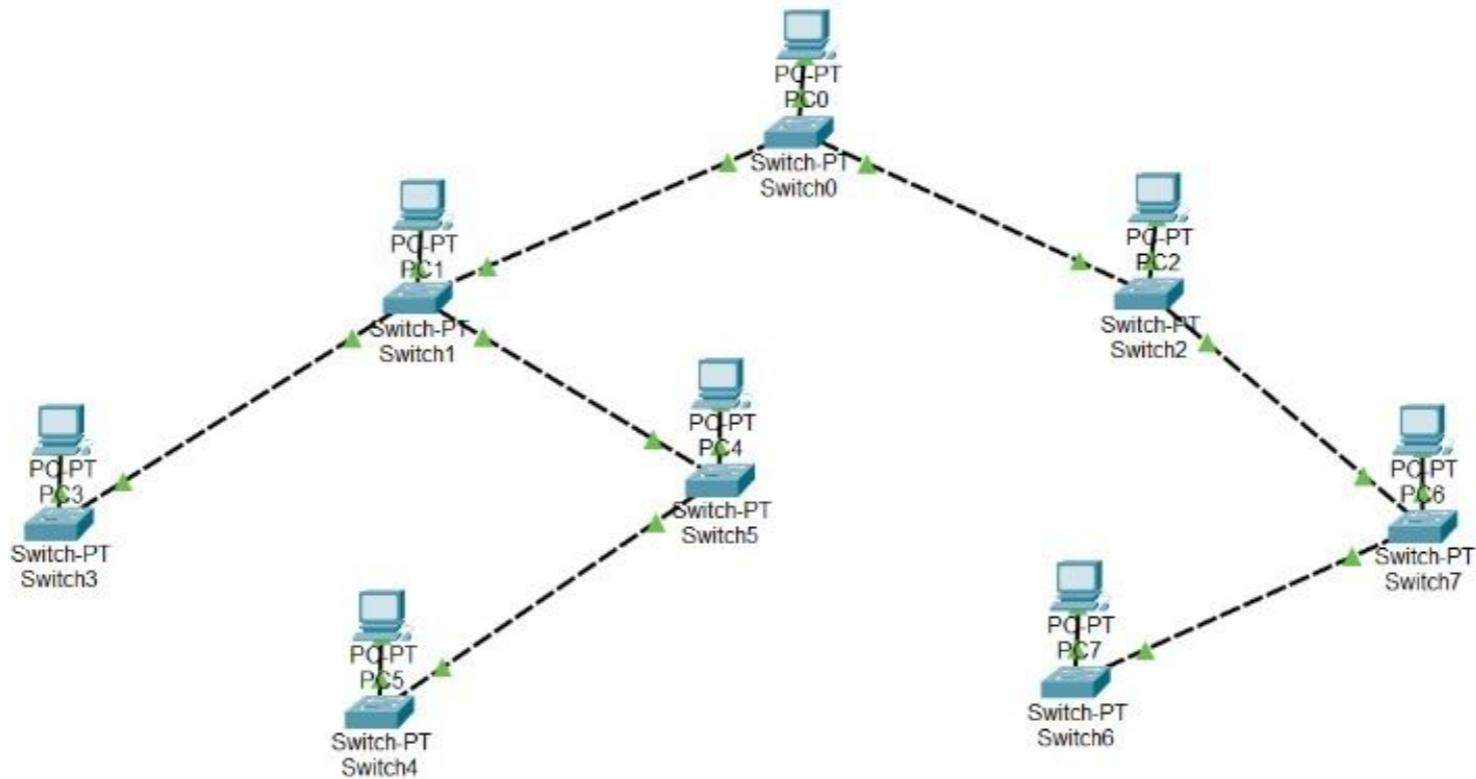


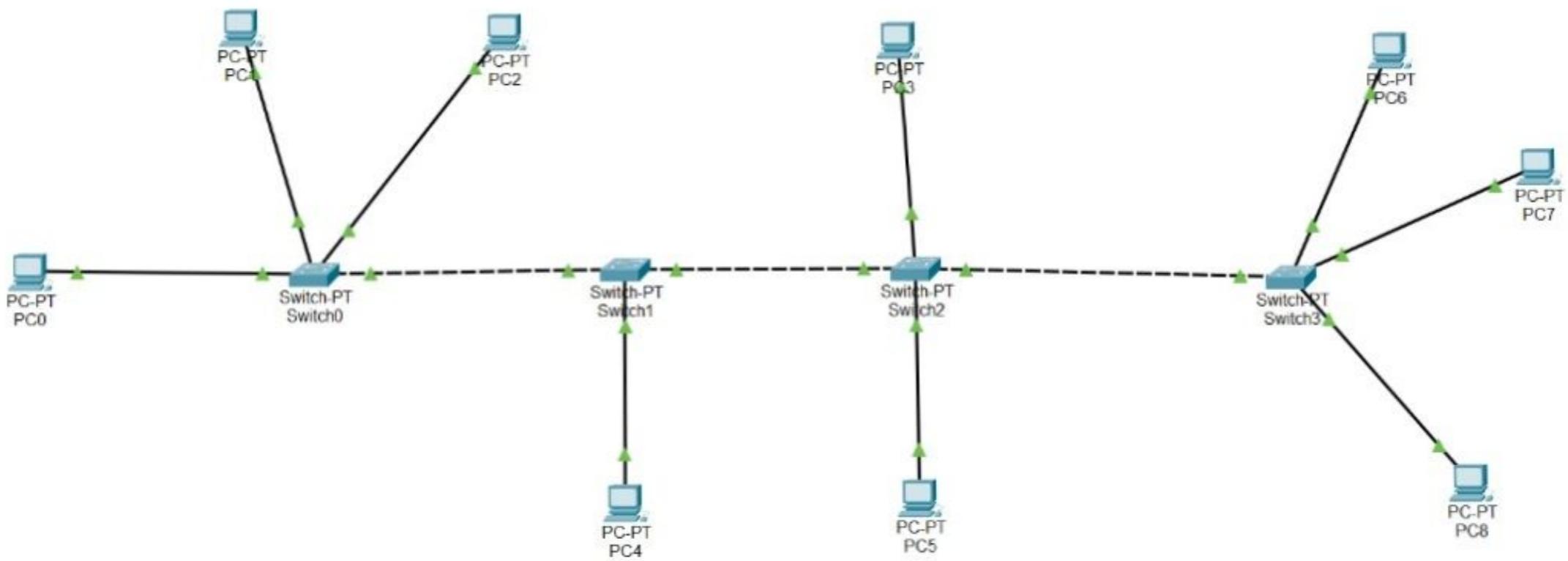
Scenario 0 New Delete

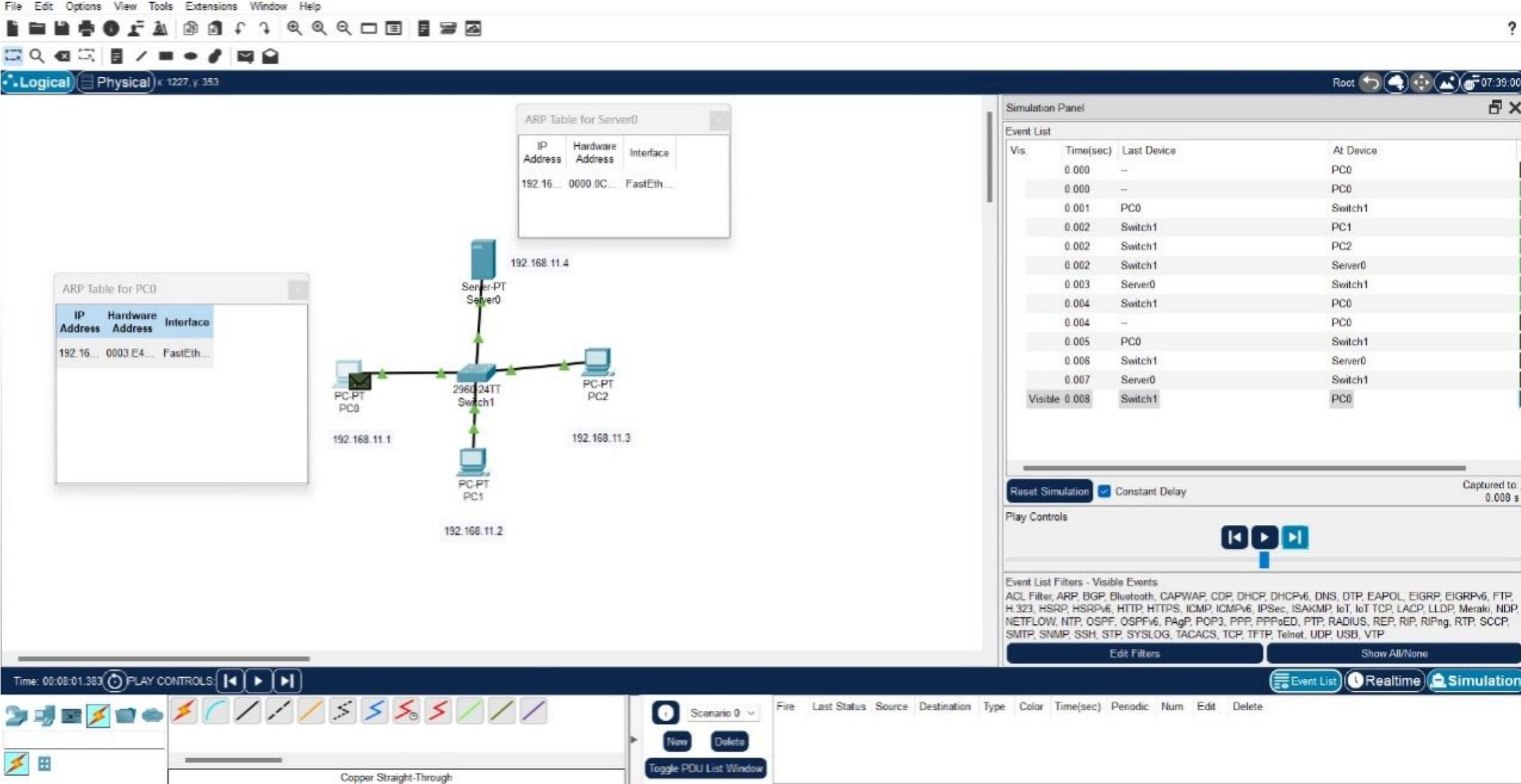
Toggle PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Period
	-	PC1	PC4	ICMP		0.000	N
	-	PC0	PC2	ICMP		0.000	N
	-	PC0	PC3	ICMP		0.000	N

(Select a Device to Drag and Drop to the Workspace)



