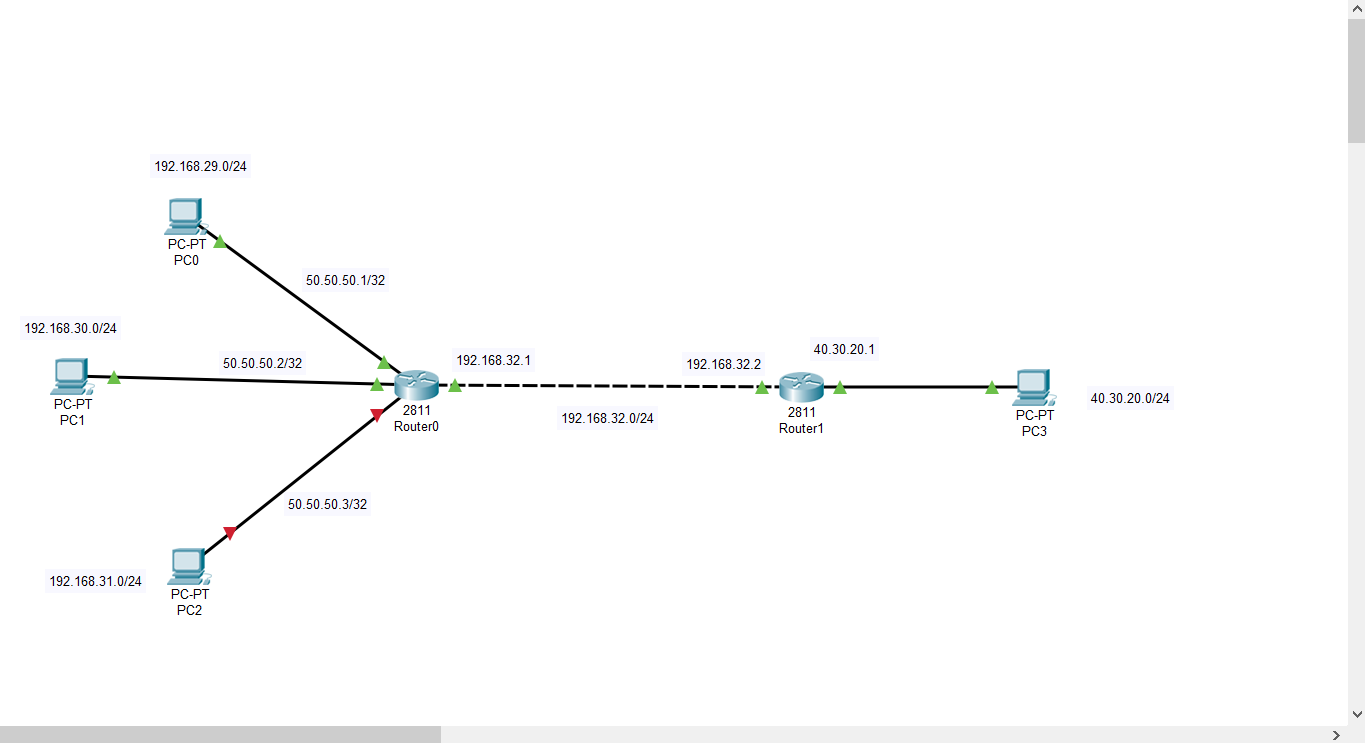
**CN\_LAB\_4\_Assignment**

**CE\_055**

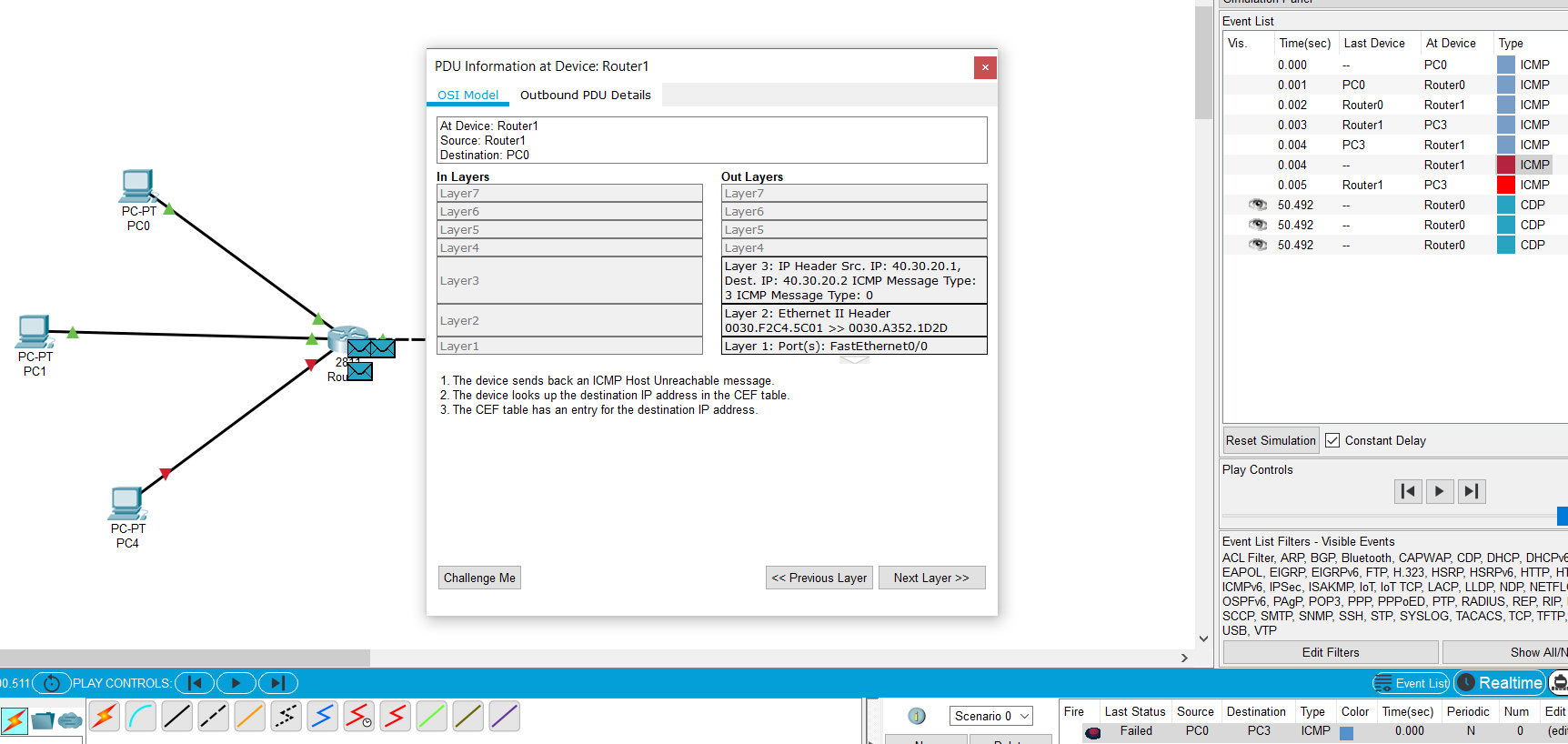
Aim:- Network address translation and implementation of dijkstra algorithm to find shortest path between routers.

1. NAT network1:-

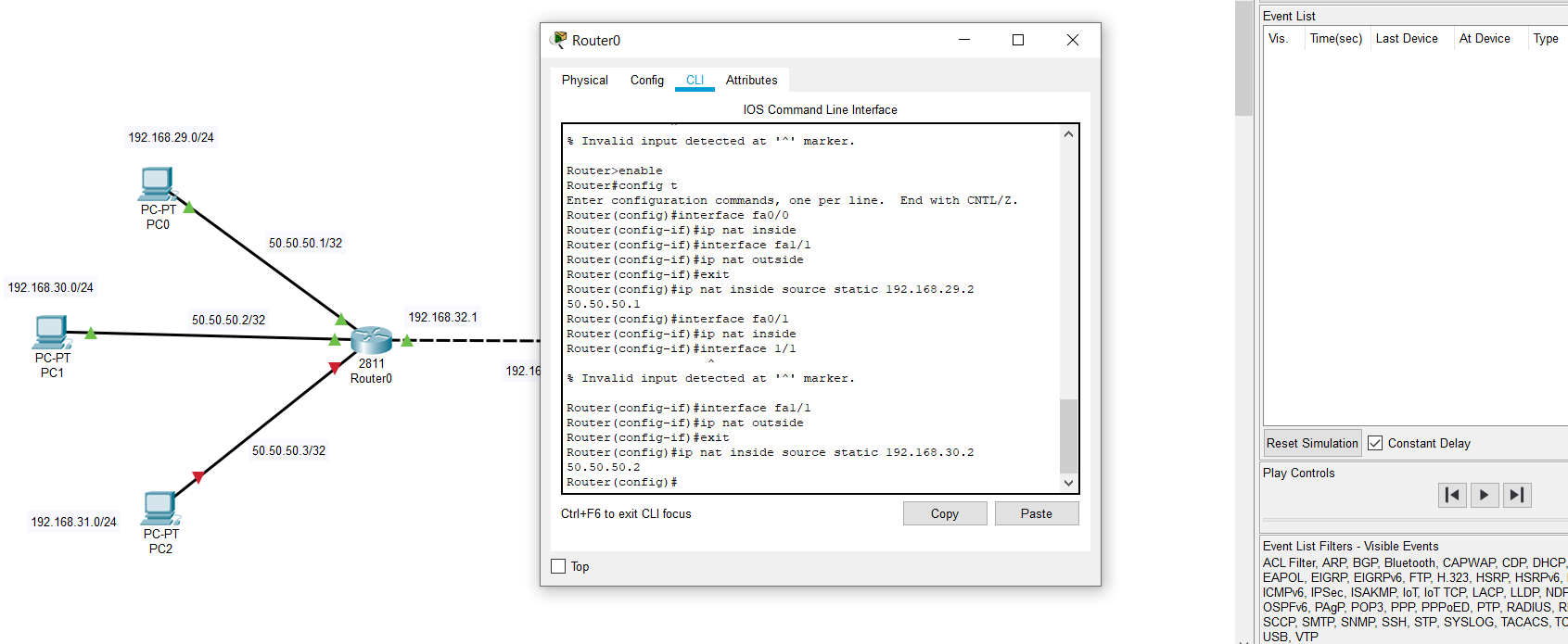
* Network Address Translation (NAT) is a process in which one or more local IP addresses are translated into one or more Global IP addresses and vice versa in order to provide Internet access to the local hosts.
* Generally, the border router is configured for NAT i.e the router which has one interface in local (inside) network and one interface in the global (outside) network. When a packet traverses outside the local (inside) network, then NAT converts that local (private) IP address to a global (public) IP address. When a packet enters the local network, the global (public) IP address is converted to a local (private) IP address.

