



Network Project 2

Sepehr Ebadi

9933243

Khordad, 1403

Wireshark یک تحلیلگر پروتکل شبکه قدرتمند است که می تواند با ضبط، تجزیه و تحلیل و تفسیر ترافیک شبکه به شناسایی مشکلات شبکه و تهدیدات امنیتی کمک کند.

در اینجا چند راه برای شناسایی مشکلات شبکه و تهدیدات امنیتی گفته میشود:

(۱) Wireshark به شما امکان می دهد ترافیک شبکه را در لحظه ضبط و تجزیه و تحلیل کنید، که می تواند به شما در شناسایی مسائلی مانند از دست دادن بسته، تأخیر، پیکربندی نادرست دستگاه های شبکه یا سایر مشکلات عملکرد شبکه کمک کند. با تجزیه و تحلیل بسته های ضبط شده، می توانید منبع مشکل را شناسایی کرده و برای حل آن اقدام مناسب انجام دهید.

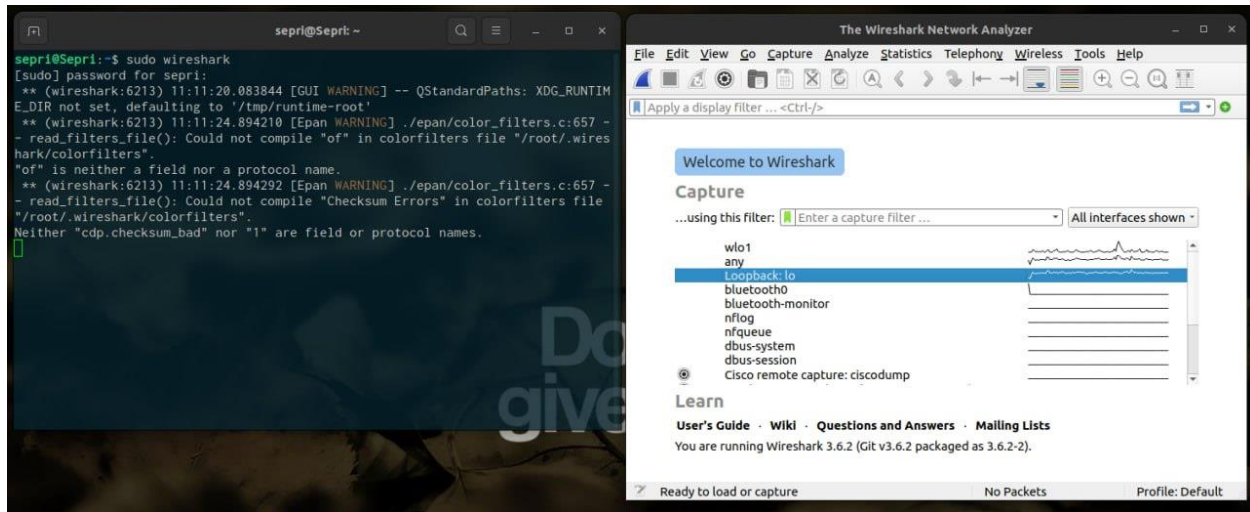
(۲) تشخیص ناهنجاری های شبکه : Wireshark می تواند با تجزیه و تحلیل الگوهای موجود در ترافیک شبکه به شما در تشخیص ناهنجاری های شبکه کمک کند. به عنوان مثال، میتوانید فیلترهایی را در Wireshark تنظیم کنید تا فقط انواع خاصی از ترافیک را ضبط کند، مانند ترافیک از یک آدرس IP خاص یا پورت، و سپس بسته های گرفته شده را برای هر الگو یا رفتار غیرعادی که ممکن است نشان دهنده یک تهدید امنیتی باشد، تجزیه و تحلیل کنید مثل پروتکل های غیرمنتظره، ترافیک بیش از حد، یا الگوهای ترافیک غیرعادی.

(۳) شناسایی تهدیدات امنیتی: Wireshark می تواند ترافیک شبکه را ضبط و تجزیه و تحلیل کند تا به شناسایی تهدیدات امنیتی مانند بدافزارها، ویروس ها یا سایر انواع فعالیت های مخرب کمک کند. برای مثال، Wireshark میتواند بسته هایی را بگیرد که حاوی بارهای مشکوک، درخواستهای DNS غیرمعمول یا اتصالات شبکه غیرمنتظره هستند که میتواند نشان دهنده نقض های امنیتی یا حملات بالقوه باشد Wireshark. همچنین میتواند به شما کمک کند تا تلاش های غیرمجاز دسترسی به شبکه، الگوهای احراز هویت غیرمعمول یا سایر نشانه های فعالیت مشکوک شبکه را شناسایی کنید.

۴) تجزیه و تحلیل پروتکل های شبکه : Wireshark دارای جداکننده های پروتکل داخلی است که به شما امکان تجزیه و تحلیل و تفسیر پروتکل های مختلف شبکه مانند UDP، DNS، HTTP، TLS/SSL، IP/TCP، و بسیاری دیگر را می دهد. با تجزیه و تحلیل جزئیات پروتکل های شبکه در بسته های ضبط شده، می توانید مسائل یا ناهنجاری هایی را شناسایی کنید که ممکن است بر عملکرد یا امنیت شبکه تأثیر بگذارد.

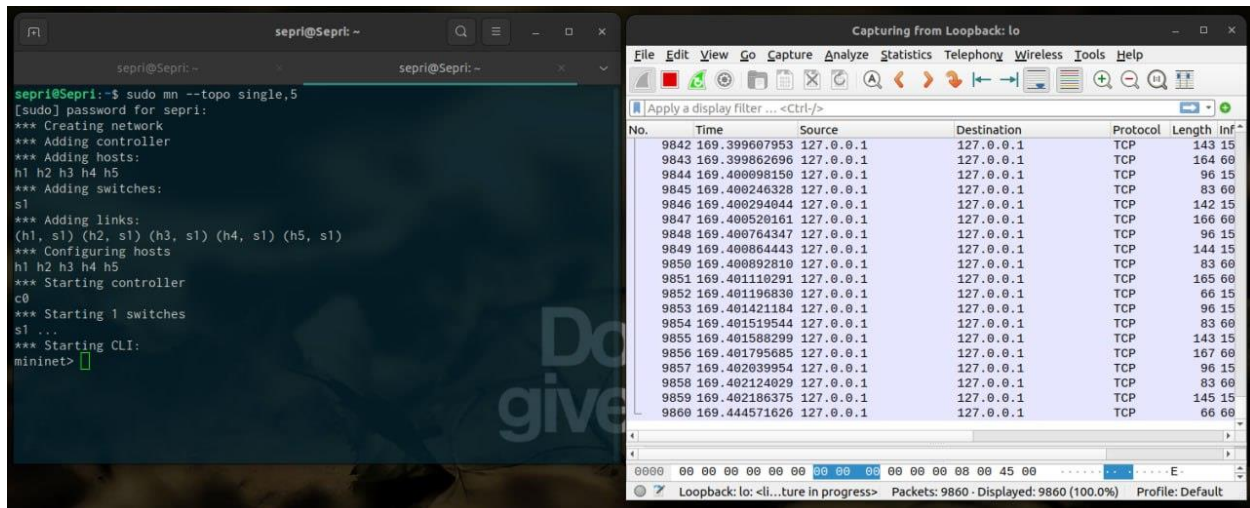
۵) نظارت بر ترافیک شبکه : Wireshark می تواند به عنوان یک ابزار نظارت بر شبکه برای ضبط و تجزیه و تحلیل ترافیک شبکه در یک دوره زمانی مورد استفاده قرار گیرد. این می تواند به شما کمک کند رفتار شبکه پایه را ایجاد کنید، تغییرات در الگوهای ترافیک را شناسایی کنید و فعالیت غیرعادی شبکه را شناسایی کنید که ممکن است نشان دهنده مشکلات شبکه یا تهدیدات امنیتی باشد.