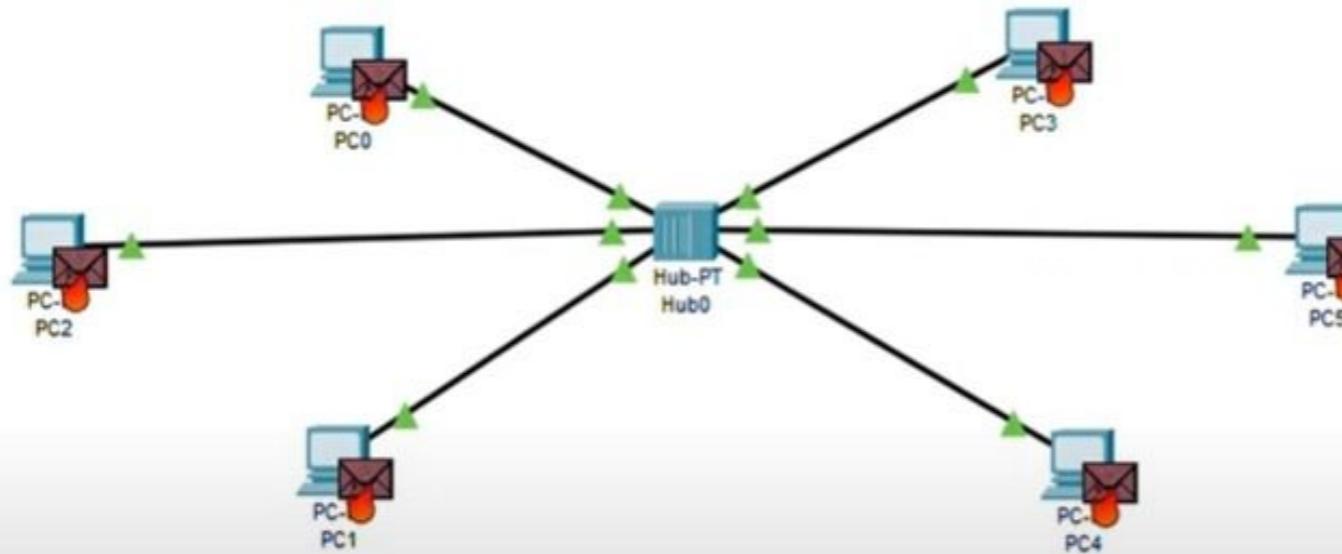






 Logical  Physical x: 1195, y: 420

Root     09:44:00



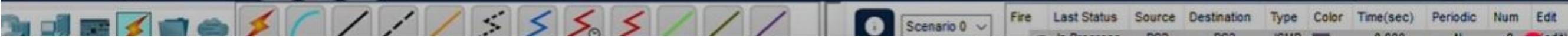
Time: 00:08:27.172

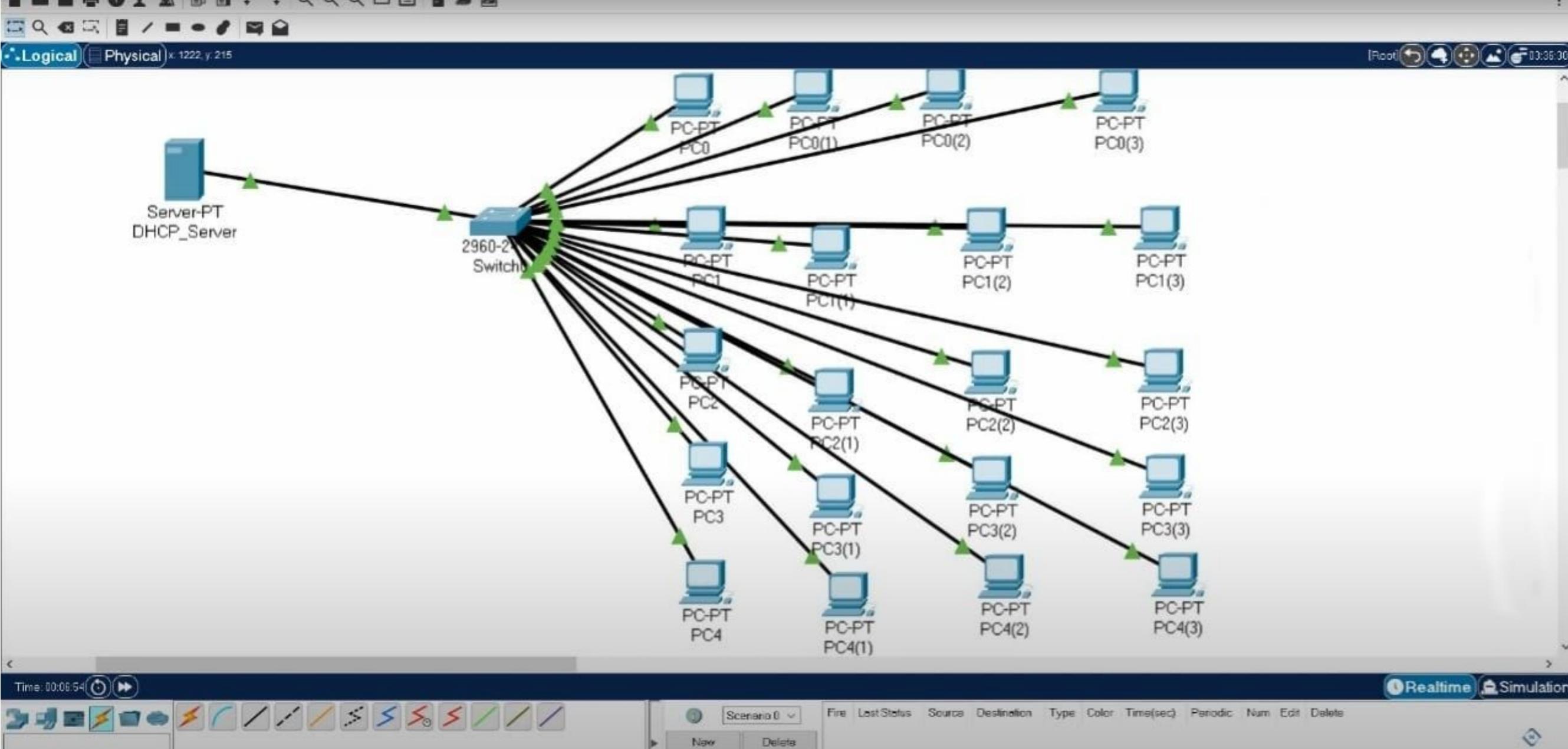
A set of three small blue navigation icons: a left arrow, a right arrow, and a double-right arrow.

