Homework 3

Question 1:

1. What are the goals of Web caching? List at least three different goals.

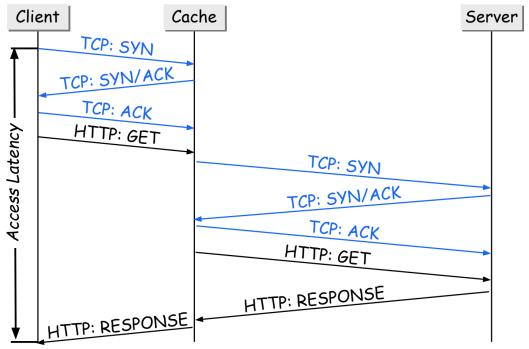
Goals: Improve user experience and network/server scalability.

- Speed up user access to Web content,
- Reduce network load,
- Reduce origin server load.

2. Which of these goals cannot be achieved directly through multicast technology?

Multicast can not speed up user access to Web content, i.e., does not address the problem of high access latency.

3. Sketch the message flow between a Client, a Web cache, and an Origin Server in case of a Web cache miss. Show HTTP messages and messages for TCP connection establishment.



4. Using the illustration from 3, explain how “access latency” is defined.

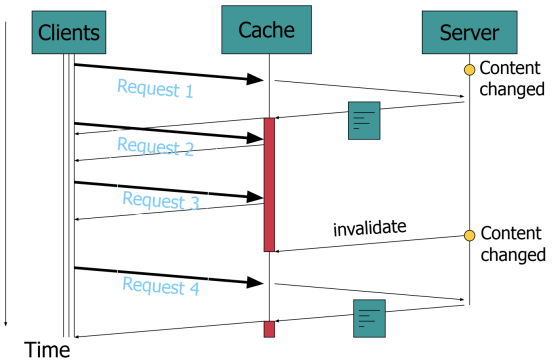
The “access latency: is defined by the y-axis in this figure.

Question 2:

Cache invalidation and cache update are two techniques to ensure cache consistency.

1. Briefly describe the cache invalidation mechanism.

- Idea: The server knows best when a Web page is updated!
- The server remembers where its pages are cached.
- When a page is modified, the server notifies the caches that have a copy of this page.
- Caches mark the page as stale.
- Subsequent client requests will fetch a fresh copy from the server.



2. Briefly describe the cache update mechanism.

- The server remembers where its pages are cached.
- When a page is modified, the server notifies the caches and gives them an updated version of the page (usually used together with the application layer multicast).
- Caches always assume that they have the latest content.

Question 3:

A streaming application is requesting video clips with a playback rate of 400kbps from a regular streaming server.

1. How big (in bytes) must the client-side buffer of the streaming application be to absorb network jitter for up to 2 seconds?

The bytes of the client side buffer of the streaming application absorb network jitter up to 2 seconds is (400*10^3*2)/8 = 100KB.

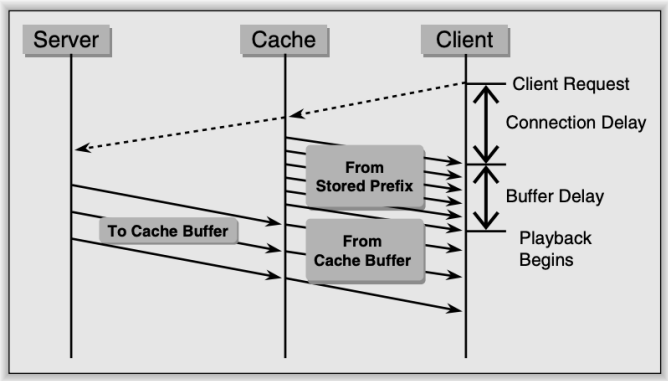
2. Ignoring any connection delay, what is the minimum start-up latency a user of the streaming application has to expect and why?

The playback rate is 400kbps, which is equivalent to 50KB/s. Since the playback starts when the playback buffer is filled and we have the buffer size is 100KB (from Question 3.1), we obtain the minimum start-up latency as the divide of buffer size by the playback rate, i.e., (100KB)/(50KB/s) = 2 seconds.

3. The streaming server now implements “fast prefix transfer” to reduce the start-up latency. What is the optimal prefix length? Why is a larger prefix undesirable? Why is a shorter prefix undesirable?

- Since the playback rate of 400kpbs, i.e., 50KBps requires 100KB of buffer space, we obtain the optimal prefix length is (100KB)/(50KB/s) 2 seconds long.
- If the prefix is larger, then the capacity of the cache is not enough and ejects files from the server.
- Conversely, the the prefix is shorter, then the buffer delay will be longer, reducing the optimal benefit of fast prefix transfer.