به طور خلاصه، Wireshark می تواند به شناسایی مشکلات شبکه و تهدیدات امنیتی با ضبط، تجزیه و تحلیل و تفسیر ترافیک شبکه برای شناسایی ناهنجاری ها، تجزیه و تحلیل پروتکل های شبکه و نظارت بر فعالیت شبکه کمک کند. این ابزار ارزشمندی برای مدیران شبکه، تحلیلگران امنیتی و سایر متخصصان فناوری اطلاعات است که نیاز به تشخیص و حل مشکلات شبکه و شناسایی تهدیدات امنیتی بالقوه دارند.



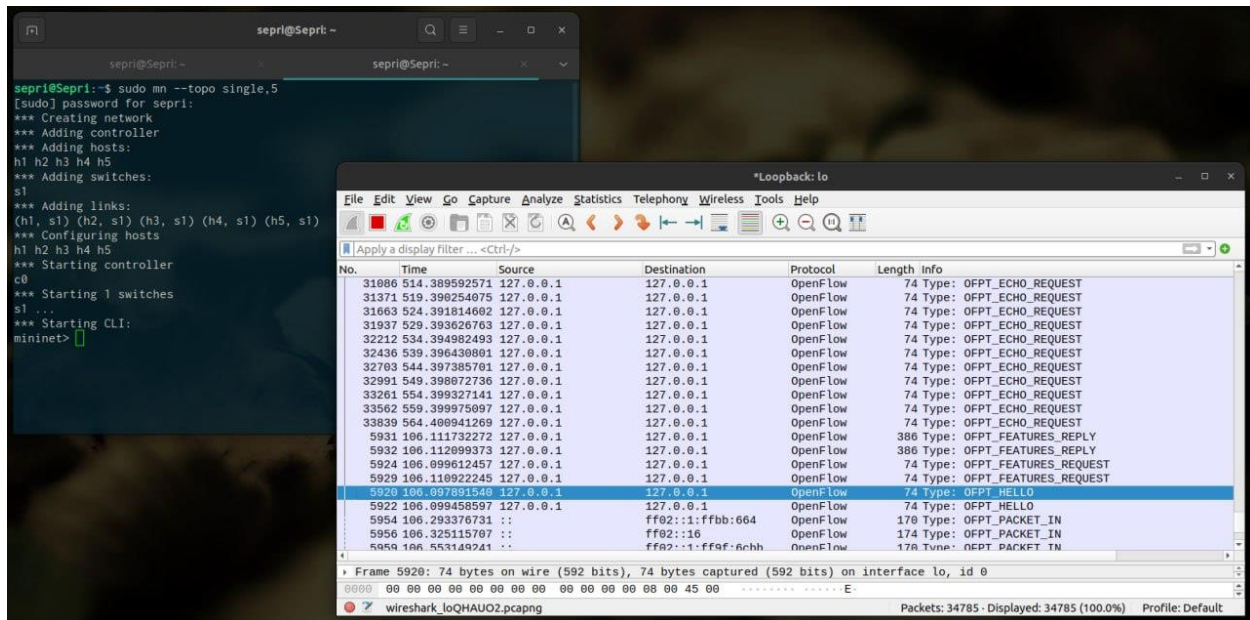
۱-۲

از tcp استفاده میکند. Openflow هم از tcp استفاده میکند.



