پروژه اول - طراحی، شبیه سازی و تحلیل پیشرفته توپولوژی های شبکه با Mininet و SDN

مجتبی ملائی ۴۰۱۳۱۳۸۳

سوال اول:

اجرای دستور : `mininet> nodes`

```
available nodes are:
h1 h2 h3 h4 h5 h6 h7 s1
```

این دستور به ما نودها (اجزای شبکه) را نشان می‌دهد. در این شبکه ۷ هاست(میزبان) با نام های **h**۱ تا **h**۷ مشاهده می‌شود. همچنین یک سوئیچ **s**۱ نیز وجود دارد.

اجرای دستور : `mininet> net`

```
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
h5 h5-eth0:s1-eth5
h6 h6-eth0:s1-eth6
h7 h7-eth0:s1-eth7
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-
eth4:h4-eth0 s1-eth5:h5-eth0 s1-eth6:h6-eth0 s1-eth7:h7-eth0
```

با اجرای این دستور توپولوژی شبکه نمایش داده خواهد شد. در این خروجی هر یک از نود ها در هر خط وجود دارد و سپس اتصالات هر نود نمایش داده شده است.

اجرای دستور : `mininet> dump`

```
<Host h1: h1-eth0:10.0.0.1 pid=7286>
<Host h2: h2-eth0:10.0.0.2 pid=7293>
```

```
<Host h3: h3-eth0:10.0.0.3 pid=7302>
<Host h4: h4-eth0:10.0.0.4 pid=7310>
<Host h5: h5-eth0:10.0.0.5 pid=7316>
<Host h6: h6-eth0:10.0.0.6 pid=7322>
<Host h7: h7-eth0:10.0.0.7 pid=7328>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-
eth3:None,s1-eth4:None,s1-eth5:None,s1-eth6:None,s1-eth7:None
pid=7281>
```

این دستور اطلاعات دقیق و کاملی در مورد هر نود شبکه به ما میدهد. از جمله آن می‌توان به نام، نوع، اینترفیس، آدرس **IP**، شماره پروسس اشاره کرد. در این خروجی سوئیچ به دلیل آنکه آدرس آیپی ندارد، به ازای هر اینترفیسش **None** آمده است. همچنین نوع سوئیچ **OVSSwitch** می‌باشد.

اجرای دستور : `mininet> pingall`

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
```

این دستور به ازای هر هاست شبکه، اتصال به هاست های دیگر شبکه را با پینگ بررسی میکند. خروجی این دستور نشان می‌دهد که تمامی هاست ها به یکدیگر متصل هستند.

بررسی پایداری لینک بین **h۱** و **s۱**:

قبل از قطع لینک داریم:

```
mininet> sh ovs-appctl fdb/show  s1
port  VLAN  MAC                Age
   3     0  5a:9a:58:7a:2d:16    2
   1     0  ee:aa:85:58:cf:6a    2
```

```
   4      0  f6:e1:b3:80:d3:32    2
   5      0  4a:f4:73:eb:66:dc    2
   6      0  2e:dd:aa:08:f1:4d    2
   7      0  3a:be:1a:42:6f:4c    2
   2      0  5e:df:71:5b:c9:9f    2
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.354 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.062 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.059 ms
^C
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7084ms
rtt min/avg/max/mdev = 0.059/0.112/0.354/0.092 ms
```

در این خروجی در پورت ۱ آدرس مک **h**۱ مشاهده می‌شود. همچنین متوسط **RTT** بین **h**۱ و **h**۲ نیز برابر با ۰.۱۱۲ میلی ثانیه است.

بعد از قطع لینک:

```
mininet> link s1 h1 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X
h2 -> X h3 h4 h5 h6 h7
h3 -> X h2 h4 h5 h6 h7
h4 -> X h2 h3 h5 h6 h7
h5 -> X h2 h3 h4 h6 h7
h6 -> X h2 h3 h4 h5 h7
h7 -> X h2 h3 h4 h5 h6
*** Results: 28% dropped (30/42 received)
```

