

实验说明

中国科学院计算技术研究所

网络技术研究中心

提纲

- Mininet实验环境
- 实验说明
 - 广播网络实验
 - 交换机转发实验
 - BGP前缀劫持攻击及检测实验
 - TCP公平性实验
 - HTTP服务器实验

Mininet实验环境

- Mininet是一个可以支持快速搭建模拟网络的平台
 - <http://mininet.org/>
- Mininet环境只能安装在Linux系统下，推荐使用Ubuntu发行版，版本号从20.04到最新都可以，64位或32位都行
- 如果物理机为Windows系统，可以使用虚拟机方式安装Linux系统，推荐使用VirtualBox虚拟机
- 运行Mininet环境时需要root权限

安装Mininet

- 两种安装方式，优先使用第一种：
 - 命令行下直接使用pip安装： `sudo python -m pip install mininet`
 - 假设Python版本为3.x
 - 如果提示找不到mininet安装包，则说明Linux系统版本较旧，使用apt安装，这时安装的mininet只支持Python2
 - `sudo apt install mininet`
- 根据安装方式，运行mininet脚本时选择不同的Python版本，第一种安装方式使用python运行，第二种使用python2运行

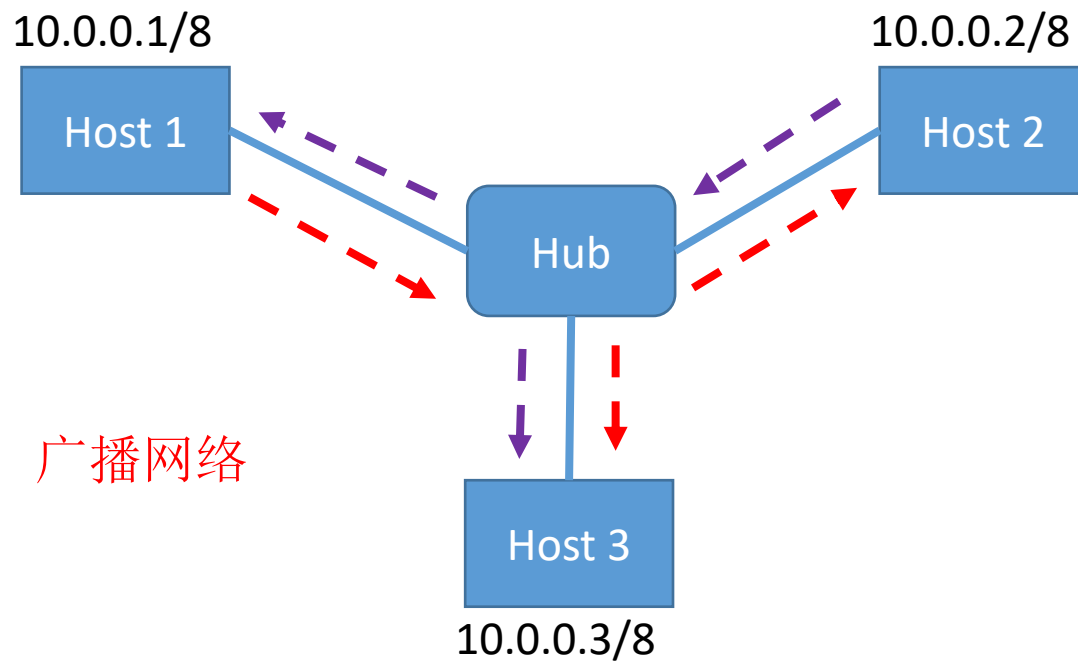
验证Mininet是否安装成功

- 执行pingall命令，如果所有节点都互通，说明安装成功
 - ~ \$ sudo mn
 - mininet> pingall
 - mininet> quit

```
alvin@alvin-ubuntu: ~/networking/mininet
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> 
```

广播网络实验

广播网络



广播网络节点（Hub）实现

- 已知网络端口数据结构和发送数据包函数

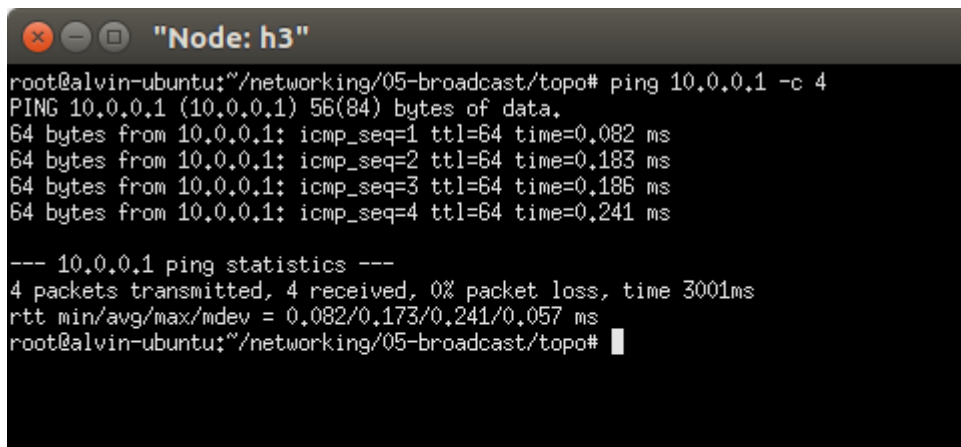
```
typedef struct {  
    struct list_head list;  
    int fd;  
    int index;  
    u8 mac[ETH_ALEN];  
    char name[16];  
} iface_info_t;  
  
void iface_send_packet(iface_info_t  
*iface, const char *packet, int len)  
{  
    struct sockaddr_ll addr;  
    // fill addr ..., omitted  
    sendto(iface->fd, packet, len, 0,  
    &addr, sizeof(addr));  
}
```

节点广播的逻辑

```
foreach iface in iface_list:  
    if iface != rx_iface:  
        iface_send_packet(iface, packet, len);
```


实现节点广播逻辑

- 实现main.c中的broadcast_packet函数
 - instance->iface_list链表中保存所有端口的信息
 - 收到每个数据包，将该包从所有其它端口发送出去
- 结果验证
 - 使用three_nodes_bw.py拓扑文件
 - 三个节点相互能够ping通



```
root@alvin-ubuntu:~/networking/05-broadcast/topo# ping 10.0.0.1 -c 4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.082 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.183 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.186 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.241 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.082/0.173/0.241/0.057 ms
root@alvin-ubuntu:~/networking/05-broadcast/topo#
```

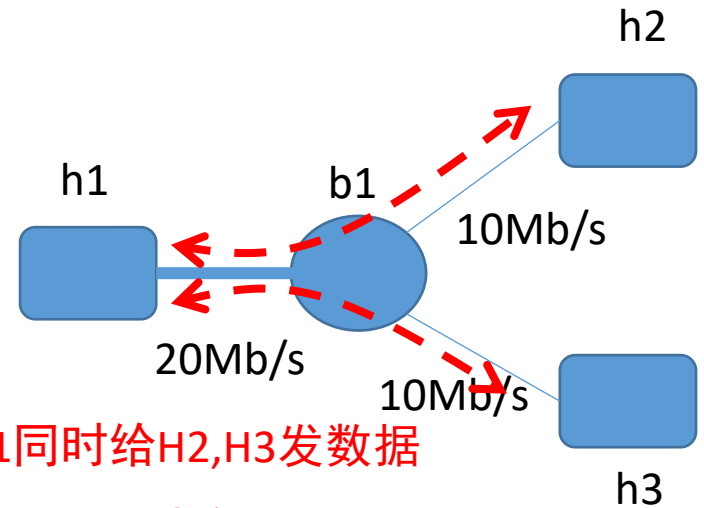
验证广播网络传输效率

- 进行iperf测试

- 实验验证广播网络的链路利用效率

- iperf测试 (Client -> Server) :