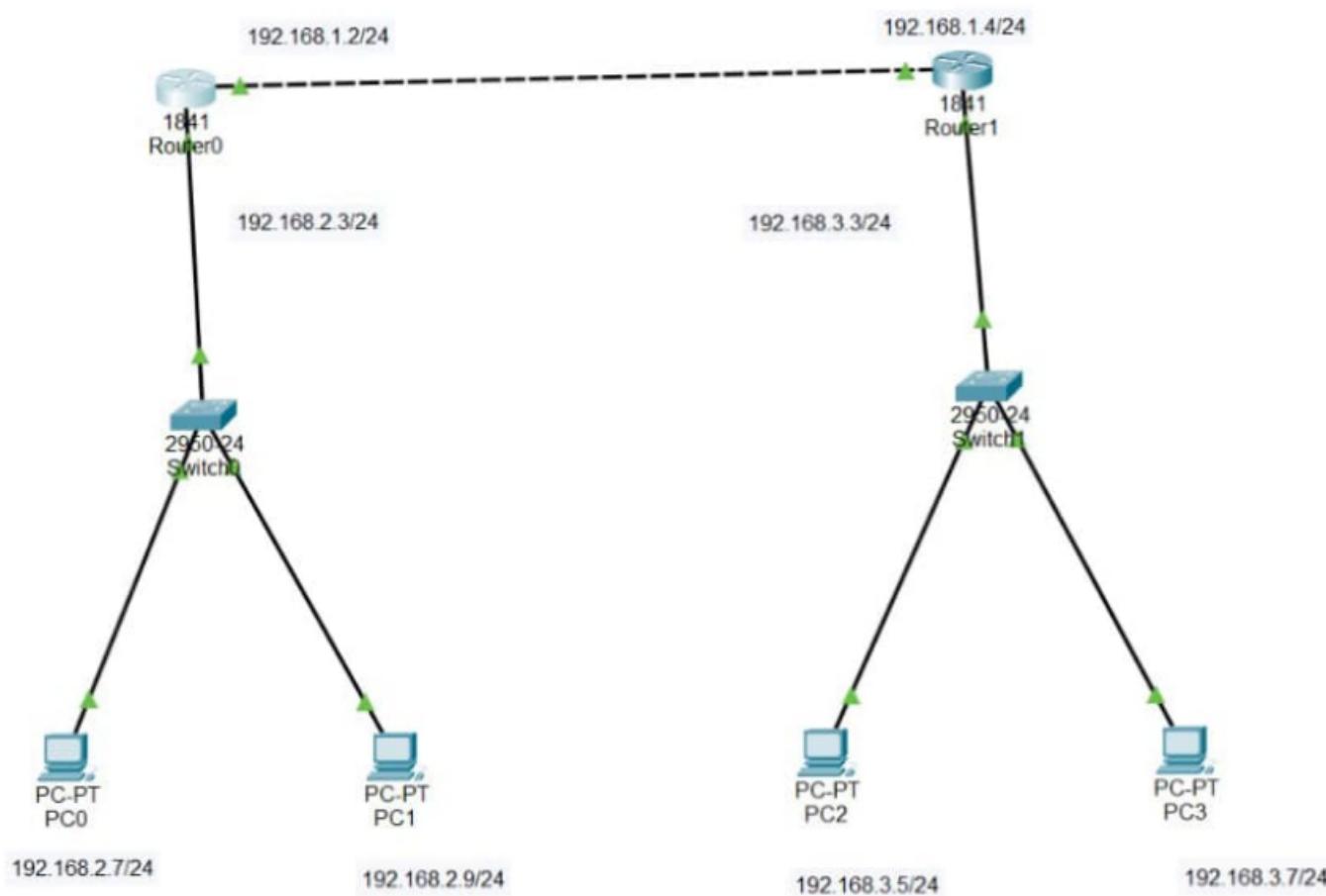
Event List

Realtime

### Simulation



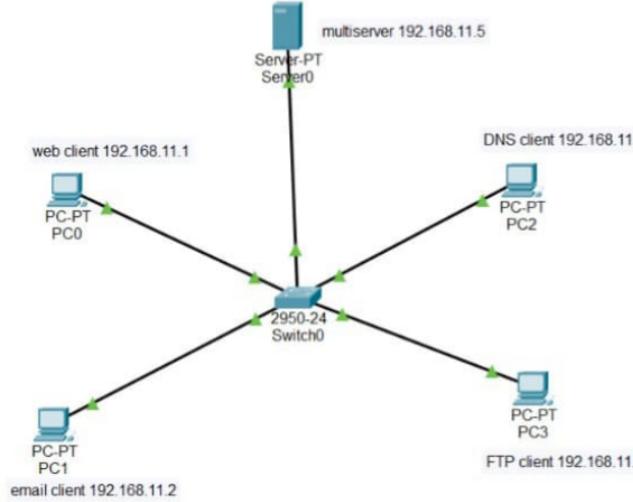






Logical Physical x: 1295, y: 399

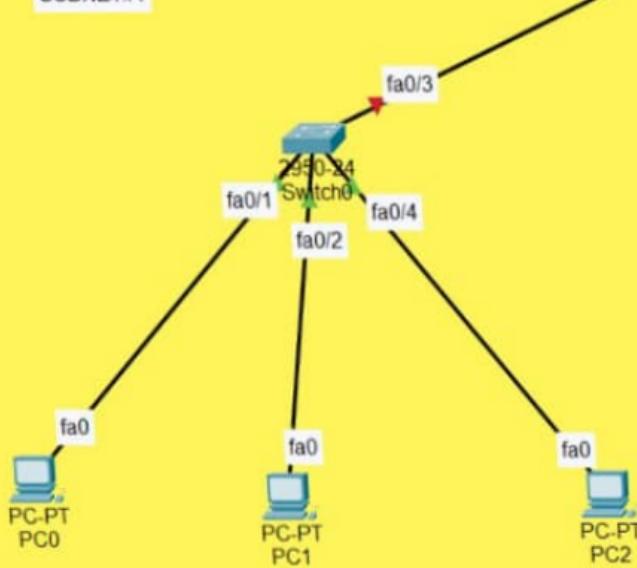
Root ↺ ⏪ ⏴ ⏵ ⏷ ⏸ ⏹ 00:51:00



## SUBNETTING C CLASS ADDRESS

NETWORK ADDRESS 192.168.10.0  
/25

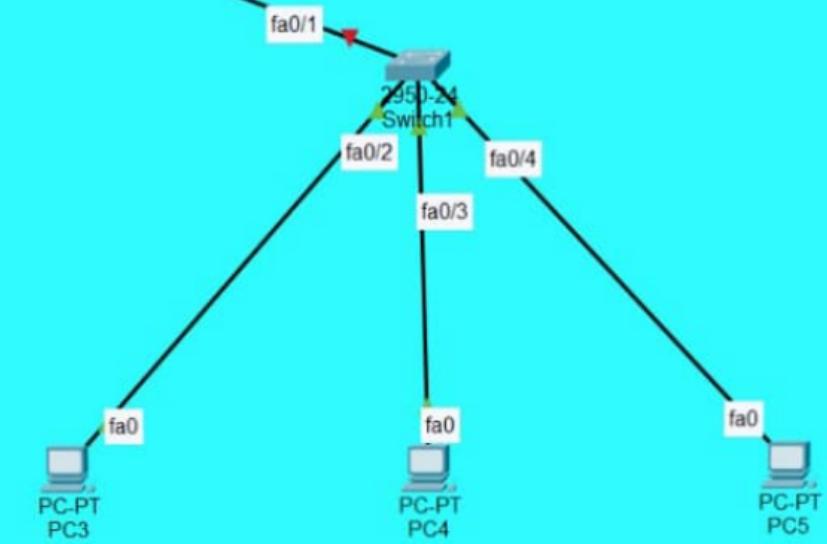
SUBNET#1



gig0/0 2911  
Router0

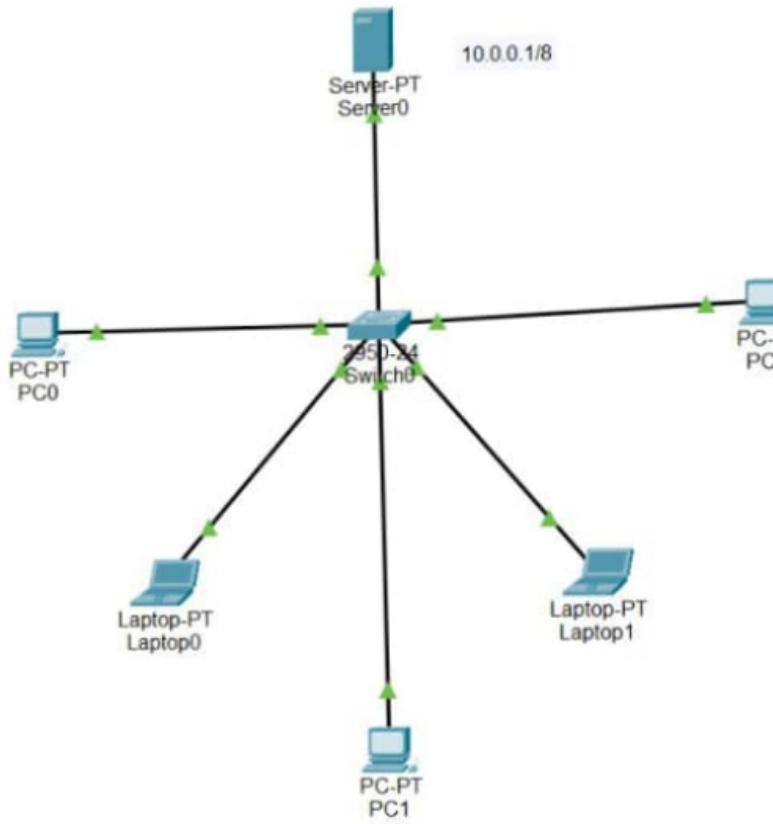
gig0/1

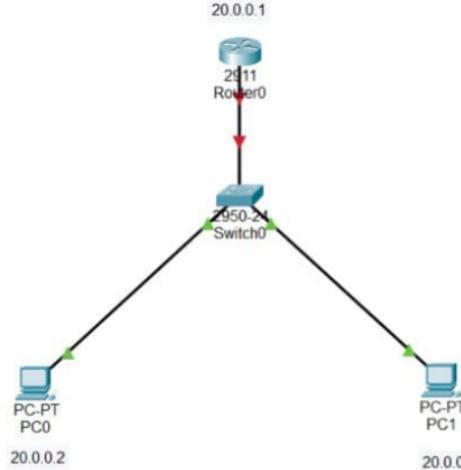
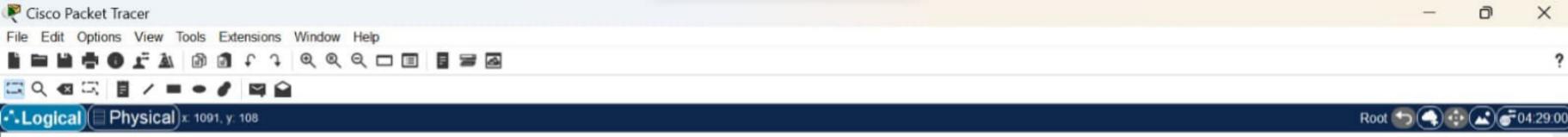
SUBNET#2

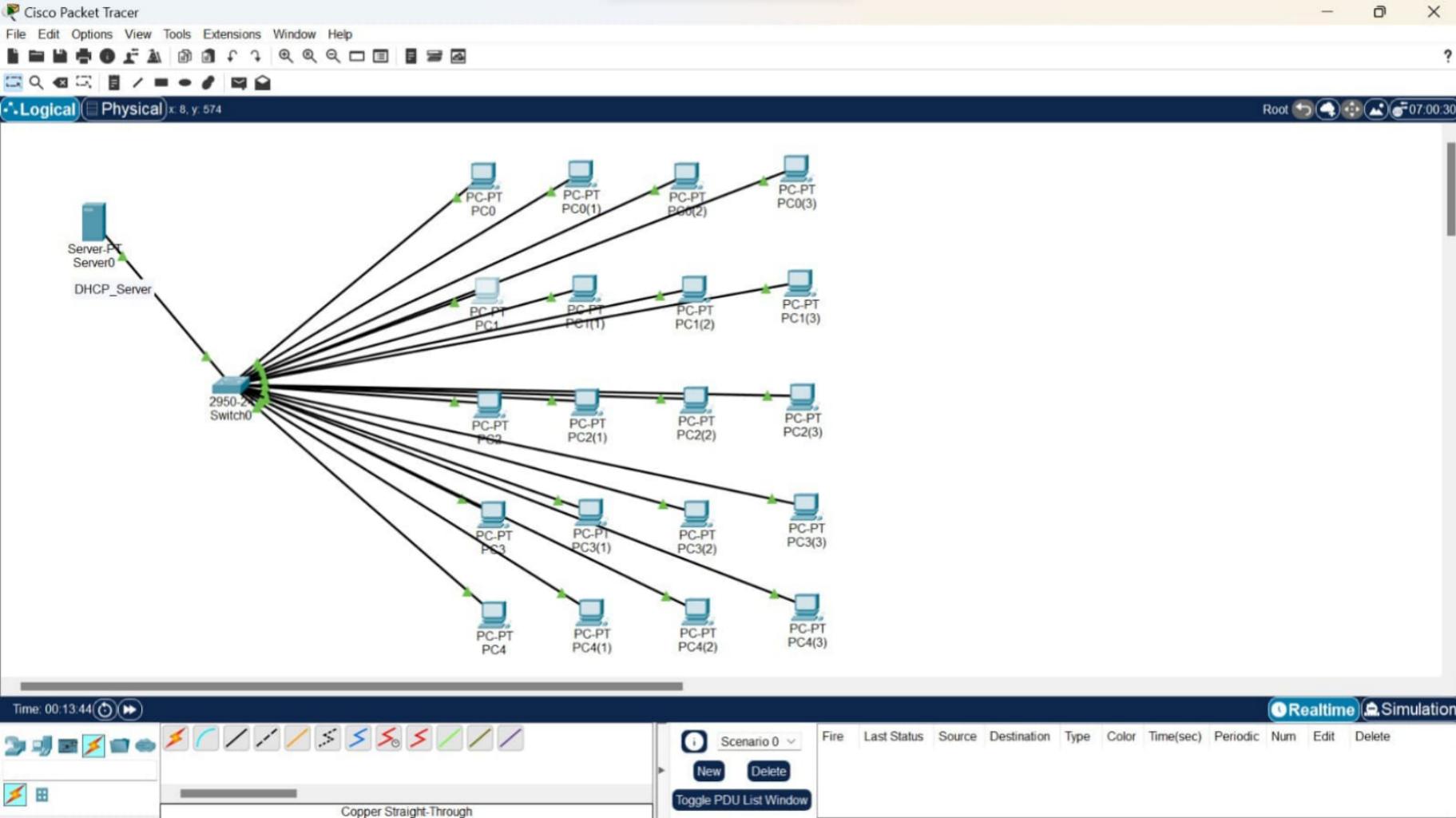


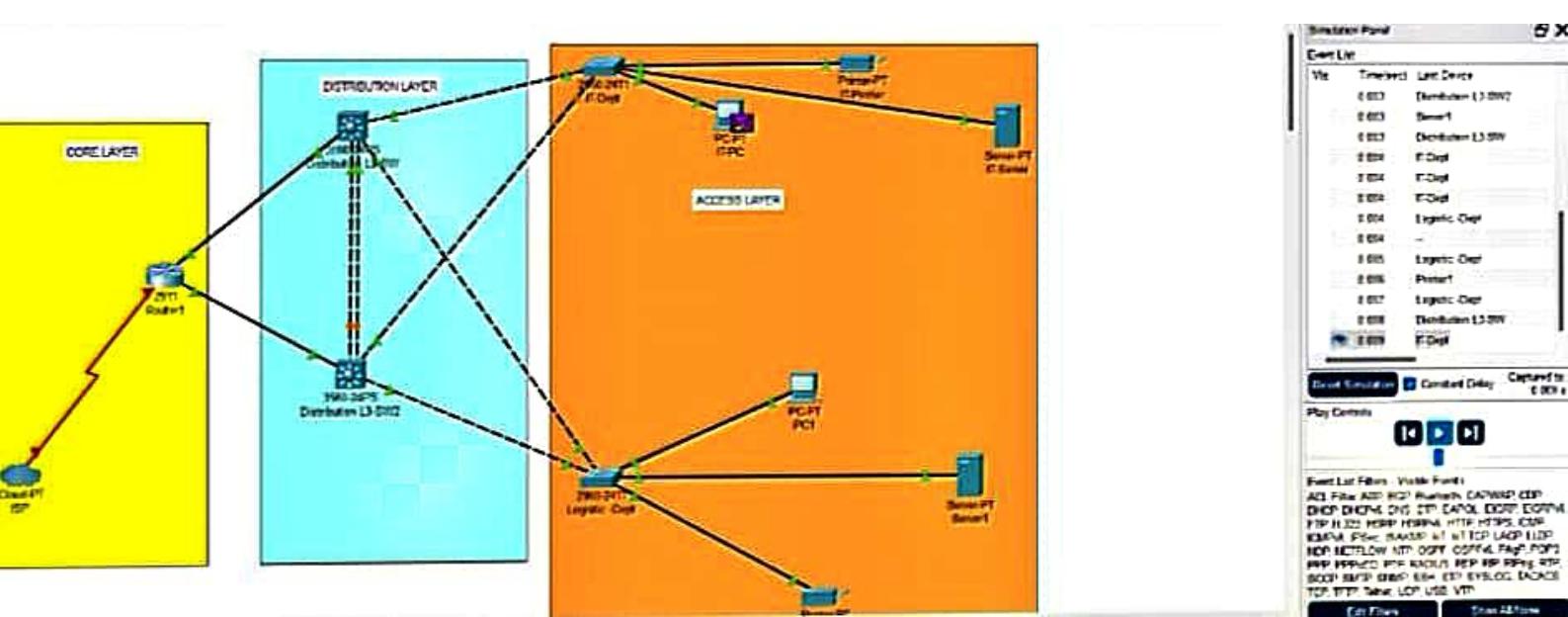


Logical Physical x: 940, y: 96









Simulation Panel

Event List

No.	Time/seq.	Unit	Device
1	0.003		Distribution L3 SW2
2	0.003		Server1
3	0.003		Distribution L3 SW
4	0.004		F-Dest
5	0.004		E-Dest
6	0.004		E-Dest
7	0.004		Logistic-Dest
8	0.004		-
9	0.005		Logistic-Dest
10	0.005		Server1
11	0.007		Logistic-Dest
12	0.008		Distribution L3 SW
13	0.009		F-Dest

