The Internet Ecosystem and Evolution

Lab 2

IP addressing

IP subnetting

- “Real” (hosts on same link) or “logical” (created by the operator for routing purposes) subnets
- **Aggregation:** hosts aggregated into a single prefix
- CIDR (Classless Interdomain Routing): flexible IP subnetting
  - first *X* bits: **subnet identifier**
  - remaining 32-X bits: **host identifier**
  - *X* is marked by the **prefix length** (pl. `/18`) or the **netmask** (e.g., `255.255.192.0`, in dotted-decimal notation)
- Convention: to identify a particular subnet we set the host identifier to zero

IP subnetting

- **Question:** how many IP addresses are aggregated in the `12.130.192.0/21` prefix?
- **Answer:** the host identifier is of *32 – 21 = 11* bits, consequently there are $2^{11}=2048$ separate IP addresses held by the prefix
- **Question:** how many host identifiers can be handed out from this prefix?
- **Answer:** the first address inside the prefix is reserved as the subnet identifier by convention, the last is for subnet multicast, so there remain 2048-2=2046 assignable IP addresses

IP subnetting

- **Question:** which is the first and the last “real” IP address in the prefix `12.130.192.0/21`?
- **Answer:** use the binary representation

CIDR notation	`12.130.192.0/21`
Prefix length	21 bits (from the MSB)
binary	`00001100 10000010 11000000 00000000`
Subnet mask (binary)	`11111111 11111111 11111000 00000000`
Subnet mask (dotted)	`255.255.248.0`
First IP address	`12.130.192.1`
binary	`00001100 10000010 11000000 00000001`
Last IP address	`12.130.199.254` (!!!!)
binary	`00001100 10000010 11000111 11111110`

IP subnetting

- **Question:** which `/19` prefix contains the IP address `73.38.171.112`?
- Note that there is exactly one such `/19`!
- **Answer:** using the binary representation
- Host identifier=`000...` (last 13 bits)

IP address	`73.38.171.112`
Binary	`01001001 00100110 10101011 01110000`
First 19 bits will be the subnet id, the rest (host id) is set to 0	`01001001 00100110 10100000 00000000`
Dotted decimal notation	`73.38.160.0/19`

IP subnetting

- **Question:** does the prefix `153.43.255.0/24` contain the IP address `153.47.255.199`?
- **Answer:** since it is a `/24`, it is enough to look at the first 3 decimal numbers
- Since these differ, the answer is no
- **Question:** does the prefix `189.208.40.0/22` contain the IP address `189.208.44.89`?
- **Answer:** no, the first 22 bits differ

Prefix in binary	10111101 11010000 00101**0**00 00000000
IP address in binary	10111101 11010000 00101**1**00 01011001
Subnet id (/22)	10111101 11010000 00101**1**

Most specific prefix

- Forwarding occurs based on the FIB

Part of the FIB of a router		
IP prefix/length	**Prefix in binary**	**Next-hop IP addr.**
189.110.0.0/15	10111101 0110111	10.0.0.1
189.111.16.0/22	10111101 01101111 000100	10.0.0.2
189.111.18.0/23	10111101 01101111 0001001	10.0.0.3
189.111.17.0/24	10111101 01101111 00010001	10.0.0.4

- **Longest Prefix Match (LPM):** from the prefixes that match on all bits of the subnet id, find the one that matches the address on the most bits
- Can impose specific routing behavior on select groups of hosts (subnets)

Most specific prefix

- **Question:** which is the most specific FIB entry for the IP address `189.111.19.10`?
- **Answer:** `189.111.19.10 = 10111101 01101111 00010011 00001010`
- The first, second, and third entries match on the subnet id, and the third is the longest matching prefix
- **Question:** LPM for `189.111.16.110`?
- **Answer:** only the first two entries match, second is longer
- For IP address `189.111.17.11` the fourth one is the LPM

Aggregation/deaggregation

- **Question:** divide the prefix `1.11.112.0/22` into two `/24` and one `/23` prefixes
- **Answer:** first, split the `/22` into two `/23`s, by setting bit 23 to `0` and `1`, respectively

  `1.11.112.0/22 =`

  `1.11.112.0/23` ∪ `1.11.114.0/23`
- Then, split the first `/23` to two `/24`s at bit 24

