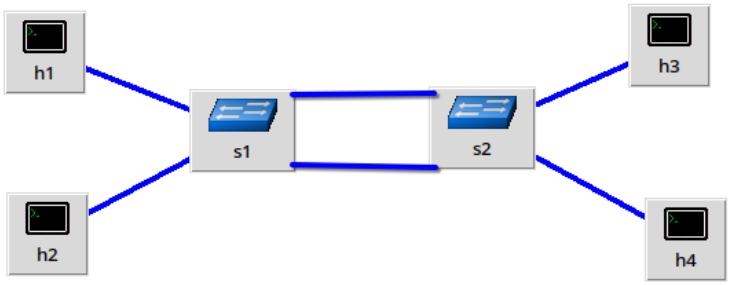
پروژه سوم

مجتبی ملائی
۴۰۱۳۱۳۸۳

الف) تصویر شماتیک از توپولوژی. در این توپولوژی سوئیچ ها در حالت `failmode="standalone"` و `stp=True` هستند.

`topo.py`



ب) نتیجه خروجی دستور `net` و دستور `pingall` :

```
~/st/term6/CN/p/phase3 main !1 ?1 sudo python topo.py
*** Adding switches
*** Adding links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller

*** Starting 2 switches
s1 s2 ...
*** Running CLI
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth3
h2 h2-eth0:s1-eth4
h3 h3-eth0:s2-eth3
h4 h4-eth0:s2-eth4
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s2-eth2 s1-eth3:h1-eth0 s1-eth4:h2-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s1-eth2 s2-eth3:h3-eth0 s2-eth4:h4-eth0
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

شکل ۱: با توجه پروتکل STP کمی طول می‌کشد تا شبکه کاملا پایدار شود

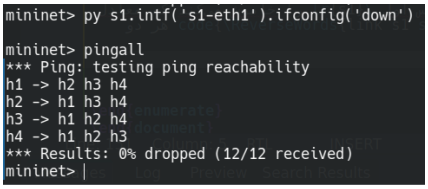
پ) جدول زیر مقدار میانگین تاخیر برای مسیر های‌مختلف نشان می‌دهد.

route	time (ms)
h1-s1-h2	0.152
h1-s1-s2-h3	0.205
h1-s1-s2-h4	0.247
h2-s1-s2-h3	0.196
h2-s1-s2-h4	0.224
h3-s2-h4	0.146

ت) جدول زیر مقدار پهنای باند (Bandwidth) بین میزبان‌های مختلف نشان می‌دهد.

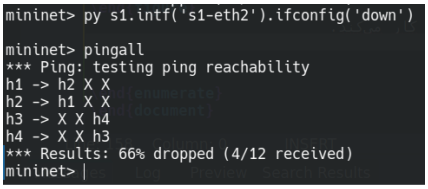
src	dst	bandwidth (Gbits/sec)
h1	h2	43.6
h1	h3	41.8
h1	h4	41.4
h2	h3	41.7
h2	h4	41.2
h3	h4	44.1

ث) با استفاده از دستور `py s1.intf('s1-eth1').ifconfig('down')` اینترفیس اول s1 را خاموش می‌کند که معادل قطع سیم اول است. (دستور `link s1 s2 down` هر دو لینک را قطع می‌کند.)



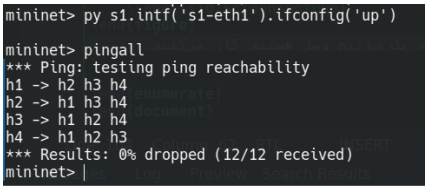
شکل ۲: شبکه بعد از قطع سیم اول

پس از گذشت مدتی زمان، همانطور که دیده می‌شود شبکه همچنان به طور کامل کار می‌کند.



شکل ۳: شبکه پس از قطع هر دو سیم

در این حالت، فقط میزبان هایی که به یک سوئیچ وصل هستند کار می‌کنند.



شکل ۴: شبکه بعد از وصل مجدد سیم اول

پس از وصل مجدد سیم اول و با گذشت کمی زمان دوباره شبکه متصل می‌شود. با توجه به خروجی‌ها و رفتار شبکه مشخص است که طبق پروتکل STP مسیر دیگر به درستی جایگزین می‌شود.

ج)

۱) برای چنین کاری، مجبور هستیم حالت STP و standalone را غیر فعال کنیم. چون STP یکی از پورت‌ها را در حالت `STP_BLOCK` قرار می‌دهد و حتی با تعریف جریان بر روی این پورت، این پورت همچنان بسته‌ها را drop خواهد کرد. بنابراین برای این قسمت از فایل `topo_no_stp.py` استفاده شده است.

۲) حال باید یکسری جریان برای کارکرد اولیه شبکه به آن اضافه کنیم.

