
Trabalho Laboratorial 3

Comunicações Móveis

OLSR and IPv6 using Mininet

Diogo Remião & Miguel Pinheiro

Abril 2021



Faculdade de Engenharia da Universidade do Porto
TEC

Bandwidth (bits/s)	PC1	PC2	PC3	PC4
Linha	442	696	721	435
Estrela	467	693	451	436

Table 1: Bandwidth em linha e em estrela

0.1 Pergunta 1

No protocolo OLSR, o pacote HELLO é um pacote enviado periodicamente por um node para os nodes adjacentes (one-hop neighbours). Este pacote contém informação relativa ao link entre o sender-node e o neighbour-node, assim como informação sobre os seus vizinhos, de forma a que todos os nodes consigam mapear a rede [1].

É de notar que o pacote HELLO é sempre enviado em broadcast para os vizinhos. Para enviar em broadcast, no caso de uma rede MANET [2], os Ips a usar para broadcast/multicast são 224.0.0.109 para IPv4 e FF02:0:0:0:0:0:6D para IPv6. Como neste caso o Ipv4Broadcast está desativado, o sistema iria fazer uso do protocolo IPv6.

```

Internet Protocol Version 6, Src: 2021::4, Dst: ff02::6d
  0110 .... = Version: 6
  > .... 1100 0000 .... = Traffic Class: 0xc0 (DSCP: CS6, ECN: Not-ECT)
  .... 0110 0001 0000 1101 0000 = Flow Label: 0x610d0
  Payload Length: 60
  Next Header: UDP (17)
  Hop Limit: 1
  Source Address: 2021::4
  Destination Address: ff02::6d

```

Figure 1: Pacote HELLO

0.2 Pergunta 2

Dado que neste cenário todos os nodes estão ligados ao mesmo switch, isto significa que eles conseguem se ver entre todos. Para simular as diferentes configurações, temos que filtrar o tráfego que "supostamente" não deve existir.

Por exemplo, analisando a tipologia em linha, no PC1 temos que filtrar todo o tráfego para o PC3 e PC4, que nesta configuração, o PC1 não devia conseguir ver.

No.	Time	Source	Destination	Protocol	Length	Info
29	44.350400359	2021::1	ff02::6d	OLSR	158	OLSR (IPv6) Packet, Length: 208 Bytes
32	40.114552887	2021::1	ff02::6d	OLSR	158	OLSR (IPv6) Packet, Length: 96 Bytes
33	40.116419904	2021::2	ff02::6d	OLSR	222	OLSR (IPv6) Packet, Length: 160 Bytes
34	44.122093850	2021::1	2021::2	ICMPv6	118	Echo (ping) request id=0x285b, seq=1, hop limit=64 (reply in 35)
35	44.122272200	2021::2	2021::1	ICMPv6	118	Echo (ping) reply id=0x285b, seq=1, hop limit=64 (request in 34)
38	45.122616883	2021::1	2021::2	ICMPv6	118	Echo (ping) request id=0x285b, seq=2, hop limit=64 (reply in 39)
39	45.122845488	2021::2	2021::1	ICMPv6	118	Echo (ping) reply id=0x285b, seq=2, hop limit=64 (request in 38)
40	45.448488030	2021::2	ff02::6d	OLSR	254	OLSR (IPv6) Packet, Length: 192 Bytes
41	46.143280500	2021::1	ff02::6d	OLSR	154	OLSR (IPv6) Packet, Length: 52 Bytes
42	46.146063177	2021::1	2021::2	ICMPv6	118	Echo (ping) request id=0x285b, seq=3, hop limit=64 (reply in 43)
43	46.146966115	2021::2	2021::1	ICMPv6	118	Echo (ping) reply id=0x285b, seq=3, hop limit=64 (request in 42)
44	47.171890955	2021::1	2021::2	ICMPv6	118	Echo (ping) request id=0x285b, seq=4, hop limit=64 (reply in 45)
45	47.173919437	2021::2	2021::1	ICMPv6	118	Echo (ping) reply id=0x285b, seq=4, hop limit=64 (request in 44)

Figure 2: Análise do tráfego no PC1 com filtros

Após aplicar os filtros correspondentes, podemos analisar a largura de banda utilizada por cada node. Os dados a considerar são apenas das mensagens de controlo, (OLSR v1), pois estas são automáticas e constantes. Se considerássemos os pings efetuados, os dados estariam errados pois os pings não são constantes e dependem do que o utilizador fez.