```
mininet> sh ovs-appctl fdb/show  s1
port  VLAN  MAC                Age
   1     0  ee:aa:85:58:cf:6a   98
   2     0  5e:df:71:5b:c9:9f    5
   3     0  5a:9a:58:7a:2d:16    5
   4     0  f6:e1:b3:80:d3:32    5
   5     0  4a:f4:73:eb:66:dc    5
   6     0  2e:dd:aa:08:f1:4d    4
   7     0  3a:be:1a:42:6f:4c    4
mininet> h1 ping h2
ping: connect: Network is unreachable
```

همانطور که دیده می‌شود، هاست **h**۱ به هیچ یک از هاست های دیگر دسترسی ندارد و همچنین هاست دیگر نیز به آن دسترسی ندارند. همچنین در جدول **MAC** نیز **age** آن ۹۸ است که نشان می‌دهد این **MAC** آدرس مربوط به خیلی وقت قبل است. همچنین **h**۱ به **h**۲ نمیتواند پینگ بگیرد و نمی‌توان **RTT** برای آن مشخص کرد.

بعد از اتصال مجدد:

```
mininet> link s1 h1 up
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7
h2 -> h1 h3 h4 h5 h6 h7
h3 -> h1 h2 h4 h5 h6 h7
h4 -> h1 h2 h3 h5 h6 h7
h5 -> h1 h2 h3 h4 h6 h7
h6 -> h1 h2 h3 h4 h5 h7
h7 -> h1 h2 h3 h4 h5 h6
*** Results: 0% dropped (42/42 received)
mininet> sh ovs-appctl fdb/show  s1
port  VLAN  MAC                Age
   6     0  2e:dd:aa:08:f1:4d    8
   7     0  3a:be:1a:42:6f:4c    8
   5     0  4a:f4:73:eb:66:dc    8
   4     0  f6:e1:b3:80:d3:32    8
```

```
   3      0  5a:9a:58:7a:2d:16    8
   2      0  5e:df:71:5b:c9:9f    8
   1      0  ee:aa:85:58:cf:6a    3
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.426 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.067 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6092ms
rtt min/avg/max/mdev = 0.056/0.121/0.426/0.124 ms
```

همانطور که دیده می‌شود پس از اتصال مجدد، همه هاست ها به **h**۱ و برعکس دسترسی دارند. همچنین **age** مربوط به پورت ۱ دوباره کم شده است که این نشان می‌دهد اخیرا از آن بسته عبور کرده است. همچنین **RTT** میانگین پس از اتصال برابر با ۰.۱۲۱ میلی ثانیه است.

بررسی دستور **dump :**

```
sudo python3 base_topo.py  --host 10
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=7818>
<Host h2: h2-eth0:10.0.0.2 pid=7824>
<Host h3: h3-eth0:10.0.0.3 pid=7835>
<Host h4: h4-eth0:10.0.0.4 pid=7841>
<Host h5: h5-eth0:10.0.0.5 pid=7847>
<Host h6: h6-eth0:10.0.0.6 pid=7853>
<Host h7: h7-eth0:10.0.0.7 pid=7859>
<Host h8: h8-eth0:10.0.0.8 pid=7865>
<Host h9: h9-eth0:10.0.0.9 pid=7871>
```