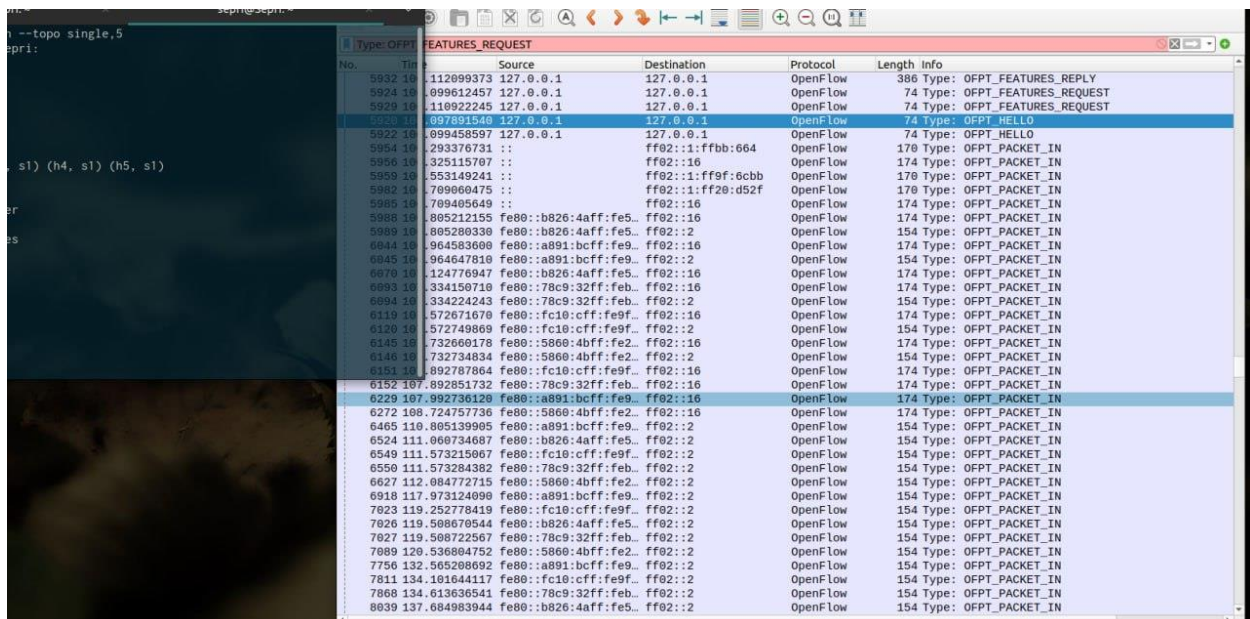
۲-۲

خط ۵۹۲۰ و ۵۹۲۲



۳-۲

خط ۵۹۲۴ و ۵۹۲۹ و ۵۹۳۲



پیام درخواست ویژگی از کنترل کننده به سویچ معمولاً به منظور درخواست اطلاعات در مورد قابلیت ها و ویژگی های پشتیبانی شده توسط سویچ است. این به کنترل کننده اجازه می دهد تا قابلیت های سویچ را درک کند و رفتار آن را بر اساس آن تنظیم کند. برای مثال، کنترل کننده ممکن است یک پیام درخواست ویژگی ارسال کند تا در مورد فیلدهای `actions, supported match fields flow table size` یا سایر قابلیت های سویچ پرس و جو کند. قالب یک پیام درخواست ویژگی در OpenFlow ممکن است بسته به نسخه خاص OpenFlow مورد استفاده و اجرای سویچ متفاوت باشد. با این حال، به طور کلی شامل فیلدهایی مانند نوع پیام، نسخه OpenFlow و هرگونه اطلاعات خاص مربوط به ویژگی درخواستی است.

نمونه ای از یک پیام درخواست ویژگی ساده در OpenFlow: ۱.۰

```
Header:
- Version: 1.0
- Type: Feature Request
- Length: <length of the message>

Body:
- Empty (no specific information requested)
```

پیام پاسخ ویژگی اطلاعاتی درباره قابلیت ها و ویژگی های پشتیبانی شده توسط سویچ ارائه می کند و به کنترل کننده اجازه می دهد تا قابلیت های سویچ را درک کند و رفتار آن را بر اساس آن تنظیم کند. قالب پیام پاسخ ویژگی در OpenFlow ممکن است بسته به نسخه خاص OpenFlow مورد استفاده و اجرای سویچ متفاوت باشد. با این حال، به طور کلی شامل فیلدهایی مانند نوع پیام، نسخه OpenFlow و اطلاعات مربوط به ویژگی ها و قابلیت های پشتیبانی شده سویچ است.

نمونه ای از پیام پاسخ ویژگی ساده در OpenFlow 1.0:

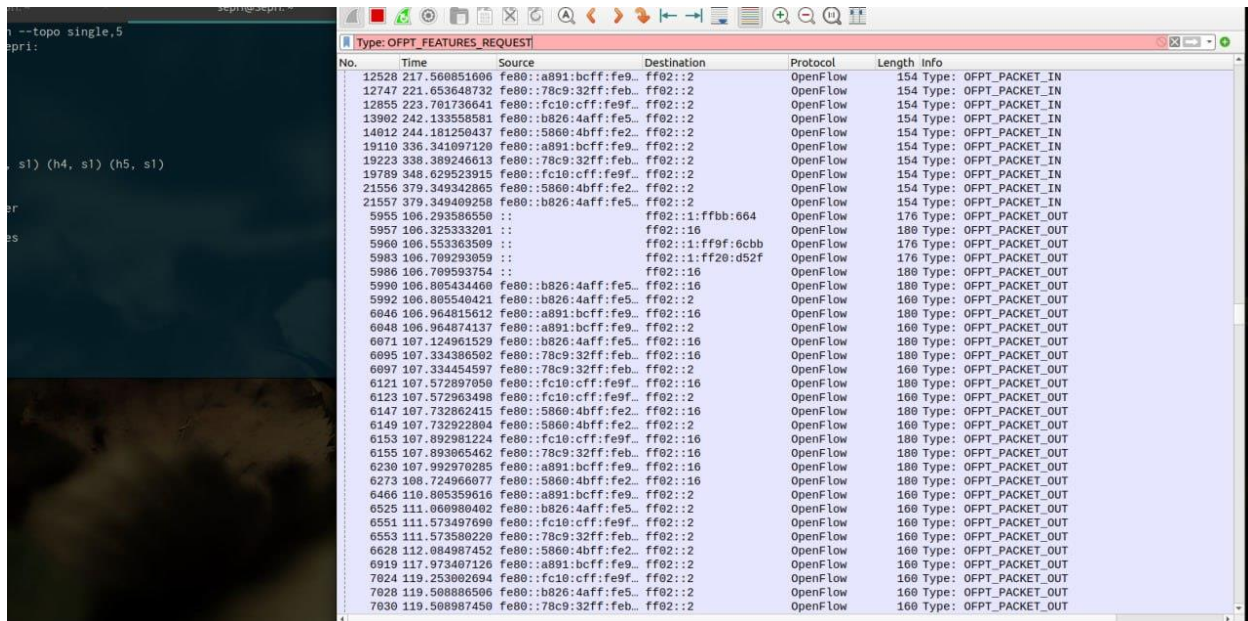
Header:

- Version: 1.0
- Type: Feature Reply
- Length: <length of the message>

Body:

- Datapath ID: <unique identifier for the switch>
- Supported Features: <list of supported features such as match fields, actions, flow table size, etc.>
- Supported Actions: <list of supported actions that can be performed on flows>
- Other capabilities: <other capabilities of the switch, such as maximum number of groups, meters, etc.>

۴-۲



The image shows a terminal window on the left with network configuration commands and a packet capture analysis window on the right. The terminal shows commands like 'topo single,5' and 'pr1:'. The packet capture window is titled 'Type: OFPT_FEATURES_REQUEST' and displays a table of network traffic.