No caso da tipologia em linha, quer o PC1 quer o PC4 só vêm um node, PC2 e PC3 respetivamente. Já o PC2 e PC3 vêm dois nodes cada. Deste modo, é de esperar que o PC2 e PC3 gerem mais tráfego que o PC1 e PC4, dado que vêm mais nodes. Isto é facilmente visível na tabela, onde o PC2 e PC3 geram aproximadamente mais 50% de tráfego com mensagens de controlo.

No caso da tipologia em estrela, o PC2 é vizinho de todos os outros PCs, enquanto os que outros PCs apenas têm um vizinho. Deste modo, é de esperar que largura de banda usada pelo PC2 seja superior aos outros computadores, o que se verifica nos dados obtidos.

0.3 Pergunta 3

Para efetuar o teste, procedemos à mudança de tipologia nos PCs e reiniciámos o protocolo OLSR após a reconfiguração. O instante $t=0$ na captura do *Wireshark* corresponde ao momento em que o protocolo foi iniciado no último PC (PC1). Imediatamente a seguir, executamos constantemente o comando `h1 ping6 2021::4` para enviar um ping entre o PC1 e o PC2. Este pedido não foi no entanto capturado pelo *Wireshark*, dado que o nesse momento o PC1 ainda não conhecia a existência do PC4. Após 32 segundos, o PC1 finalmente já tinha informação relativamente ao PC4 e ao caminho e os pings passaram a ser lidos pelo *Wireshark*.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	2021::1	ff02::6d	OLSR v1	114	OLSR (IPv6) Packet, Length: 52 Bytes
3	2.497168772	2021::1	ff02::6d	OLSR v1	114	OLSR (IPv6) Packet, Length: 52 Bytes
5	5.534299166	2021::1	ff02::6d	OLSR v1	158	OLSR (IPv6) Packet, Length: 98 Bytes
7	8.139798387	2021::1	ff02::6d	OLSR v1	214	OLSR (IPv6) Packet, Length: 152 Bytes
9	11.150843821	2021::1	ff02::6d	OLSR v1	114	OLSR (IPv6) Packet, Length: 52 Bytes
11	12.861673634	2021::1	ff02::6d	OLSR v1	114	OLSR (IPv6) Packet, Length: 52 Bytes
13	16.965874797	2021::1	ff02::6d	OLSR v1	158	OLSR (IPv6) Packet, Length: 98 Bytes
15	17.688985528	2021::1	ff02::6d	OLSR v1	214	OLSR (IPv6) Packet, Length: 152 Bytes
17	22.411945111	2021::1	ff02::6d	OLSR v1	158	OLSR (IPv6) Packet, Length: 98 Bytes
19	26.753111461	2021::1	ff02::6d	OLSR v1	114	OLSR (IPv6) Packet, Length: 52 Bytes
21	27.264444755	2021::1	ff02::6d	OLSR v1	114	OLSR (IPv6) Packet, Length: 52 Bytes
23	27.264444755	2021::1	ff02::6d	OLSR v1	178	OLSR (IPv6) Packet, Length: 118 Bytes
25	31.866774938	2021::1	ff02::6d	OLSR v1	158	OLSR (IPv6) Packet, Length: 98 Bytes
26	32.140845278	2021::1	2021::4	ICMPv6	118	Echo (ping) request id=0x30ba, seq=1, hop limit=64 (reply in 28)
27	32.140845278	2021::1	2021::1	ICMPv6	214	Redirect is at 00:00:00:00:00:04
28	32.140845278	2021::1	2021::1	ICMPv6	118	Echo (ping) reply id=0x30ba, seq=1, hop limit=63 (request in 26)
30	32.573573877	2021::1	ff02::6d	OLSR v1	158	OLSR (IPv6) Packet, Length: 98 Bytes
31	33.175826389	2021::1	2021::4	ICMPv6	118	Echo (ping) request id=0x30ba, seq=2, hop limit=64 (reply in 33)
32	33.175826389	2021::1	2021::1	ICMPv6	214	Redirect is at 00:00:00:00:00:04
33	33.175826389	2021::1	2021::1	ICMPv6	118	Echo (ping) reply id=0x30ba, seq=2, hop limit=63 (request in 31)
34	34.199478882	2021::1	2021::4	ICMPv6	118	Echo (ping) request id=0x30ba, seq=3, hop limit=64 (reply in 36)
35	34.199478882	2021::1	2021::1	ICMPv6	214	Redirect is at 00:00:00:00:00:04
36	34.199478882	2021::1	2021::1	ICMPv6	118	Echo (ping) reply id=0x30ba, seq=3, hop limit=63 (request in 34)
37	35.224251687	2021::1	2021::4	ICMPv6	118	Echo (ping) request id=0x30ba, seq=4, hop limit=64 (reply in 38)
38	35.224251687	2021::1	2021::1	ICMPv6	214	Redirect is at 00:00:00:00:00:04
39	35.224251687	2021::1	2021::1	ICMPv6	118	Echo (ping) reply id=0x30ba, seq=4, hop limit=63 (request in 37)
40	36.248371817	2021::1	2021::4	ICMPv6	118	Echo (ping) request id=0x30ba, seq=5, hop limit=64 (reply in 42)
41	36.248371817	2021::1	2021::1	ICMPv6	214	Redirect is at 00:00:00:00:00:04
42	36.248371817	2021::1	2021::1	ICMPv6	118	Echo (ping) reply id=0x30ba, seq=5, hop limit=63 (request in 40)