  `1.11.112.0/23 =`

  `1.11.112.0/24` ∪ `1.11.113.0/24`
- We could have split the second `/23` as well if we wanted

  `1.11.114.0/23 =`

  `1.11.114.0/24` ∪ `1.11.115.0/24`

IP subnetting: Useful tools

- `ipcalc(1)`: conversion between arbitrary formats: `http://jodies.de/ipcalc`

```
$ ipcalc 203.123.64.0/19
Address:   203.123.64.0          11001011.01111011.010 00000.00000000
Netmask:   255.255.224.0 = 19    11111111.11111111.111 00000.00000000
Wildcard:  0.0.31.255            00000000.00000000.000 11111.11111111
=>
Network:   203.123.64.0/19       11001011.01111011.010 00000.00000000
HostMin:   203.123.64.1          11001011.01111011.010 00000.00000001
HostMax:   203.123.95.254        11001011.01111011.010 11111.11111110
Broadcast: 203.123.95.255        11001011.01111011.010 11111.11111111
Hosts/Net: 8190                   Class C
$ ipcalc 203.123.64.0/19 -s 4000 4000
```

- `libc: inet_aton(3), inet_ntoa(3),` ...
- python: `from netaddr import *`

Typical exam exercises

- How many IP addresses are aggregated into the prefix `120.1.32.0/19`? How many hosts can be assigned an IP address from this prefix? Which s the first and the last assignable IP address within the prefix?
- Which `/14` prefix contains the address `3.41.11.12`?
- Can one aggregate the prefixes `177.143.96.0/21` and `177.143.104.0/21` into a single `/20`?
- Split the prefix `107.14.64.0/19` into a subnet that contains at least 2000 host identifiers plus two other subnets that contain at least 1000 addresses each!

Typical exam exercises

- Which one is the most specific entry in the below FIB for the IP address `10.100.45.1`, `10.100.27.111`, and `10.99.5.5` respectively?

Sample from the FIB of a router	
IP prefix/prefix length	**Next-hop IP addr.**
10.96.0.0/12	10.0.0.1
10.100.0.0/17	10.0.0.2
10.100.16.0/20	10.0.0.3
10.100.32.0/20	10.0.0.4

Generating IP packets: Scapy

Scapy

- Assembling and sending packets easily with essentially arbitrary header fields and content
- Simple packet decoding and dumping (even to pdf!)
- Scanning, fuzzing, traceroute, unit tests
- Protocols/applications/formats supported from the link layer to the application layer
- Security testing of protocol implementations woth specially crafted packets
- Integrated into the powerful `python` programming language

Scapy

- Scapy is readily available in the OpenWRT images inside GNS3
- Start the project from the last lab and enter `R1`

```
root@OpenWrt:/# scapy
Welcome to Scapy (2.3.1)
>>>
```

- Packet to host `10.0.1.2` with maximal TTL

```
>>> packet=IP(dst="10.0.1.2", ttl=255)
>>> packet
<IP  ttl=255 dst=10.0.1.2 |>
>>> packet.show()
```

Scapy

- It is worth saving packets assembled into a variable
- Enough to set essential fields only (rest is automatic)

```
>>> p = IP(dst="10.0.1.2")
>>> p.ttl
64
```

- Show all headers: `p.show()`
- Byte stream: `str(p)`
- Hexadecimal dump: `hexdump(p)`
- Sending the packet: `send(p)` (routing table lookup based on the IP destination address, needs a valid routing table!)

Scapy

- Protocols can be combined with the op. "/"
- Send an HTTP packet to host `10.0.1.2`
- Putting an HTTP header into TCP, Scapy sets the TCP destination port to 80 automatically

```
>>> send(IP(dst="10.0.2.2")/TCP()/"GET / HTTP/1.0\r\n\r\n")
```

- Capturing (sniffing) packets on `R2`

```
root@OpenWrt:/# tcpdump -nvvvv -s 256 -i eth0
[  …] device eth0 entered promiscuous mode
tcpdump: listening on eth0, link-type EN10MB (Ethernet), ...
21:09:00.922471 IP (tos 0x0, ttl 64, …, proto TCP (6), length 58)
    10.0.2.1.20 > 10.0.2.2.80: Flags [S], …, length 18
21:09:00.922508 IP (tos 0x0, ttl 64, …, proto TCP (6), length 40)
    10.0.2.2.80 > 10.0.2.1.20: Flags [R.], …, length 0
```