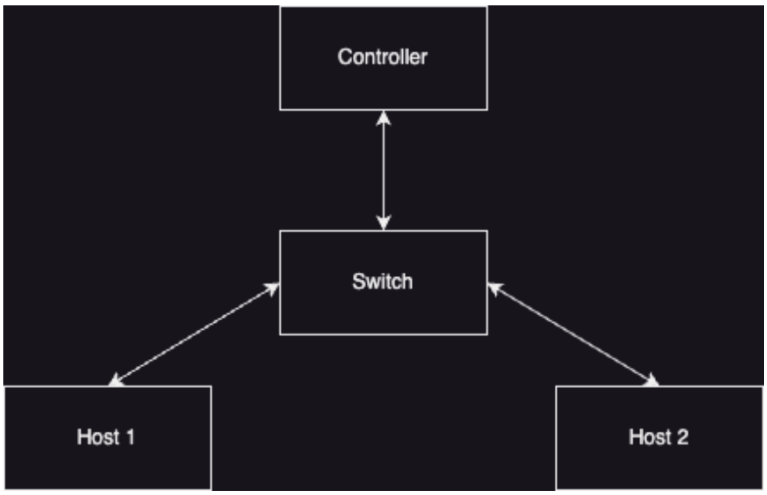
4. Using the optimal prefix length and assuming no other network traffic, what is the minimum bandwidth required between client and server to reduce the buffer delay to 1 second?



To reduce the buffer delay from 2 to 1 second, we need to increase the bandwidth from 400kbps to (400*2) = 800kpbs.

Question 4:

Using Mininet and the POX library, implement a network topology as shown in the following figure:

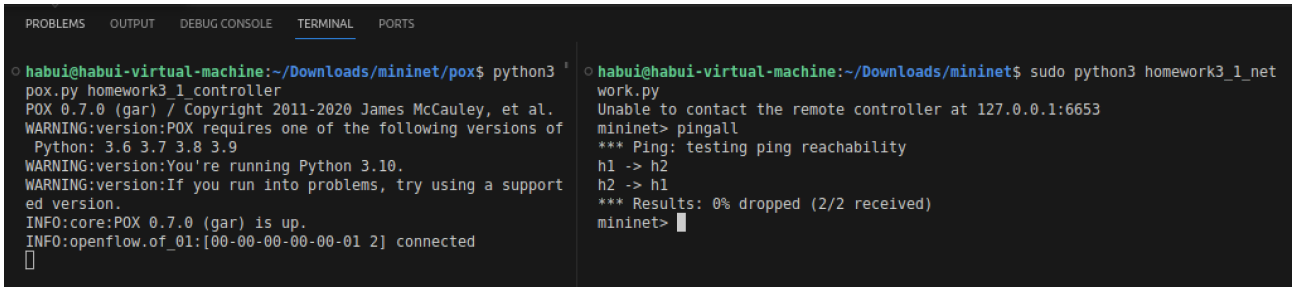


Use the POX controller to update the flow table of the switch. Only ARP, ICMP packets, and TCP packets with port 80 are allowed. All other packets will be dropped.
You can evaluate your firewall using these steps:

1. For ICMP packets, test the reachability of ICMP packets using Mininet's **pingall** command.
2. For TCP packets, evaluate your firewall by running a simple HTTP server on the host. First, run the command in Mininet to open terminals for Host 1 and Host 2.
    a. “xterm h1 h2”
3. Then run the command in Host 1's terminal to start a simple HTTP server on port 80.
    a. “python3 -m http.server 80”
4. Finally, use the **curl** command to test the TCP packet.
    a. “curl 10.0.0.1”
5. If everything works fine, you will see the response from the server.
6. Then change the port of the HTTP server to another port, such as 8080. Modify and run the curl command again to send the request to the new port. You will see failure information appear.