Event Simulation  Committed Delay: 0.000 s

Play Controls

Event List Filter: Watch Point

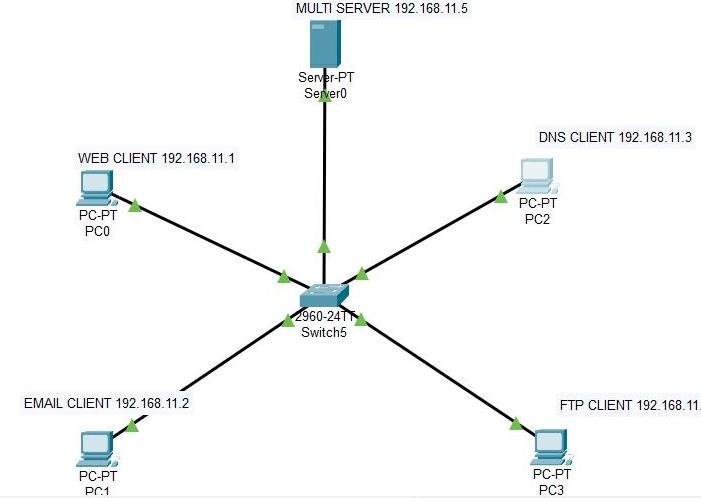
AQ, File, ADD, RCP, RunWith, CAPTURE, CDP, DHCP, DHCPOF, DNS, STP, EAPOL, EIGRP, EIGRPv6, FIP, FLSU, HSRP, HSRPv6, HTTP, HTTPS, ICMP, IGMP, ISIC, IS-IS, LACP, LLDP, LDP, LDP-METISLOW, MTU, OIF, OSPF, OSPFv3, PING, PPP, PPPoE, PPPoEV, RCP, RADIUS, REP, RIP, RIPng, RTSP, SCOP, SBCP, SBCPv6, STP, SYSLOG, TACACS, TDR, TFTP, Telnet, LDP, USB, VTP

Edit Filter Show All Items

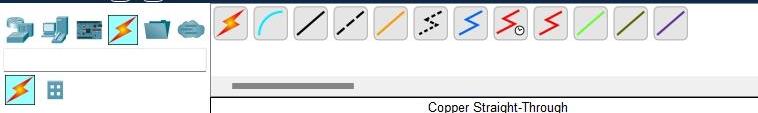


Logical Physical x 936, y 3799

Root ↺ ⏪ ⏴ ⏵ ⏷ 19:28:30



Time: 00:38:35 ⏪ ⏩



Fire Last Status Source Destination Type Color Time(sec) Periodic Num Edit Delete

New Delete

Toggle PDU List Window

Realtime Simulation

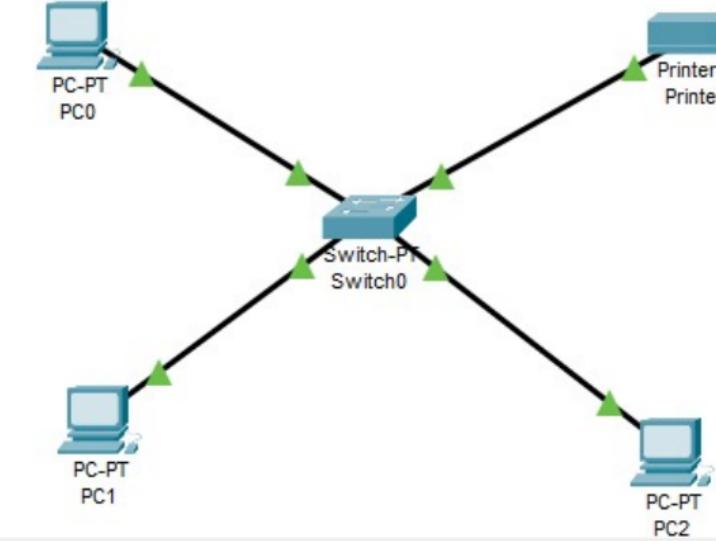


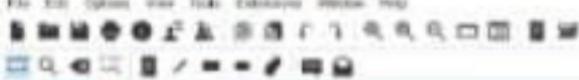
Logical Physical x: 522, y: 124

Root

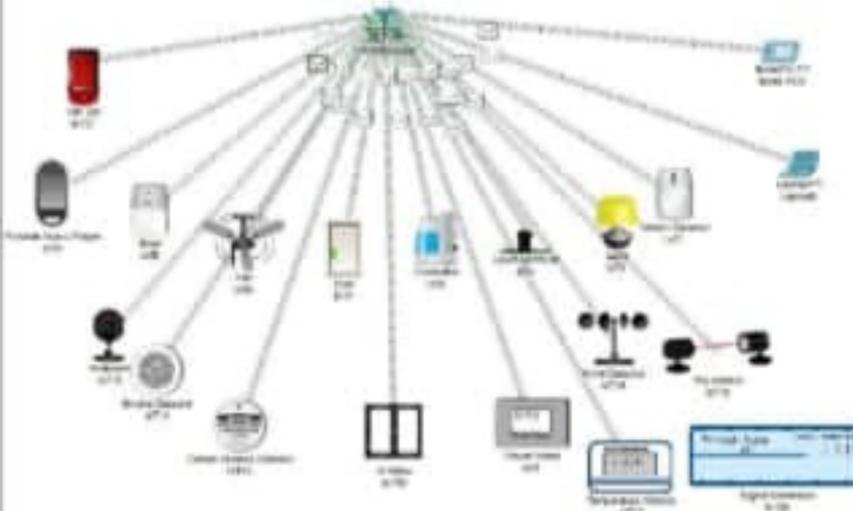


18:05:30





Logical Physics 1401 : 46



Inventory Report		
Item List		
No.	Item Code	Last Checked
1	Vinilo 2 RTV	2023-09-15
2	Vinilo 2 RTV	2023-09-15
3	Vinilo 2 RTV	2023-09-15
4	Vinilo 2 RTV	2023-09-15
5	Vinilo 2 RTV	2023-09-15
6	Vinilo 2 RTV	2023-09-15
7	Vinilo 2 RTV	2023-09-15
8	Vinilo 2 RTV	2023-09-15
9	Vinilo 2 RTV	2023-09-15
10	Vinilo 2 RTV	2023-09-15
11	Vinilo 2 RTV	2023-09-15
12	Vinilo 2 RTV	2023-09-15
13	Vinilo 2 RTV	2023-09-15
14	Vinilo 2 RTV	2023-09-15
15	Vinilo 2 RTV	2023-09-15
16	Vinilo 2 RTV	2023-09-15
17	Vinilo 2 RTV	2023-09-15
18	Vinilo 2 RTV	2023-09-15
19	Vinilo 2 RTV	2023-09-15
20	Vinilo 2 RTV	2023-09-15

Printed: Tuesday, January 20, 2016

Page 10

Play Games

10

如想了解更多的信息

SHOOT SHOTGUN GUN, SHOT GUN, SHOTGUN

中国科学院植物研究所植物学国家重点实验室

新嘉坡三月廿四日正午時分

1994-1995-1996-1997-1998-1999-2000

Wolff Wolff Wolff Wolff Wolff Wolff Wolff Wolff

TOP 行動 Team 10P 的點評

[Edit Filters](#) [Search All now](#)

第十一章 项目管理与组织行为

第1章 项目管理概论

www.ijerpi.org

www.wiley.com/go/teachingandlearning



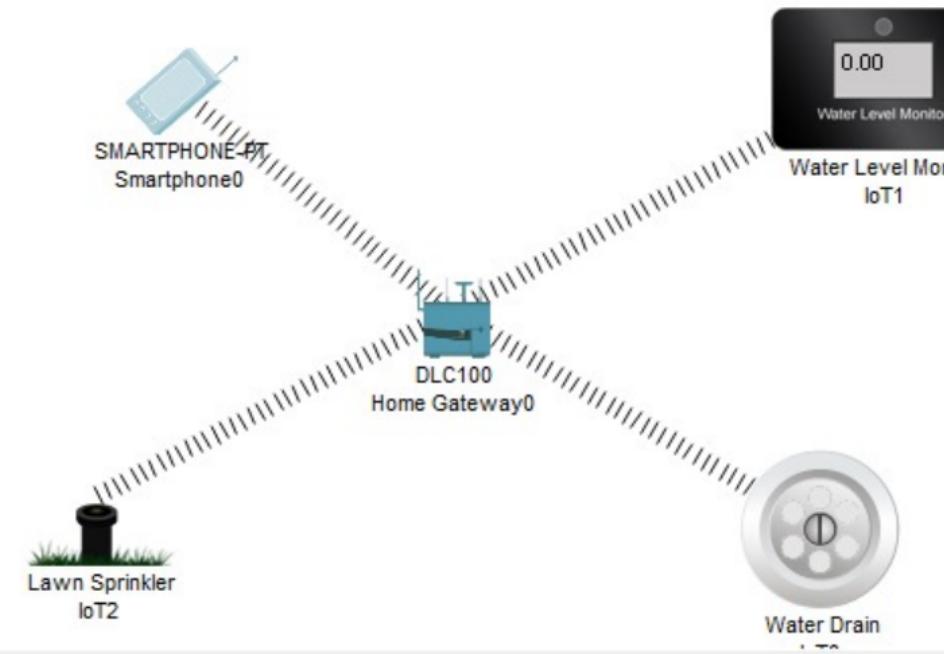
bioRxiv preprint doi: <https://doi.org/10.1101/2023.09.11.570000>; this version posted September 11, 2023. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under a [aCC-BY-ND 4.0 International license](https://creativecommons.org/licenses/by-nd/4.0/).