- H1: iperf client; H2, H3: iperf servers, **H1同时给H2,H3发数据**
 - H1: iperf server; H2, H3: iperf clients, **H2, H3同时给H1发数据**



```

"Node: h1"
root@alvin-ubuntu:~/networking/05-broadcast/topo# iperf -c 10.0.0.2 -t 30
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.1 port 42412 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.3 sec  17.5 MBytes  4.84 Mbits/sec
root@alvin-ubuntu:~/networking/05-broadcast/topo#

"Node: h1"
root@alvin-ubuntu:~/networking/05-broadcast/topo# iperf -c 10.0.0.3 -t 30
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.1 port 48660 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.2 sec  11.0 MBytes  3.05 Mbits/sec
root@alvin-ubuntu:~/networking/05-broadcast/topo#

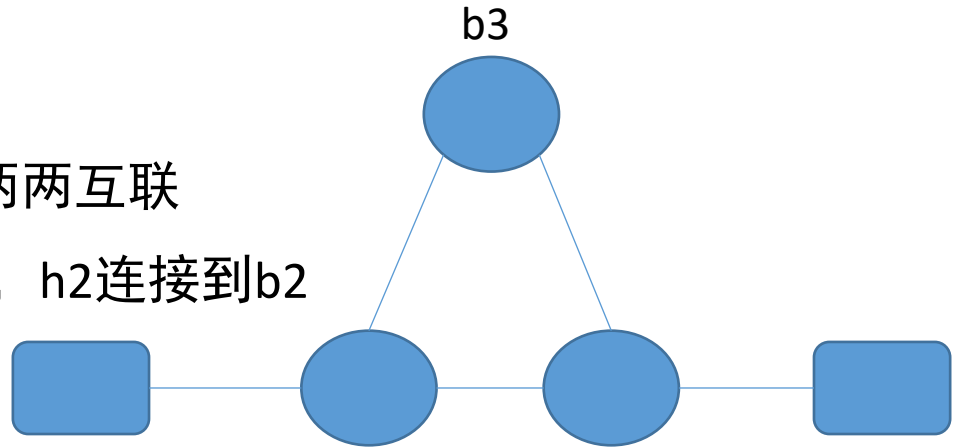
h2 # iperf -s
h3 # iperf -s
```

验证广播网络产生数据环路

- 环形网络拓扑

- 三个Hub节点，b1, b2, b3，两两互联
- 两个主机节点，h1连接到b1，h2连接到b2

- 由h1向h2发送一个数据包



- h1# ping -c 1 10.0.0.2

- 抓包看到一个数据包

```
root@alvin-ubuntu:~/networking/backup/2018-autumn/04-broadcast# ping -c 1 10.0.0.2
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=0.373 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.373/0.373/0.373/0.000 ms
root@alvin-ubuntu:~/networking/backup/2018-autumn/04-broadcast#
```

No.	Time	Source	Destination	Protocol	Length	Info
347	0.025926267	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
348	0.025953517	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
349	0.026242051	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
350	0.026272627	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
351	0.026297554	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
352	0.026324763	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
353	0.026451732	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
354	0.026481301	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
355	0.026506159	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
356	0.026753550	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
357	0.026780034	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	10.0.0.2 is at 96:d9:eb:ee:47:cc
358	5.060358971	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
359	6.067934799	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
360	7.092262423	96:d9:eb:ee:47:cc	9e:fa:31:0b:b2:f9	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2

Frame 355: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: 96:d9:eb:ee:47:cc (96:d9:eb:ee:47:cc), Dst: 9e:fa:31:0b:b2:f9 (9e:fa:31:0b:b2:f9)
Address Resolution Protocol (reply)

0000 9e fa 31 0b b2 f9 96 d9 eb ee 47 cc 08 06 00 01 ..1.....6....
0010 08 00 06 04 00 02 96 d9 eb ee 47 cc 0a 00 00 026.....
0020 9e fa 31 0b b2 f9 0a 00 00 01 ..1.....

实验内容

- 实现节点广播的broadcast_packet函数
- 验证广播网络能够正常运行
 - 从一个端节点ping另一个端节点
- 验证广播网络的效率
 - 在three_nodes_bw.py进行iperf测量
 - 两种场景： H1: iperf client; H2, H3: servers; H1: iperf server; H2, H3: clients
- 验证环形拓扑下节点广播会产生数据包环路

注意事项

1. 需要先在集线器节点(b1-b3)上运行hub(或hub-reference)，然后在主机节点(h1-h3)上运行相应网络程序(ping或iperf)
2. 实验依赖ethtool工具，可用apt进行安装
3. 执行iperf测试时，是iperf client给iperf server发送数据
4. 网络链路的带宽是双向的
 - 节点h1与b1之间的链路带宽为20Mbps，h1以20Mbps速率向b1传输数据的同时，b1也能以20Mbps速率向h1传输数据