No.	Time	Source	Destination	Protocol	Length	Info
12528	217.650051606	fe80::a891:bcff:fe9...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
12747	221.653648732	fe80::78c9:32ff:feb...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
12855	223.701736641	fe80::fc10:cff:fe9f...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
13902	242.133558581	fe80::b826:4aff:fe5...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
14012	244.181250437	fe80::5860:4bfff:fe2...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
19110	336.341097120	fe80::a891:bcff:fe9...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
19223	338.389246613	fe80::78c9:32ff:feb...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
19789	348.629523915	fe80::fc10:cff:fe9f...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
21556	379.349342865	fe80::5860:4bfff:fe2...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
21557	379.349409258	fe80::b826:4aff:fe5...	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
5955	106.293586550	::	ff02::1:ffbb:664	OpenFlow	176	Type: OFPT_PACKET_OUT
5957	106.325333201	::	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
5960	106.553363509	::	ff02::1:ff9f:6cbb	OpenFlow	176	Type: OFPT_PACKET_OUT
5983	106.709293059	::	ff02::1:ff20:d52f	OpenFlow	176	Type: OFPT_PACKET_OUT
5986	106.709593754	::	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
5990	106.805434460	fe80::b826:4aff:fe5...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
5992	106.805540421	fe80::b826:4aff:fe5...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6046	106.964815612	fe80::a891:bcff:fe9...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6048	106.964874137	fe80::a891:bcff:fe9...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6071	107.124961529	fe80::b826:4aff:fe5...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6095	107.334386502	fe80::78c9:32ff:feb...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6097	107.334454597	fe80::78c9:32ff:feb...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6121	107.572097050	fe80::fc10:cff:fe9f...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6123	107.572963498	fe80::fc10:cff:fe9f...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6147	107.732862415	fe80::5860:4bfff:fe2...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6149	107.732922804	fe80::5860:4bfff:fe2...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6153	107.892981224	fe80::fc10:cff:fe9f...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6155	107.893065462	fe80::78c9:32ff:feb...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6230	107.992970285	fe80::a891:bcff:fe9...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6273	108.724966077	fe80::5860:4bfff:fe2...	ff02::16	OpenFlow	180	Type: OFPT_PACKET_OUT
6460	110.085359616	fe80::a891:bcff:fe9...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6525	111.086980402	fe80::b826:4aff:fe5...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6551	111.573497690	fe80::fc10:cff:fe9f...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6553	111.573580220	fe80::78c9:32ff:feb...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6628	112.084987452	fe80::5860:4bfff:fe2...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
6919	117.973407126	fe80::a891:bcff:fe9...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
7024	119.253002694	fe80::fc10:cff:fe9f...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
7028	119.508886506	fe80::b826:4aff:fe5...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT
7030	119.508987450	fe80::78c9:32ff:feb...	ff02::2	OpenFlow	160	Type: OFPT_PACKET_OUT

۵-۲

این بسته در دو حالت ارسال میشود و missing flow control و reverse connection

حال یا در قسمت action ذکر شده یا matching برای آن یافت نشده.

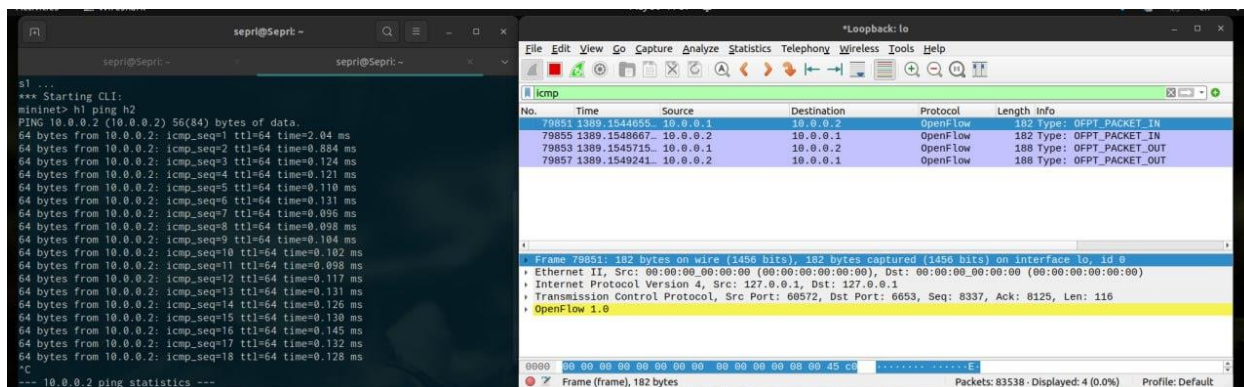
حالت واکنشی (Reactive mode):

در این حالت، سوئیچ هنگامی که بسته ای را دریافت می کند که با هیچ ورودی جریان موجود در جدول جریان خود مطابقت ندارد، پیام "packet_in" را به کنترل کننده ارسال می کند. سپس کنترل کننده نحوه مدیریت بسته را تعیین می کند، مانند نصب یک flow entry جدید در flow table سوئیچ یا انجام برخی اقدامات دیگر، و دستورالعمل های مناسب را به سوئیچ ارسال می کند.

حالت فعال (proactive mode):

در این حالت، سوئیچ برای هر بسته ای که دریافت می کند، پیام "packet_in" را به کنترل کننده ارسال نمی کند. در عوض، کنترلر flow entries را در flow table سوئیچ برای الگوهای ترافیک مورد انتظار از قبل نصب می کند، به طوری که سوئیچ می تواند بسته ها را مستقیماً بدون دخالت کنترلر ارسال کند. فقط بسته هایی که با هیچ flow entry موجود در flow table سوئیچ مطابقت ندارند به عنوان پیام "packet_in" برای پردازش بیشتر به کنترل کننده ارسال می شوند.

۶-۲



بسته های icmp تحت پروتکل openflow از هاست ۱ با آیدی ۱۰.۰.۰.۱ به هاست ۲ با آیدی ۱۰.۰.۰.۲ فرستاده شده اند و در جواب نیز هاست ۲ برای هاست یک پاسخ را ارسال کرده است. به طور کلی ping شامل دیتاهای زیر میباشد:

An ICMP request is a layered packet which is sent over the internet. It contains the Ether layer, which has the target and source MAC address in it. It also contains the IP layer, which has the source and target IP and also a couple of flags included. And at last it contains the ICMP data. This contains a type, a subtype, then a checksum and the rest of the header, which can vary from type and subtype (E.g. The code for echo is 8 and reply is 0).



```

sepri@Sepri:~$ sudo mn --topo single,6 --mac --switch ovsk
[sudo] password for sepri:
*** Creating network
*** Adding controller
-----
Caught exception. Cleaning up...
Exception: Please shut down the controller which is running on port 6653:
Active Internet connections (servers and established)
tcp        0      0 0.0.0.0:6653          0.0.0.0:*               LISTEN     6486/controller
tcp        0      0 127.0.0.1:66572       127.0.0.1:6653          ESTABLISHED 1026/ovs-vsitchd
tcp        0      0 127.0.0.1:6653        127.0.0.1:66572        ESTABLISHED 6486/controller
-----
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller ovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller ovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
kill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]*' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br sl
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br sl
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_[:alnum:]]+-eth[[:digit:]]+)'
( ip link del sl-eth1; ip link del sl-eth2; ip link del sl-eth3; ip link del sl-eth4; ip link del sl-eth5 ) 2> /dev/null
ip link show
*** Killing stale mininet node processes
killall -9 -f mininet:
*** Shutting down stale tunnels
killall -9 -f Tunnel-Ethernet
killall -9 -f .ssh/rm
rm -f -/.ssh/rm/*
*** Cleanup complete.
sepri@Sepri:~$

```

دستور `sudo mn --topo single,6 --mac --switch ovsk` برای ایجاد یک شبیه‌سازی شبکه با توپولوژی مشخصی استفاده می‌شود. این دستور چندین گزینه را ترکیب می‌کند تا شبکه‌ای با ویژگی‌های خاص ایجاد کند. در ادامه توضیح عملکرد هر یک از این گزینه‌ها آمده است:

`--topo single,6`

---topo - برای مشخص کردن توپولوژی شبکه استفاده می شود.

single,6 - به این معنی است که یک سویچ (switch) و ۶ هاست (host) در این توپولوژی وجود دارد. به

عبارت دیگر، همه ۶ هاست به یک سویچ واحد متصل خواهند بود.