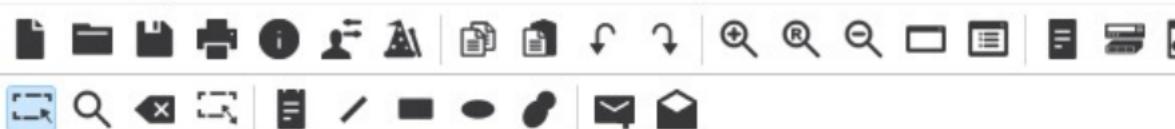
www.ijerpi.org



Logical Physical x: 816, y: 197

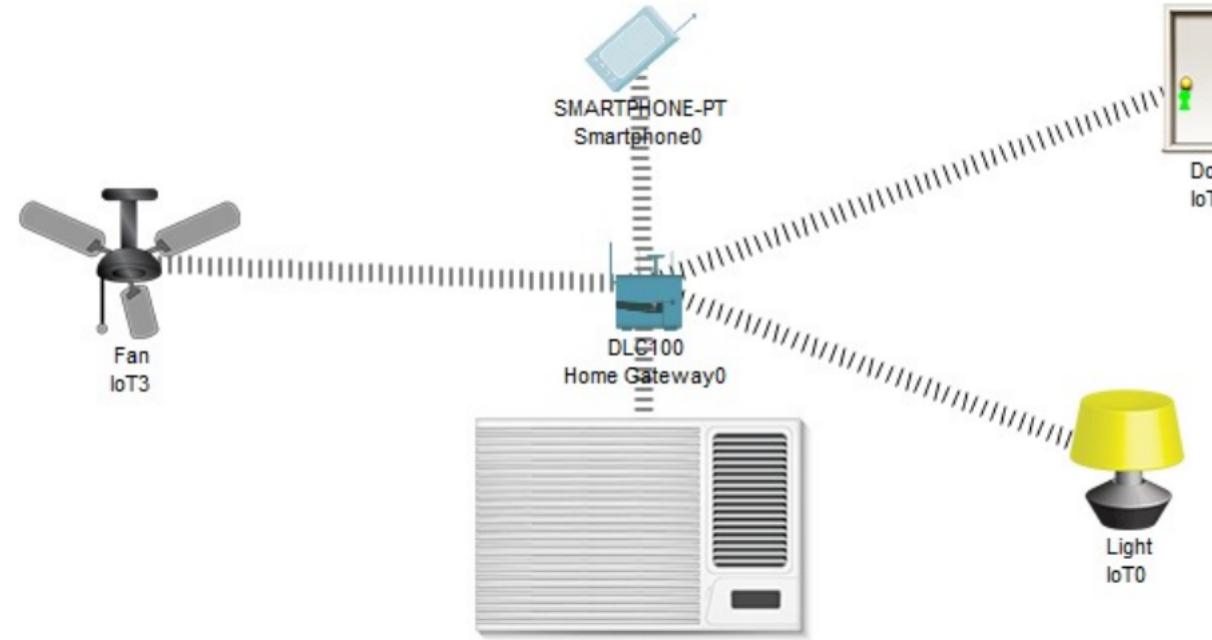
Root ↺ ⚡ ⌂ ⌃ ⌄ 01:57:00





Logical Physical x: 782, y: 55

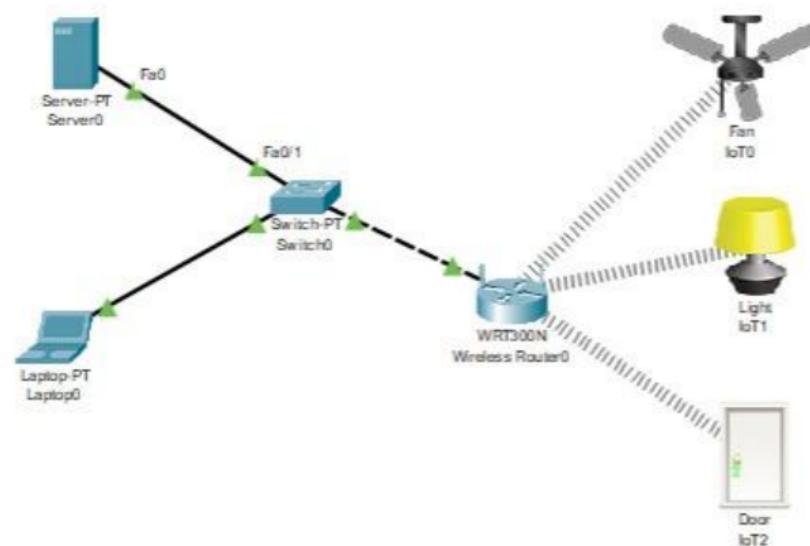
Root 01:09:30





Logical Physical x: 519, y: 188

Root ⏪ ⏴ ⏵ ⏶ ⏷ 07:06:30



No.	Time	Source	Destination	Protocol	Length	Info
11	0.244413	2405:201:e01b:205b:... 2405:200:1607:1731:...	TCP	74	51674 → 443 [ACK] Seq=1753 Ack=452 Win=1022 Len=0	
12	3.628912	2405:200:1607:1731:... 2405:201:e01b:205b:...	TCP	86	443 → 51680 [SYN, ACK] Seq=0 Ack=1 Win=64800 Len=0 MSS=1344 SACK_PERM WS=128	
13	3.629433	2405:200:1607:1731:... 2405:201:e01b:205b:...	TCP	86	443 → 51676 [SYN, ACK] Seq=0 Ack=1 Win=64800 Len=0 MSS=1344 SACK_PERM WS=128	
14	3.629433	2405:200:1607:1731:... 2405:201:e01b:205b:...	TCP	86	443 → 51678 [SYN, ACK] Seq=0 Ack=1 Win=64800 Len=0 MSS=1344 SACK_PERM WS=128	
15	10.077682	192.168.29.61	3.111.224.186	TCP	54	51577 → 443 [RST, ACK] Seq=25 Ack=1 Win=0 Len=0
16	14.607220	192.168.29.61	204.79.197.203	TCP	54	51672 → 443 [FIN, ACK] Seq=1 Ack=1 Win=1018 Len=0
17	14.607467	192.168.29.61	49.44.183.139	TCP	54	51671 → 443 [FIN, ACK] Seq=1 Ack=1 Win=1020 Len=0
18	14.629760	49.44.183.139	192.168.29.61	TLSv1.2	78	Application Data
19	14.629760	49.44.183.139	192.168.29.61	TCP	54	443 → 51671 [FIN, ACK] Seq=25 Ack=2 Win=501 Len=0
20	14.629853	192.168.29.61	49.44.183.139	TCP	54	51671 → 443 [RST, ACK] Seq=2 Ack=25 Win=0 Len=0
21	14.630024	204.79.197.203	192.168.29.61	TCP	54	443 → 51672 [ACK] Seq=1 Ack=2 Win=16383 Len=0
22	14.630024	204.79.197.203	192.168.29.61	TCP	54	443 → 51672 [FIN, ACK] Seq=1 Ack=2 Win=16383 Len=0
23	14.630081	192.168.29.61	204.79.197.203	TCP	54	51672 → 443 [ACK] Seq=2 Ack=2 Win=1018 Len=0

> Frame 1: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits) on interface \Device\NPF\_{...}

✓ Ethernet II, Src: AzureWaveTec\_9c:a1:40 (94:bb:43:9c:a1:40), Dst: ServercomPri\_6e:f0:1 (08:00:27:6e:f0:11)

- > Destination: ServercomPri\_6e:f0:11 (8c:a3:99:6e:f0:11)
- > Source: AzureWaveTec\_9c:a1:40 (94:bb:43:9c:a1:40)
- Type: IPv6 (0x86dd)
- [Stream index: 0]

> Internet Protocol Version 6, Src: 2405:201:e01b:205b:e83d:b683:4369:37a1, Dst: 2405:200:1607:1731:... (08:00:27:6e:f0:11)

> Transmission Control Protocol, Src Port: 51674, Dst Port: 443, Seq: 1, Ack: 1, Len: 27

> Transport Layer Security

0000	8c a3 99 6e f0 11 94 bb	43 9c a1 40 86 dd 60 03	..n.. C @`
0010	c9 14 01 28 06 40 24 05	02 01 e0 1b 20 5b e8 3d	..( @\$ .. [ =
0020	b6 83 43 69 37 a1 24 05	02 00 16 07 17 31 00 00	.Ci7 \$ .. 1 ..
0030	00 00 31 2c 74 d8 c9 da	01 bb 0d df 75 05 57 7b	.1,t .. uW{
0040	ec f0 50 18 04 00 f9 75	00 00 17 03 03 01 0f a4	.P .. u ..
0050	b2 0e 11 f3 bc 81 b8 8a	2b 1d 2c 6f 00 df 1c de	..... + ,o ..
0060	7f 25 53 3a 5b c5 9e 98	d6 d9 50 4a 4e 32 52 9a	.%S:[ .. PjN2R ..
0070	ed c7 b9 05 5c 54 83 81	80 7d e8 e4 8c 21 8c c8	.. \T .. } .. ! ..
0080	9e 9f 55 07 a4 99 69 d3	18 f6 bb 9a 96 1b bc 39	.U .. i .. 9 ..
0090	47 3c 44 ee fa 84 f8 2d	6a 9d 08 40 4f f8 dc e8	G<D .. j .. @0 ..
00a0	8f e7 e0 fd d9 24 83 7d	23 0f 06 f1 b1 19 39 74	.... \$ .. # .. 9t ..
00b0	9c 6d ac e2 68 df fc 1b	92 3a 63 91 63 aa 7f fd	m .. h .. :c .. c ..
00c0	1e b6 0c 12 26 f1 c2 2a	b8 25 11 51 79 2e 5a 55	.... & .. * .. %Q .. ZU ..
00d0	53 f9 dd eb fd 76 22 72	c1 5e 72 52 cc e3 08 88	S .. v"r .. ^rR ..
00e0	c6 82 a9 2d 68 45 7c 51	43 9c b3 2a 34 45 bd c6	.... -hE Q .. C .. *4E ..
00f0	77 e6 13 5c fc eb dc 45	0e 54 35 28 ec 2c d8 4f	w .. \ .. E .. T5( .. , .. 0 ..
0100	cb 8f 02 f5 c4 47 0b d8	31 c0 2c 4e 63 9e 35 34	.... G .. 1 .. , .. Nc .. 54 ..
0110	fe c0 bd 60 79 ff d8 47	1a 7d e0 1d 82 19 90 6d	.... `y .. G .. } .. m ..



Apply a display filter ... &lt;Ctrl-/&gt;



No.	Time	Source	Destination	Protocol	Length	Info
246	1.068613	2405:200:1607:2885:...	2405:201:e01b:205b:...	TLSv1.3	97	[TCP Previous segment not captured] , Application Data
247	1.068659	2405:201:e01b:205b:...	2405:200:1607:2885:...	TCP	86	[TCP Dup ACK 245#1] 51740 → 443 [ACK] Seq=808 Ack=3630 Win=261632 Len=0 SLE=3999 SRE=4022
248	1.069277	2405:200:1607:2885:...	2405:201:e01b:205b:...	TCP	1466	443 → 51740 [ACK] Seq=4022 Ack=808 Win=31744 Len=1392 [TCP PDU reassembled in 251]
249	1.069299	2405:201:e01b:205b:...	2405:200:1607:2885:...	TCP	86	[TCP Dup ACK 245#2] 51740 → 443 [ACK] Seq=808 Ack=3630 Win=261632 Len=0 SLE=3999 SRE=5414
250	1.085003	2405:200:1607:2885:...	2405:201:e01b:205b:...	TCP	1466	443 → 51740 [PSH, ACK] Seq=5414 Ack=808 Win=31744 Len=1392 [TCP PDU reassembled in 251]
251	1.085003	2405:200:1607:2885:...	2405:201:e01b:205b:...	TLSv1.3	848	Application Data
252	1.085037	2405:201:e01b:205b:...	2405:200:1607:2885:...	TCP	86	[TCP Dup ACK 245#3] 51740 → 443 [ACK] Seq=808 Ack=3630 Win=261632 Len=0 SLE=3999 SRE=6806
253	1.085075	2405:201:e01b:205b:...	2405:200:1607:2885:...	TCP	86	[TCP Dup ACK 245#4] 51740 → 443 [ACK] Seq=808 Ack=3630 Win=261632 Len=0 SLE=3999 SRE=7580
254	1.097225	2405:200:1607:2885:...	2405:201:e01b:205b:...	TLSv1.3	443	[TCP Fast Retransmission] , Application Data
255	1.097908	2405:201:e01b:205b:...	2405:200:1607:2885:...	TCP	74	51740 → 443 [ACK] Seq=808 Ack=7580 Win=262144 Len=0
256	1.105337	2a03:2880:f34e:120:...	2405:201:e01b:205b:...	TCP	74	443 → 51746 [ACK] Seq=3605 Ack=619 Win=67840 Len=0
257	4.764335	2405:201:e01b:205b:...	2405:200:1607:1731:...	TCP	74	51674 → 443 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
258	4.764500	2405:201:e01b:205b:...	2405:200:1607:1731:...	TCP	74	51675 → 443 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