Figure 3: Ping do PC1 para PC4 após mudança de tipologia

Entre o primeiro ping e o a primeira resposta temos um intervalo de aproximadamente 32 segundos. Este tempo faz sentido dado que a soma do tempo do um pacote HELLO (10s) e TC (6s) é de 16 segundos. Este processo tem que ser efetuado duas vezes, do PC4 para o 2 e depois do PC2 para o PC1, daí os 32 segundos.

0.4 Pergunta 4

A resposta a esta pergunta está parcialmente presente na secção 0.2.

A função dos MPR é de otimização da transmissão dos pacotes HELLO e TC dentro da rede MANET. Deste modo, minimizamos o tráfego desnecessário na rede e diminuimos a largura de banda necessária [3].

Na tipologia de linha, os MPR serão o PC2, escolhido pelo PC1 e PC3, e o PC3, escolhido pelo PC2 e PC4, que são os PCs que permitem comunicação 2-hop aos PCs que os escolheram. Como observamos na tabela 1, os PC2 e PC3 geram mais tráfego, que é um bom indicador do comportamento de MPRs.

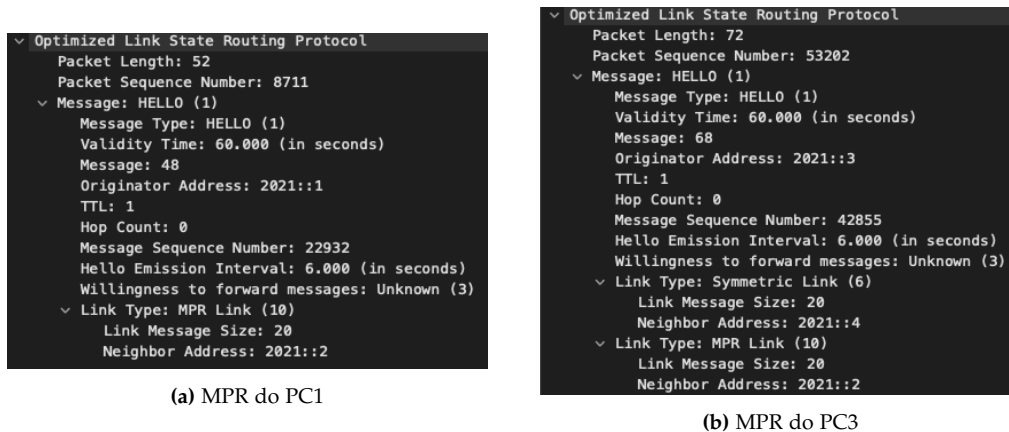


Figure 4: PC2 como MPR

Na figura 4a, o PC1 escolhe o PC2 como o seu MPR. Na figura 4b, o PC3 tem o PC2 como um *MPR Link* e o PC4 como um *Symmetric Link*.