---mac :

- این گزینه باعث می شود که MAC آدرس ها به صورت خود کار و متوالی به هاست ها اختصاص داده شوند. این کار به جلوگیری از تداخل و برخورد MAC آدرس ها کمک می کند و مدیریت شبکه را ساده تر می کند.

ovsk ---switch :

---switch - برای مشخص کردن نوع سویچ استفاده می شود.

ovsk - نشان دهنده استفاده از Open vSwitch (OVS) به عنوان سویچ مجازی است. Open vSwitch یک سویچ مجازی پیشرفته است که قابلیت های متعددی برای مدیریت ترافیک شبکه و پیاده سازی پروتکل های مختلف ارائه می دهد.

با اجرای این دستور، Mininet یک شبکه با توپولوژی زیر ایجاد می کند:

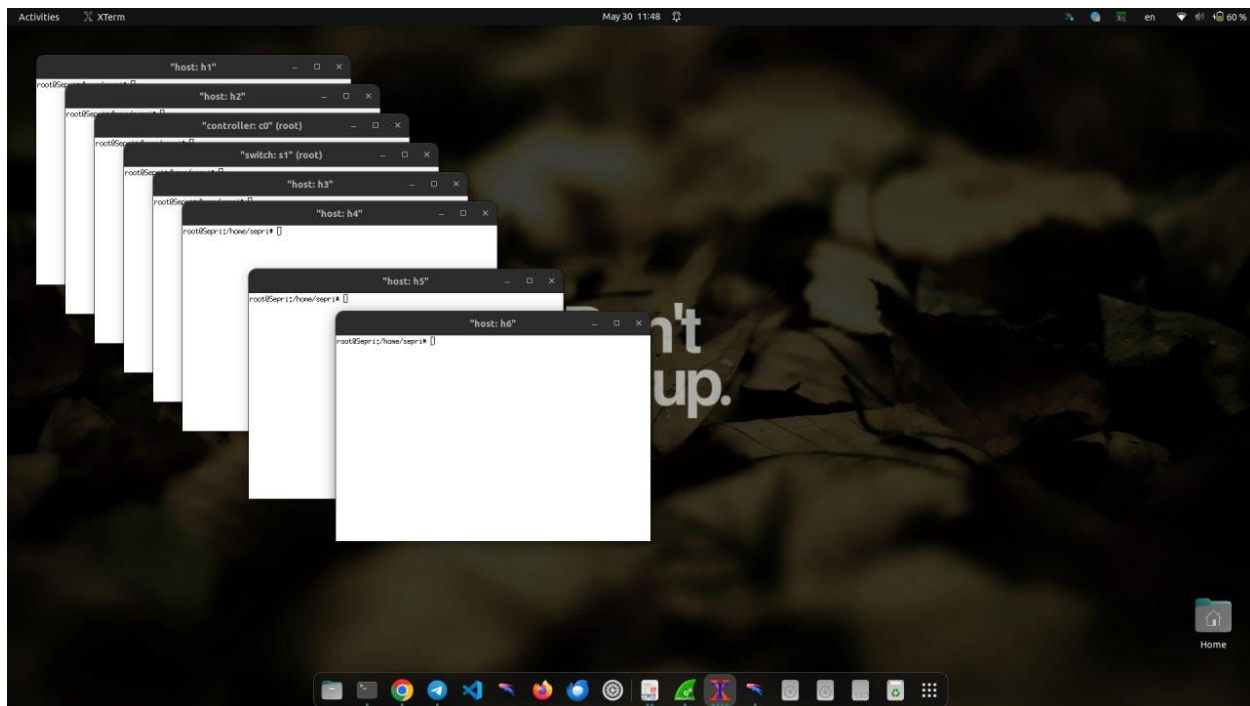
-یک سویچ مجازی که با استفاده از Open vSwitch پیاده سازی شده است.

-شش هاست که هر کدام به این سویچ متصل هستند.

MAC -آدرس های هاست ها به صورت خود کار و متوالی اختصاص داده می شوند.

```
Active internet connections (servers and established)
tcp        0      0 0.0.0.0:6653->0.0.0.0: LISTEN    6486/controller
tcp        0      0 127.0.0.1:60572->127.0.0.1:6653 ESTABLISHED 1026/ovs-vswitchd
tcp        0      0 127.0.0.1:6653->127.0.0.1:6653 ESTABLISHED 6486/controller

*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflow ovs-controller ovs-testcontroller udpbtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflow ovs-controller ovs-testcontroller udpbtest mnexec ivs ryu-manager 2> /dev/null
killall -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -rf /tmp/ovs/* /tmp/vlogs* /tmp/*out /tmp/*log "host: h1"
*** Removing old x11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]*' | sed 's/dp/nl:/'
*** Removing Ovs datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br s1
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([0-9a-z]*-eth[0-9])' | sed 's/eth[0-9]/ethX/'
( ip link del s1-eth1; ip link del s1-eth2; ip link del s1-eth3; ip link del s1-eth4; ip link del s1-eth5 ) 2> /dev/null
ip link show
*** Killing stale mininet node processes
killall -9 -f mininet
*** Shutting down stale tunnels
killall -9 -f Tunnel-Ethernet
killall -9 -f .ssh/rm
rm -rf ~/.ssh/rm/
*** Cleanup complete.
sepr1@sepr1:~$ sudo mn --topo single,6 --controller remote -x
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6653
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Running terms on:
c0
*** Starting controller
c0
*** Starting switches
s1 ...
*** Starting CLI:
mininet>
```



دستور `x sudo mn --topo single,6 --controller remote` در Mininet برای ایجاد یک شبیه‌سازی شبکه با توپولوژی مشخص و کنترلر راه دور استفاده می‌شود. در ادامه توضیح عملکرد هر یک از اجزای این دستور آمده است:

`--topo single,6` :

`--topo` - برای مشخص کردن توپولوژی شبکه استفاده می‌شود.

`single,6` - به این معنی است که یک سویچ (switch) و ۶ هاست (host) در این توپولوژی وجود دارد. به عبارت دیگر، همه ۶ هاست به یک سویچ واحد متصل خواهند بود.

`--controller remote` :

- این گزینه مشخص می‌کند که کنترلر شبکه به صورت راه دور (remote) مدیریت شود. به طور پیش فرض، Mininet از کنترلر محلی استفاده می‌کند، اما با این گزینه، می‌توان کنترلر را به یک سرور راه دور یا دستگاه دیگری متصل کرد که پروتکل‌های کنترل شبکه را اجرا می‌کند (مثل OpenFlow).