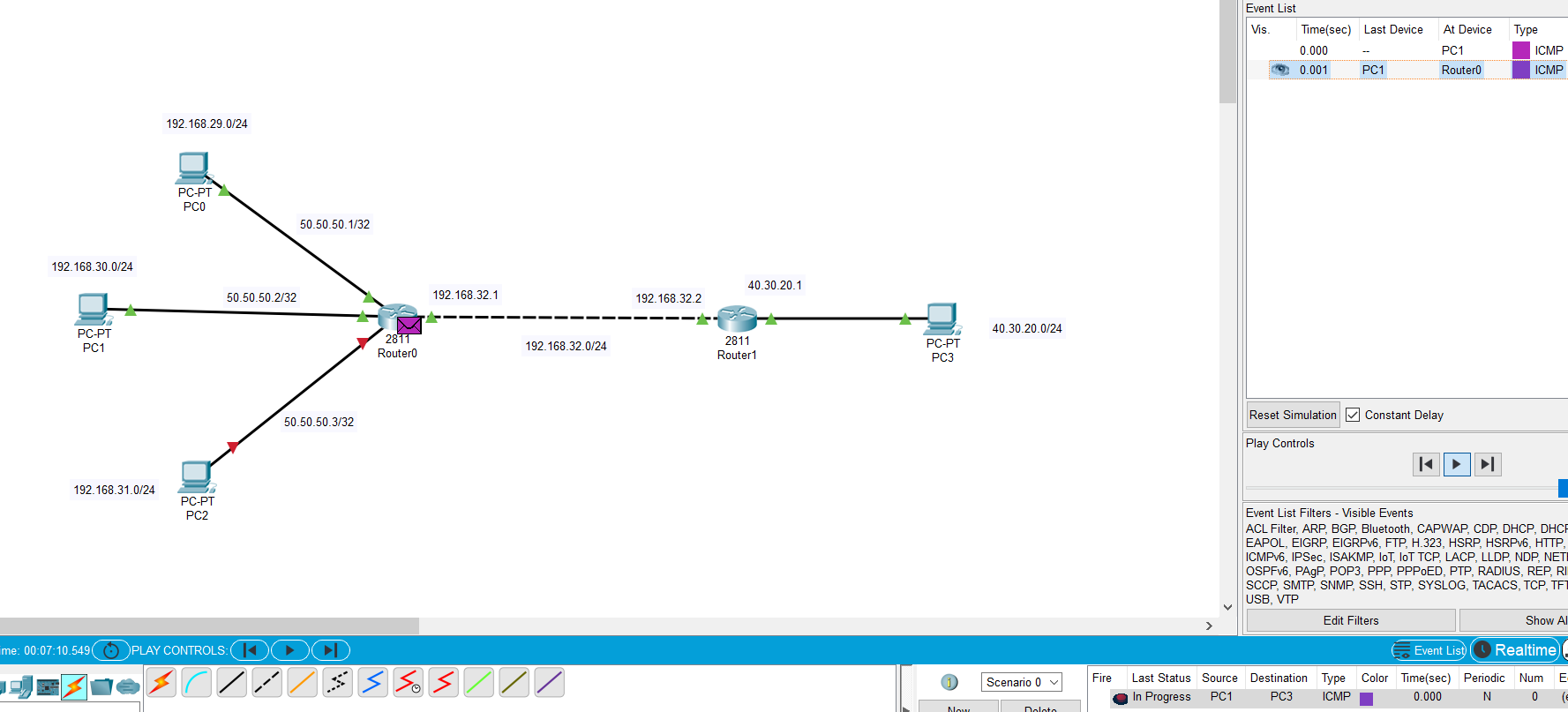


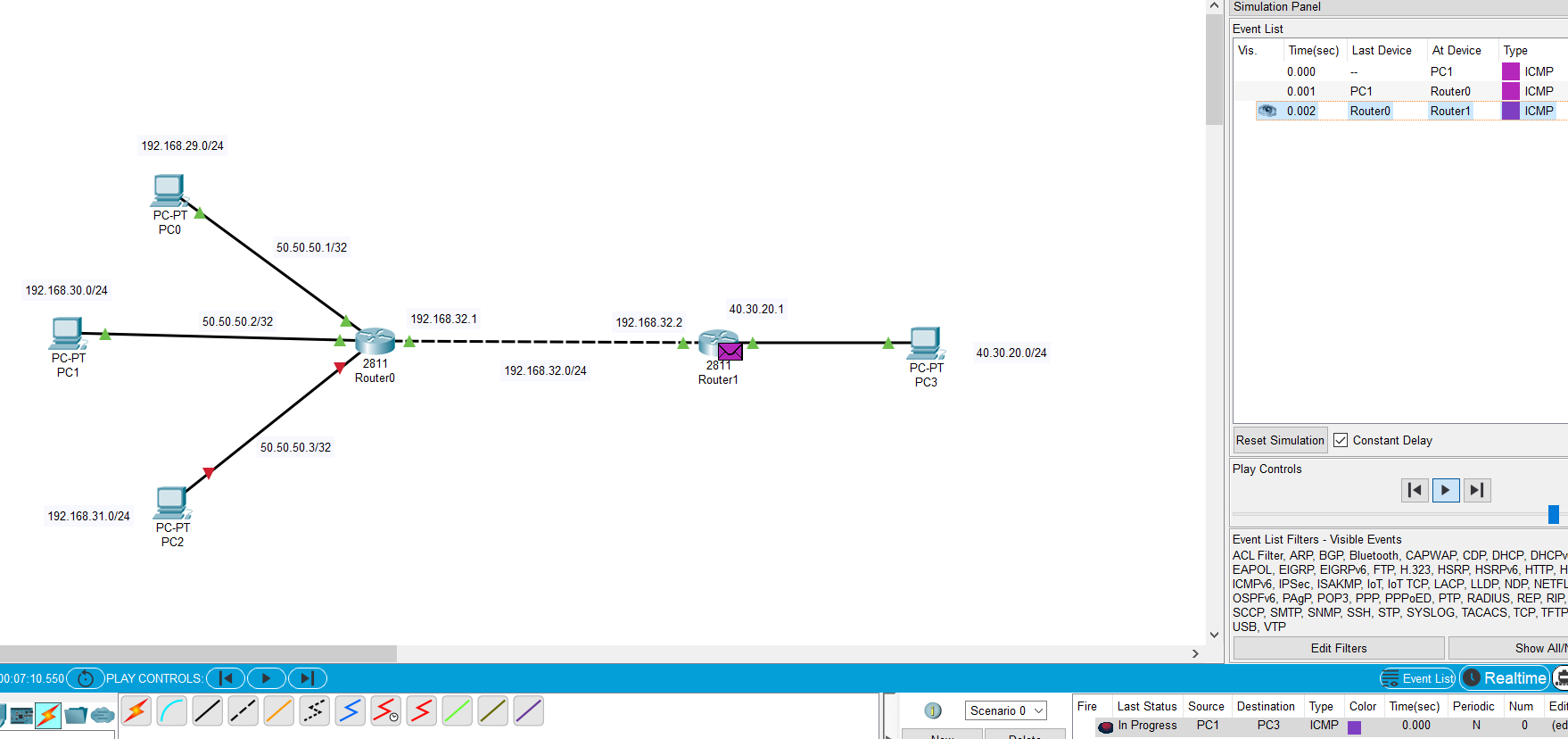
* Error while message transfer.