Exercises

- Set up a GNS3 project with two routers `R1` and `R2`, create IP-layer connectivity (`10.5.0.1/24 – 10.5.0.2/24`), generate packets with Scapy on `R1` and sniff traffic with `tcpdump` on `R2`!
- Determine the MAC address of the other side!

```
>>> send(ARP(op=ARP.who_has, psrc="10.5.0.1",pdst="10.5.0.2"))
```

- Send a valid `ICMP „Echo request”` packet and observe the response!

```
>>> send(IP(dst="10.5.0.2")/ICMP(type=8)/"AAAAAAAAAAA")
```

- Protocol „fuzzing”: security-test protocol implementations with invalid packets!

```
>>> send(fuzz(IP(dst="10.5.0.2",ttl=2)), count=5)
```

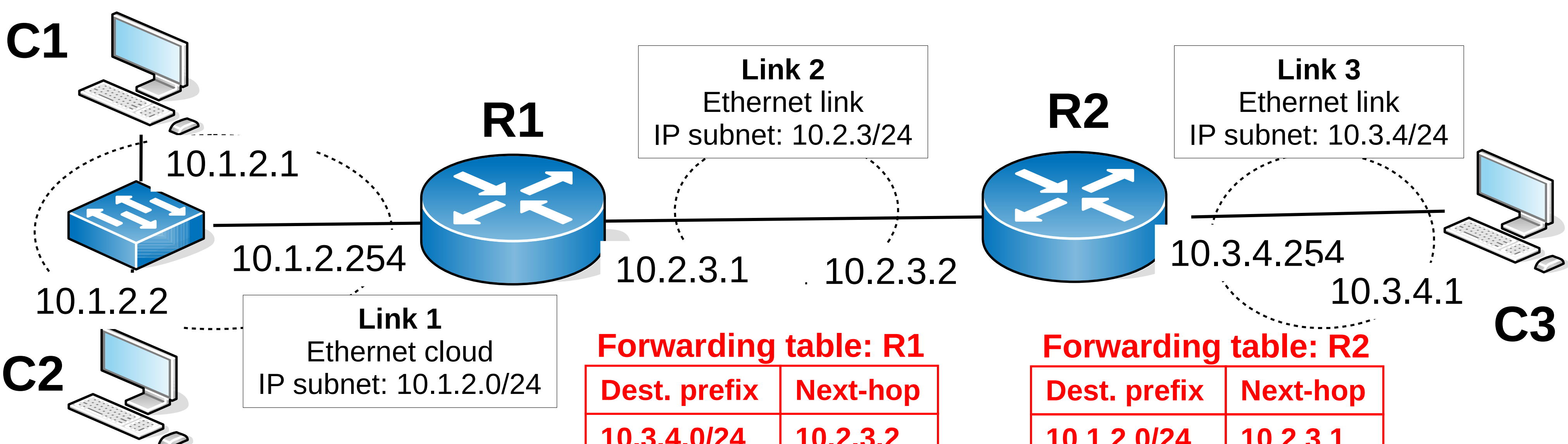
- BGP fuzz testing: (let R2 run BGP: vtysh/”conf t”/”router bgp 10”/exit/exit/exit)

```
>>> sck=socket.socket()                  # python socket
>>> sck.connect(("10.5.0.2",179))        # connecting to the BGP daemon
>>> str=StreamSocket(sck)                # scapy stream for sending pkts
>>> p=IP(dst="10.5.0.2")/TCP(dport=179)/fuzz(Raw()) # „fuzz” BGP pkt
>>> str.send(p)                          # send packet
273                                      # BGP resets the connection
>>> sck.close()                          # close the reset connection
```

IP forwarding

IP Forwarding

- IP forwarding **inside** links (`C1` ↔ `C2`) and **between directly connected links** (`R1: Link1` ↔ `Link2`, `R2: Link2` ↔ `Link3`) is automatic
- IP forwarding **between remote links** (`Link1` ↔ `Link3`) needs a forwarding table (FIB) to be set up at the intermediate routers (`R1` and `R2`)