`-x` :

- این گزینه Mininet را به گونه‌ای اجرا می‌کند که یک ترمینال گرافیکی `xterm` برای هر هاست و سویچ باز شود. این ترمینال‌ها به کاربران امکان می‌دهند تا دستورات را مستقیماً بر روی هر دستگاه شبکه وارد کنند و خروجی‌ها را مشاهده کنند.

با اجرای این دستور، Mininet یک شبکه با توپولوژی زیر ایجاد می‌کند:

- یک سویچ که به ۶ هاست متصل است.

- کنترلر شبکه به صورت راه دور مدیریت می‌شود.

- برای هر هاست و سویچ، یک پنجره ترمینال `xterm` باز می‌شود.

```
Activities Terminal May 30 11:52
sepr1@Sepri:~$ sudo mn --topo tree,6 --mac --arp
[sudo] password for sepr1:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55
h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s30 s31 s32 s33 s34 s35 s36 s37 s38 s39 s40 s41 s42 s43 s44 s45 s46 s47 s48 s49 s50 s51 s52 s53 s54 s55
s56 s57 s58 s59 s60 s61 s62 s63
*** Adding links:
*****
Caught exception. Cleaning up...
Exception: Error creating interface pair (s1-eth1,s2-eth3): RTNETLINK answers: File exists
*****
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflow ovs-controller ovs-testcontroller udpbtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflow ovs-controller ovs-testcontroller udpbtest mnexec ivs ryu-manager 2> /dev/null
killall -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]*' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br s1
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([0-9a-z]{4}-[0-9a-z]{4})-eth[0-9]{1,2}'
( ip link del s1-eth1; ip link del s1-eth2; ip link del s1-eth3; ip link del s1-eth4; ip link del s1-eth5; ip link del s1-eth6 ) 2> /dev/null
ip link show
*** Killing stale mininet node processes
killall -9 -f mininet
*** Shutting down stale tunnels
killall -9 -f TunnelEthernet
killall -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
sepr1@Sepri:~$
```

دستور `sudo mn --topo tree,6 --mac --arp` در Mininet برای ایجاد یک شبیه‌سازی شبکه با توپولوژی درختی، تنظیم خودکار MAC آدرس‌ها و فعال کردن پروتکل ARP استفاده می‌شود. در ادامه، تجزیه و تحلیل و عملکرد هر یک از اجزای این دستور آمده است:

`--topo tree,6`:

`--topo` - برای مشخص کردن توپولوژی شبکه استفاده می‌شود.

`tree,6` - به این معنی است که یک توپولوژی درختی با ۶ هاست ایجاد خواهد شد. در توپولوژی درختی،

سوئیچ‌ها و هاست‌ها به صورت سلسله‌مراتبی به یکدیگر متصل می‌شوند، به طوری که یک یا چند سوئیچ مرکزی به سوئیچ‌های سطح پایین‌تر و هاست‌ها متصل می‌شوند.

`--mac`:

- این گزینه باعث می‌شود که MAC آدرس‌ها به صورت خودکار و متوالی به هاست‌ها و سوئیچ‌ها اختصاص داده شوند. این کار به جلوگیری از تداخل و برخورد MAC آدرس‌ها کمک می‌کند و مدیریت شبکه را ساده‌تر می‌کند.

:--arp

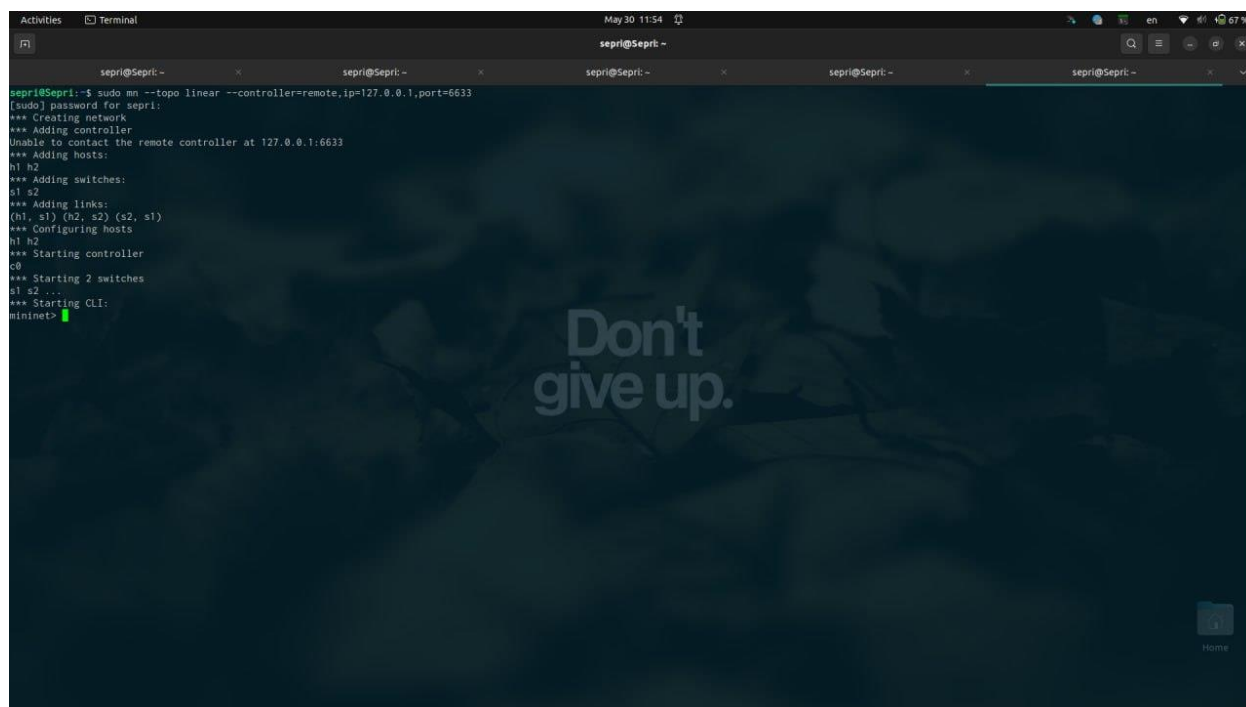
- این گزینه پروتکل ARP (Address Resolution Protocol) را در شبکه فعال می کند. برای تبدیل آدرس های IP به MAC آدرس ها استفاده می شود. با فعال کردن این گزینه، Mininet جدول های ARP را برای هاست ها پر می کند، که این کار باعث می شود تا ارتباطات در شبکه به صورت بهینه تری انجام شوند.