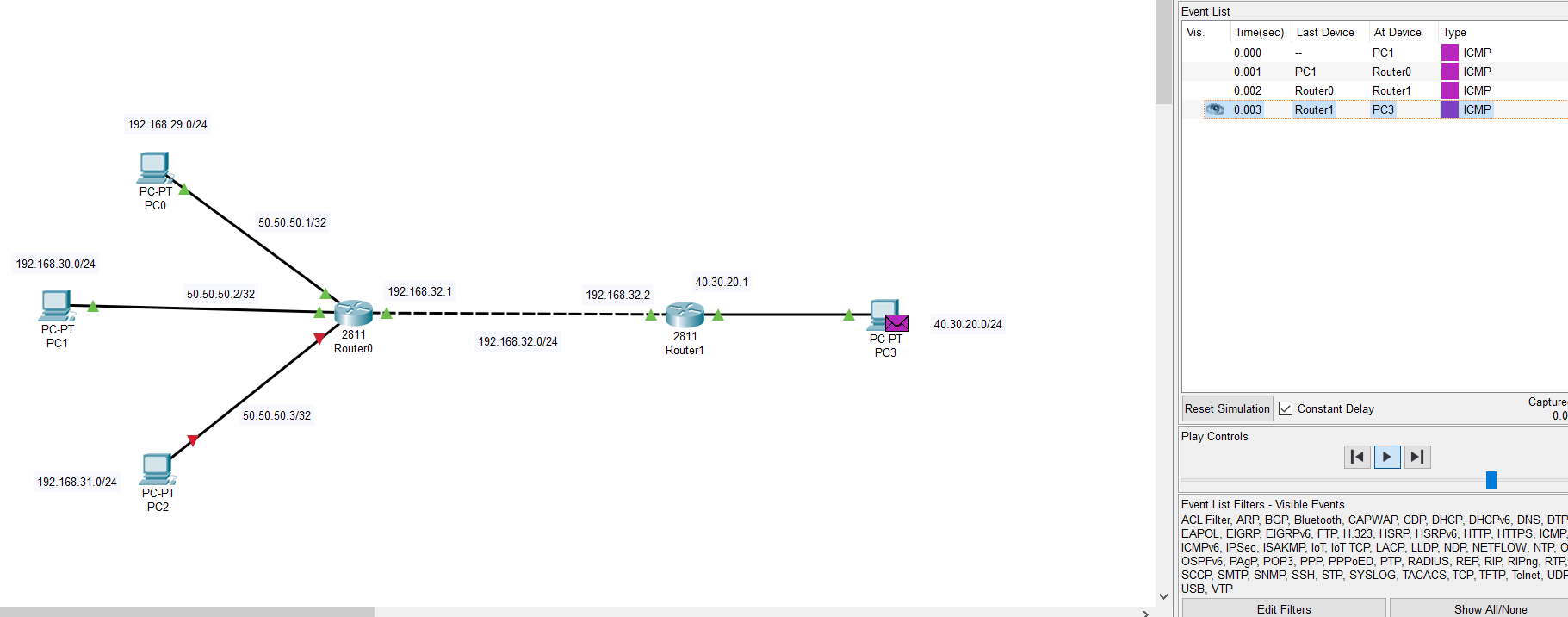


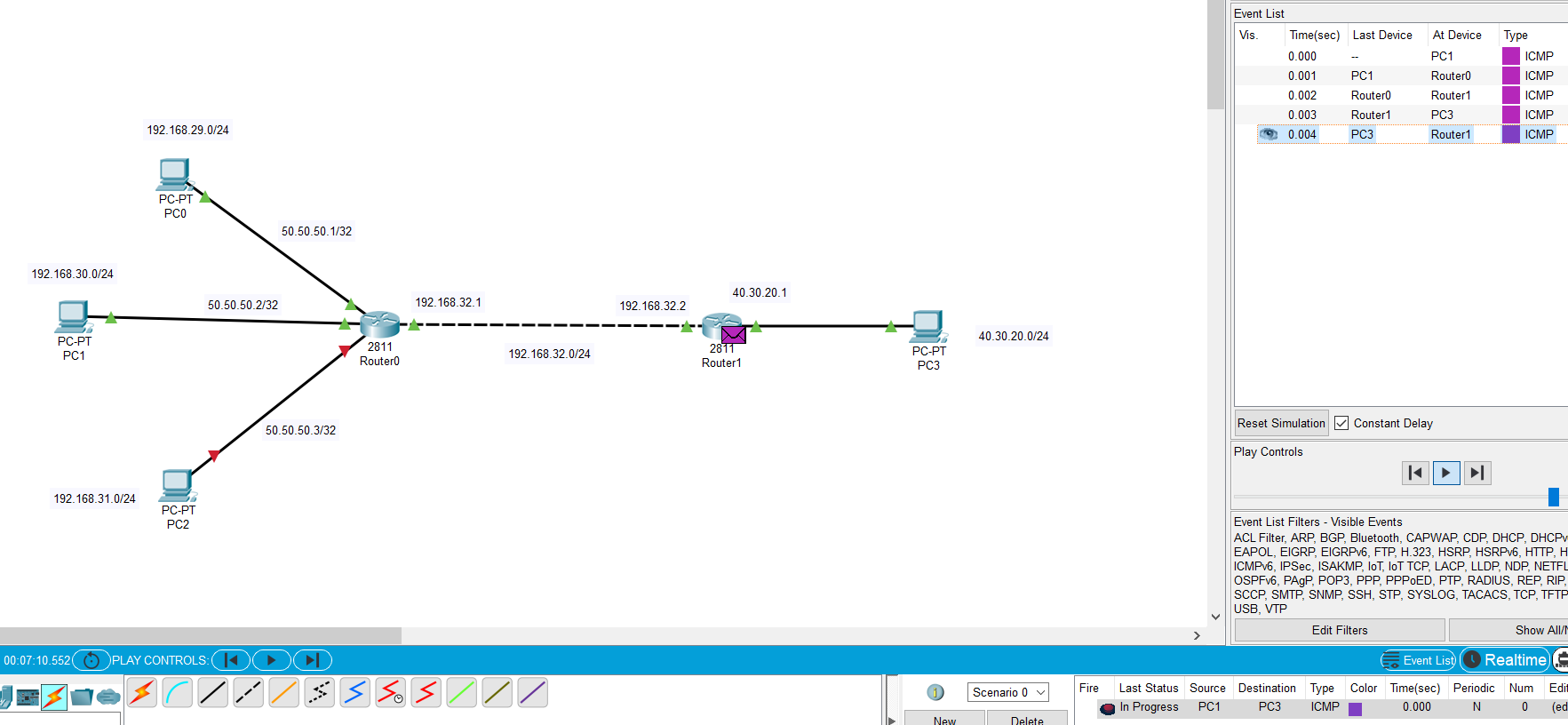
* Unreachable!
* Commands for solution and network translation:-