```
<Host h10: h10-eth0:10.0.0.10 pid=7879>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-
eth3:None,s1-eth4:None,s1-eth5:None,s1-eth6:None,s1-
eth7:None,s1-eth8:None,s1-eth9:None,s1-eth10:None pid=7813>
```

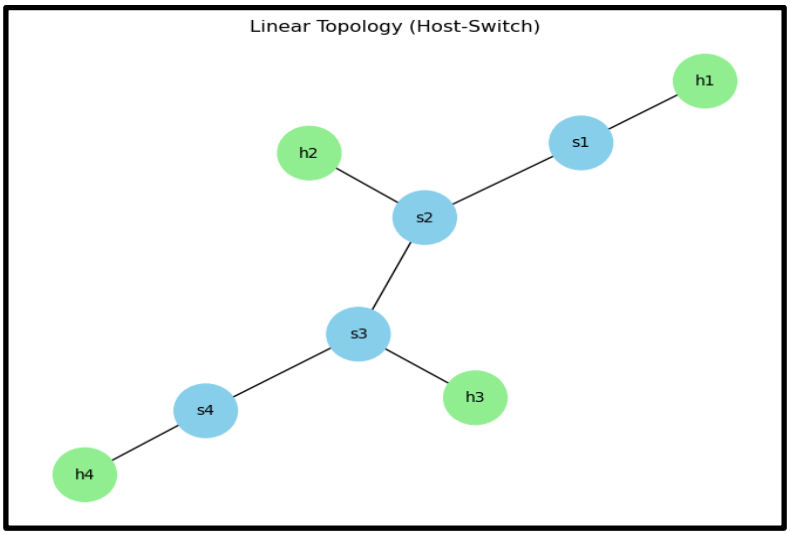
همانند حالت قبل است با این تفاوت که این‌بار ۱۰ هاست داریم و در **s**۱ نیز ۱۰ **interface** داریم.

سوال مفهومی:

در توپولوژی **Single**، هر میزبان مستقیماً به یک سوئیچ مرکزی متصل است. بنابراین، بیشترین فاصله بین هر دو میزبان (قطر شبکه) همیشه **۲** است (از میزبان اول به سوئیچ و از سوئیچ به میزبان دوم). اما در توپولوژی **Linear**، سوئیچ ها به صورت زنجیره‌ای و پشت‌سرهم متصل هستند و هر میزبان نیز به یک سوئیچ متصل میگردد. در این حالت، فاصله بین دورترین میزبان‌ها برابر با **n+1** (که **n** تعداد میزبان‌هاست) می‌شود. بنابراین، با افزایش تعداد میزبان‌ها، قطر شبکه نیز بزرگ‌تر می‌شود.

سوال دوم:

توپولوژی **Linear**:

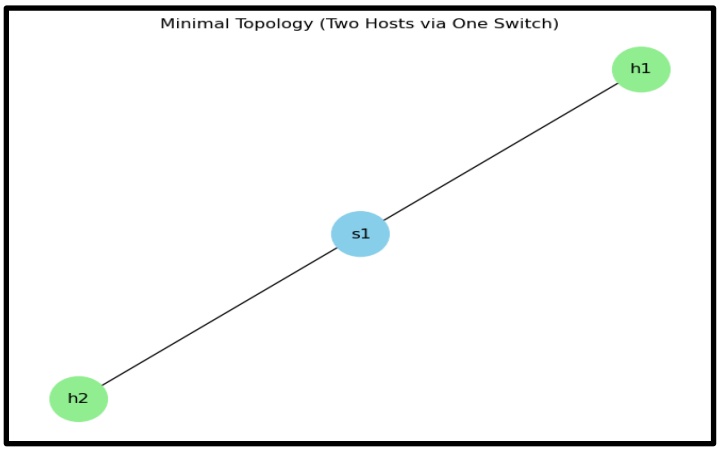


درجه هر نود: **h**۱ تا **h**۴: یک — **s**۱ و **s**۴: دو — **s**۲ و **s**۳: سه

