TCP/IP Networking, Autumn 2019-2020
Lab1 Research Exercise

Furkan KARAKAS, Onur VEYISOGLU

October 4, 2019

In this research exercise, we chose to analyze the topology of a generic tree. A generic tree has a depth a, and each node in the tree has b child nodes. In this topology, only the nodes in the deepest part of the tree are the hosts, and all other nodes are switches which are responsible for forwarding the IP packets from Host X to Host Y. An example of a generic tree with $a = 3$ and $b = 2$ is illustrated in Figure 1.

The Python code in order to generate the tree is given below:

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


class GenericTree(Topo):
    """Simple topology example."""

    def build(self, depth=1, fanout=2):
        # Numbering: h1..N, s1..M
        self.hostNum = 1
        self.switchNum = 1

    def build(self, depth=1, fanout=2):
        # Numbering: h1..N, s1..M
        self.hostNum = 1
        self.switchNum = 1
        # Build topology
        self.addTree(depth, fanout)

    def addTree(self, depth, fanout):
        """Use recursion to build the generic tree."""
```

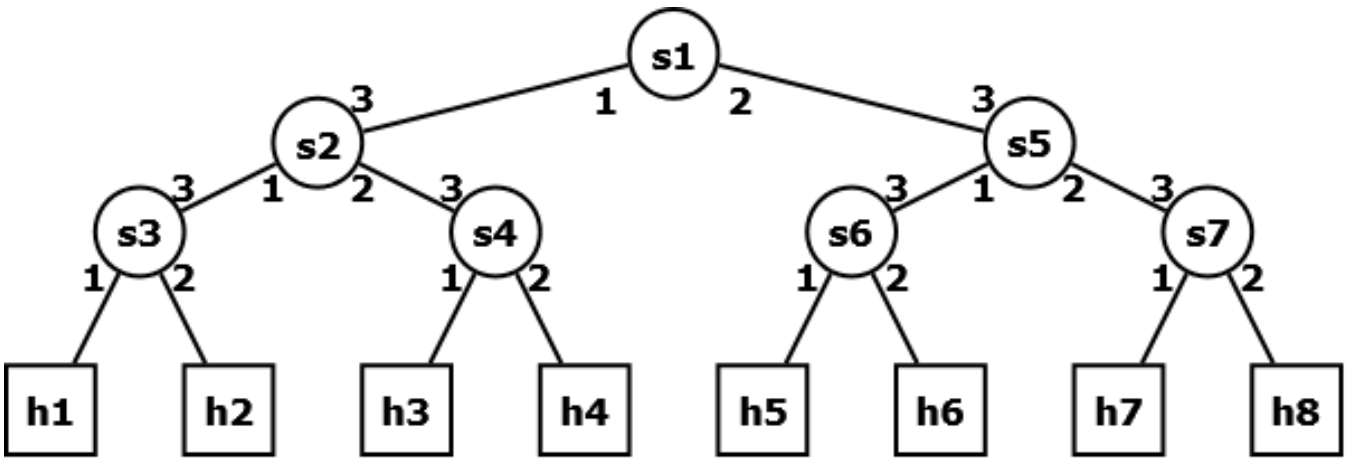

Figure 1: A generic tree with depth 3 and child nodes 2 (i.e. a binary tree with depth 3)

```
        isSwitch = depth > 0
        if isSwitch:
            node = self.addSwitch('s%s' % self.switchNum)
            self.switchNum += 1
            for _ in range(fanout):
                child = self.addTree(depth - 1, fanout)
                self.addLink(node, child)
        else:
            node = self.addHost('h%s' % self.hostNum)
            self.hostNum += 1
        return node


def run(a, b):
    c = Controller('c')
    net = Mininet(topo=GenericTree(depth=int(a), fanout=int(b)), host=CPULimitedHost,
                  controller=c)
    net.start()

    CLI(net)
    net.stop()


if __name__ == '__main__':
    setLogLevel('info')
    a = input("Write tree depth:")
    b = input("Write the number of children:")
    run(a, b)
```

Running the program with the input pair $(a, b) = (3, 2)$, and trying to use command `pingall`, we get:

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

We notice that it ran pretty quickly.
With the input pair $(a, b) = (5, 2)$:

```
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
h2 -> h1 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
h3 -> h1 h2 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
h4 -> h1 h2 h3 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
h5 -> h1 h2 h3 h4 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
h6 -> h1 h2 h3 h4 h5 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
h7 -> h1 h2 h3 h4 h5 h6 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
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
```

```
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28
    h29 h30 h31 h32
h16 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h17 h18 h19 h20 h21 h22 ............
```

With this input pair, we noticed that it took more than 5 minutes for the program to ping all devices, which is a drastic increase in the running time. Pinging h2 from h1 would yield:

```
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=0.050 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.134 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.097 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.078 ms
```

On the other hand, pinging h32 from h1 would yield:

```
mininet> h1 ping h32
PING 10.0.0.32 (10.0.0.32) 56(84) bytes of data.
64 bytes from 10.0.0.32: icmp_seq=1 ttl=64 time=86.5 ms
64 bytes from 10.0.0.32: icmp_seq=2 ttl=64 time=1.30 ms
64 bytes from 10.0.0.32: icmp_seq=3 ttl=64 time=0.125 ms
64 bytes from 10.0.0.32: icmp_seq=4 ttl=64 time=0.160 ms
64 bytes from 10.0.0.32: icmp_seq=5 ttl=64 time=0.121 ms
64 bytes from 10.0.0.32: icmp_seq=6 ttl=64 time=0.109 ms
64 bytes from 10.0.0.32: icmp_seq=7 ttl=64 time=0.146 ms
```

Note that there is a significant time delay in the second case.