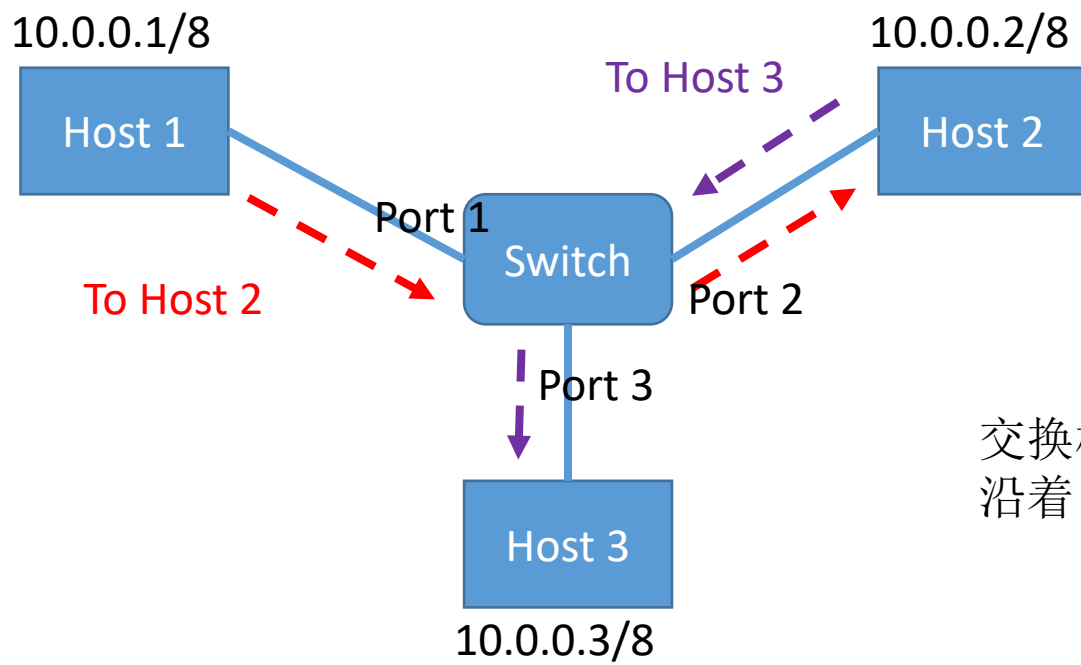
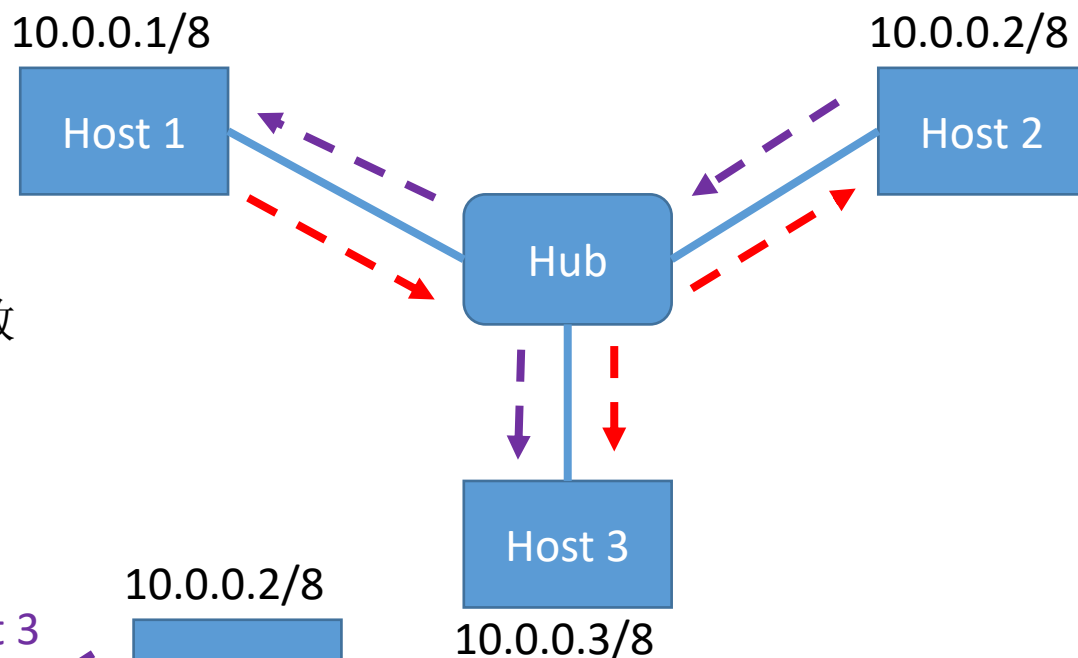
附件文件列表

- disable_offloading.sh # 禁止TCP Offloading
- disable_ipv6.sh # 禁止IPv6协议
- include # 所有相关头文件
- main.c # 待实现程序
- Makefile
- ring_topo.py # 环形拓扑
- three_nodes_bw.py # 设置了带宽限制的拓扑

交换机转发实验

交换机转发

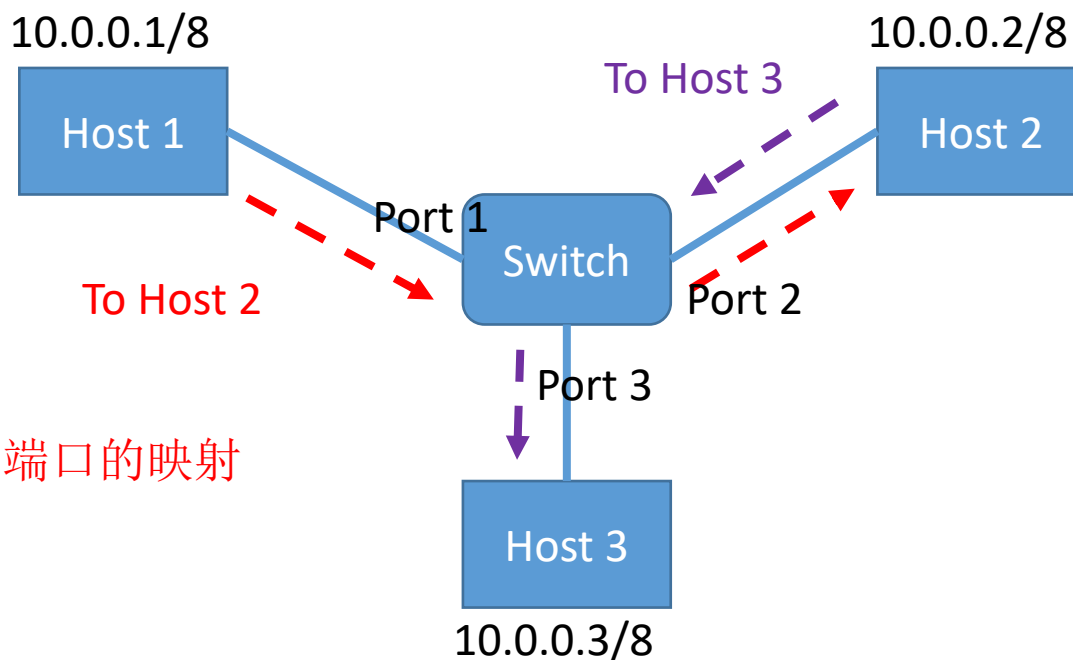
广播网络中，广播节点将每个数据包从所有其他端口广播出去



交换机（Switch）将收到的数据包沿着目的主机方向转发（Forward）

交换机转发表

交换机转发



- 交换机将目的地址与转出端口的映射存储在转发表中

转发表示意

目的地址	转发端口
Host 1	Port 1
Host 2	Port 2
Host 3	Port 3

实际转发表

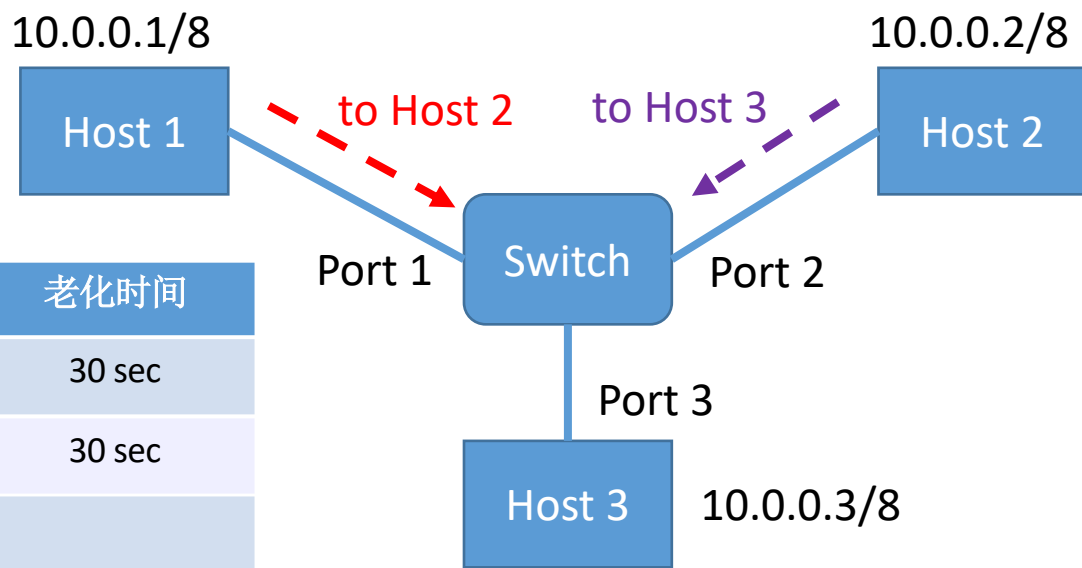
目的地址	转发端口	老化时间
Host 1 MAC Addr	Port 1	30 sec
Host 2 MAC Addr	Port 2	30 sec
Host 3 MAC Addr	Port 3	30 sec