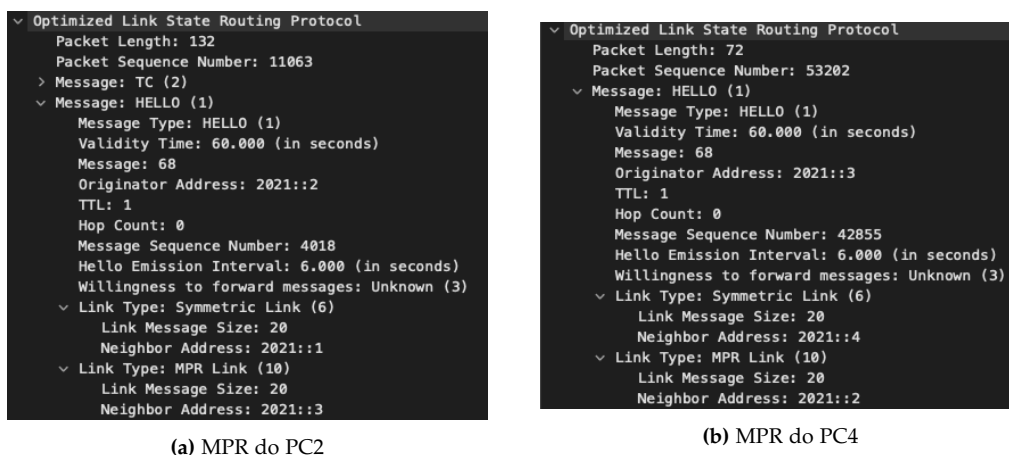


Figure 5: PC3 como MPR

Uma situação semelhante se pode verificar quando ao PC3. Na figura 4a, o PC2 escolhe o PC3 como o seu MPR. Na figura 4b, o PC4 tem o PC3 como um *MPR Link* e o PC2 como um *Symmetric Link*.

Na tipologia em estrela, o PC2 será escolhido por todos os PCs como MPR dado que é o centro da configuração em Estrela. O PC2 é o CP que permite comunicações 2-hop para todos. Enquanto que o PC2 vê todos os PCs, estes só vêm o PC2.

```

Optimized Link State Routing Protocol
Packet Length: 96
Packet Sequence Number: 5206
> Message: TC (2)
  Message: HELLO (1)
    Message Type: HELLO (1)
    Validity Time: 60.000 (in seconds)
    Message: 48
    Originator Address: 2021::1
    TTL: 1
    Hop Count: 0
    Message Sequence Number: 52178
    Hello Emission Interval: 6.000 (in seconds)
    Willingness to forward messages: Unknown (3)
  < Link Type: MPR Link (10)
    Link Message Size: 20
    Neighbor Address: 2021::2

```

(a) MPR do PC1

```

Optimized Link State Routing Protocol
Packet Length: 52
Packet Sequence Number: 55767
  Message: HELLO (1)
    Message Type: HELLO (1)
    Validity Time: 60.000 (in seconds)
    Message: 48
    Originator Address: 2021::3
    TTL: 1
    Hop Count: 0
    Message Sequence Number: 14797
    Hello Emission Interval: 6.000 (in seconds)
    Willingness to forward messages: Unknown (3)
  < Link Type: MPR Link (10)
    Link Message Size: 20
    Neighbor Address: 2021::2

```

(b) MPR do PC3

```

Optimized Link State Routing Protocol
Packet Length: 86
Packet Sequence Number: 22818
> Message: TC (2)
  Message: HELLO (1)
    Message Type: HELLO (1)
    Validity Time: 60.000 (in seconds)
    Message: 48
    Originator Address: 2021::4
    TTL: 1
    Hop Count: 0
    Message Sequence Number: 31132
    Hello Emission Interval: 6.000 (in seconds)
    Willingness to forward messages: Unknown (3)
  < Link Type: MPR Link (10)
    Link Message Size: 20
    Neighbor Address: 2021::2