قطر شبکه: از **h**۱ تا **h**۴: پنج

قابلیت تحمل خرابی: اگر سوئیچ های انتهایی دچار مشکل شوند، فقط یک هاست از کار می‌افتد اما اگر یک لینک بین دو سوئیچ میانی یا یکی از سوئیچ های میانی دچار مشکل شود، ارتباط نیمه اول شبکه با نیمه دوم شبکه از دست می‌رود.

توپولوژی **Minimal**:

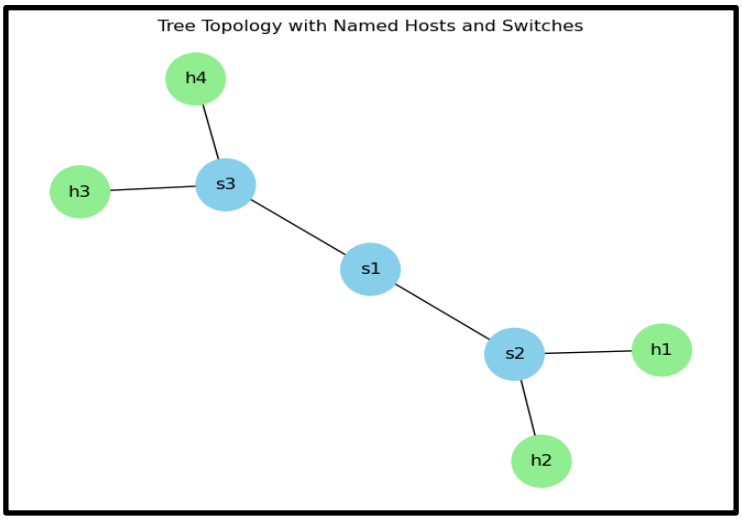


درجه هر نود: **h**۱ تا **h**۴: یک — **s**۱: دو

قطر شبکه: از **h**۱ تا **h**۲: دو

قابلیت تحمل خرابی: تمام شبکه وابسته به **s**۱ است و اگر از کار بی‌افتد ارتباط بین **h**۱ و **h**۲ از بین می‌رود.

توپولوژی **Tree**:

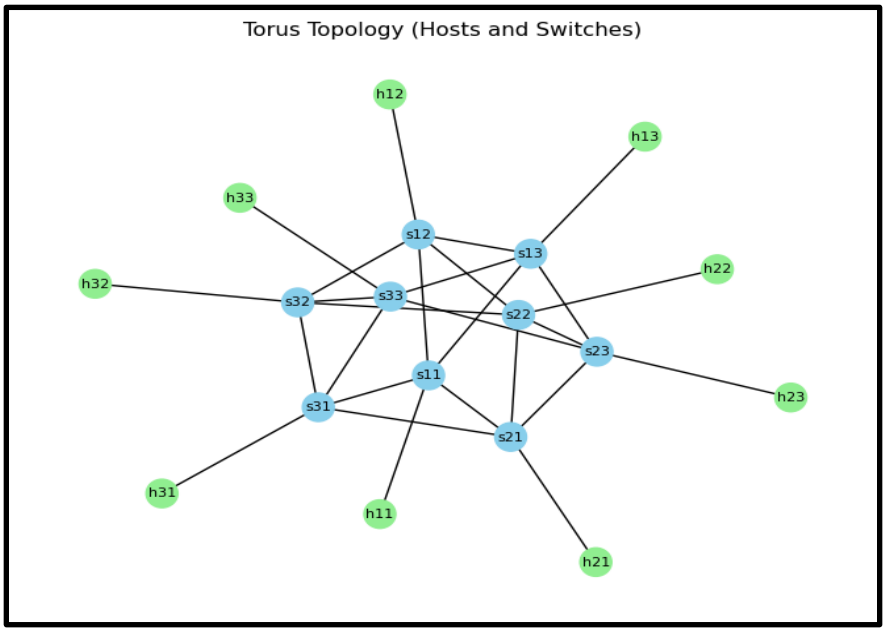


درجه هر نود: h۱ تا h۴: یک — s۲ و s۳: سه — s۱: یک

قطر شبکه: از h۱ تا h۴: چهار

قابلیت تحمل خرابی: اگر نود روت یعنی s۱ دچار اختلال شود، نیمه راست و چپ شبکه دیگر نمی‌توانند ارتباط برقرار کنند. به صورت کلی تر هر نودی که دچار اختلال شود، ارتباط بین نیمه راست، چپ و بقیه شبکه مختل می‌شود. اگر فقط یک لینک دچار اختلال شود، ارتباط بین درخت پایین لینک و بقیه درخت اصلی قطع می‌شود.

توپولوژی Torus:

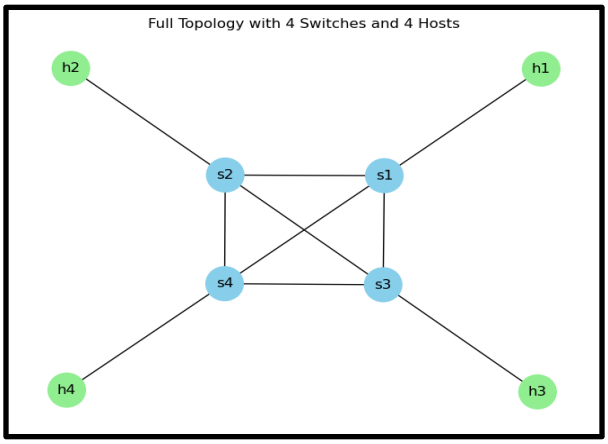