交换机学习转发表

- 当交换机从某端口收到源MAC地址为X的数据包时，可以确定：将目的MAC地址为X的数据包从该端口转出可以达到目的主机。

转发表条目

目的地址	转发端口	老化时间
Host 1 MAC Addr	Port 1	30 sec
Host 2 MAC Addr	Port 2	30 sec



- 收到数据包后，交换机根据转发表中对应的转发端口转出数据包
- 交换机转发数据包时查不到对应端口时，直接广播该数据包

交换机学习转发表

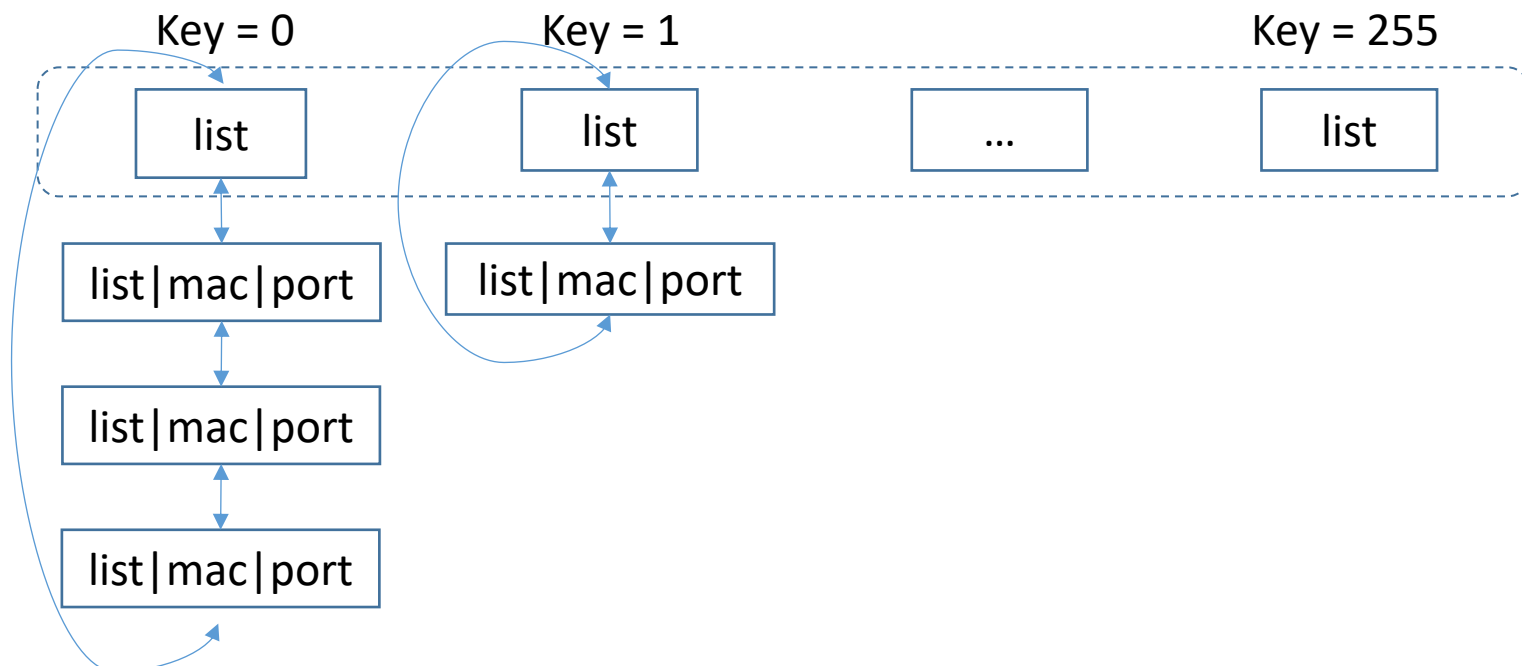
转发表

目的地址	转发端口	老化时间
Host 1 MAC Addr	Port 1	30 sec
Host 2 MAC Addr	Port 2	30 sec
Host 3 MAC Addr	Port 3	30 sec

- ①查询操作：每收到一个数据包，根据目的MAC地址查询相应转发条目：如果查询到对应条目，则根据相应转发端口转发数据包；否则，广播该数据包
- ②插入操作：每收到一个数据包，如果其源MAC地址在转发表中，更新访问时间；否则，将该地址与入端口的映射关系写入转发表
- ③老化操作：每秒钟运行一次老化操作，删除超过30秒未访问的转发条目

转发表结构

- 如果将所有mac->port映射存到一个链表中，则每次查找需要遍历整个链表
- 可以先对MAC地址Hash，根据key值到对应的链表中查找



交换机转发实现

- 实现对数据结构`mac_port_map`的所有操作，以及数据包的转发和广播操作
 - `iface_info_t *lookup_port(u8 mac[ETH_ALEN]);`
 - `void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface);`
 - `int sweep_aged_mac_port_entry();`
 - `void broadcast_packet(iface_info_t *iface, const char *packet, int len);`
 - `void handle_packet(iface_info_t *iface, char *packet, int len);`

实验内容

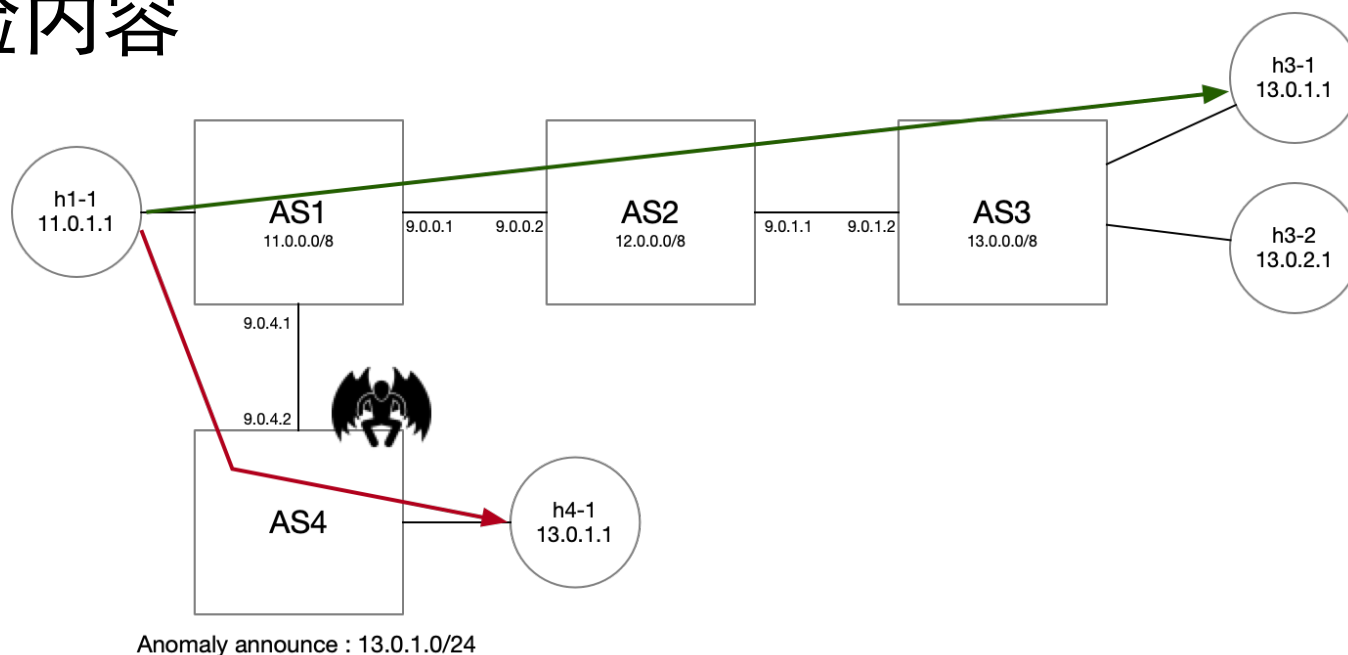
- 在主机h2和h3上分别打开wireshark程序，监听各自主机的eth0端口(h2-eth0和h3-eth0)
- 在h1主机上分别ping h2和h3两个主机
 - 在h2和h3两个主机上的wireshark捕获的应该只包含自己节点和h1产生的数据包