▼ Transmission Control Protocol, Src Port: 80, Dst Port: 51738, Seq: 1, Ack: 1, Len: : Source Port: 80 Destination Port: 51738 [Stream index: 0] [Stream Packet Number: 1]	0000 94 bb 43 9c a1 40 8c a3 99 6e f0 11 86 dd 68 c6 ..C@.. n...h. 0010 26 b1 00 95 06 38 2a 03 28 80 f3 4e 01 21 fa ce &...8* (..N!.. 0020 b0 0c 00 00 72 60 24 05 02 01 e0 1b 20 5b e8 3d ...r\$ .. [:= 0030 b6 83 43 69 37 a1 00 50 ca 1a d0 6f 96 81 62 8a ..Ci7..P ..o..b.. 0040 4a 55 50 18 01 05 ae e2 00 00 48 54 54 50 2f 31 JUP..... HTTP/1 0050 2e 31 20 32 30 30 20 4f 4b 0d 0a 43 61 63 68 65 .1 200 O K..Cache 0060 2d 43 6f 6e 74 72 6f 6c 3a 20 6e 6f 2d 63 61 63 -Control : no-cac 0070 68 65 2c 20 6e 6f 2d 73 74 6f 72 65 2c 20 6d 75 he, no-s tore, mu 0080 73 74 2d 72 65 76 61 6c 69 64 61 74 65 0d 0a 50 st-reval idate..P 0090 72 61 67 6d 61 3a 20 6e 6f 2d 63 61 63 68 65 0d ragma: n o-cache.. 00a0 0a 45 78 70 69 72 65 73 3a 20 30 0d 0a 54 72 61 ..Expires : 0..Tra 00b0 6e 73 66 65 72 2d 45 6e 63 6f 64 69 6e 67 3a 20 nsfer-En coding: 00c0 63 68 75 6e 6b 65 64 0d 0a 0d 0a chunked... 0101 .... = Header Length: 20 bytes (5)
> Flags: 0x018 (PSH, ACK)	...

Time	Source	Destination	Protocol	Length	Info
388 48.696789	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	1514	Ignored Unknown Record
389 48.696789	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	1514	Ignored Unknown Record
390 48.696789	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	1514	Ignored Unknown Record
391 48.696789	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	1514	Ignored Unknown Record
392 48.696789	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	2954	Ignored Unknown Record
393 48.696941	2405:201:e01b:205b::	2405:200:1630:caf::	TCP	74	51754 → 443 [ACK] Seq=1038 Ack=62165 Win=66048 Len=0
394 48.710219	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	1514	Ignored Unknown Record
395 48.710288	2405:201:e01b:205b::	2405:200:1630:caf::	TCP	74	51754 → 443 [ACK] Seq=1038 Ack=63605 Win=66048 Len=0
396 48.711452	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	1514	Ignored Unknown Record
397 48.711452	2405:200:1630:caf::...	2405:201:e01b:205b::	TLSv1.2	1514	Ignored Unknown Record
398 48.711452	2405:200:1630:caf::...	2405:201:e01b:205b::	TCP	1514	443 → 51754 [PSH, ACK] Seq=66485 Ack=1038 Win=64128 Len=1440
399 48.711554	2405:201:e01b:205b::	2405:200:1630:caf::	TCP	74	51754 → 443 [ACK] Seq=1038 Ack=67925 Win=66048 Len=0
400 48.736571	2405:200:1630:caf::...	2405:201:e01b:205b::	TCP	1514	443 → 51754 [ACK] Seq=67925 Ack=1038 Win=64128 Len=1440

Acknowledgment Number: 1 (relative ack number)

Acknowledgment number (raw): 1653230165

0101 .... = Header Length: 20 bytes (5)

> Flags: 0x018 (PSH, ACK)

Window: 261

[Calculated window size: 261]

[Window size scaling factor: -1 (unknown)]

Checksum: 0xae2 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

> [SEQ/ACK analysis]

TCP payload (129 bytes)

TCP segment data (129 bytes)

0000	94	bb	43	9c	a1	40	8c	a3	99	6e	f0	11	86	dd	68	c6	..C..@..	-n....h..
0010	26	b1	00	95	06	38	2a	03	28	80	f3	4e	01	21	fa	ce	&...-8*	(..N.. ..
0020	b0	0c	00	00	72	60	24	05	02	01	e0	1b	20	5b	e8	3d	....r`\$..	....[..=
0030	b6	83	43	69	37	a1	00	50	ca	1a	d0	6f	96	81	62	8a	..Ci7..P..	...o..b..
0040	4a	55	50	18	01	05	ae	e2	00	00	48	54	54	50	2f	31	JUP.....	..HTTP/1
0050	2e	31	20	32	30	30	20	4f	4b	0d	0a	43	61	63	68	65	.1	200 O K..Cache
0060	2d	43	6f	6e	74	72	6f	6c	3a	20	6e	6f	2d	63	61	63	-Control :	no-cac
0070	68	65	2c	20	6e	6f	2d	73	74	6f	72	65	2c	20	6d	75	he, no-s tore,	mu
0080	73	74	2d	72	65	76	61	6c	69	64	61	74	65	0d	0a	50	st-reval idate..P	
0090	72	61	67	6d	61	3a	20	6e	6f	2d	63	61	63	68	65	0d	pragma: n o-cache..	
00a0	0a	45	78	70	69	72	65	73	3a	20	30	0d	0a	54	72	61	+Expires : 0..Tra	
00b0	6e	73	66	65	72	2d	45	6e	63	6f	64	69	6e	67	3a	20	nsfer-En coding:	
00c0	63	68	75	6e	6b	65	64	0d	0a	0d	0a						chunked..	



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define PORT 8080
8 #define BUFFER_SIZE 1024
9
10 void send_file(FILE *fp, int sockfd) {
11     char buffer[BUFFER_SIZE];
12
13     while (fgets(buffer, BUFFER_SIZE, fp) != NULL) {
14         if (send(sockfd, buffer, strlen(buffer), 0) == -1) { // Send only valid data
15             perror("Error sending file");
16             exit(1);
17         }
18         memset(buffer, 0, BUFFER_SIZE);
19     }
20 }
```

Listening on port 8080...

== Session Ended. Please Run the code again ==



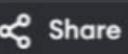
```
21
22 int main() {
23     int sockfd, new_sock;
24     struct sockaddr_in server_addr, new_addr;
25     socklen_t addr_size;
26
27     sockfd = socket(AF_INET, SOCK_STREAM, 0);
28     if (sockfd < 0) {
29         perror("Socket creation failed");
30         exit(EXIT_FAILURE);
31     }
32
33     server_addr.sin_family = AF_INET;
34     server_addr.sin_addr.s_addr = INADDR_ANY;
35     server_addr.sin_port = htons(PORT);
36
37     if (bind(sockfd, (struct sockaddr *)&server_addr, sizeof
38             (server_addr)) < 0) {
39         perror("Binding failed");
40         close(sockfd);
41         exit(EXIT_FAILURE);
42     }
```

Listening on port 8080...

==== Session Ended. Please Run the code again ===



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define PORT 8080
8 #define BUFFER_SIZE 1024
9
10 void receive_file(int sockfd) {
11     char buffer[BUFFER_SIZE];
12     FILE *fp = fopen("received_file.txt", "w");
13     if (fp == NULL) {
14         perror("File open failed");
15         exit(1);
16     }
17
18     int n;
19     while ((n = recv(sockfd, buffer, BUFFER_SIZE, 0)) > 0) {
20         fwrite(buffer, sizeof(char), n, fp);
21         memset(buffer, 0, BUFFER_SIZE);
22     }
}
```

Connection failed: Connection refused

==== Code Exited With Errors ===



main.c



Run

Output

Clear

```
22     }
23
24     if (n < 0) {
25         perror("Error receiving file");
26     }
27
28     fclose(fp);
29 }
30
31 int main() {
32     int sockfd;
33     struct sockaddr_in server_addr;
34
35     sockfd = socket(AF_INET, SOCK_STREAM, 0);
36     if (sockfd < 0) {
37         perror("Socket creation failed");
38         exit(EXIT_FAILURE);
39     }
40
41     server_addr.sin_family = AF_INET;
42     server_addr.sin_port = htons(PORT);
43     server_addr.sin_addr.s_addr = inet_addr("127.0.0.1") //
```

Connection failed: Connection refused

==== Code Exited With Errors ===



main.c



Run

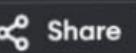
Output

Clear

```
38         exit(EXIT_FAILURE);
39     }
40
41     server_addr.sin_family = AF_INET;
42     server_addr.sin_port = htons(PORT);
43     server_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); //
44             Change to server IP if necessary
45
45 -    if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof
46             (server_addr)) < 0) {
46     perror("Connection failed");
47     close(sockfd);
48     exit(EXIT_FAILURE);
49 }
50
51     receive_file(sockfd);
52     printf("File received successfully.\n");
53
54     close(sockfd);
55     return 0;
56 }
57 |
```

Connection failed: Connection refused

==== Code Exited With Errors ===



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define PORT 5353
8 #define BUFFER_SIZE 512
9 #define DOMAIN "example.com"
10 #define IP_ADDRESS "93.184.216.34"
11
12 int main() {
13     int sockfd;
14     struct sockaddr_in server_addr, client_addr;
15     socklen_t addr_len = sizeof(client_addr);
16     char buffer[BUFFER_SIZE];
17
18     if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
19         perror("Socket creation failed");
20         exit(EXIT_FAILURE);
21     }
22 }
```

DNS Server is listening on port 5353



main.c



Share

Run

Output

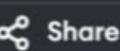
Clear

```
22
23     memset(&server_addr, 0, sizeof(server_addr));
24     server_addr.sin_family = AF_INET;
25     server_addr.sin_addr.s_addr = INADDR_ANY;
26     server_addr.sin_port = htons(PORT);
27
28     if (bind(sockfd, (const struct sockaddr *)&server_addr,
29               sizeof(server_addr)) < 0) {
30         perror("Bind failed");
31         close(sockfd);
32         exit(EXIT_FAILURE);
33     }
34
35     printf("DNS Server is listening on port %d\n", PORT);
36
37     while (1) {
38         int n = recvfrom(sockfd, buffer, BUFFER_SIZE - 1, 0,
39                           (struct sockaddr *)&client_addr,
40                           &addr_len);
41
42         if (n < 0) {
43             perror("Receive failed");
44             continue;
45         }
46
47         // Process received DNS request
48
49         // Construct response
50
51         if (sendto(sockfd, response, strlen(response), 0,
52                    (struct sockaddr *)&client_addr,
53                    addr_len) < 0) {
54             perror("Send failed");
55             continue;
56         }
57
58     }
59
60 }
```

^ DNS Server is listening on port 5353



main.c



Run

Output

Clear

```
    continue;

42 }
43
44 buffer[n] = '\0'; // Null-terminate safely
45 printf("Received request for: %s\n", buffer);
46
47 if (strncmp(buffer, DOMAIN, BUFFER_SIZE) == 0) {
48     if (sendto(sockfd, IP_ADDRESS, strlen(IP_ADDRESS),
49         0,
50             (struct sockaddr *)&client_addr,
51             addr_len) < 0) {
52         perror("Send failed");
53     } else {
54         printf("Responded with IP: %s\n", IP_ADDRESS);
55     }
56 } else {
57     const char *not_found = "Not found";
58     if (sendto(sockfd, not_found, strlen(not_found), 0,
59             (struct sockaddr *)&client_addr,
60             addr_len) < 0) {
61         perror("Send failed");
62     }
63 }
```

\* DNS Server is listening on port 5353



main.c



Run

Output

Clear