درجه هر نود: h۱۱ تا h۳۳: یک — s۱۱ تا s۳۳: پنج

قطر شبکه: از h۱۱ تا h۳۳: چهار

قابلیت تحمل خرابی: قابلیت تحمل خرابی بسیار بالایی دارد. اگر یکی از سوئیچ ها خراب شود، فقط هاست مربوط به آن قطع می‌شود و اگر یک لینک بین دو سوئیچ خراب شود، هیچ مشکلی برای شبکه به وجود نمی‌آید زیرا ۳ لینک جایگزین دیگر برای آن وجود دارد. در کل به دلیل مسیر های زیاد بین سوئیچ ها تحمل خرابی بسیار زیادی دارد.

توپولوژی **Full**:

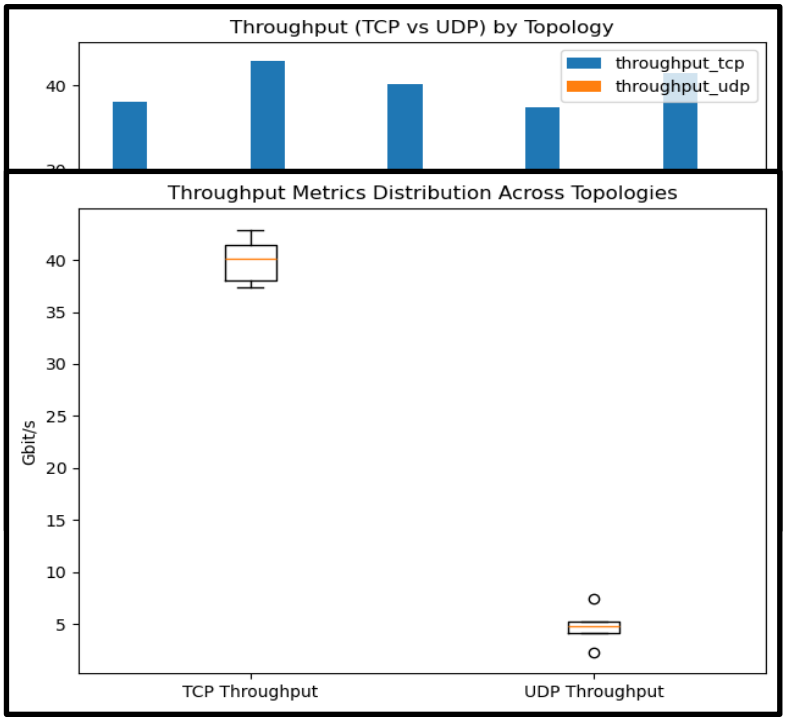


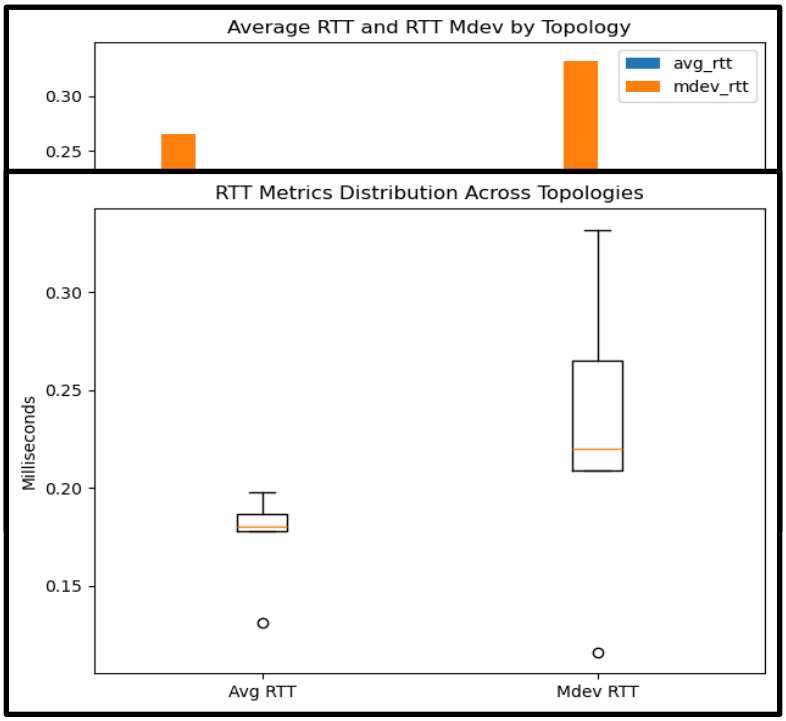
درجه هر نود: **h**۱ تا **h**۴: یک — **s**۱ تا **s**۴: چهار

قطر شبکه: از **h**۱ تا **h**۴: سه

قابلیت تحمل خرابی: از قابلیت تحمل خرابی بالایی برخوردار است. اگر یکی از سوئیچ ها خراب شود، فقط همان هاست وصل شده به سوئیچ قطع می‌شود. اگر لینک بین دو سوئیچ قطع شود، دو لینک جایگزین دیگر وجود دارد. هرچقدر توپولوژی بزرگ تر شود، تحمل خرابی آن نیز افزایش می‌یابد. به طور کلی به دلیل اینکه هر سوئیچ به تمام سوئیچ های دیگر شبکه متصل است، تحمل خرابی آن بسیار زیاد است.

مقایسه نتایج:





مقایسه توپولوژی ها:

Linear به دلیل سادگی و هزینه پایین، بیشتر در شبکه‌های کوچک مانند آزمایشگاه‌ها، سنسورها یا سیستم‌های موقت کاربرد دارد. **Minimal** که طراحی بهینه‌تری دارد، در مراکزی که بودجه یا منابع محدود است و همچنان عملکرد قابل قبول نیاز دارند، مانند شبکه‌های داخلی یا برخی طراحی‌های مرکز داده، استفاده می‌شود. **Tree** به خاطر ساختار سلسله‌مراتبی‌اش، در شبکه‌های سازمانی، سیستم‌های دانشگاهی و مراکز داده بزرگ که نیاز به مدیریت چند لایه دارند، رایج است.

در سطح پیشرفته‌تر، **Torus** به دلیل قابلیت تحمل خطا و مسیرهای متعدد بین گره‌ها، در سیستم‌های محاسبات با عملکرد بالا (**HPC**) مثل ابررایانه‌ها یا خوشه‌های محاسباتی استفاده می‌شود. و در نهایت، **Full** که همه گره‌ها را مستقیماً به هم متصل می‌کند، با وجود هزینه و پیچیدگی زیاد، در سیستم‌های حیاتی که نیاز به ارتباط بی‌وقفه، کم‌تأخیر و بسیار پایدار دارند، مانند شبکه‌های نظامی، زیرساخت‌های مالی یا کنترل صنعتی کاربرد دارد.

سوال سوم:

برای این سوال از توپولوژی **full** به دلیل وجود مسیر های متنوع و مسیریابی ساده تر استفاده شده است.

کاهش پهنای باند:

Controller: step3/c_qos.py, Mininet: step3/full.py