附件文件列表

- disable_ipv6.sh # 禁止IPv6协议
- disable_offloading.sh # 禁止TCP Offloading
- include # 所有相关头文件
- mac.c # 待实现mac_port_mac相关操作
- main.c # 待实现转发函数
- Makefile
- packet.c # 待实现广播函数
- three_nodes_bw.py # Mininet topo脚本

BGP前缀劫持攻击及检测实验

实验内容



- 实验拓扑：如图，AS1-AS4 是 4 个边界网关路由器。AS1 拥有 11.0.0.0/8，该网段内有一个 host（h1-1），它被用作客户端访问位于 AS3 的 h3-1 和 h3-2 服务器。AS3 拥有 13.0.0.0/8。正常情况下 h1-1 访问 13.0.1.1 的流量按照绿色路径所示。
- 发动攻击：AS4 是一个恶意 AS，它会劫持 h1-1 通往 13.0.1.1 的流量，转发到自己域内的恶意服务器。劫持方式是 AS4 宣告一条更长的网络前缀 13.0.1.0/24。劫持后 h1-1 的请求会按照红色路径到达恶意服务器 h4-1。通往 h3-2 的流量不受影响。

实验工具环境搭建

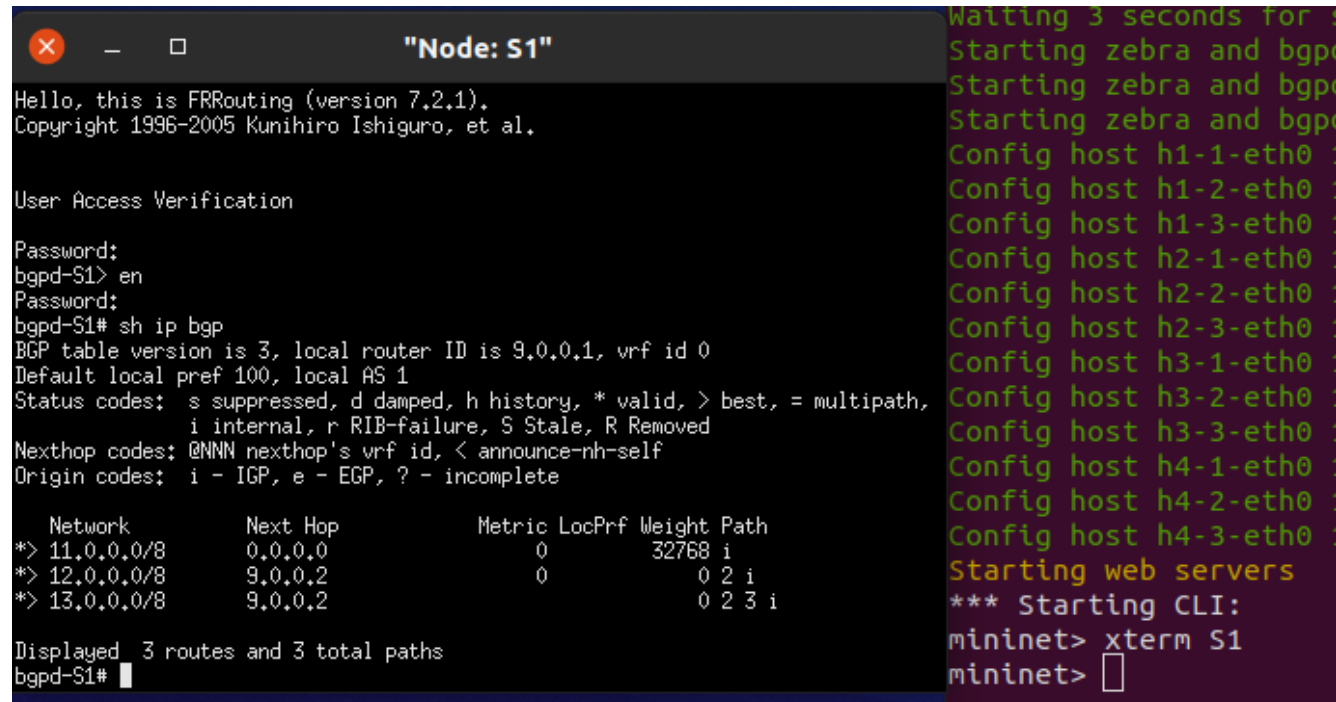
- 实验环境
 - 工具： FRRouting（需要 bgpd 实现 BGP 协议交互、zebra 将路由表项下发到 Linux 内核）、bgpdump（分析 bgp updates 消息）
- 工具安装 `sudo bash ./install.sh`
 - FRRouting
 - bgpdump

启动实验

- 运行 `sudo bash ./auto_configure.sh` 配置环境（只需执行一次，配置后就不要再更改实验目录）
- 修改 BGP configure 文件：
 - 仿照 AS1 和 AS2 的 configure 文件补充 AS3 和 AS4 的 bgp 配置文件（bgpd-S3.conf bgpd-S4.conf）
- 启动 mininet 环境，进行实验
 - `$ sudo python bgp.py`

实验步骤：查看 S1 边界路由器的转发表

- 启动 S1 的 xterm 终端 `mininet> xterm S1`
- (S1 xterm terminal)# `./connect.sh S1`
- Password: `en`
- `bgpd-S1> en`
- Password: `en`
- `bgpd-S1# sh ip bgp`



```
Node: S1
Hello, this is FRRouting (version 7.2.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification
Password:
bgpd-S1> en
Password:
bgpd-S1# sh ip bgp
BGP table version is 3, local router ID is 9.0.0.1, vrf id 0
Default local pref 100, local AS 1
Status codes: s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete

   Network        Next Hop        Metric LocPrf Weight Path
  *> 11.0.0.0/8    0.0.0.0          0         32768 i
  *> 12.0.0.0/8    9.0.0.2          0           0 2 i
  *> 13.0.0.0/8    9.0.0.2          0           0 2 3 i

Displayed 3 routes and 3 total paths
bgpd-S1#
```

Waiting 3 seconds for s
Starting zebra and bgpd
Starting zebra and bgpd
Starting zebra and bgpd
Config host h1-1-eth0 1
Config host h1-2-eth0 1
Config host h1-3-eth0 1
Config host h2-1-eth0 1
Config host h2-2-eth0 1
Config host h2-3-eth0 1
Config host h3-1-eth0 1
Config host h3-2-eth0 1
Config host h3-3-eth0 1
Config host h4-1-eth0 1
Config host h4-2-eth0 1
Config host h4-3-eth0 1
Starting web servers
*** Starting CLI:
mininet> xterm S1
mininet>