```
        addr_len) < 0) {
50            perror("Send failed");
51        } else {
52            printf("Responded with IP: %s\n", IP_ADDRESS);
53        }
54    } else {
55        const char *not_found = "Not found";
56        if (sendto(sockfd, not_found, strlen(not_found), 0,
57                    (struct sockaddr *)&client_addr,
58                    addr_len) < 0) {
59            perror("Send failed");
60        } else {
61            printf("Responded with: Not found\n");
62        }
63    }
64
65    close(sockfd);
66    return 0;
67 }
68 }
```

DNS Server is listening on port 5353



main.c



Run

Output

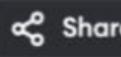
Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define SERVER_IP "127.0.0.1"
8 #define PORT 5353
9 #define DOMAIN "google.com"
10 #define BUFFER_SIZE 512
11
12 int main() {
13     int sockfd;
14     struct sockaddr_in server_addr;
15     char buffer[BUFFER_SIZE];
16     socklen_t addr_len = sizeof(server_addr);
17
18     // Create UDP socket
19     if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
20         perror("Socket creation failed");
21         exit(EXIT_FAILURE);
22     }
```

▲ Sent query for: google.com



main.c



Run

Output

Clear

```
21         exit(EXIT_FAILURE);
22     }
23
24     memset(&server_addr, 0, sizeof(server_addr));
25     server_addr.sin_family = AF_INET;
26     server_addr.sin_port = htons(PORT);
27
28     // Convert IP address
29 -    if (inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr) <=
29 +        0) {
30         perror("Invalid address/ Address not supported");
31         close(sockfd);
32         exit(EXIT_FAILURE);
33     }
34
35     // Send DNS-like query
36 -    if (sendto(sockfd, DOMAIN, strlen(DOMAIN), 0, (struct
36 +        sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
37         perror("Send failed");
38         close(sockfd);
39         exit(EXIT_FAILURE);
40     }
```

Sent query for: google.com



main.c



Run

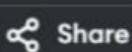
Output

Clear

```
    sockaddr *)&server_addr, sizeof(server_addr)) < 0) { ^ Sent query for: google.com
37     perror("Send failed");
38     close(sockfd);
39     exit(EXIT_FAILURE);
40 }
41
42     printf("Sent query for: %s\n", DOMAIN);
43
44 // Receive response
45     int n = recvfrom(sockfd, buffer, BUFFER_SIZE - 1, 0,
46     (struct sockaddr *)&server_addr, &addr_len);
47     if (n < 0) {
48         perror("Receive failed");
49     } else {
50         buffer[n] = '\0'; // Null-terminate the received data
51         safely
52         printf("Received response: %s\n", buffer);
53     }
54     close(sockfd);
55 }
```



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <arpa/inet.h>
5 #include <unistd.h>
6 #include <netinet/in.h>
7
8 #define DNS_PORT 53
9 #define DNS_SERVER "8.8.8.8" // Google's public DNS server
10 #define BUFFER_SIZE 65536
11
12 // DNS header structure
13 struct DNS_HEADER {
14     unsigned short id;
15     unsigned char rd :1;
16     unsigned char tc :1;
17     unsigned char aa :1;
18     unsigned char opcode :4;
19     unsigned char qr :1;
20     unsigned char rcode :4;
21     unsigned char cd :1;
22     unsigned char ad :1;
```

Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Run

Output

Clear

```
22     unsigned char ad :1;
23     unsigned char z :1;
24     unsigned char ra :1;
25     unsigned short q_count;
26     unsigned short ans_count;
27     unsigned short auth_count;
28     unsigned short add_count;
29 };
30
31 // DNS question structure
32 struct QUESTION {
33     unsigned short qtype;
34     unsigned short qclass;
35 };
36
37 // DNS resource data structure
38 struct R_DATA {
39     unsigned short type;
40     unsigned short _class;
41     unsigned int ttl;
42     unsigned short data_len;
43 }
```

^ Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Run

Output

Clear

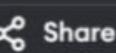
```
    unsigned short data_len;
43  };
44
45 // Structure of a response record
46 struct RES_RECORD {
47     unsigned char *name;
48     struct R_DATA *resource;
49     unsigned char *rdata;
50 };
51
52 // Convert hostname into DNS format
53 void ChangeToDnsNameFormat(unsigned char *dns, const char
    *host) {
54     char host_copy[256]; // Safe buffer for modification
55     strcpy(host_copy, host);
56     strcat(host_copy, ".");
57
58     int lock = 0;
59     for (int i = 0; i < strlen(host_copy); i++) {
60         if (host_copy[i] == '.') {
61             *dns++ = i - lock;
62             for (; lock < i; lock++) {
63                 *dns++ = '.';
64             }
65         }
66     }
67 }
```

^ Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Run

Output

Clear

```
51     "DNS++ = i - LOCK,  
52     for (; lock < i; lock++) {  
53         *dns++ = host_copy[lock];  
54     }  
55     lock++;  
56 }  
57 }  
58 *dns = '\0';  
59 }  
60  
61 // Perform a DNS query  
62 void PerformDnsQuery(const char *host) {  
63     unsigned char buf[BUFFER_SIZE], *qname;  
64     struct sockaddr_in dest;  
65     struct DNS_HEADER *dns;  
66     struct QUESTION *qinfo;  
67  
68     // Create UDP socket  
69     int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);  
70     if (sock < 0) {  
71         perror("Socket creation failed");  
72         return;
```

^ Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Run

Output

Clear

```
92         return;
93     }
94
95     // Setup DNS query packet
96     memset(buf, 0, BUFFER_SIZE);
97     dns = (struct DNS_HEADER *)buf;
98     dns->id = (unsigned short)htonl(getpid());
99     dns->qr = 0; // Query
100    dns->opcode = 0; // Standard query
101    dns->aa = 0;
102    dns->tc = 0;
103    dns->rd = 1; // Recursion desired
104    dns->ra = 0;
105    dns->z = 0;
106    dns->ad = 0;
107    dns->cd = 0;
108    dns->rcode = 0;
109    dns->q_count = htons(1); // One question
110
111    // Setup query name
112    qname = &buf[sizeof(struct DNS_HEADER)];
113    ChangeToDnsNameFormat(qname, host);
```

Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Run

Output

Clear

```
111 // Setup query name
112 qname = &buf[sizeof(struct DNS_HEADER)];
113 ChangeToDnsNameFormat(qname, host);
114
115 // Setup query type and class
116 qinfo = (struct QUESTION *)&buf[sizeof(struct DNS_HEADER)
117     + (strlen((const char *)qname) + 1)];
118 qinfo->qtype = htons(1); // Type A query
119 qinfo->qclass = htons(1); // Class IN
120
121 // Send the query
122 if (sendto(sock, buf, sizeof(struct DNS_HEADER) + (strlen
123     ((const char *)qname) + 1) + sizeof(struct QUESTION),
124     0,
125     (struct sockaddr *)&dest, sizeof(dest)) < 0) {
126     perror("Query sending failed");
127     close(sock);
128     return;
129 }
130
131 printf("Sent query for: %s\n", host);
132 }
```

^ Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Share

Run

Output

Clear

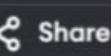
```
128     printf("Sent query for: %s\n", host);
129
130     // Receive the response
131     socklen_t addr_len = sizeof(dest);
132     int n = recvfrom(sock, buf, BUFFER_SIZE, 0, (struct
133         sockaddr *)dest, &addr_len);
134     if (n < 0) {
135         perror("Response reception failed");
136         close(sock);
137         return;
138     }
139
140     dns = (struct DNS_HEADER *)buf;
141     unsigned char *reader = &buf[sizeof(struct DNS_HEADER) +
142         (strlen((const char *)qname) + 1) + sizeof(struct
143             QUESTION)];
144
145     printf("Resolved IP Addresses:\n");
146     // Read each answer record
147     for (int i = 0; i < ntohs(dns->ans_count); i++) {
148         struct RES_RECORD answer;
```

^ Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Run

Output

Clear

```
...
145     for (int i = 0; i < ntohs(dns->ans_count); i++) {
146         struct RES_RECORD answer;
147         answer.name = reader;
148         reader += 2; // Move past name
149
150         answer.resource = (struct R_DATA *)reader;
151         reader += sizeof(struct R_DATA);
152
153         if (ntohs(answer.resource->type) == 1) { // IPv4
154             address
155             answer.rdata = (unsigned char *)malloc(ntohs
156                                         (answer.resource->data_len));
157             if (!answer.rdata) {
158                 fprintf(stderr, "Memory allocation failed\n");
159                 close(sock);
160                 return;
161             }
162             memcpy(answer.rdata, reader, ntohs(answer.resource
163                                         ->data_len));
164             reader += ntohs(answer.resource->data_len);
165 }
```

^ Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===



main.c



Run

Output

Clear

```
160
161     memcpy(answer.rdata, reader, ntohs(answer.resource
162             ->data_len));
163     reader += ntohs(answer.resource->data_len);
164
165     // Print the resolved IP address
166     printf("%d.%d.%d.%d\n", answer.rdata[0], answer
167             .rdata[1], answer.rdata[2], answer.rdata[3]);
168
169     free(answer.rdata);
170 }
171 close(sock);
172 }
173
174 int main(int argc, char *argv[]) {
175     if (argc < 2) {
176         printf("Usage: %s <hostname>\n", argv[0]);
177         return 1;
178     }
179 }
```

^ Usage: /tmp/gyme5pnJbB/main.o <hostname>

==== Code Exited With Errors ===

29)

The image shows a Java code editor interface with two tabs open. Both tabs are titled "TCPEcho.java".

**Server Side (Top Tab):**

```
1- import java.io.*;
2 import java.net.*;
3 import java.util.Scanner;
4
5- public class TCPEcho {
6
7     // TCP Echo Server
8-     public static void startServer(int port) {
9-         try (ServerSocket serverSocket = new ServerSocket(port)) {
10            System.out.println("Echo Server is running on port " +
11                port + "...");
12            while (true) {
13                Socket clientSocket = serverSocket.accept();
14                System.out.println("Client connected: " +
15                    clientSocket.getInetAddress());
16
17                BufferedReader input = new BufferedReader(new
18                    InputStreamReader(clientSocket.getInputStream()
19                        .getOutputStream(), true));
20
21                String receivedMessage;
22                while ((receivedMessage = input.readLine()) != null)
23                {
24
25                    System.out.println("Received: " + receivedMessage
26                        );
27                    output.println("Echo: " + receivedMessage); // /
28                    // Send back the same message
29
30                }
31            }
32        } catch (IOException e) {
33            e.printStackTrace();
34        }
35    }
36}
```

**Client Side (Bottom Tab):**

```
20-     {
21         System.out.println("Received: " + receivedMessage
22             );
23         output.println("Echo: " + receivedMessage); // /
24         // Send back the same message
25
26         clientSocket.close();
27         System.out.println("Client disconnected.");
28     }
29 }
30
31 // TCP Echo Client
32 public static void startClient(String serverAddress, int port) {
33     try (Socket socket = new Socket(serverAddress, port);
34         BufferedReader userInput = new BufferedReader(new
35             InputStreamReader(System.in));
36         PrintWriter output = new PrintWriter(socket
37             .getOutputStream(), true);
38         BufferedReader input = new BufferedReader(new
39             InputStreamReader(socket.getInputStream())));
40
41     System.out.println("Connected to Echo Server. Type
42         
```

The screenshot shows a Java code editor with the file 'TCPEcho.java' open. The code is a client-side application for a TCP echo server. It includes logic to handle user input and print responses to the console. A message at the bottom of the code indicates it's for choosing between Server and Client modes.

```
39     System.out.println("Connected to Echo Server. Type
40         messages to send (type 'exit' to quit):");
41
42     String message;
43     while (true) {
44         System.out.print("You: ");
45         message = userInput.readLine();
46         if (message.equalsIgnoreCase("exit")) {
47             break;
48         }
49         output.println(message);
50         System.out.println("Server: " + input.readLine()); // Read echoed response
51
52     System.out.println("Closing connection...");
53 } catch (IOException e) {
54     e.printStackTrace();
55 }
56 }
57 // Main method to choose between Server and Client
58 public static void main(String[] args) {
59     Scanner scanner = new Scanner(System.in);
60     System.out.println("Choose mode: 1) Server 2) Client");
61     int choice = scanner.nextInt();
62 }
```

The screenshot shows a Java code editor with the file 'TCPEcho.java' open. The code is a server-side application for a TCP echo server. It includes logic to start the server and handle client connections. A message at the bottom of the code indicates it's for choosing between Server and Client modes.