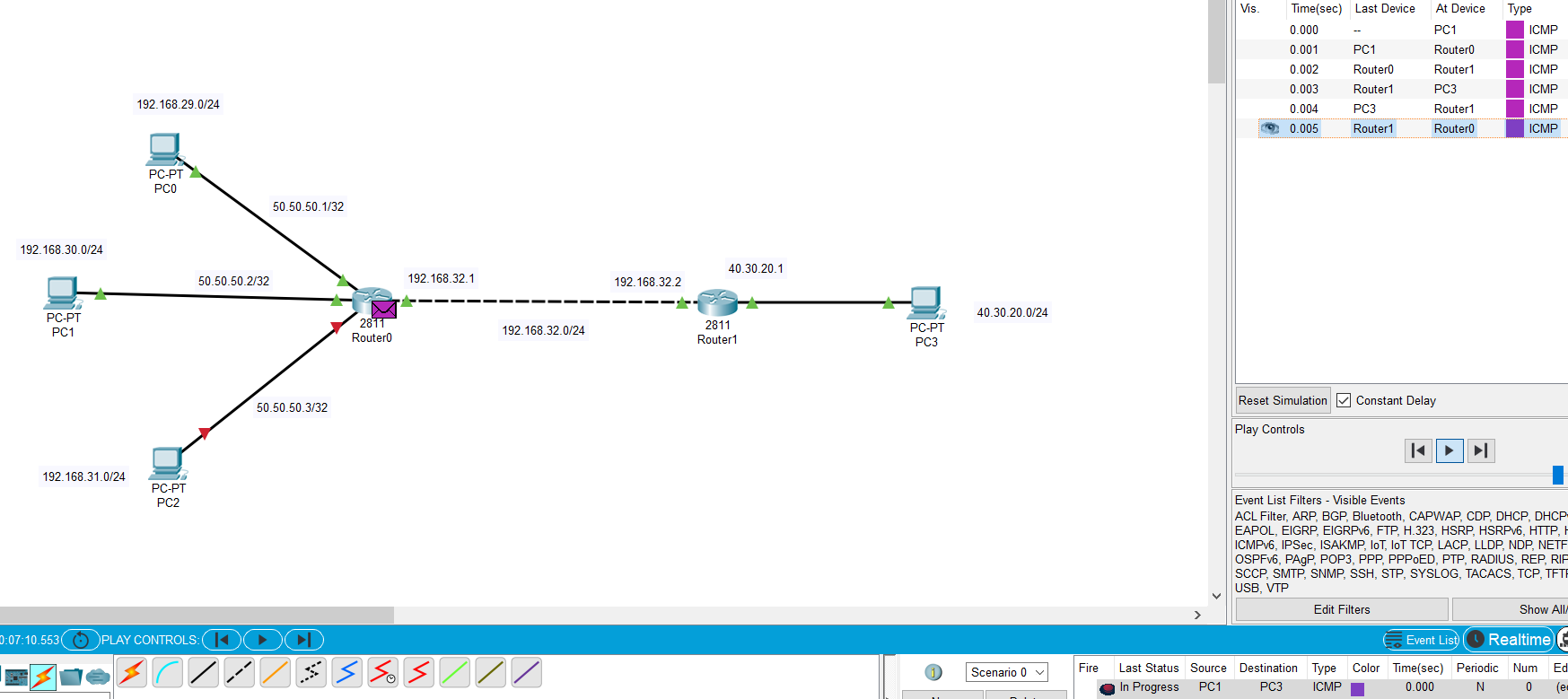


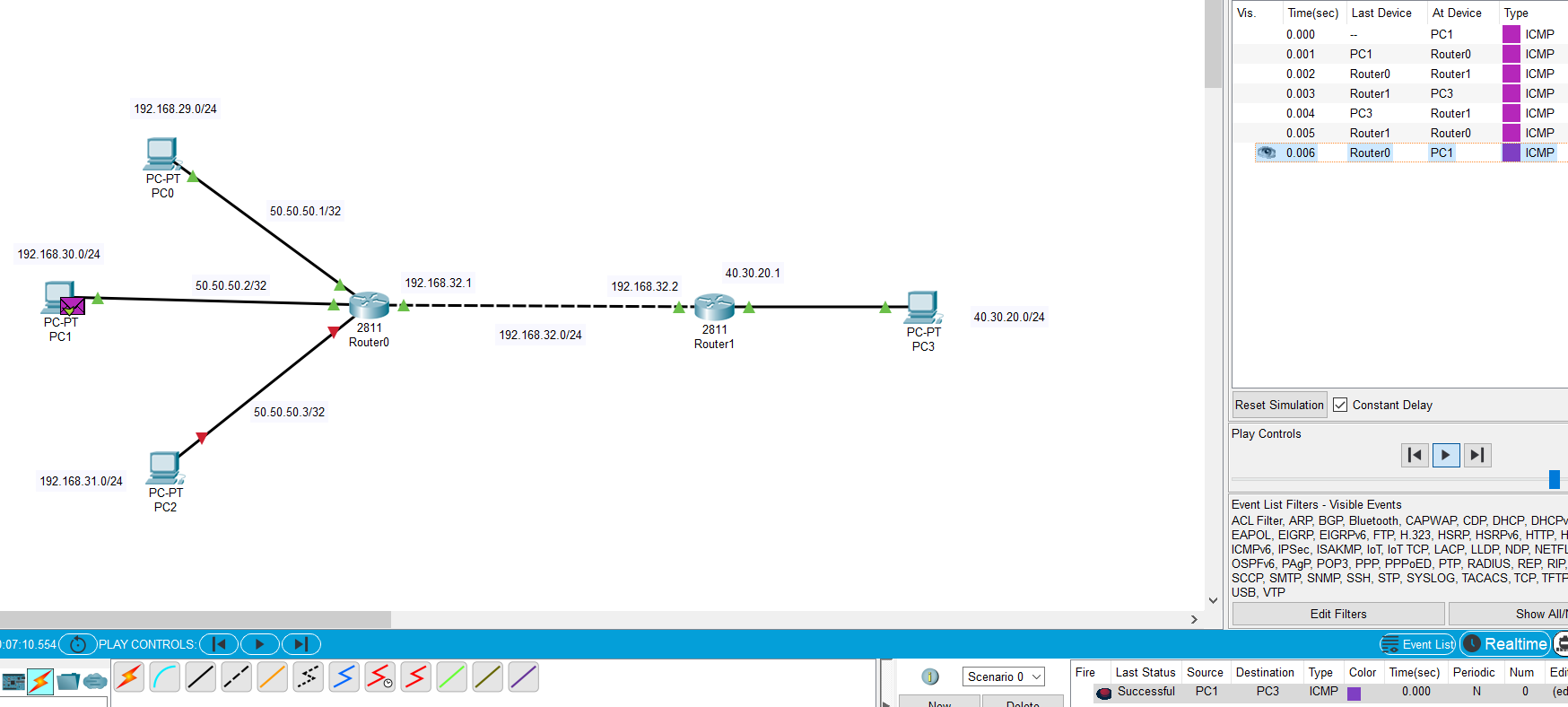




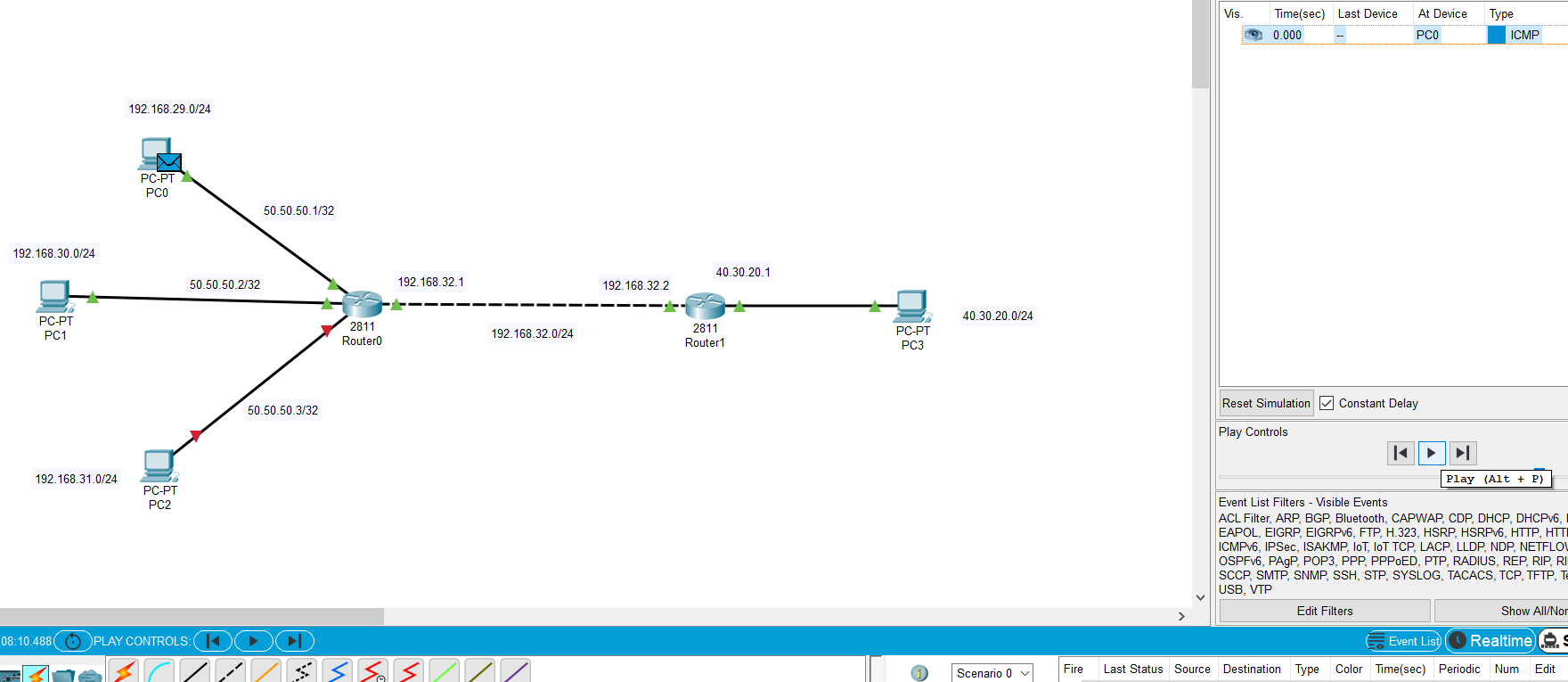


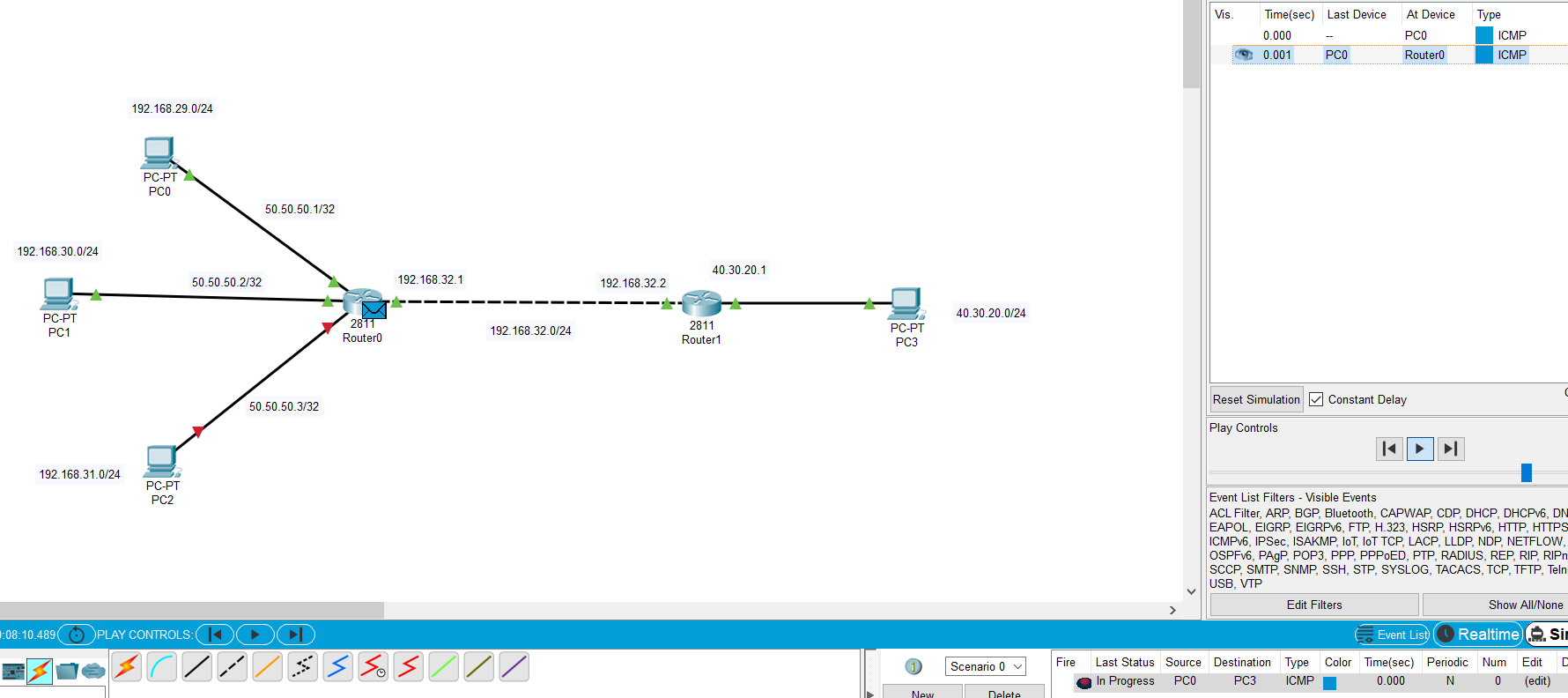


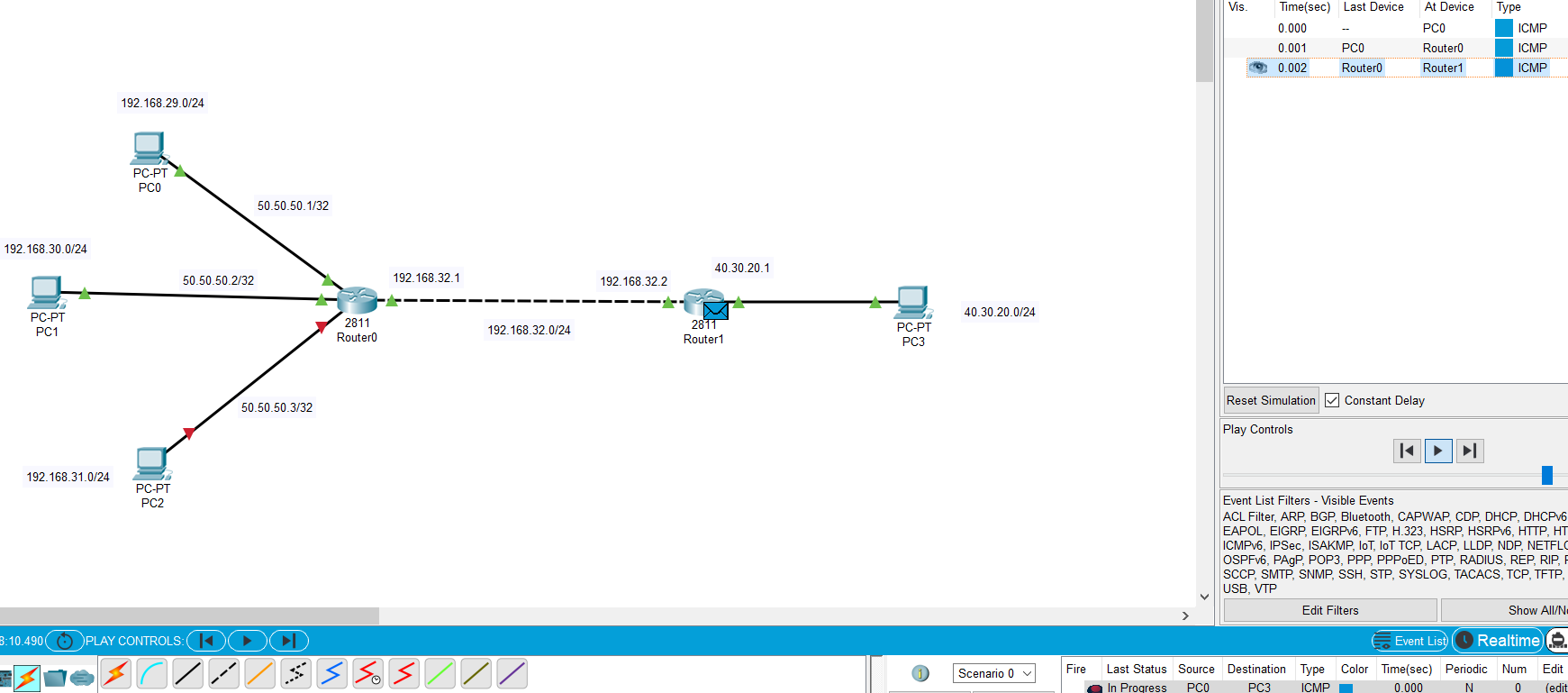


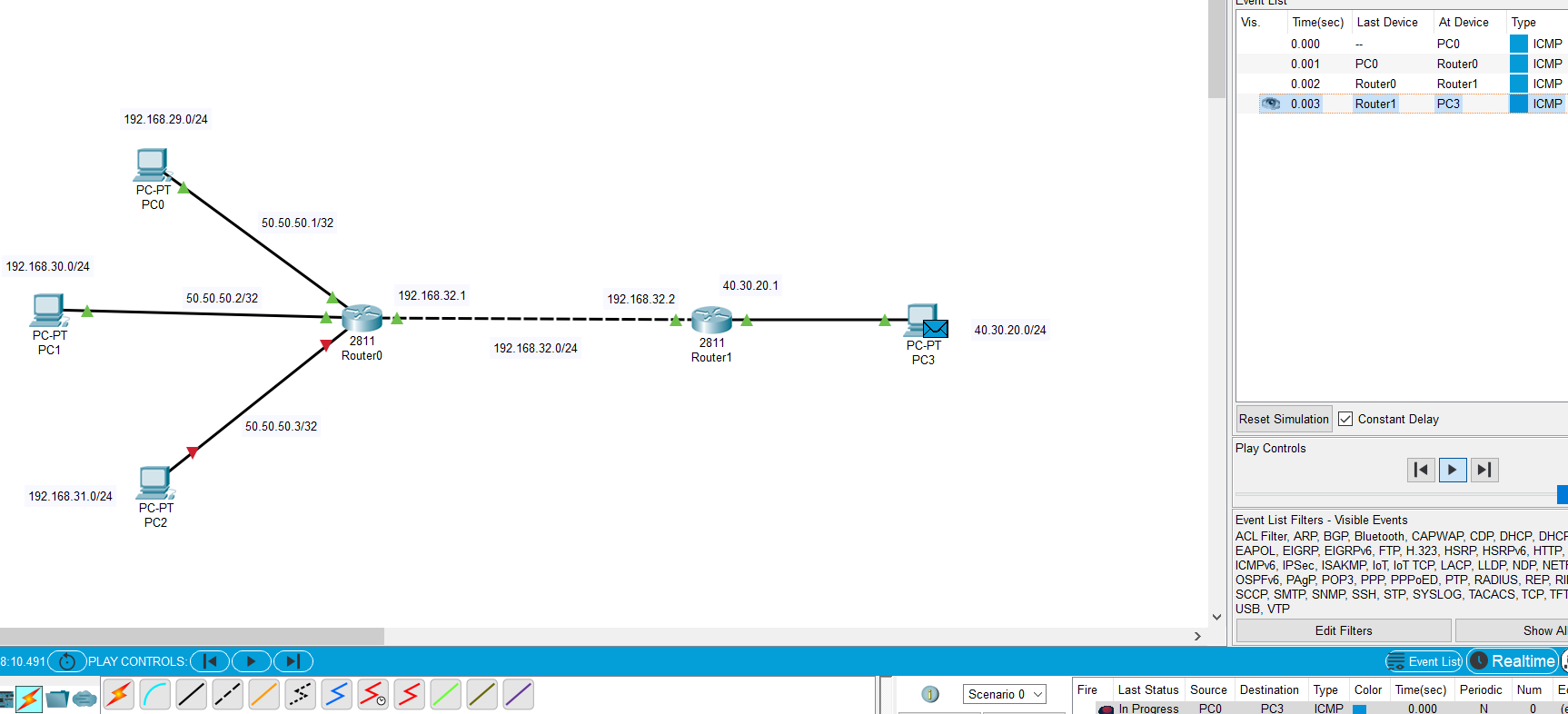


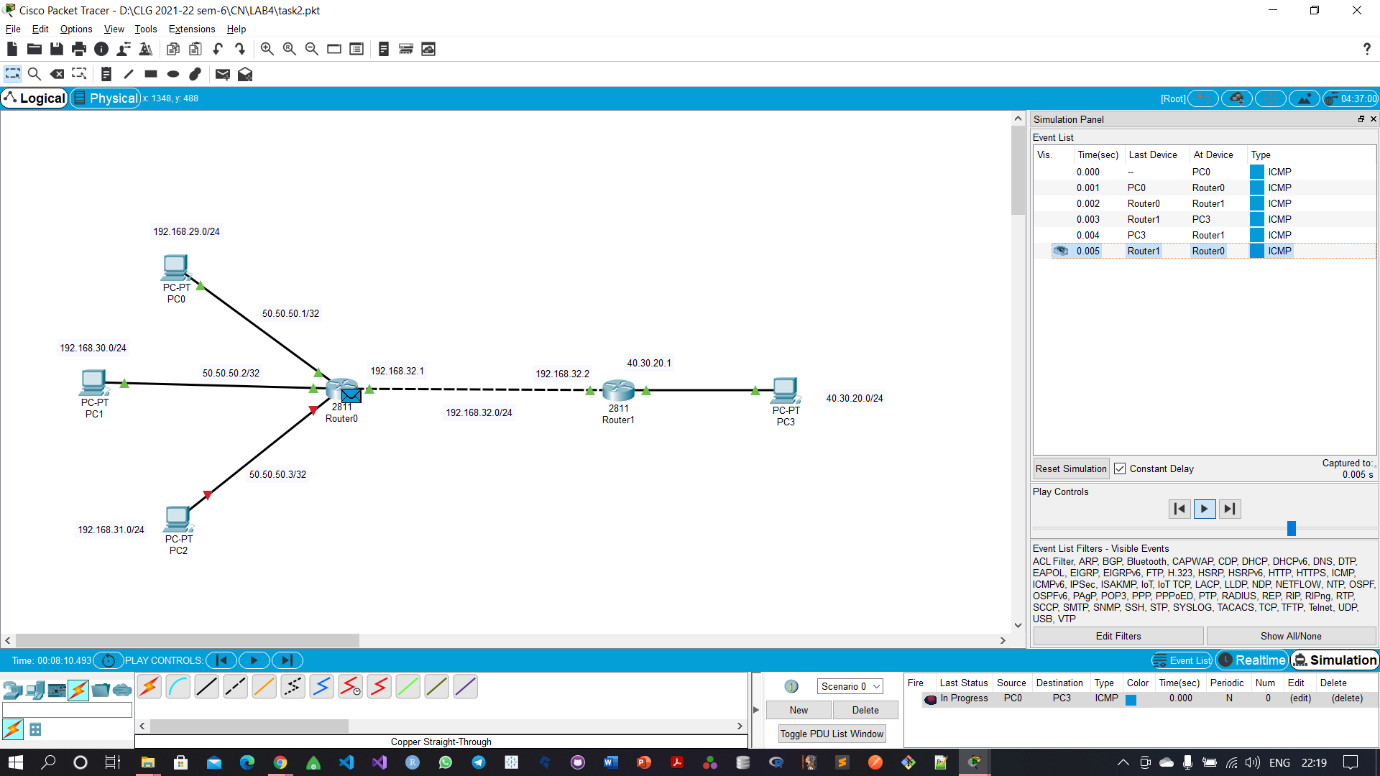
* From pc0 to pc3:-

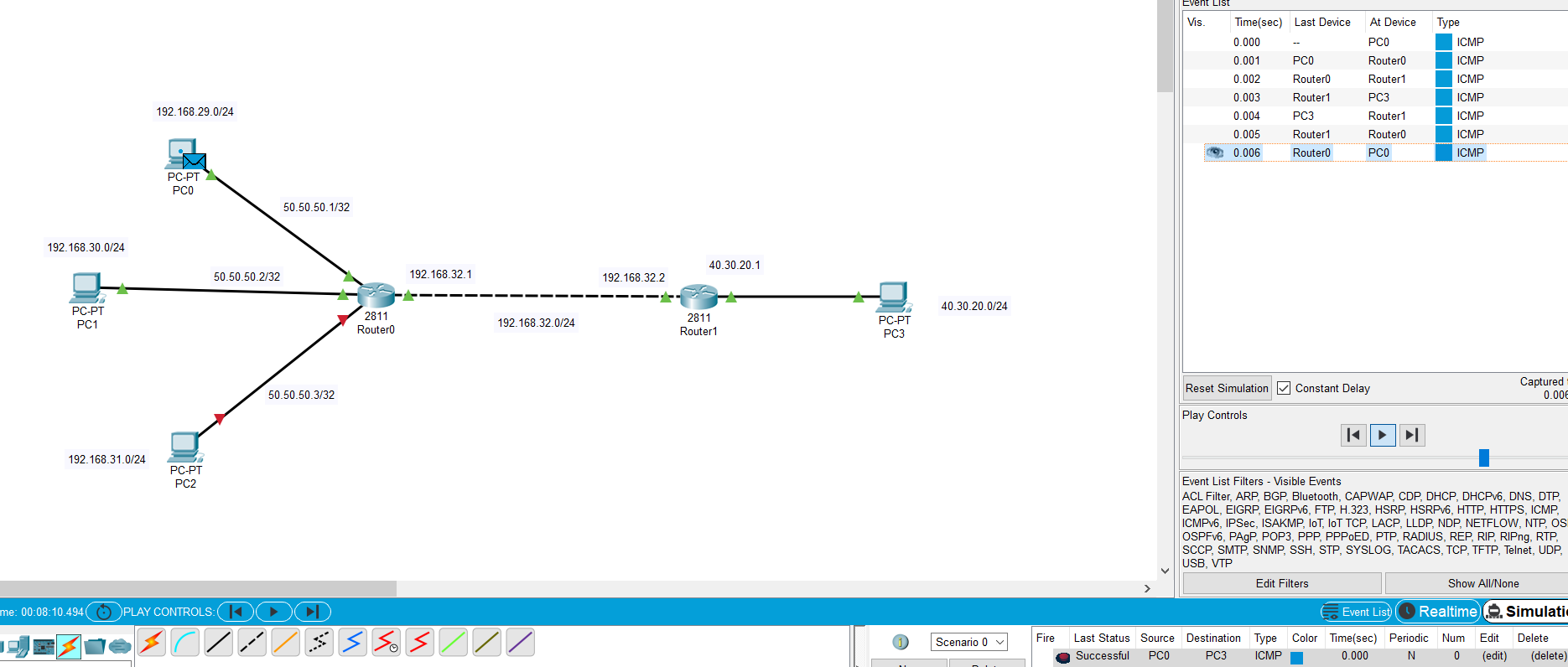




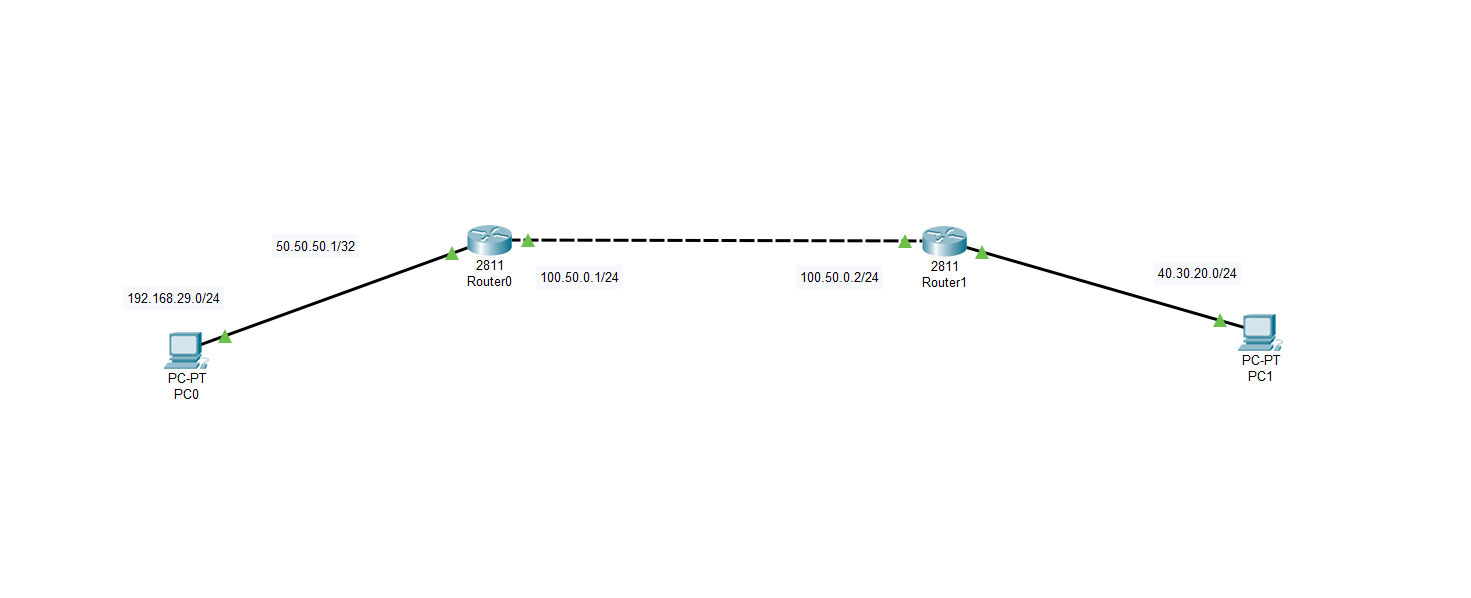


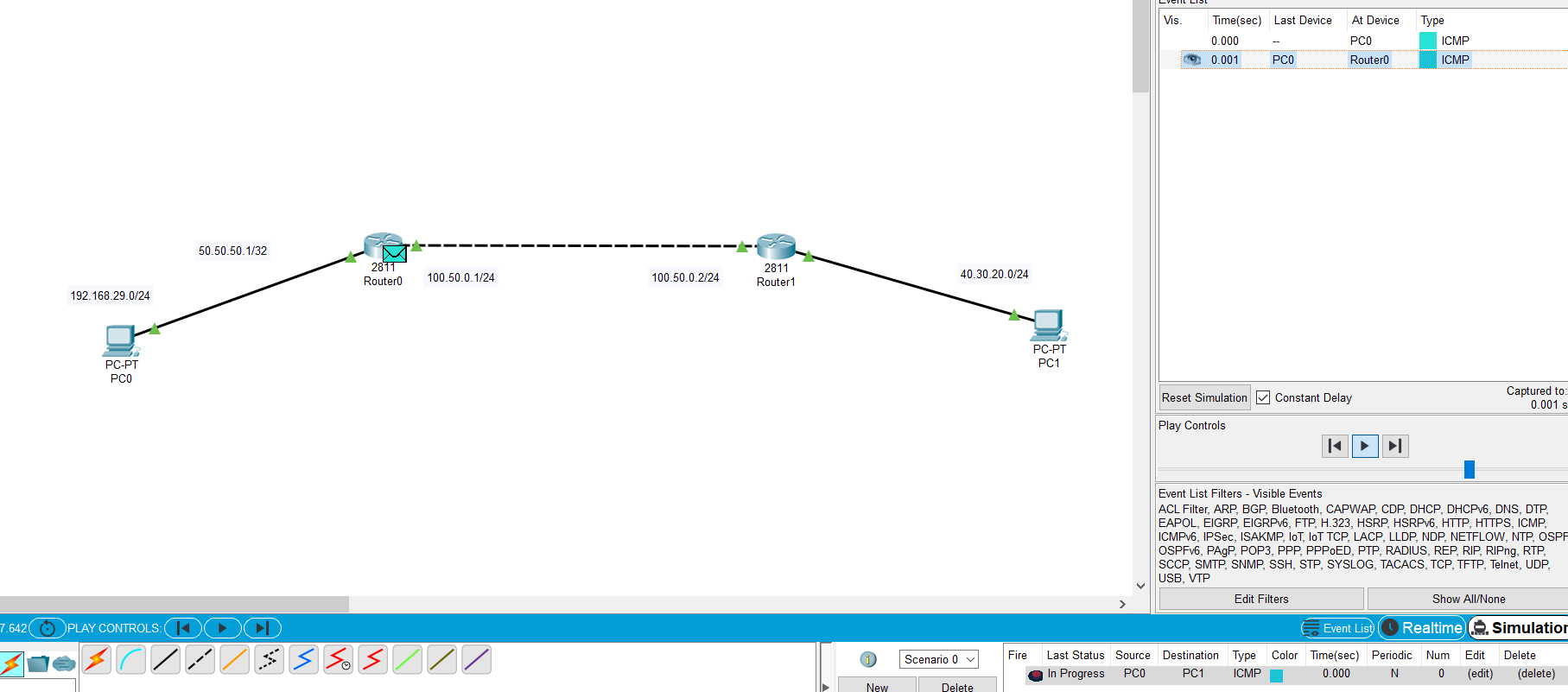


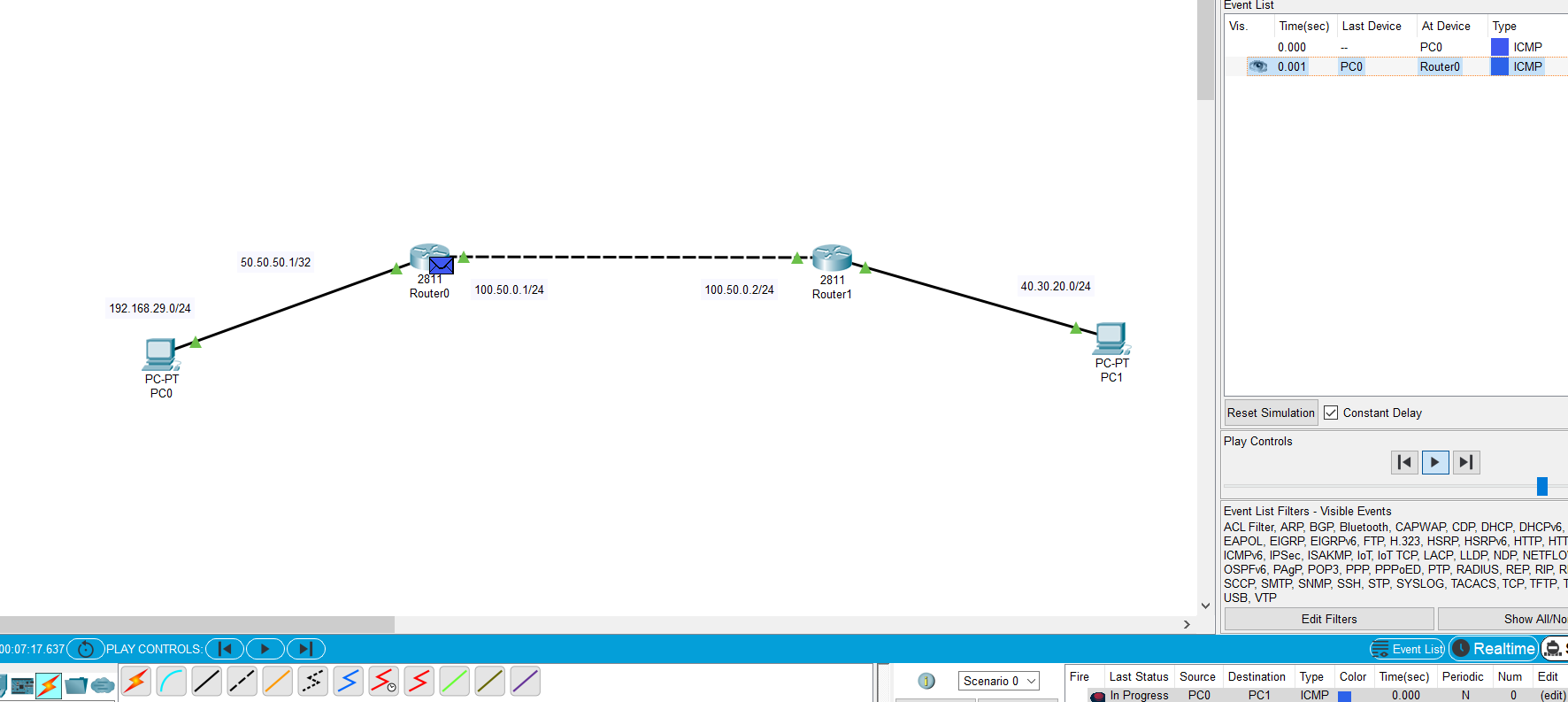


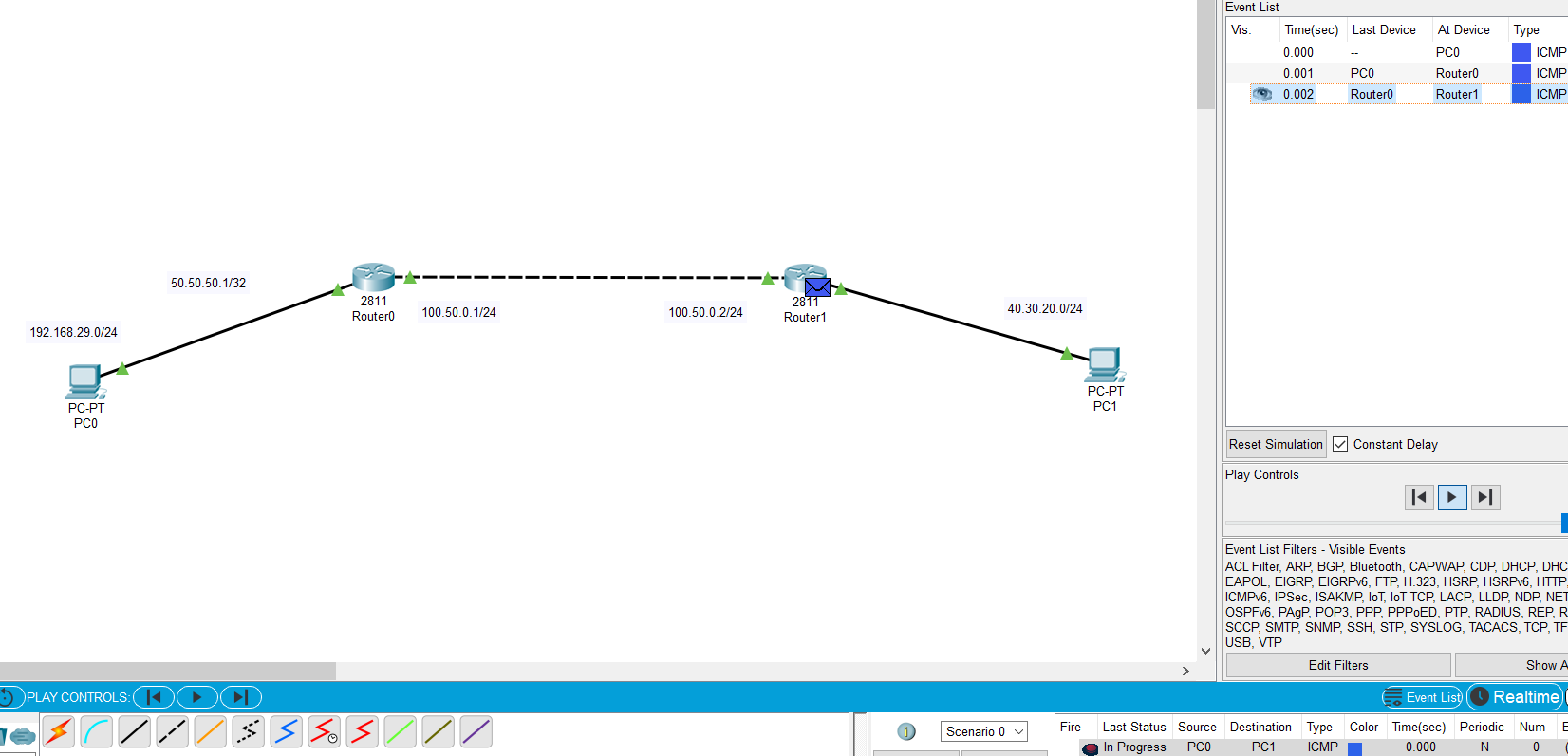


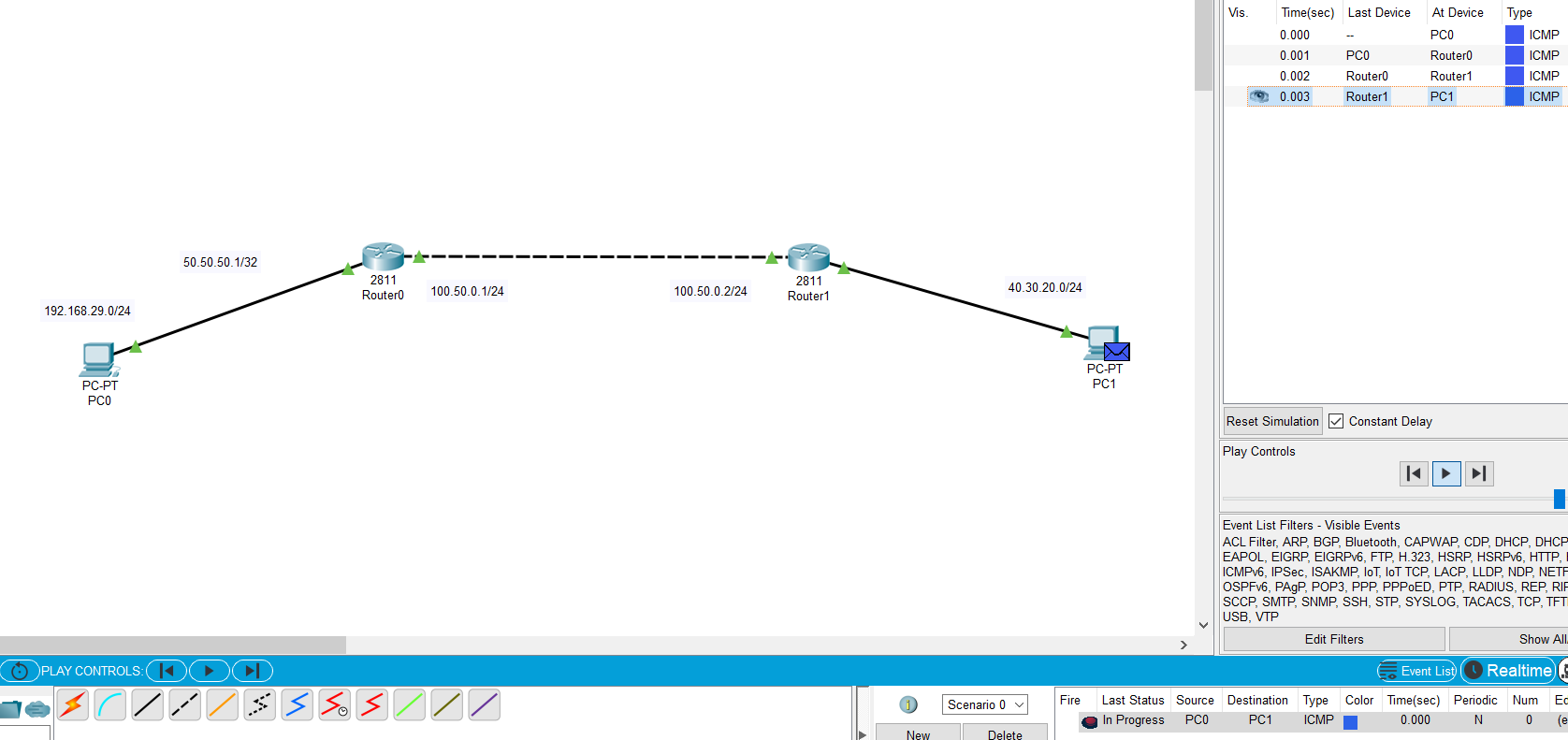
1. Network2:-

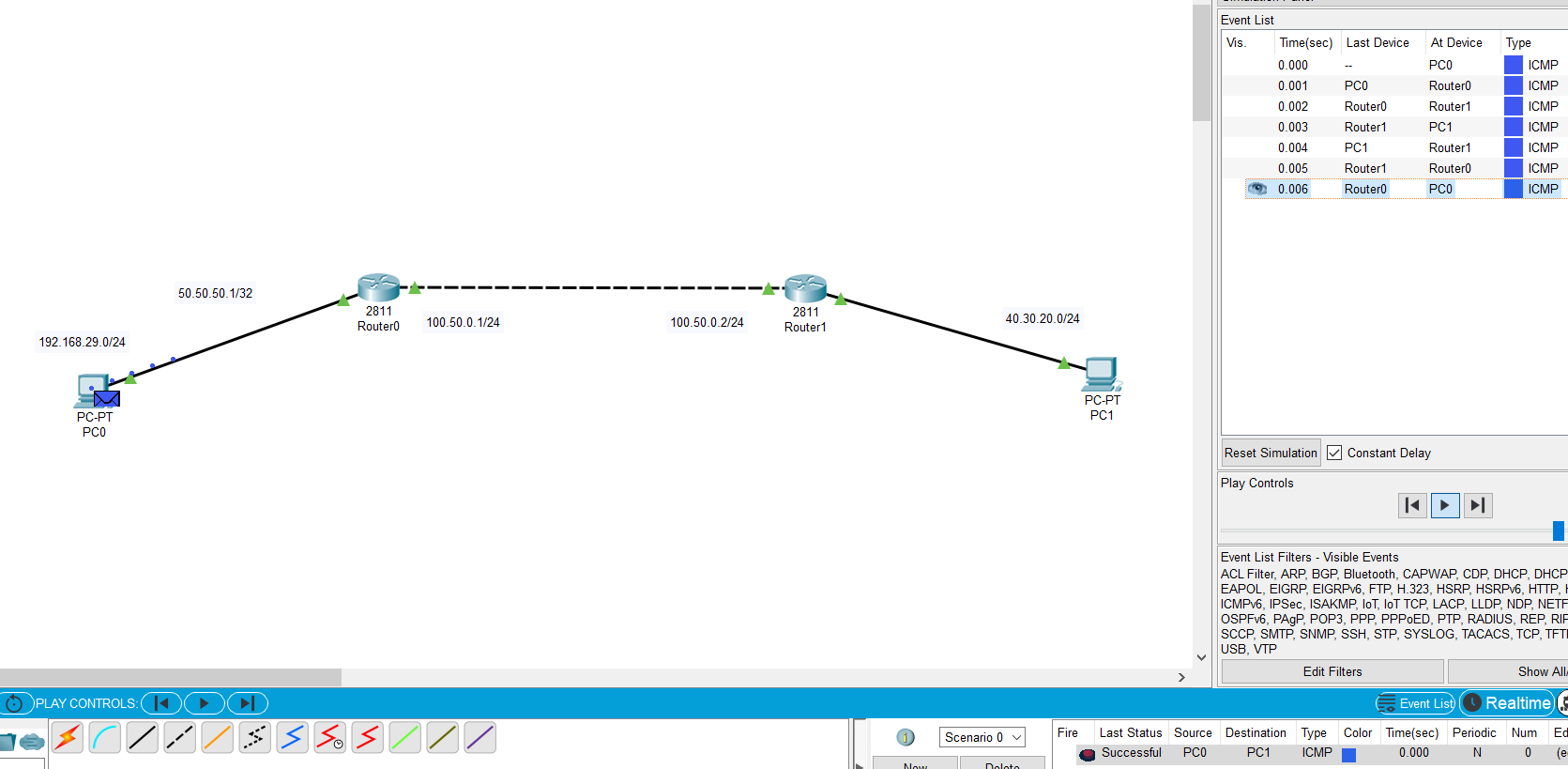




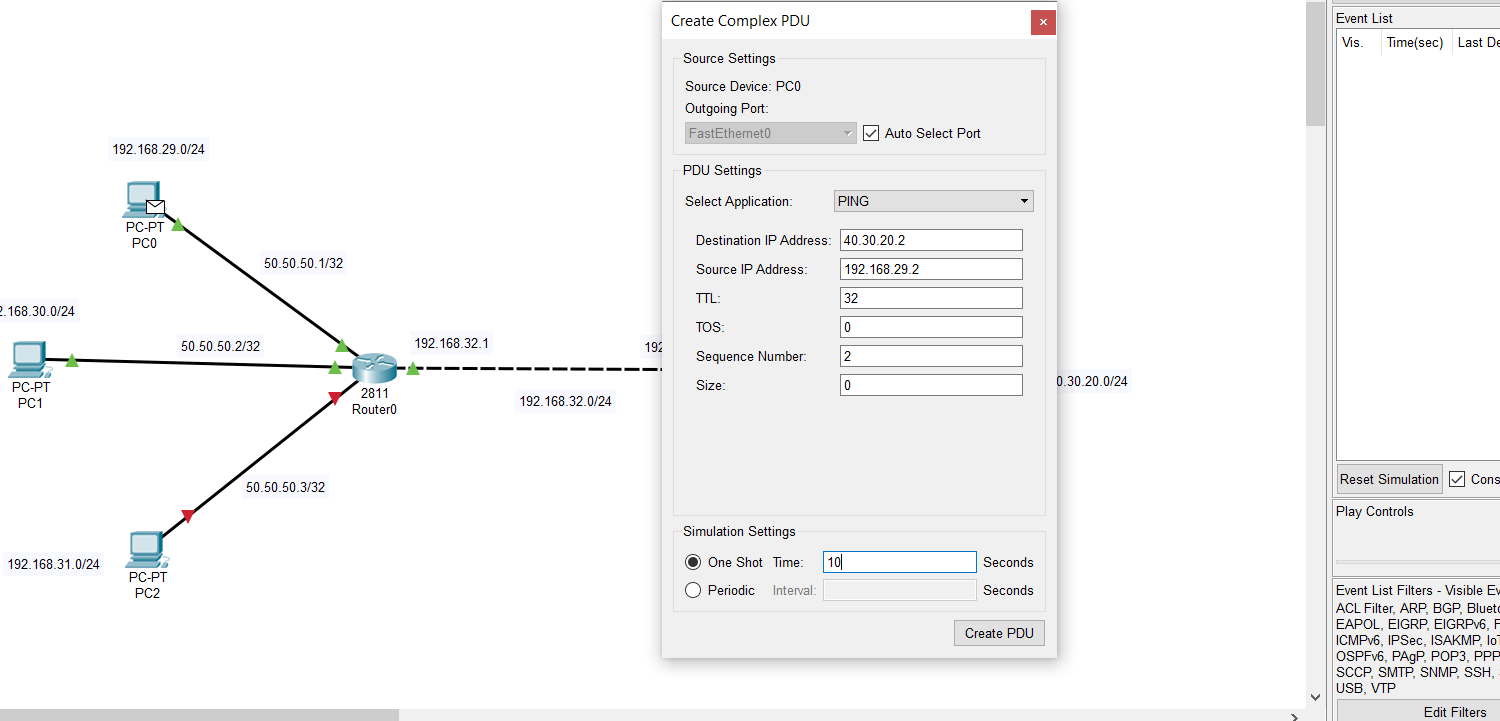


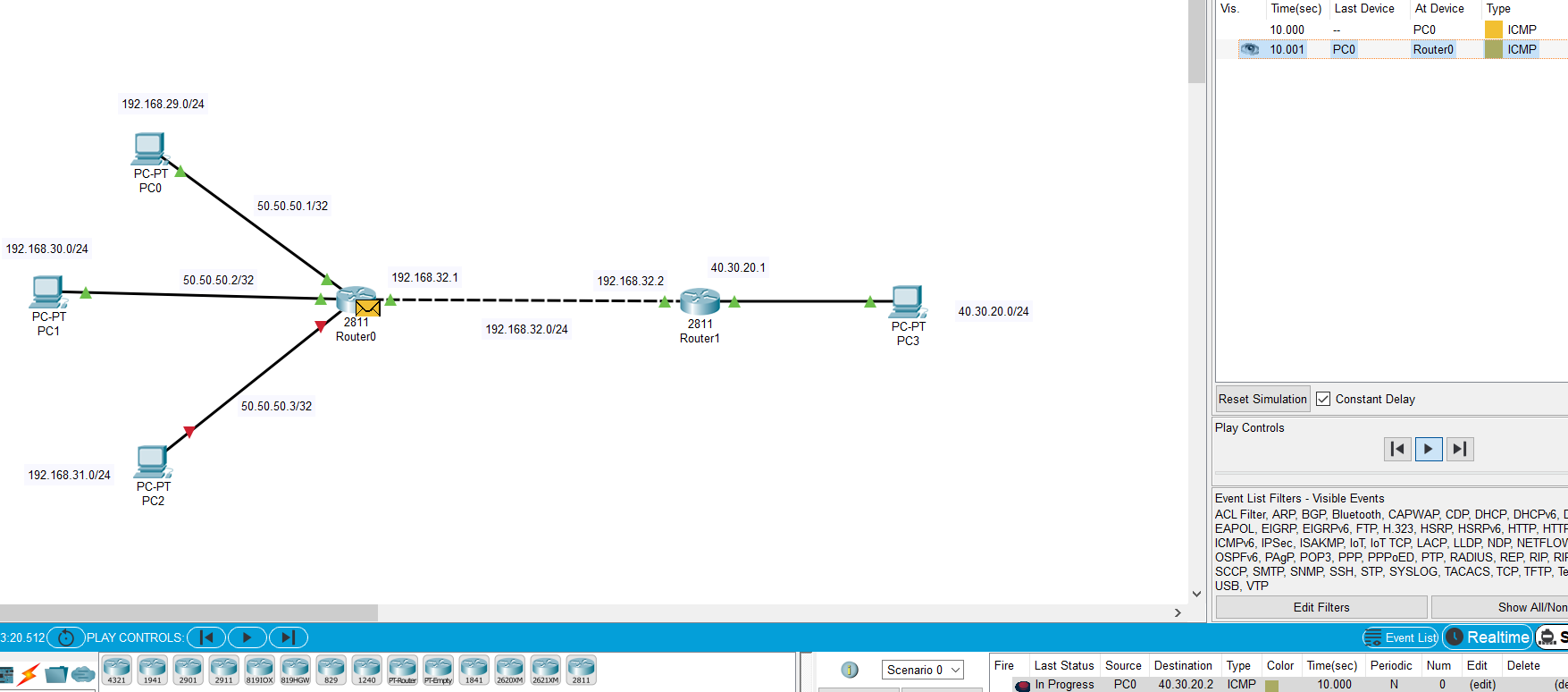