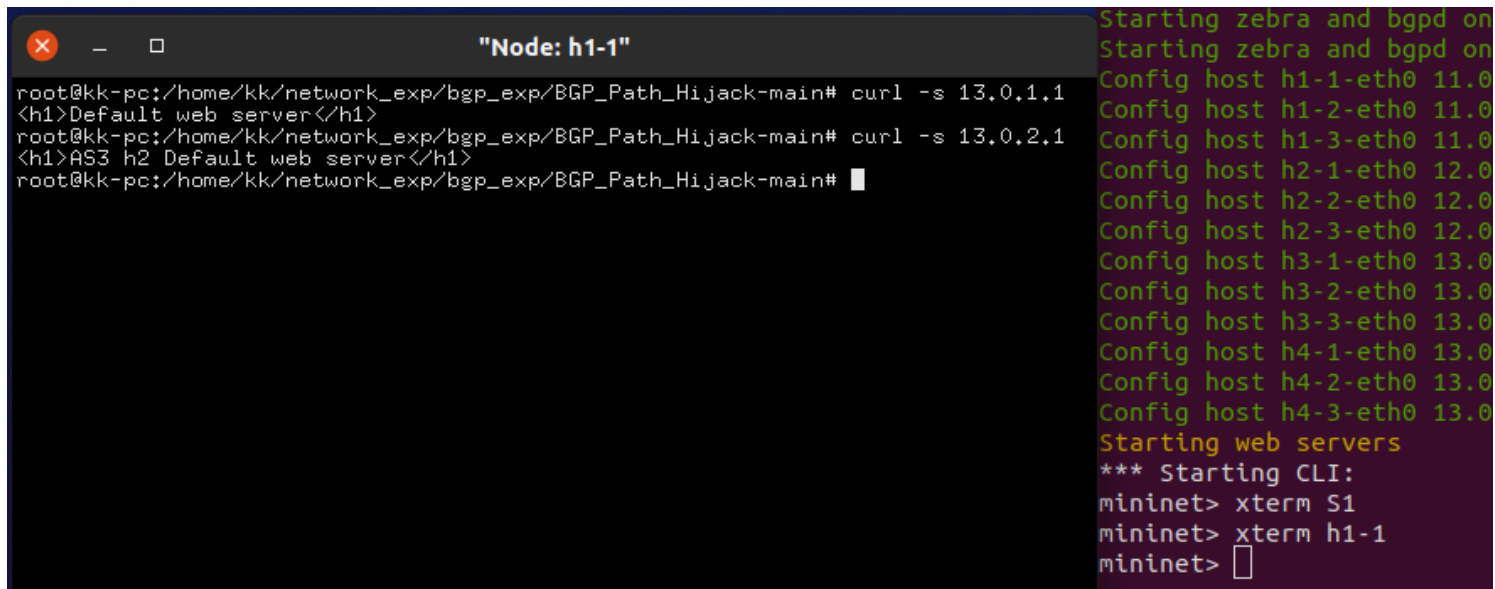
实验步骤： h1-1 正常访问 13.0.1.1 和 13.0.2.1

- mininet> *xterm h1-1*
- (h1-1 xterm terminal) # *curl -s 13.0.1.1*

<h1>Default web server</h1>

- (h1-1 xterm terminal) # *curl -s 13.0.2.1*

<h1>AS3 h2 Default web server</h1>



```
root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# curl -s 13.0.1.1
<h1>Default web server</h1>
root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# curl -s 13.0.2.1
<h1>AS3 h2 Default web server</h1>
root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main#
```

```
Starting zebra and bgpd on
Starting zebra and bgpd on
Config host h1-1-eth0 11.0
Config host h1-2-eth0 11.0
Config host h1-3-eth0 11.0
Config host h2-1-eth0 12.0
Config host h2-2-eth0 12.0
Config host h2-3-eth0 12.0
Config host h3-1-eth0 13.0
Config host h3-2-eth0 13.0
Config host h3-3-eth0 13.0
Config host h4-1-eth0 13.0
Config host h4-2-eth0 13.0
Config host h4-3-eth0 13.0
Starting web servers
*** Starting CLI:
mininet> xterm S1
mininet> xterm h1-1
mininet>
```

实验步骤：AS4 发动前缀劫持

- mininet> xterm S4
- (S4 terminal) #./start_rogue.sh
- (h1-1 terminal) # curl -s 13.0.1.1
- (h1-1 terminal) # curl -s 13.0.2.1

```
"Node: S4"
root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# ./start_rogue.sh
Killing any existing rogue AS
Executed cmd: mnexec -a 391388 pgrep -f zebra-S4 | sudo xargs kill -9
./stop_rogue.sh: line 3: 403181 Killed                  sudo python run.py --nod
e S4 --cmd "pgrep -f zebra-S4 | sudo xargs kill -9"
Executed cmd: mnexec -a 391388 pgrep -f bgpd-S4 | sudo xargs kill -9
./stop_rogue.sh: line 4: 403189 Killed                  sudo python run.py --nod
e S4 --cmd "pgrep -f bgpd-S4 | sudo xargs kill -9"
Starting rogue AS
Executed cmd: mnexec -a 391388 /usr/lib/frr/zebra -f conf/zebra-S4.conf -d -i /t
mp/zebra-S4.pid > logs/S4-zebra-stdout
2022/09/29 16:59:38 warnings: ZEBRA: [EC 4043309105] Disabling MPLS support (no
kernel support)
Executed cmd: mnexec -a 391388 /usr/lib/frr/bgpd -f conf/bgpd-S4.conf -d -i /tmp
/bgpd-S4.pid > logs/S4-bgpd-stdout
root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main#

"Node: h1-1"
root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# curl -s 13.0.1.1
(h1)*** Attacker web server ***</h1>
root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main#

h1-1 h1-2 h1-3 h2-1 h2-2 h2-3
*** Done
kk@kk-pc:~/network_exp/bgp_exp/BGP_Path_Hijack-main$
*** Creating network
*** Adding controller
*** Adding hosts:
h1-1 h1-2 h1-3 h2-1 h2-2 h2-3
*** Adding switches:
S1 S2 S3 S4 S5
*** Adding links:
(S1, S2) (S1, S4) (S1, S5) (S2, S3) (S2, S4) (S2, S5) (S3, S4) (S3, S5) (S4, S5)
*** Configuring hosts
h1-1 h1-2 h1-3 h2-1 h2-2 h2-3
*** Starting controller
c0
*** Starting 5 switches
S1 S2 S3 S4 S5
Waiting 3 seconds for sysctl
Starting zebra and bgpd on S1
Starting zebra and bgpd on S2
Starting zebra and bgpd on S3
Config host h1-1-eth0 11.0.0.1
Config host h1-2-eth0 11.0.0.2
Config host h1-3-eth0 11.0.0.3
Config host h2-1-eth0 12.0.0.1
Config host h2-2-eth0 12.0.0.2
Config host h2-3-eth0 12.0.0.3
Config host h3-1-eth0 13.0.0.1
Config host h3-2-eth0 13.0.0.2
Config host h3-3-eth0 13.0.0.3
Config host h4-1-eth0 13.0.0.4
Config host h4-2-eth0 13.0.0.5
Config host h4-3-eth0 13.0.0.6
Starting web servers
*** Starting CLI:
mininet> xterm S1
mininet> xterm h1-1
mininet> xterm S4
mininet>
```

实验步骤：检查 S1 路由表的变化

The image shows a network experiment setup with three terminal windows. The top-left window shows the BGP configuration and route table for S1. The top-right window shows the initialization of the network topology. The bottom-left window shows the execution of a script to start a rogue AS for hijacking. The bottom-right window shows the execution of a script to start a rogue AS for hijacking.