```

(c) MPR do PC4

Figure 6: PC2 como MPR

Como podemos observar na figura 6, todos os PCs escolhem o PC2 como o seu MPR.

```

Optimized Link State Routing Protocol
Packet Length: 248
Packet Sequence Number: 44456
> Message: TC (2)
> Message: TC (2)
> Message: TC (2)
  Message: HELLO (1)
    Message Type: HELLO (1)
    Validity Time: 60.000 (in seconds)
    Message: 80
    Originator Address: 2021::2
    TTL: 1
    Hop Count: 0
    Message Sequence Number: 14580
    Hello Emission Interval: 6.000 (in seconds)
    Willingness to forward messages: Unknown (3)
  < Link Type: Symmetric Link (6)
    Link Message Size: 52
    Neighbor Address: 2021::4
    Neighbor Address: 2021::3
    Neighbor Address: 2021::1

```

Figure 7: Links do PC2

Dado que o PC2 consegue comunicar com todos os PCs em one-hop, não precisa de MPR. Desse modo, assume todos os links que tem com os restantes PCs como *Symmetric Links*.

0.5 Pergunta 5

```

diogoremiao@diogoremiao-VirtualBox:~/Desktop/new script$ sudo python2 mininet_olsr_topology.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h1, s2) (h2, s1) (h3, s1) (h4, s1) (h5, s2)
*** Configuring hosts
h1 h2 h3 h4 h5
Configuring Network*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
Running
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth1 h1-eth1:s2-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
h5 h5-eth0:s2-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
s2 lo: s2-eth1:h1-eth1 s2-eth2:h5-eth0
c0
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 2021::1 prefixlen 128 scopeid 0x0<global>
    inet6 fe80::200:ff:fe00:1 prefixlen 64 scopeid 0x20<link>
    ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
    RX packets 40 bytes 4779 (4.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 1132 (1.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

h1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 3000::1 prefixlen 64 scopeid 0x0<global>
    inet6 fe80::f8d9:4ff:fe96:971f prefixlen 64 scopeid 0x20<link>
    ether fa:d9:04:96:97:1f txqueuelen 1000 (Ethernet)
    RX packets 24 bytes 2955 (2.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 10 bytes 952 (952.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X
h2 -> h1 h3 h4 X
h3 -> h1 h2 h4 X
h4 -> h1 h2 h3 X
h5 -> X X X X
*** Results: 40% dropped (12/20 received)
mininet>

```

Figure 8: Output do script Python

O Mininet oferece uma command-line interface que nos permite correr comandos específicos de cada host [4]. No entanto, para podermos utilizar o comando é preciso um *Mininet Object*, especificando a classe topo do OLSR e o controlador OVS. Desta forma, conseguimos agora resolver os diferentes hosts acedendo a este objeto e podemos correr agora os comandos indicados para cada host (Anexo A).

Na figura 8 podemos observar que correndo por exemplo *ifconfig* no host 1, a interface h1-eth0 está configurada para IPv6 como indicado. O comando *net* dá como output as diferentes interfaces que foram configuradas. Finalmente, correndo *Pingall* verificamos a conexão entre todos os hosts, garantindo que os comandos foram de facto executados para cada hosts, e o sistema funcional.

0.6 Pergunta 6

Mais uma vez, o novo *Mininet Object* prova-se útil, dado que nos permite aceder diretamente à network e aos diferentes hosts. Deste modo, usando mais uma vez a CLI do Mininet, podemos fazer uso da API Python disponibilizada pelo Mininet para fazer as alterações necessárias. Para dizer ao Mininet que se está a tentar correr um comando python, tem que se por o prefixo *py* antes de todos os comandos.

O comando `net.addHost('h6')` permite adicionar mais um host à rede, neste caso o h6.

De seguida é preciso ligar o host ao switch, utilizando o comando `net.addLink(net.get('s1'), net.get('h6'))`.