با اجرای این دستور، Mininet یک شبکه با توپولوژی زیر ایجاد می کند:

-یک توپولوژی درختی با ۶ هاست.

MAC -آدرس های هاست ها و سویچ ها به صورت خود کار و متوالی تنظیم می شوند.

-پروتکل ARP در شبکه فعال شده و جدول های ARP هاست ها پر می شود.



```
seprl@seprl:~$ sudo mn --topo linear --controller=remote,ip=127.0.0.1,port=6633
[sudo] password for seprl:
*** Creating network
*** Adding controller
*** Adding hosts
*** Adding switches
*** Adding links
*** Configuring hosts
*** Starting controller
*** Starting 2 switches
*** Starting CLI:
mininet>
```

دستور `sudo mn --topo linear --controller=remote,ip=127.0.0.1,port=6633` در Mininet برای ایجاد یک شبیه‌سازی شبکه با توپولوژی خطی (linear) و استفاده از یک کنترلر راه دور با تنظیمات مشخص استفاده می‌شود. در ادامه، تجزیه و تحلیل و عملکرد هر یک از اجزای این دستور آمده است:

`--topo linear` :

`--topo` - برای مشخص کردن توپولوژی شبکه استفاده می‌شود.

`linear` - به این معنی است که توپولوژی شبکه به صورت خطی خواهد بود. در این توپولوژی، سویچ‌ها و هاست‌ها به صورت سریال به یکدیگر متصل می‌شوند. به عنوان مثال، هاست ۱ به سویچ ۱، سویچ ۱ به سویچ ۲، سویچ ۲ به هاست ۲ و به همین ترتیب ادامه دارد.

`--controller=remote,ip=127.0.0.1,port=6633` :

`--controller` - برای مشخص کردن تنظیمات کنترلر استفاده می‌شود.

`remote` - نشان‌دهنده این است که کنترلر شبکه به صورت راه دور قرار دارد.

`ip=127.0.0.1` - نشان می‌دهد که کنترلر در همان ماشینی قرار دارد که Mininet اجرا می‌شود (localhost).

`port=6633` - پورت استفاده شده برای ارتباط با کنترلر را مشخص می‌کند. پورت ۶۶۳۳ به طور پیش فرض برای OpenFlow کنترلرها استفاده می‌شود.

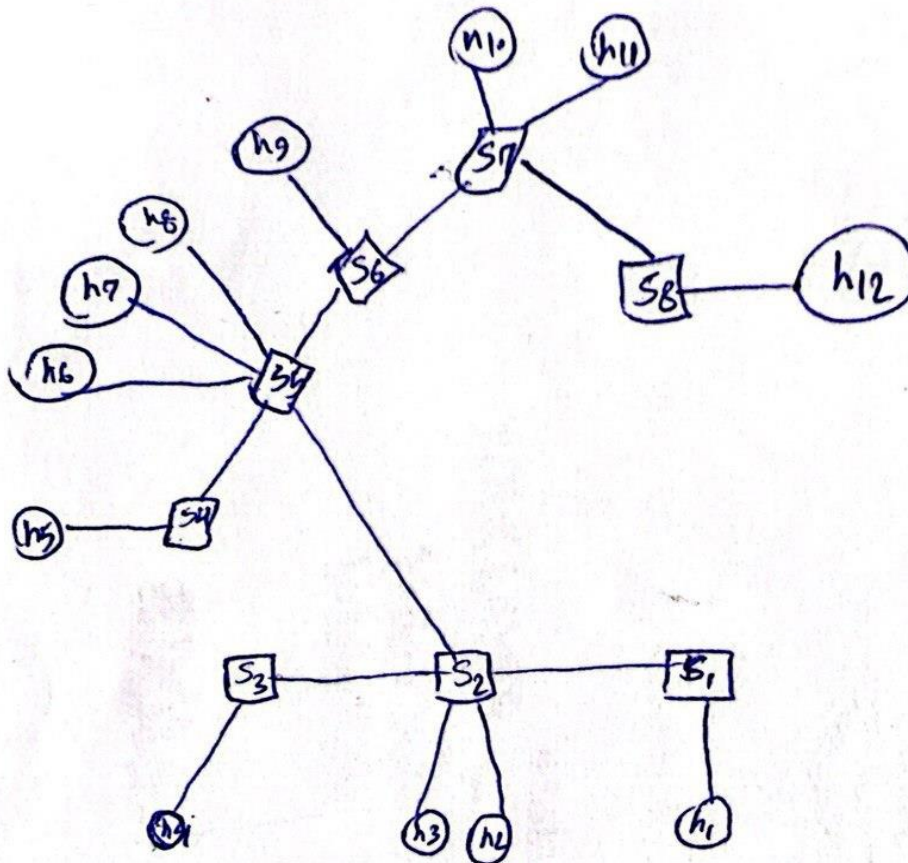
با اجرای این دستور، Mininet یک شبکه با توپولوژی زیر ایجاد می‌کند:

-یک توپولوژی خطی با سویچ‌ها و هاست‌ها که به صورت سریال به هم متصل شده‌اند.

-کنترلر شبکه به صورت راه دور بر روی IP آدرس ۱۲۷.۰.۰.۱ (localhost) و پورت ۶۶۳۳ تنظیم می‌شود.

ε

1-ε



```

link = cls(model1, node2, **options...)
File "/usr/local/lib/python3.10/dist-packages/mininet/link.py", line 567, in __init__
link.__init__(self, sargs, **kwargs)
File "/usr/local/lib/python3.10/dist-packages/mininet/link.py", line 457, in __init__
self.makeIntfPair(intfName1, intfName2, addr1, addr2)
File "/usr/local/lib/python3.10/dist-packages/mininet/link.py", line 502, in makeIntfPair
return makeIntfPair(intfName1, intfName2, addr1, addr2, model1, node2)
File "/usr/local/lib/python3.10/dist-packages/mininet/util.py", line 282, in makeIntfPair
raise Exception("Error creating interface pair (%s,%s): %s" %
Exception: Error creating interface pair (s2-eth2,s3-eth1): RTNETLINK answers: File exists

seprl@seprl:~/Documents/VSCode$ sudo mn -c
*** Removing excess controllers/ofdatapaths/pings/noxes.
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflow ovs-controller ovs-testcontroller udpbtest mnexec ivs ryu-manager 2> /dev/null
killall -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[8-9]*' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br s1 --if-exists del-br s2
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_[:alnum:]]+)-eth[[:digit:]]+'
( ip link del s1-eth2; ip link del s2-eth2; ip link del s2-eth2; ip link del s1-eth2; ip link del s1-eth1; ip link del s2-eth1 ) 2> /dev/null
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet
*** Shutting down stale tunnels
pkill -9 -f TunnelEthernet
pkill -9 -f ssh/am
rm -f ~/.ssh/am/*
*** Cleanup complete.
seprl@seprl:~/Documents/VSCode$ sudo python Miniset-Python-Q4.py
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding switches
*** Adding hosts
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controller
c0
*** Starting 8 switches
s1 s2 s3 s4 s5 s6 s7 s8 ...
*** Running CLI
*** Starting CLI:
mininet>