Node: S1

Origin codes: i - IGP, e - EGP, ? - incomplete

Network	Next Hop	Metric	LocPrf	Weight	Path
*> 11.0.0.0/8	0.0.0.0	0		32768	i
*> 12.0.0.0/8	9.0.0.2	0		0	2 i
*> 13.0.0.0/8	9.0.0.2	0		0	2 3 i

Displayed 3 routes and 3 total paths

bgpd-S1# sh ip bgp

BGP table version is 4, local router ID is 9.0.0.1, vrf id 0

Default local pref 100, local AS 1

Status codes: s suppressed, d damped, h history, * valid, > best, = multipath, i internal, r RIB-failure, S Stale, R Removed

Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self

Origin codes: i - IGP, e - EGP, ? - incomplete

Network	Next Hop	Metric	LocPrf	Weight	Path
*> 11.0.0.0/8	0.0.0.0	0		32768	i
*> 12.0.0.0/8	9.0.0.2	0		0	2 i
*> 13.0.0.0/8	9.0.0.2	0		0	2 3 i
*> 13.0.1.0/24	9.0.4.2	0		0	4 i

Displayed 4 routes and 4 total paths

bgpd-S1#

Node: h1-1

root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# curl -s 13.0.1.1

<h1>Default web server</h1>

root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# curl -s 13.0.1.1

<h1>*** Attacker web server ***</h1>

root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# curl -s 13.0.2.1

<h1>AS3 h2 Default web server</h1>

root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main#

Node: S4

*** Starting 4 switches

S1 S2 S3 S4

Waiting 3 seconds for systemctl changes to take effect...

Starting zebra and bgpd on S1

Starting zebra and bgpd on S2

Starting zebra and bgpd on S3

Config host h1-1-eth0 11.0.1.1/24, gateway: 11.0.1.254

Config host h1-2-eth0 11.0.2.1/24, gateway: 11.0.2.254

Config host h1-3-eth0 11.0.3.1/24, gateway: 11.0.3.254

Config host h2-1-eth0 12.0.1.1/24, gateway: 12.0.1.254

Config host h2-2-eth0 12.0.2.1/24, gateway: 12.0.2.254

Config host h2-3-eth0 12.0.3.1/24, gateway: 12.0.3.254

Config host h3-1-eth0 13.0.1.1/24, gateway: 13.0.1.254

Config host h3-2-eth0 13.0.2.1/24, gateway: 13.0.2.254

Config host h3-3-eth0 13.0.3.1/24, gateway: 13.0.3.254

Config host h4-1-eth0 13.0.1.1/24, gateway: 13.0.1.254

Config host h4-2-eth0 13.0.2.1/24, gateway: 13.0.2.254

Config host h4-3-eth0 13.0.3.1/24, gateway: 13.0.3.254

Starting web servers

*** Starting CLI:

mininet> xterm S1

mininet> xterm h1-1

mininet> xterm S4

mininet>

Node: S4

root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main# ./start_rogue.sh

Killing any existing rogue AS

Executed cmd: mnexec -a 323109 pgrep -f zebra-S4 | sudo xargs kill -9

./stop_rogue.sh: line 3: 323310 Killed sudo python run.py --nod

a S4 --cmd "pgrep -f zebra-S4 | sudo xargs kill -9"

Executed cmd: mnexec -a 323109 pgrep -f bgpd-S4 | sudo xargs kill -9

./stop_rogue.sh: line 4: 323318 Killed sudo python run.py --nod

a S4 --cmd "pgrep -f bgpd-S4 | sudo xargs kill -9"

Starting rogue AS

Executed cmd: mnexec -a 323109 /usr/lib/frr/zebra -f conf/zebra-S4.conf -d -i /t

mp/zebra-S4.pid > logs/S4-zebra-stdout

2022/09/26 10:17:27 warnings: ZEBRA: [EC 4043309105] Disabling MPLS support (no

kernel support)

Executed cmd: mnexec -a 323109 /usr/lib/frr/bgpd -f conf/bgpd-S4.conf -d -i /t

mp/bgpd-S4.pid > logs/S4-bgpd-stdout

root@kk-pc:/home/kk/network_exp/bgp_exp/BGP_Path_Hijack-main#

攻击检测

- 问题：作为 AS3 的管理员，如何检测自己遭到了前缀劫持攻击？
- 查看节点收到的 update 消息：
 - 每个 router 收到的 update 消息保存在 updates 目录下
 - `$ cd updates`
 - `$ sudo chmod +r ./*` (使用 bgpdump 读取需要添加读权限)
 - `$ bgpdump <file-name>` 查看文件内容
 - `$ bgpdump -m <file-name>` 可以使用紧凑格式输出，方便后续在攻击检测工具中处理数据
- 设计一个实时检测工具，持续读取 AS3 收到的 update 消息并提出警告

实验要求

- 补充bgpd-S3/4.conf配置文件，完成BGP前缀劫持攻击
- 设计实现一个实时检测工具，持续读取 AS3 收到的 update 消息并提出警告
- 提交配置文件、工具代码和实验报告

附件文件列表

- auto_configure.sh # 配置设置，只需要执行一次
- bgp.py # BGP实验主文件
- conf # 各节点BGP配置文件
- connect.sh # 连接到bgpd查看路由信息
- install.sh # 安装实验相关工具
- logs
- start_rogue.sh # 发起BGP前缀劫持攻击
- stop_rogue.sh # 停止BGP前缀劫持攻击
- updates
- webserver.py # HTTP服务器

参考资料

- <https://docs.frrouting.org/en/latest/bgp.html>

TCP公平性实验

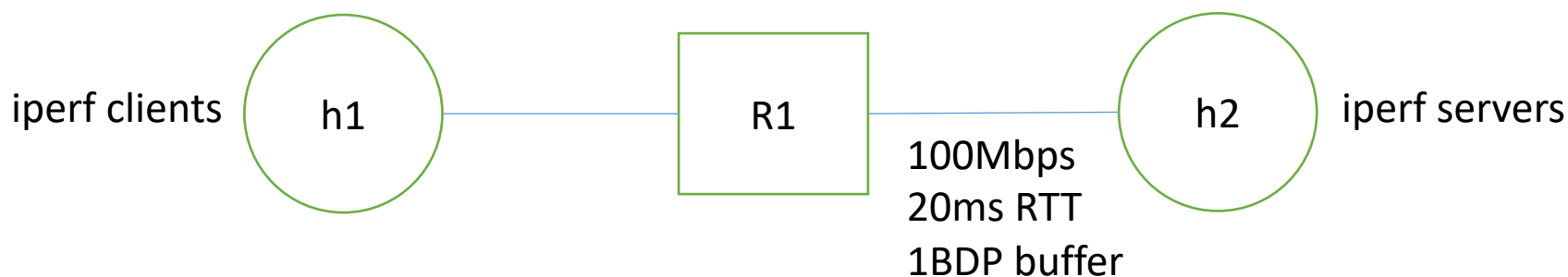
背景

- TCP协议作为应用需求和网络环境之间的性能适配器，其机制和算法一直在演进中
- Linux网络协议栈
 - 将TCP拥塞控制接口抽象出来，支持不同的TCP拥塞控制算法实现
 - 用户只需要实现特定接口，就可以自定义实现面向特定网络场景的拥塞控制逻辑
- Cubic算法：只使用丢包作为拥塞信号，收敛速度慢
- BBR算法：周期性探测带宽和延迟，在高带宽变化环境下收敛性差、公平性差