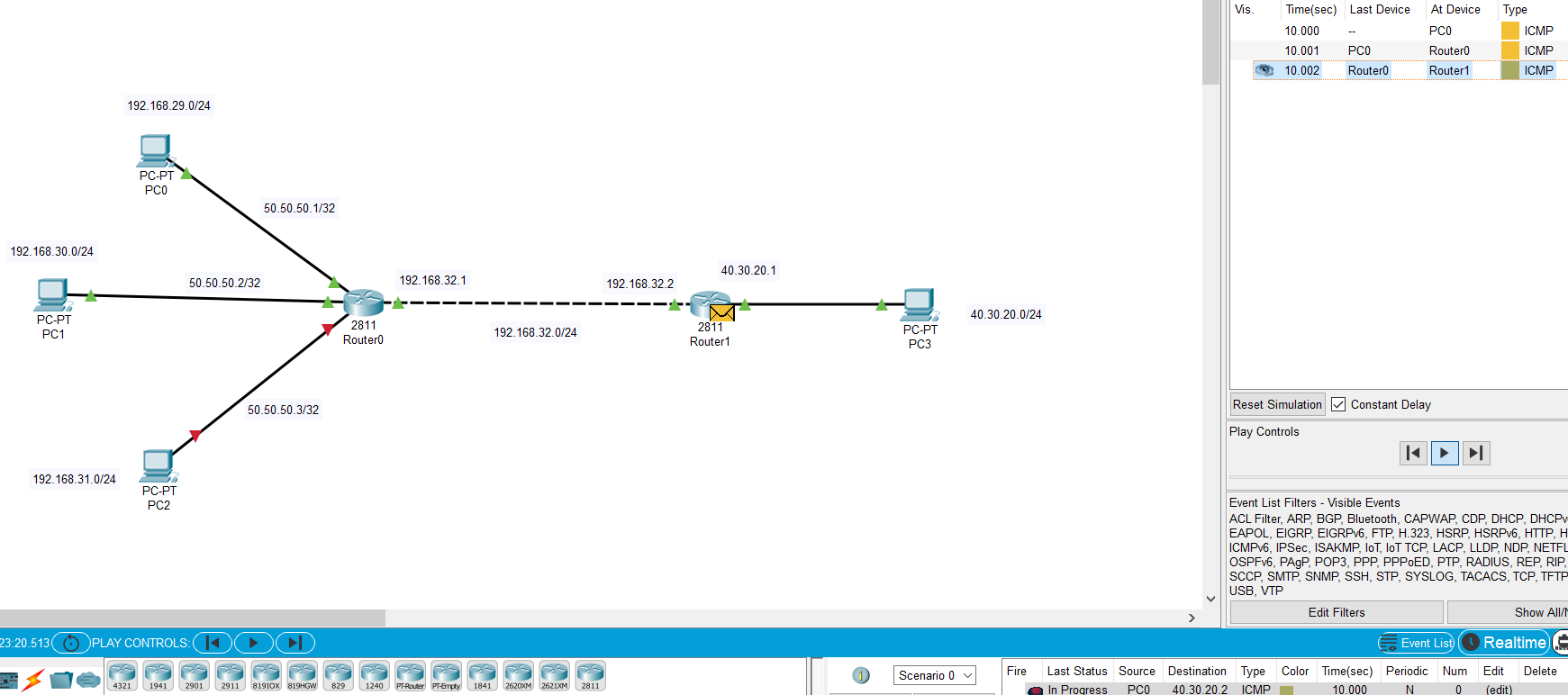


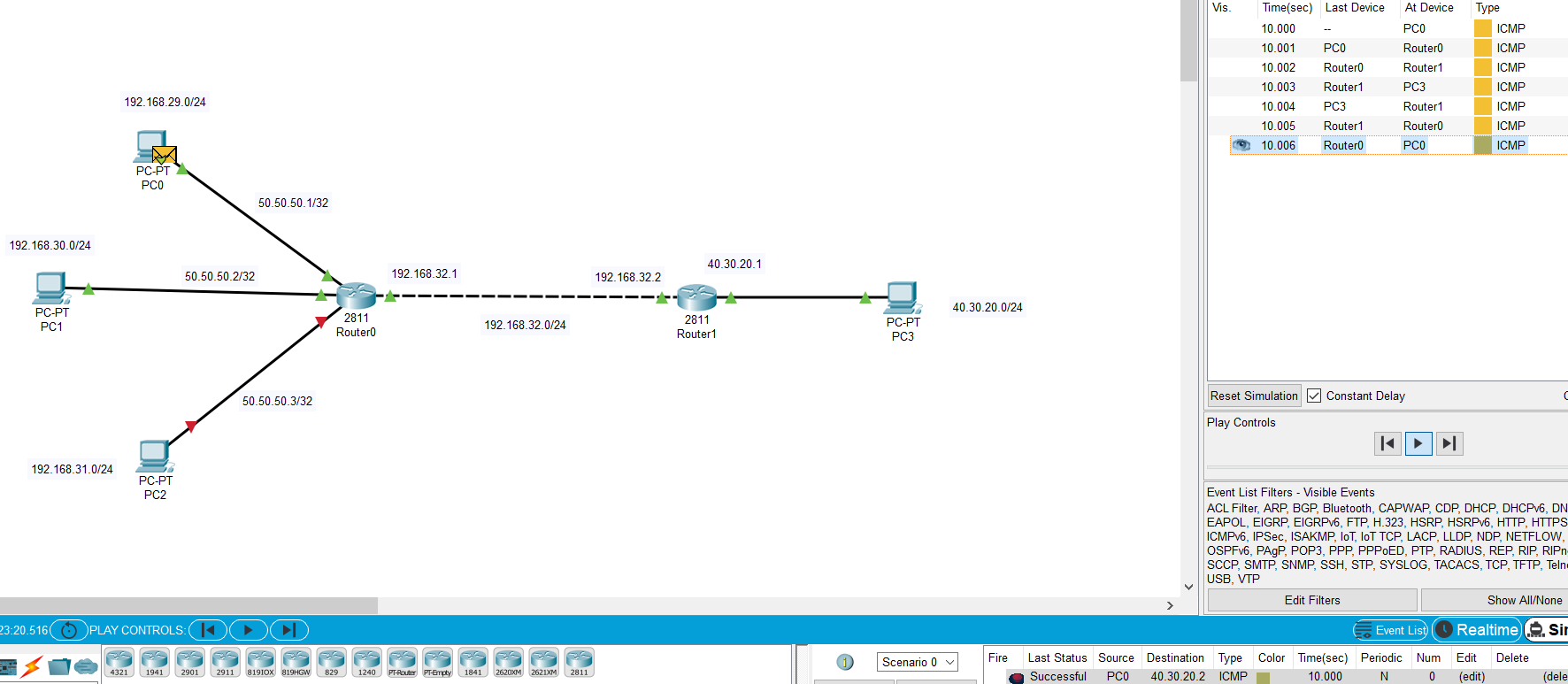


1. Complex PDU:-









1. Dijkstra Algorithm:-

Code:-

'''

Author : Dhruv B kakadiya

'''

class Graph:

    def min\_distance(self, dist, queue):

        minimum = float("Inf")

        min\_index = -1

        for i in range(len(dist)):

            if ((dist[i] < minimum) and (i in queue)):

                minimum = dist[i]

                min\_index = i

        return min\_index

    # print path from source to all node reccursively

    def printPath(self, parent, j):

        if (parent[j] == -1):

            print(j)

            return

        self.printPath(parent, parent[j])

        print(j)

    # for printing solution for shortest path