```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.0 Mbits/sec', '1.0 Mbits/sec']
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['1.0 Mbits/sec', '1.0 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['84.1 Gbits/sec', '83.8 Gbits/sec']
```

پهنای باند هاست **h۱** به یک مگابیت بر ثانیه کاهش یافته است.

تفکیک ترافیک برای load balancing:

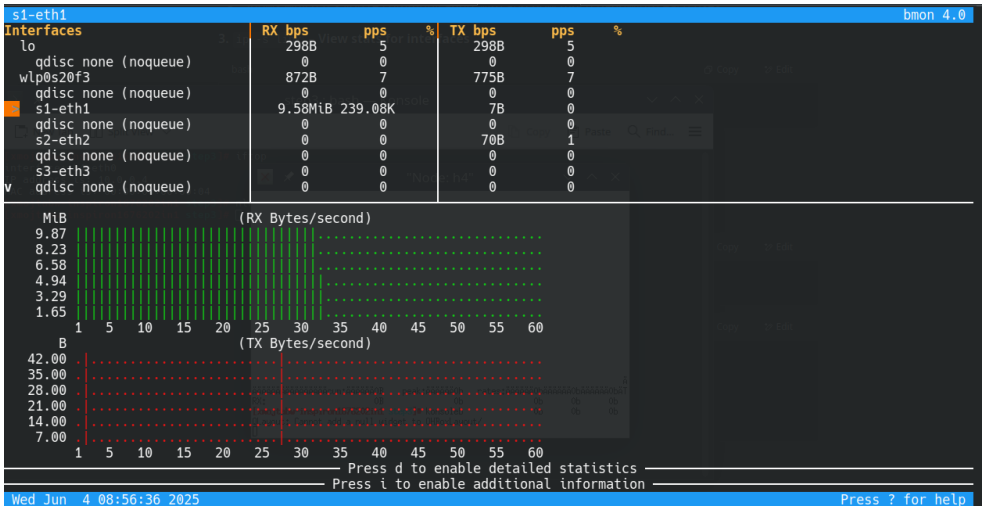
برای اینکه ترافیک **ssh** و **http** تفکیک شوند و از دو مسیر متفاوت به مقصد برسند نیاز داریم تا فلو های متفاوتی برای هر کدام تعریف کنیم. در اینجا برای بر روی **s۱** و **s۴** یک فلوی جداگانه برای **ssh** در نظر گرفته شد. هنگامی که ترافیک **ssh** از **h۱** به **h۴** یا برعکس داشته باشیم، بجای استفاده از پورت ۴ در **s۱** و پورت ۱ در **s۴** از پورت ۲ هر دو سوئیچ (ارسال به **s۲**) استفاده می کنیم. بدین ترتیب ترافیک مربوط به **ssh** از سوئیچ **s۲** عبور می‌کند و ترافیک های دیگر از جمله **Http** از لینک مستقیم بین **s۱** و **s۴**. فلو های گفته شده با بررسی شماره پورت مقصد و یا منبع **TCP** میتوانند این ترافیک ها را تفکیک کنند. فایل **s1-2.pcap** شامل پکت های ارسالی بر روی پورت دوم **s۱** (متصل به **s۲**)است که در آن ترافیک **ssh** مشاهده می‌شود و فایل **s1-4.pcap** مربوط به پورت ۴ **s۱** (متصل به**s۴**) است که شامل **ping** ها و **http** است.

حمله UDP Flooding:

در این حمله از کنترلر **c_base.py** و توپولوژی **full.py** استفاده شد. در هاست **h۱** از کامند زیر استفاده شد:

```
hping3 --udp -p 53 -a 10.0.0.2 --flood 10.0.0.4
```

در این کامند، بسته های**UDP** به پورت ۵۳ آیپی ۱۰.۰.۰.۴ میروند و آدرس مبدا آنها برای ناشناس ماندن حمله کننده



۱۰.۰.۰.۲ است.

در حالتی که حمله در جریان نیست:

```
[  7]   0.00-10.00  sec  50.4 GBytes  42.4 Gbits/sec receiver
```

در حالتی که حمله در جریان است:

```
[  7]   0.00-10.00  sec  47.5 GBytes  40.8 Gbits/sec receiver
```

نشان میدهد که پهنای باند حدود ۲.۹ گیگابایت کاهش یافته است. *(هرچند به دلیل اینکه این آزمایش در* **miminet** *به صورت محلی و بدون عبور از سیم انجام میشود، این کاهش شاید کم بنظر برسد(نسبت به ۴. ۵۰ گیگابایت پهنای باند اصلی) اما در عمل این عدد بسیار بزرگ است. همچنین واقع* **TCP** *در کرنل لینوکس به دلیل بهینه سازی های انجام شده تقریبا به صورت* **zero-copy** *انجام می‌شود و سرعت بسیار بالایی نسبت به* **UDP** *دارد. و در این پروژه هم در مقایسه* **throughput** *ها سرعت* **TCP** *بسیار بیشتر بود.)*

مراحل تشخیص:

1. اشغال زیاد پهنای باند.
2. بسته های **UDP** غیر عادی زیاد.

تحلیل بسته ها:

1. بسته ها را به کمک **tcpdump** ذخیره می‌کنیم
2. بررسی بازه زمانی بسته ها (آیا این بسته ها به صورت **burst** کوتاه هستند و یا به صورت مداوم و با نرخ زیاد ارسال شده اند.)
3. بررسی آدرس مبدا و مقصد (چه کسی بسته ها را ارسال می‌کند؟ آیا از **IP** جعلی **(spoofing)** استفاده کرده؟ آیا یک مقصد خاص هدف حمله است یا کل شبکه؟)
4. بررسی پورت مبدا و مقصد (آیا یک سرویس/سرویس های خاص مدنظر است یا به همه پورت ها ارسال می‌شود؟)
5. بررسی **payload** ها (چه چیزی ارسال می‌شود؟)

دفاع:

برای یک دفاع سریع می‌توان با تنظیم یک فایروال نرخ بسته های **UDP** را در مقصد و یا مبدا کاهش داد. اگر از مبدا مطمئن هستیم با تنظیم فایروال بر روی سوئیچ **access** و یا در **OpenFlow**، حمله کننده را مسدود نماییم. اگر بخواهیم از چنین حملاتی جلوگیری شود، میتوانیم با استفاده از **OpenFlow** تعیین کنیم که بسته هایی فقط ارسال شوند که **IP** مبدا آنها با **IP** ارسالی آن هاست متصل یکسان باشد. در این حالت حمله کننده نمی‌تواند **IP** خود را جعل کند اما همچنان قادر به ارسال بسته های **UDP** و **UDP flooding** می‌باشد ولی تشخیص حمله کننده ساده تر می‌شود و میتوان آن را مسدود کرد. اگر بخواهیم بیشتر جلوگیری کنیم، میتوانیم نرخ ارسال بسته های **UDP** را با **OpenFlow** کاهش دهیم.