```

۲-۴

```

seprl@seprl:~/Documents/VSCode$ cd Documents/
seprl@seprl:~/Documents$ cd VS
bash: cd: VS: No such file or directory
seprl@seprl:~/Documents$ cd VSCode/
seprl@seprl:~/Documents/VSCode$ ls
Miniset-Python-Q4.py test.py
seprl@seprl:~/Documents/VSCode$ sudo python Miniset-Python-Q4.py
[sudo] password for seprl:
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding switches
*** Adding hosts
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12
*** Starting controller
c0
*** Starting 8 switches
s1 s2 s3 s4 s5 s6 s7 s8 ...
*** Running CLI
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5097ms
pipe 4
mininet> h8 ping h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
From 10.0.0.8 icmp_seq=1 Destination Host Unreachable
From 10.0.0.8 icmp_seq=2 Destination Host Unreachable
From 10.0.0.8 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.0.7 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5107ms
pipe 4
mininet> s2 ping s5
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.076 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.054 ms
^C
--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3065ms
rtt min/avg/max/mdev = 0.028/0.058/0.076/0.019 ms
mininet> h2 ping h7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable

```

مشاهده اینکه زمان پینگ برای اولین تلاش بیشتر از تلاش‌های بعدی است، ممکن است به دلایل زیر باشد:

۱. راه‌اندازی اولیه شبکه: وقتی شبکه برای اولین بار راه‌اندازی می‌شود، دستگاه‌های مختلف (سوئیچ‌ها، میزبان‌ها و کنترلر) ممکن است هنوز در حال راه‌اندازی و پیکربندی باشند. این فرایند می‌تواند زمان ببرد و باعث شود پینگ اولیه بیشتر طول بکشد.

۲. آرپ (ARP) درخواست: اولین پینگ بین دو دستگاه معمولاً شامل درخواست ARP برای پیدا کردن آدرس MAC مقصد است. این درخواست و پاسخ آن زمان‌بر است و باعث می‌شود پینگ اولیه بیشتر طول بکشد. پس از دریافت آدرس MAC، پینگ‌های بعدی سریع‌تر انجام می‌شوند زیرا دستگاه‌ها می‌دانند چگونه به مقصد برسند.

۳. ساخت جدول جریان در سوئیچ‌ها: در شبکه‌های SDN و مخصوصاً وقتی از سوئیچ‌های OVS استفاده می‌کنیم، اولین بسته‌ای که از طریق یک سوئیچ عبور می‌کند، ممکن است نیاز به مشورت با کنترلر برای تصمیم‌گیری مسیر داشته باشد. این فرایند زمان می‌برد و باعث می‌شود اولین پینگ طولانی‌تر شود. پس از اینکه جریان (Flow) در سوئیچ نصب شد، بسته‌های بعدی بدون نیاز به مشورت با کنترلر فوراً مسیریابی می‌شوند.

به طور کلی، اولین پینگ شامل فعالیت‌های پیکربندی و اکتشافی است که در تلاش‌های بعدی نیاز به تکرار ندارد. به همین دلیل زمان اولین پینگ بیشتر از تلاش‌های بعدی است.


```

Activities Terminal May 30 14:14
seprl@Sepri: ~/Documents/VSCode

4 packets transmitted, 4 received, 0% packet loss, time 3065ms
rtt min/avg/max/mdev = 0.028/0.058/0.076/0.019 ms
mininet> h2 ping h7
PING 10.0.0.7 (10.0.0.7) 56(64) bytes of data:
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.0.7 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4080ms
pipe 4
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=4776>
<Host h2: h2-eth0:10.0.0.2 pid=4778>
<Host h3: h3-eth0:10.0.0.3 pid=4780>
<Host h4: h4-eth0:10.0.0.4 pid=4782>
<Host h5: h5-eth0:10.0.0.5 pid=4784>
<Host h6: h6-eth0:10.0.0.6 pid=4786>
<Host h7: h7-eth0:10.0.0.7 pid=4788>
<Host h8: h8-eth0:10.0.0.8 pid=4790>
<Host h9: h9-eth0:10.0.0.9 pid=4792>
<Host h10: h10-eth0:10.0.0.10 pid=4794>
<Host h11: h11-eth0:10.0.0.11 pid=4796>
<Host h12: h12-eth0:10.0.0.12 pid=4798>
<OVSSwitch s1: 10.127.0.0.1,s1-eth1:None,s1-eth2:None,s2-eth3:None,s2-eth4:None,s2-eth5:None pid=4750>
<OVSSwitch s2: 10.127.0.0.1,s2-eth1:None,s2-eth2:None,s3-eth3:None,s3-eth4:None,s3-eth5:None pid=4753>
<OVSSwitch s3: 10.127.0.0.1,s3-eth1:None,s3-eth2:None,s4-eth3:None,s4-eth4:None,s4-eth5:None pid=4756>
<OVSSwitch s4: 10.127.0.0.1,s4-eth1:None,s4-eth2:None,s5-eth3:None,s5-eth4:None,s5-eth5:None pid=4759>
<OVSSwitch s5: 10.127.0.0.1,s5-eth1:None,s5-eth2:None,s6-eth3:None,s6-eth4:None,s6-eth5:None pid=4762>
<OVSSwitch s6: 10.127.0.0.1,s6-eth1:None,s6-eth2:None,s7-eth3:None,s7-eth4:None,s7-eth5:None pid=4765>
<OVSSwitch s7: 10.127.0.0.1,s7-eth1:None,s7-eth2:None,s8-eth3:None,s8-eth4:None,s8-eth5:None pid=4768>
<OVSSwitch s8: 10.127.0.0.1,s8-eth1:None,s8-eth2:None,s9-eth3:None,s9-eth4:None,s9-eth5:None pid=4771>
<RemoteController c0: 127.0.0.1:6633 pid=4743>
mininet> nodes
available nodes are:
c0 h1 h10 h11 h12 h2 h3 h4 h5 h6 h7 h8 h9 s1 s2 s3 s4 s5 s6 s7 s8
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X X X X
h2 -> X X X X X X X X X X
h3 -> X X X X X X X X X X
h4 -> X X X X X X X X X X
h5 -> X X X X X X X X X X
h6 -> X X X X X X X X X X
h7 -> X X X X X X X X X X
h8 -> X X X X X X X X X X
h9 -> X X X X X X X X X X
h10 -> X X X X X X X X X X
h11 -> X X X X X X X X X X
h12 -> X X X X X X X X X X
*** Results: 100% dropped (0/132 received)
mininet>

```