    def printSolution(self, dist, parent):

        src = self.src

        print("Vertex \t\t\nSrc -> Des\tDistance\tNext Hop\tPath\t\t\tTotal Hops")

        for i in range(1, len(dist)):

            print("\n%d --> %d \t%d\t" % (src, i, dist[i]), end = "")

            if(dist[i] != float('inf')):

                temp\_path = []

                temp\_path\_string = ""

                currentNode = i

                hop\_count = 0

                while(currentNode != src and currentNode >= 0 ):

                    hop\_count += 1

                    temp\_path.insert(0, currentNode)

                    temp\_path\_string = str(currentNode) + " ->" +  temp\_path\_string

                    currentNode = parent[currentNode]

                if(i == src):

                    temp\_path.insert(0, src)

                temp\_path\_string = str(src) + "->" + temp\_path\_string

                temp\_path\_string = temp\_path\_string[ : -2]

                print(f"Next Hop:{temp\_path[0]}\t\t", temp\_path\_string, "\t\tTotal\_Hops: ", hop\_count)

    def dijkstra(self, graph, src):

        self.src = src

        row = len(graph)

        col = len(graph[0])

        dist = [float("Inf")] \* row

        parent = [-1] \* row

        dist[src] = 0

        queue = []

        for i in range(row):

            queue.append(i)

        while (queue):

            u = self.min\_distance(dist,queue)

            queue.remove(u)

            for i in range(col):

                if ((graph[u][i]) and (i in queue)):

                    if (dist[u] + graph[u][i] < dist[i]):

                        dist[i] = dist[u] + graph[u][i]

                        parent[i] = u

        self.printSolution(dist,parent)

g= Graph()

graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],

        [4, 0, 8, 0, 0, 0, 0, 11, 0],

        [0, 8, 0, 7, 0, 4, 0, 0, 2],

        [0, 0, 7, 0, 9, 14, 0, 0, 0],

        [0, 0, 0, 9, 0, 10, 0, 0, 0],

        [0, 0, 4, 14, 10, 0, 2, 0, 0],

        [0, 0, 0, 0, 0, 2, 0, 1, 6],

        [8, 11, 0, 0, 0, 0, 1, 0, 7],

        [0, 0, 2, 0, 0, 0, 6, 7, 0]

        ]

# Print the solution

src = int(input("Enter the source node :"))

g.dijkstra(graph,src)

* Output:-