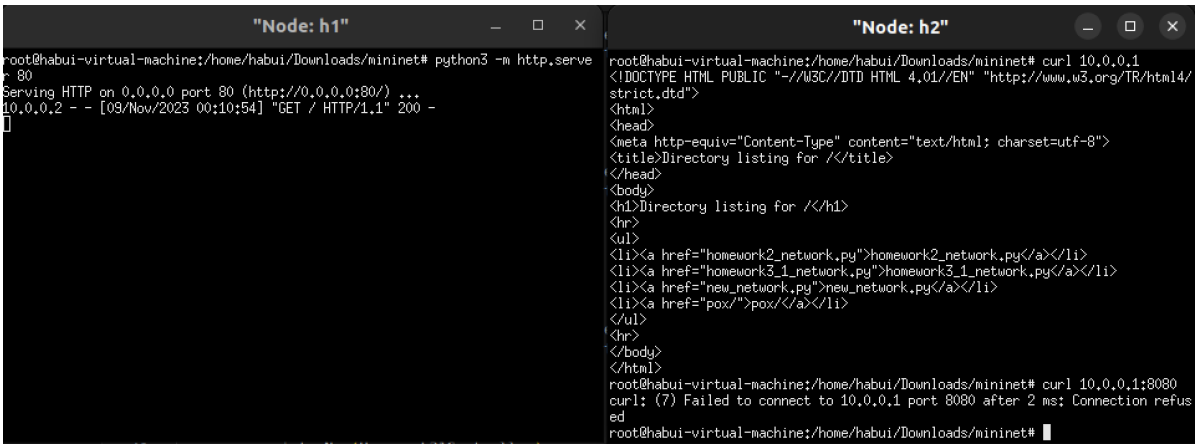
Submit a screenshot of the **pingall** and the results of two **curl** executions, along with your source code. The materials you submit should demonstrate that your firewall works as expected.

Screenshot of the “pingall” command

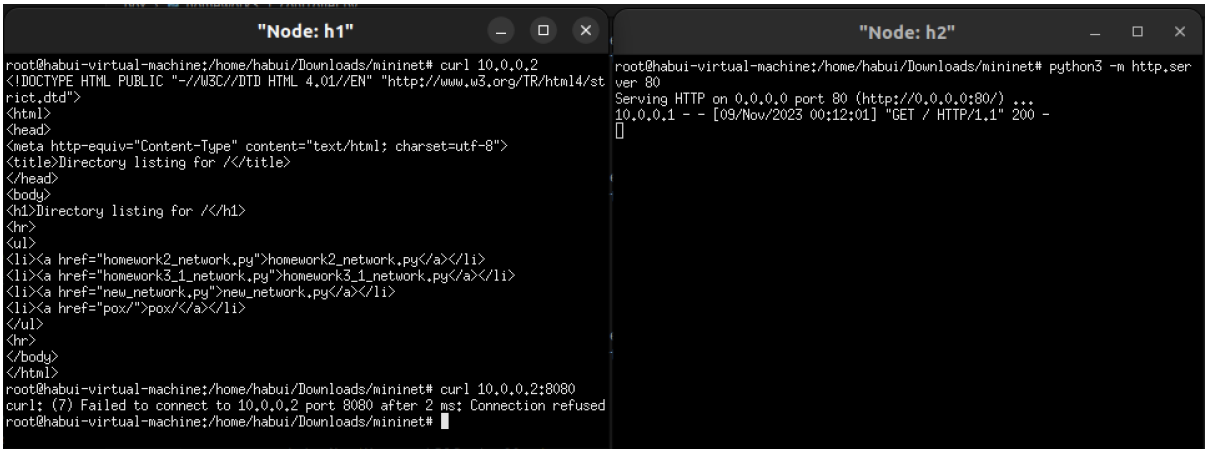


Screenshot of the results of two “curl” executions:

- “curl 10.0.0.1” and “curl 10.0.0.1:8080”



- “curl 10.0.0.1” and “curl 10.0.0.1:8080”



Source code:

- homework3_1_network.py

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.node import RemoteController

class Homework31Network(Topo):
    def build(self):
        switch1 = self.addSwitch('s1')
        host1 = self.addHost('h1', ip = '10.0.0.1')
        host2 = self.addHost('h2', ip = '10.0.0.2')

        self.addLink(switch1, host1)
        self.addLink(switch1, host2)

if __name__ == '__main__':
    net = Mininet(Homework31Network(), controller = RemoteController)
    net.start()
```

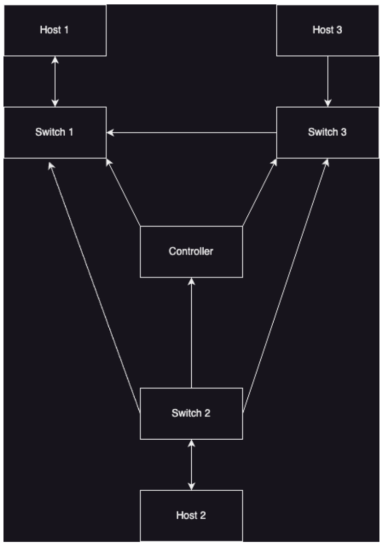
```
    CLI(net)
    net.stop()
```

- **pox/homework3_1_controller.py**