```
sh ovs-ofctl add-flow s1 "priority=70,in_port=2,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:3,4"
sh ovs-ofctl add-flow s2 "priority=70,in_port=2,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:3,4"
sh ovs-ofctl add-flow s1 "priority=70,in_port=3,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,4"
sh ovs-ofctl add-flow s1 "priority=70,in_port=4,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:3,2"
sh ovs-ofctl add-flow s2 "priority=70,in_port=3,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:2,4"
sh ovs-ofctl add-flow s2 "priority=70,in_port=4,dl_dst=ff:ff:ff:ff:ff:ff,actions=output:3,2"
```

این جریان‌ها برای مدیریت broadcast ها در درخواست های arp در نظر گرفته شده است و به نحوی است که حلقه ایجاد نشود. از لینک دوم بین دو سوئیچ برای انتقال این بسته ها استفاده شده است.

۳) برای مسیر های عادی بین میزبان‌ها نیز جریان‌های زیر را داریم:

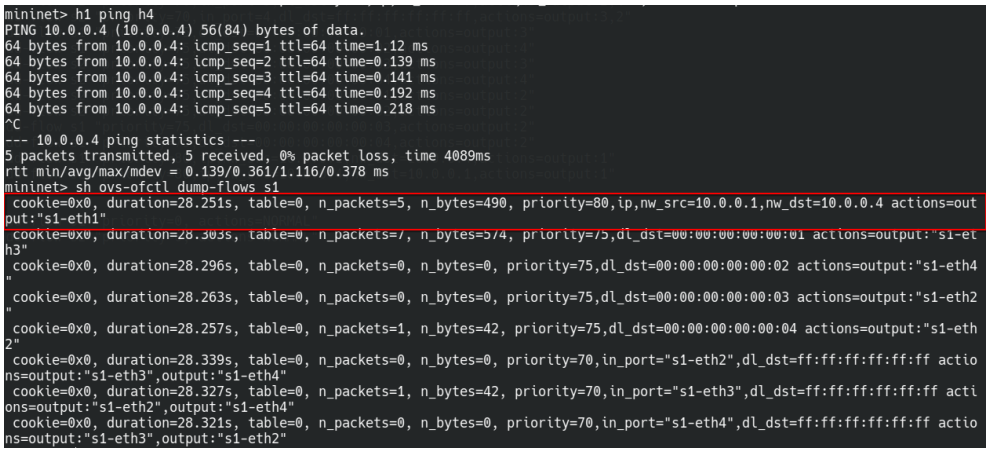
```
sh ovs-ofctl add-flow s1 "priority=75,dl_dst=00:00:00:00:00:01,actions=output:3"
sh ovs-ofctl add-flow s1 "priority=75,dl_dst=00:00:00:00:00:02,actions=output:4"
sh ovs-ofctl add-flow s2 "priority=75,dl_dst=00:00:00:00:00:03,actions=output:3"
sh ovs-ofctl add-flow s2 "priority=75,dl_dst=00:00:00:00:00:04,actions=output:4"
sh ovs-ofctl add-flow s2 "priority=75,dl_dst=00:00:00:00:00:01,actions=output:2"
sh ovs-ofctl add-flow s2 "priority=75,dl_dst=00:00:00:00:00:02,actions=output:2"
sh ovs-ofctl add-flow s1 "priority=75,dl_dst=00:00:00:00:00:03,actions=output:2"
sh ovs-ofctl add-flow s1 "priority=75,dl_dst=00:00:00:00:00:04,actions=output:2"
```

۴) نهایتا برای هدایت ترافیک بین دو میزبان (مثلا h1 و h4) می‌توانیم جریان هایی به این شکل بنویسیم:

```
sh ovs-ofctl add-flow s1 "priority=80,ip,nw_src=10.0.0.1,nw_dst=10.0.0.4,actions=output:1"
sh ovs-ofctl add-flow s2 "priority=80,ip,nw_src=10.0.0.4,nw_dst=10.0.0.1,actions=output:1"
```

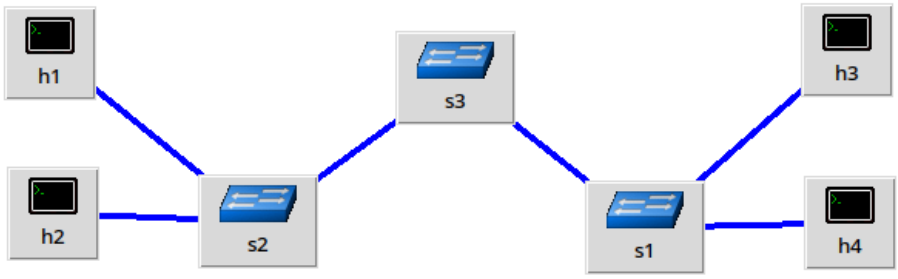
در این حالت این جریان ها به دلیل اولویت بیشترشان اول بررسی می‌شوند. این جریان‌ها ترافیک بین h1 و h4 را از طرق لینک اول عبور می‌دهند. درحالی که بقیه ترافیک ها از سیم دوم عبور می‌کنند.

۵) برای بررسی صحت این موضوع از پینگ و `dump-flows` استفاده می‌کنیم.



همانطور که دیده می‌شود، بسته های پینگ از جریانی که از لینک اول عبور می‌کند، استفاده می‌کند.

چ) برای اینکار، از یک سوئیچ سوم استفاده می‌کنیم. و سوئیچ دوم و اول را به آن وصل می‌کنیم. اگر این سوئیچ یا یکی از لینک های متصل به آن دچار خطا شود، ارتباط بین نیمه سمت چپ و راست قطع می‌شود. `singel_point_of_failure.py`



برای امتحان کردن این موضوع لینک بین s1 و s3 را قطع کرده و `pingall` را اجرا می‌کنیم.

```
mininet> link s1 s3 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X
h2 -> h1 X X
h3 -> X X h4
h4 -> X X h3
*** Results: 66% dropped (4/12 received)
mininet>
```

برای حل این مشکل کافی است یک لینک بین s1 و s2 ایجاد کنیم. حال دوباره آن را با `pingall` تست می‌کنیم.

`single_point_of_failure_redundancy.py`

```
mininet> link s1 s3 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

همانطور که دیده می‌شود. شبکه بدون مشکل به کار خود ادامه می‌دهد چون از سیم اضافی که بین سوئیچ یک و دو وجود دارد استفاده کرده است.