Depois deste passo, é preciso ligar a interface no switch, que é feito automaticamente na classe *topo* mas não na classe *net*. Isto é feito com o comando `net.get('s1').attach('s1-eth0')` na classe *node*.

Por fim, é atribuído um IP ao novo host com o comando `net.get('h6').setIP('10.0.0.6')`, mais um vez na classe *node*.

```
diogoremiao@diogoremiao-VirtualBox:~/Desktop/new script$ sudo python mininet_olsr_topology.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h1, s2) (h2, s1) (h3, s1) (h4, s1) (h5, s2)
*** Configuring hosts
h1 h2 h3 h4 h5
Configuring Network*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
Running
*** Starting CLI:
mininet> py net.addHost('h6')
<Host h6: pid=5702>
mininet> py net.addLink(net.get('s1'), net.get('h6'))
<mininet.link.Link object at 0x7f67a3c1d290>
mininet> py net.get('h6').setIP('10.0.0.6')
mininet> net
h1 h1-eth0:s1-eth1 h1-eth1:s2-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
h5 h5-eth0:s2-eth2
h6 h6-eth0:s1-eth5
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0 s1-eth5:h6-eth0
s2 lo: s2-eth1:h1-eth1 s2-eth2:h5-eth0
c0
mininet> py net.get('s1').attach('s1-eth5')
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X h6
h2 -> h1 h3 h4 X h6
h3 -> h1 h2 h4 X h6
h4 -> h1 h2 h3 X h6
h5 -> X X X X X
h6 -> h1 h2 h3 h4 X
*** Results: 33% dropped (20/30 received)
mininet>
```

Figure 9: Uso da API Python do Mininet para adicionar um novo host

Bibliography

- [1] *Optimized Link State Routing Protocol (OLSR)*. IETF. [Secção 6]. Outubro 2003.
- [2] *IANA Allocations for Mobile Ad Hoc Network (MANET) Protocols*. IETF. [Secção 5]. Março 2009.
- [3] *Optimized Link State Routing Protocol (OLSR)*. IETF. [Secção 1]. Outubro 2003.
- [4] *Mininet Python API Reference Manual*. Mininet.

A Python Script

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.node import OVSController

class OlsrTopo(Topo):
    """ COMO OLSR topology """

    def __init__(self):
        "Create custom topo."

        # Initialize topology
        Topo.__init__(self)

        # Add hosts and switches
        pc1 = self.addHost('h1', mac='00:00:00:00:00:01')
        pc2 = self.addHost('h2', mac='00:00:00:00:00:02')
        pc3 = self.addHost('h3', mac='00:00:00:00:00:03')
        pc4 = self.addHost('h4', mac='00:00:00:00:00:04')
        pc5 = self.addHost('h5', mac='00:00:00:00:00:05')

        sw1 = self.addSwitch('s1')
        sw2 = self.addSwitch('s2')

        # Add links
        self.addLink(pc1, sw1)
        self.addLink(pc2, sw1)
        self.addLink(pc3, sw1)
        self.addLink(pc4, sw1)

        self.addLink(pc1, sw2)
        self.addLink(pc5, sw2)

def Olsr():
    topo= OlsrTopo()
    net = Mininet (topo=topo, controller = OVSController)
    net.start()

    for i in range(1,4):
        net.get('h'+str(i)).cmd('ifconfig h'+str(i)+'-eth0 ↵
                               inet6 add 2021::'+str(i)+'/128')
        net.get('h'+str(i)).cmd('echo 0 > /proc/sys/net/ipv6/↵
                               conf/h'+str(i)+'-eth0/accept_ra')
```

```

        net.get('h'+str(i)).cmd('echo 1 > /proc/sys/net/ipv6/↵
        conf/h'+str(i)+'-eth0/forwarding')

net.get('h1').cmd('ifconfig h1-eth1 up')
net.get('h1').cmd('ifconfig h1-eth1 inet6 add 3000::1/64'↵
)

net.get('h5').cmd('ifconfig h5-eth0 inet6 add3000::254/64↵
')
net.get('h5').cmd('route -A inet6 add 2021:0:0::/64 gw ↵
3000::')
info('Running\n')
CLI(net)
info('Stopping\n')
net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    Olsr()

```