```
52     System.out.println("Closing connection...");
53 } catch (IOException e) {
54     e.printStackTrace();
55 }
56 }
57 // Main method to choose between Server and Client
58 public static void main(String[] args) {
59     Scanner scanner = new Scanner(System.in);
60     System.out.println("Choose mode: 1) Server 2) Client");
61     int choice = scanner.nextInt();
62
63     if (choice == 1) {
64         startServer(5000);
65     } else if (choice == 2) {
66         System.out.print("Enter Server IP (localhost for local
67             testing): ");
68         String serverAddress = scanner.next();
69         startClient(serverAddress, 5000);
70     } else {
71         System.out.println("Invalid choice.");
72     }
73     scanner.close();
74 }
75 }
76 }
```

30) server

The screenshot shows a C programming environment on the Programiz platform. The code is a simple TCP server in C, designed to handle client connections and echo back the received data.

```
main.c

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7
8 #define PORT 8080
9 #define BUFFER_SIZE 1024
10
11 void handle_client(int client_socket) {
12     char buffer[BUFFER_SIZE];
13     int bytes_read;
14
15     while (1) {
16         memset(buffer, 0, BUFFER_SIZE);
17         bytes_read = recv(client_socket, buffer, BUFFER_SIZE, 0
18                         );
19         if (bytes_read <= 0) {
20             printf("Client disconnected.\n");
21             break;
22         }
23
24         printf("Client: %s\n", buffer);
25         send(client_socket, buffer, bytes_read, 0);
26     }
27
28     close(client_socket);
29 }
30
31 int main() {
32     int server_socket, client_socket;
33     struct sockaddr_in server_addr, client_addr;
34     socklen_t client_len = sizeof(client_addr);
35
36     // Create socket
37     server_socket = socket(AF_INET, SOCK_STREAM, 0);
38     if (server_socket == -1) {
39         perror("Failed to create socket");
40     }
41
42     bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
43
44     listen(server_socket, 5);
45
46     while (1) {
47         client_socket = accept(server_socket, (struct sockaddr *)NULL, NULL);
48
49         if (client_socket < 0) {
50             perror("Failed to accept connection");
51             continue;
52         }
53
54         handle_client(client_socket);
55     }
56 }
```

The output window displays the message "Server is listening on port 8080..." twice, indicating that the server has successfully started and is waiting for incoming connections.

main.c

```
39     perror("Failed to create socket");
40     exit(EXIT_FAILURE);
41 }
42
43 // Define server address
44 server_addr.sin_family = AF_INET;
45 server_addr.sin_addr.s_addr = INADDR_ANY;
46 server_addr.sin_port = htons(PORT);
47
48 // Bind the socket to the specified IP and port
49 if (bind(server_socket, (struct sockaddr*)&server_addr,
50         sizeof(server_addr)) < 0) {
51     perror("Bind failed");
52     close(server_socket);
53     exit(EXIT_FAILURE);
54 }
55 // Listen for incoming connections
56 if (listen(server_socket, 3) < 0) {
57     perror("Listen failed");
58 }
```

main.c

```
57     perror("Listen failed");
58     close(server_socket);
59     exit(EXIT_FAILURE);
60 }
61
62 printf("Server is listening on port %d...\n", PORT);
63
64 while (1) {
65     // Accept a new connection
66     client_socket = accept(server_socket, (struct sockaddr
67     *)&client_addr, &client_len);
68     if (client_socket < 0) {
69         perror("Accept failed");
70         close(server_socket);
71         exit(EXIT_FAILURE);
72     }
73     printf("New client connected.\n");
74     handle_client(client_socket);
75 }
```

```
63
64  while (1) {
65      // Accept a new connection
66      client_socket = accept(server_socket, (struct sockaddr
67      *)&client_addr, &client_len);
68      if (client_socket < 0) {
69          perror("Accept failed");
70          close(server_socket);
71          exit(EXIT_FAILURE);
72      }
73      printf("New client connected.\n");
74      handle_client(client_socket);
75  }
76
77  close(server_socket);
78  return 0;
79 }
80
```

## Client response

```
main.c
```

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define PORT 8080
8 #define BUFFER_SIZE 1024
9
10 int main() {
11     int client_socket;
12     struct sockaddr_in server_addr; // Fixed this line
13     char buffer[BUFFER_SIZE];
14
15     // Create socket
16     client_socket = socket(AF_INET, SOCK_STREAM, 0);
17     if (client_socket == -1) {
18         perror("Socket creation failed");
19         exit(EXIT_FAILURE);
20     }
```

Output

```
Connection failed: Connection refused
== Code Exited With Errors ==
```

main.c

```
20     }
21
22     // Define server address
23     server_addr.sin_family = AF_INET;
24     server_addr.sin_port = htons(PORT);
25     server_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // Connect to localhost
26
27     // Connect to the server
28     if (connect(client_socket, (struct sockaddr*)&server_addr,
29                  sizeof(server_addr)) < 0) {
30         perror("Connection failed");
31         close(client_socket);
32         exit(EXIT_FAILURE);
33     }
34     printf("Connected to the server.\n");
35
36     while (1) {
37         // Send message to server
```

Output

```
- Connection failed: Connection refused
== Code Exited With Errors ==
```

main.c

```
33
34     printf("Connected to the server.\n");
35
36     while (1) {
37         // Send message to server
38         printf("You: ");
39         fgets(buffer, BUFFER_SIZE, stdin);
40         send(client_socket, buffer, strlen(buffer), 0);
41
42         // Receive response from server
43         memset(buffer, 0, BUFFER_SIZE);
44         recv(client_socket, buffer, BUFFER_SIZE, 0);
45         printf("Server: %s\n", buffer);
46     }
47
48     close(client_socket);
49     return 0;
50 }
```

Output

```
- Connection failed: Connect
== Code Exited With Error
```

main.c      31

Run      Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <arpa/inet.h>
5 #include <netinet/if_ether.h>
6 #include <sys/socket.h>
7 #include <unistd.h>
8
9 #define CACHE_SIZE 10
10
11 typedef struct {
12     char ip[INET_ADDRSTRLEN];
13     unsigned char mac[ETH_ALEN];
14 } arp_cache_entry;
15
16 arp_cache_entry arp_cache[CACHE_SIZE];
17
18 void add_to_cache(const char *ip, unsigned char *mac) {
19     for (int i = 0; i < CACHE_SIZE; i++) {
20         if (strlen(arp_cache[i].ip) == 0) {
21             strcpy(arp_cache[i].ip, ip);
22             memcpy(arp_cache[i].mac, mac, ETH_ALEN);
23             break;
24         }
25     }
26 }
```

/tmp/8J6GoxASNj.o  
IP: 192.168.1.1, MAC: 00:11:22:33:44:55  
==== Code Execution Successful ===

main.c

32



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define FLAG_SEQUENCE "01111110"
6
7 // Function to perform bit stuffing
8 void bit_stuffing(const char *input, char *output) {
9     int count = 0; // Count of consecutive '1's
10    int j = 0; // Index for output array
11
12    for (int i = 0; input[i] != '\0'; i++) {
13        output[j++] = input[i]; // Copy the current bit to output
14
15        // Count consecutive '1's
16        if (input[i] == '1') {
17            count++;
18        } else {
19            count = 0; // Reset count if '0' is found
20        }
21
22        // If we have five consecutive '1's, insert a '0'
23        if (count == 5) {
24            output[j++] = '0'; // Stuff a '0'
25            count = 0; // Reset count after stuffing

```

/tmp/gudr89sSou.o

Input Data: 11111011101111101011111

Stuffed Data: 111110011101111101010111110

Destuffed Data: 111110111011111101011111

== Code Execution Successful ==

33



Run

Output

```
main.c 33
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <arpa/inet.h>
5 #include <unistd.h>
6 #define PORT 8080
7 #define BUFFER_SIZE 1024
8
9 int main() {
10     int server_fd, new_socket;
11     struct sockaddr_in address;
12     int addrlen = sizeof(address);
13     char buffer[BUFFER_SIZE] = {0};
14     FILE *received_file;
15     int bytes_received;
16     // Create socket file descriptor
17     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
18         perror("Socket failed");
19         exit(EXIT_FAILURE);
20     }
21     // Set address structure
22     // Bind the socket to the port
23     bind(server_fd, (struct sockaddr *)&address, addrlen);
24     // Listen for incoming connections
25     listen(server_fd, 5);
26     // Accept a connection
27     new_socket = accept(server_fd, (struct sockaddr *)NULL, NULL);
28     // Read from the client
29     bytes_received = read(new_socket, buffer, BUFFER_SIZE);
30     // Write to the client
31     write(new_socket, buffer, bytes_received);
32     // Close the connection
33 }
```

Server is listening on port 8080...

main.c      33 (Client)      Run      Output      Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <arpa/inet.h>
5 #include <unistd.h>
6
7 #define PORT 8080
8 #define BUFFER_SIZE 1024
9
10 int main() {
11     int sock = 0;
12     struct sockaddr_in serv_addr;
13     char buffer[BUFFER_SIZE] = {0};
14     FILE *file_to_send;
15     int bytes_read;
16
17     // Create socket
18     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
19         printf("\n Socket creation error \n");
20         return -1;
21     }
22
23     // Set server address structure
24     serv_addr.sin_family = AF_INET;
25     serv_addr.sin_port = htons(PORT);
26 }
```

/tmp/icGZYMLT01.o  
Connection Failed  
==== Code Exited With Errors ===

main.c

34



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define POLYNOMIAL 0x07
5
6 unsigned char compute_crc(const char *data) {
7     unsigned char crc = 0; // Initial CRC value
8     size_t len = strlen(data);
9
10    for (size_t i = 0; i < len; i++) {
11        crc ^= (data[i] & 0xFF);
12
13        for (int j = 0; j < 8; j++) {
14            if (crc & 0x80) {
15                crc = (crc << 1) ^ POLYNOMIAL;
16            } else {
17                crc <<= 1;
18            }
19        }
20    }
21    return crc;
22 }
23
24 void simulate_error(char *data) {
25     size_t len = strlen(data);
26     if (len > 0) {
```

```
/tmp/GJirRrSc5d.o
ERROR!
Original Data: Hello, World!
Computed CRC: 0x87
Data after error: Iello, World!
Error detected in data!
```

```
== Code Execution Successful ==
```

35

```
main.c  | 35  | Run | Output | Clear
```

```
2 #include <stdlib.h>
3 #include <stdbool.h>
4
5 #define WINDOW_SIZE 4
6 #define TOTAL_PACKETS 10
7
8 void sendPackets(int windowSize) {
9     int packetsSent = 0;
10    while (packetsSent < TOTAL_PACKETS) {
11        printf("Sending packets: ");
12        for (int i = 0; i < windowSize && packetsSent <
13            TOTAL_PACKETS; i++) {
14            printf("%d ", packetsSent + 1);
15            packetsSent++;
16        }
17        printf("\n");
18        printf("Acknowledgment received for packets up to %d\n",
19             packetsSent);
20    }
21    int main() {
22        sendPackets(WINDOW_SIZE);
23        return 0;
24    }
25 }
```

```
/tmp/DnuTpVxr6m.o
Sending packets: 1 2 3 4
Acknowledgment received for packets up to 4
Sending packets: 5 6 7 8
Acknowledgment received for packets up to 8
Sending packets: 9 10
Acknowledgment received for packets up to 10
```

```
==== Code Execution Successful ===
```