```
class Homework31Controller:
    def __init__(self) -> None:
        core.openflow.addListeners(self)

    def _handle_ConnectionUp(self, event):
        connection = event.connection

        connection.send(
            of.ofp_flow_mod(
                match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = 6, nw_dst = '10.0.0.1'),
                action = of.ofp_action_output(port = 1),
            )
        )

        connection.send(
            of.ofp_flow_mod(
                match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = 6, nw_dst = '10.0.0.2'),
                action = of.ofp_action_output(port = 2),
            )
        )

        connection.send(
            of.ofp_flow_mod(
                match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.1'),
                action = of.ofp_action_output(port = 1),
            )
        )

        connection.send(
            of.ofp_flow_mod(
                match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.2'),
                action = of.ofp_action_output(port = 2),
            )
        )

        # discovre the link layer address
        connection.send(
            of.ofp_flow_mod(
                match = of.ofp_match(dl_type=pkt.ethernet.ARP_TYPE),
                action = of.ofp_action_output(port = 65531),
            )
        )

def launch():
    core.registerNew(Homework31Controller)
```

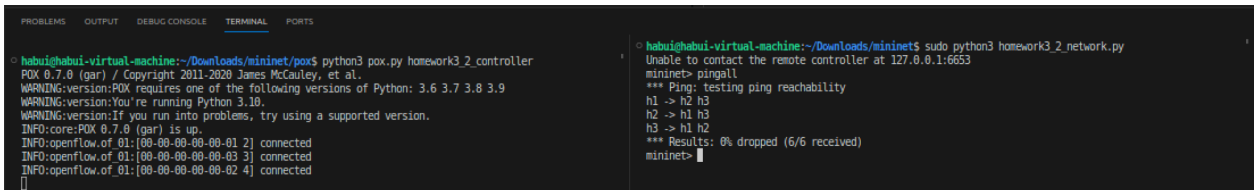
Question 5:

By using Mininet and the POX library, implement a network topology as shown in the following figure:



Use the POX controller to update the flow tables of Switches. Ensure that all hosts can ping each other without any packets being dropped due to TTL. Submit the screenshot of **pingall** command and source code.

Screenshot of the "pingall" command



Source code:

- **homework3_2_network.py**

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.node import RemoteController

class Homework32Network(Topo):
    def build(self):
        switch1 = self.addSwitch('s1')
        switch2 = self.addSwitch('s2')
        switch3 = self.addSwitch('s3')
        host1 = self.addHost('h1', ip = '10.0.0.1')
        host2 = self.addHost('h2', ip = '10.0.0.2')
        host3 = self.addHost('h3', ip = '10.0.0.3')

        self.addLink(switch1, host1, port1 = 4, port2 = 1)
        self.addLink(switch2, host2, port1 = 4, port2 = 1)
        self.addLink(switch3, host3, port1 = 4, port2 = 1)
        self.addLink(switch1, switch2, port1 = 1, port2 = 1)
```

```
        self.addLink(switch1, switch3, port1 = 2, port2 = 2)
        self.addLink(switch2, switch3, port1 = 3, port2 = 3)

if __name__ == '__main__':
    net = Mininet(Homework32Network(), controller = RemoteController)
    net.start()
    CLI(net)
    net.stop()
```

- **pox/homework3_2_controller.py**

```
from pox.core import core
import pox.lib.packet as pkt
import pox.openflow.libopenflow_01 as of

class Homework32Controller:
    def __init__(self) -> None:
        core.openflow.addListeners(self)

    def _handle_ConnectionUp(self, event):
        connection = event.connection
        if event.dpid == 1:
            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.1'),
                    action = of.ofp_action_output(port = 4),
                )
            )

            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.2'),
                    action = of.ofp_action_output(port = 1),
                )
            )

            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.3'),
                    action = of.ofp_action_output(port = 2),
                )
            )

        if event.dpid == 2:
            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.1'),
                    action = of.ofp_action_output(port = 1),
                )
            )
```

```
            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.2'),
                    action = of.ofp_action_output(port = 4),
                )
            )

            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.3'),
                    action = of.ofp_action_output(port = 3),
                )
            )

        if event.dpid == 3:
            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.1'),
                    action = of.ofp_action_output(port = 2),
                )
            )

            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.2'),
                    action = of.ofp_action_output(port = 3),
                )
            )

            connection.send(
                of.ofp_flow_mod(
                    match = of.ofp_match(dl_type=pkt.ethernet.IP_TYPE,
nw_proto = pkt.ipv4.ICMP_PROTOCOL, nw_dst = '10.0.0.3'),
                    action = of.ofp_action_output(port = 4),
                )
            )

        # discovre the link layer address
        connection.send(
            of.ofp_flow_mod(
                match = of.ofp_match(dl_type=pkt.ethernet.ARP_TYPE),
                action = of.ofp_action_output(port = 65531),
            )
        )

def launch():
    core.registerNew(Homework32Controller)
```