实验要求

- 设计并实现一种基于丢包和延迟的TCP拥塞控制机制
 - 能够在Linux 4.15+内核环境下编译成模块，并可以加载到内核中
 - 相比于Cubic/BBR算法，所设计的拥塞控制机制具有更好的收敛速度和公平性
- 提交：代码 & 实验报告

实验内容



- 实验拓扑： h1-R1-h2 直连，在 R1-h2 路径上设置速率限制
- 公平性和收敛性测试过程：在 h1 上每隔时间 t 启动 iperf 启动一条 TCP 流，直到有 n 条流同时发送。之后每隔时间 t 结束一条流，直到所有流停止发送
- 测试期间使用 tcpdump 抓包，供后续公平性分析使用

实验工具

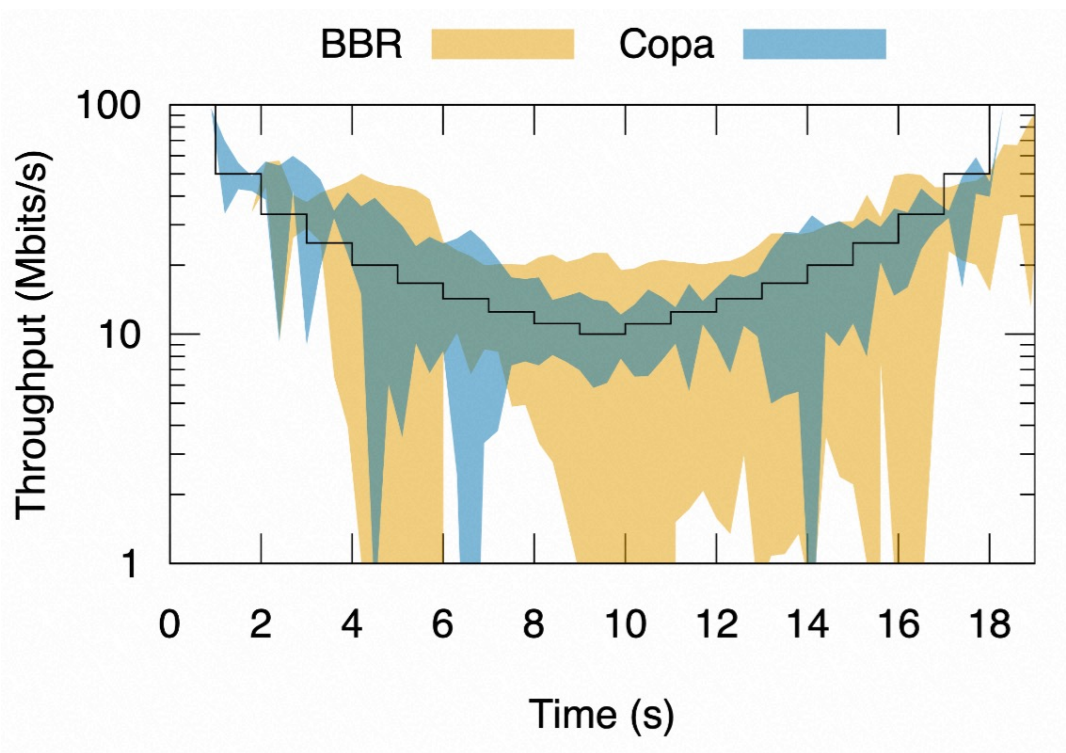
- # python topo.py -c alg_name 创建实验拓扑并测试目标算法
 - 公平性与收敛速度测试函数:
fairness_evaluation(net,cc_name,inter_flow_time = 1,num_flows = 10)每隔 inter_flow_time 加入一条流，直到加入 num_flows 条流为止。之后每 inter_flow_time 一条流结束发送，直到所有流都结束发送
 - 实验过程中， tcpdump 会抓取 h2-eth0 的数据包，供后续公平性分析使用

实验工具

- 公平性分析工具 (`pcap_analyse_tool.py`)
 - `def analyse_algorithm(pcap_file_path, cache=True)` 会分析目标 pcap 文件，由于数据处理会花费大量时间，为了方便后续处理设置了缓存机制
 - `def draw(ax,alg,alpha=0.3,color="r")` 函数说明：以 0.1s 为单位，统计每个时间区间内所有流的平均吞吐量(mean) 和所有流吞吐量的标准差(standard deviation)，绘制出 $\text{mean} \pm \text{standard deviation}$ 的范围

draw() 效果示意图

- 效果如下图所示，图中应有三种拥塞控制算法：Cubic、BBR，以及自己实现的算法。黑色线是理想值。



附件文件列表

- pcap_analyse_tool.py # 分析Pcap文件工具
- topo.py # 实验拓扑
- util.py # 实验相关辅助功能

HTTP服务器实验

HTTP服务器实验

- 实现：使用C语言实现最简单的HTTP服务器
 - 同时支持HTTP（80端口）和HTTPS（443端口）
 - 使用两个线程分别监听各自端口
 - 只需支持GET方法，解析请求报文，返回相应应答及内容

需支持的状态码	场景
200 OK	对于443端口接收的请求，如果程序所在文件夹存在所请求的文件，返回该状态码，以及所请求的文件
301 Moved Permanently	对于80端口接收的请求，返回该状态码，在应答中使用Location字段表达相应的https URL
206 Partial Content	对于443端口接收的请求，如果所请求的为部分内容（请求中有Range字段），返回该状态码，以及相应的部分内容
404 Not Found	对于443端口接收的请求，如果程序所在文件夹没有所请求的文件，返回该状态码

实验流程

- 根据上述要求，实现HTTP服务器程序
- 执行`sudo python topo.py`命令，生成包括两个端节点的网络拓扑
- 在主机h1上运行HTTP服务器程序，同时监听80和443端口
 - `h1 # ./http-server`
- 在主机h2上运行测试程序，验证程序正确性
 - `h2 # python3 test/test.py`
 - 如果没有出现`AssertionError`或其他错误，则说明程序实现正确

HTTP服务器分发视频

- 在主机h1上运行http-server，所在目录下有一个小视频（30秒左右）
- 在主机h2上运行vlc（注意切换成普通用户），通过网络获取并播放该小视频
 - 媒体 -> 打开网络串流 -> 网络 -> 请输入网络URL -> 播放

实验要求

- 提交：代码和实验报告
 - 实现越完整越好，测试越充分越好
 - 通过抓包分析，说明HTTP服务器和VLC客户端是如何传输视频文件的

附件文件列表

- dir
- index.html # 主页文件
- keys # 私钥和证书
- Makefile
- test/test.py # （客户端）测试脚本
- topo.py # 测试拓扑

实验概况

实验题目	选择范围	是否有基础代码	团队人数上限
广播网络实验	非计算机	是	1人
交换机转发实验	非计算机	是	2人
BGP前缀劫持攻击和检测实验	计算机/非计算机	是	2人
TCP公平性实验	计算机	否	2人
HTTP服务器实验	计算机	否	2人

作业在课程网站上提交，截止时间：11月6日（4周时间）