Forwarding table: R1

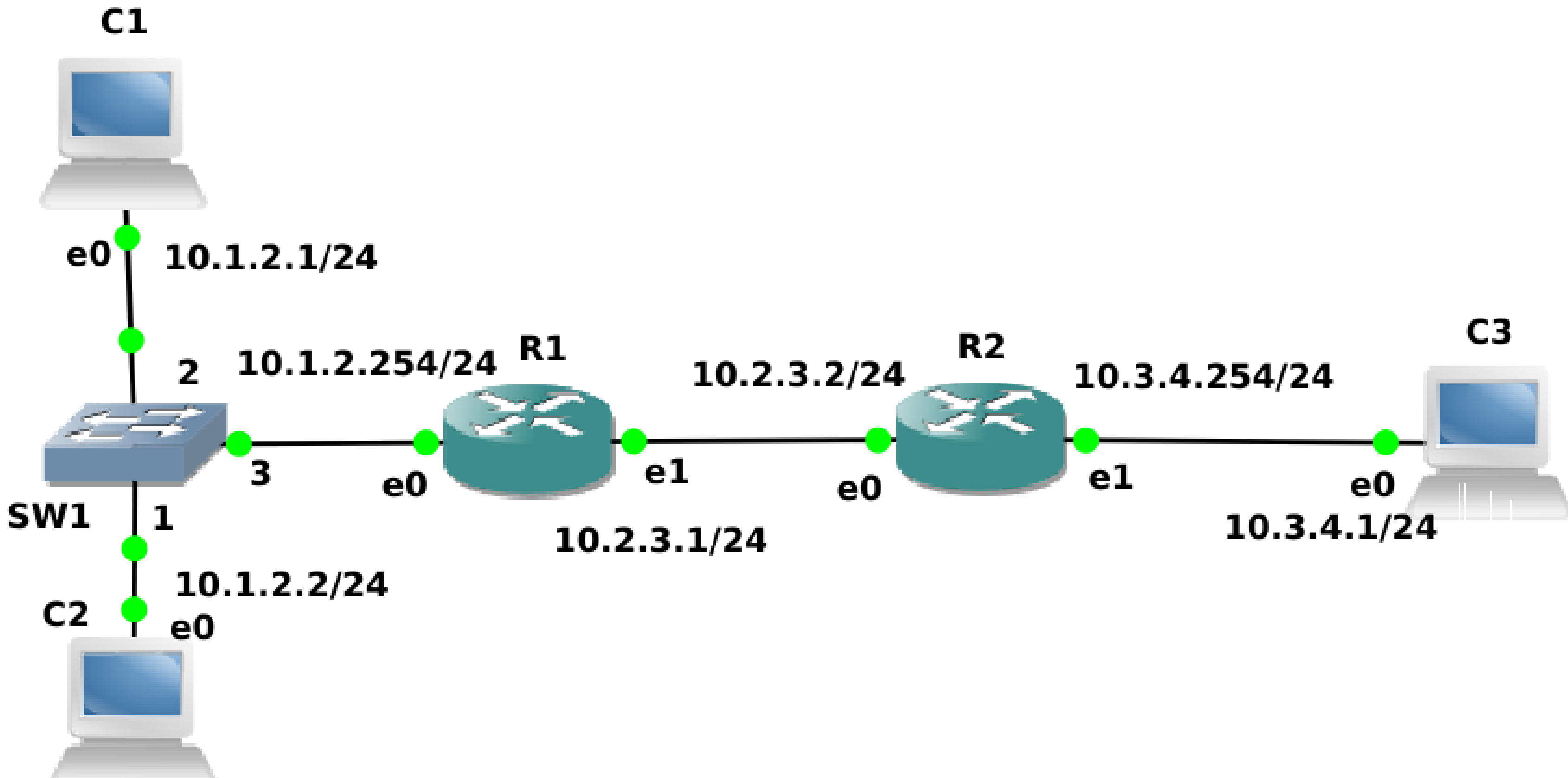
Dest. prefix	Next-hop
10.3.4.0/24	10.2.3.2

Forwarding table: R2

Dest. prefix	Next-hop
10.1.2.0/24	10.2.3.1

Exercise

- Set up the below topology in GNS3, let all hosts and routers run the OpenWRT image (use “Change hostname” and “Change symbol”) and assign interface IP addresses marked in the figure as learned in Lab1!



Exercise

- Set `R1` as default gateway on `C1`

```
OpenWrt# vtysh
OpenWrt# conf t
OpenWrt(config)# ip rou 0.0.0.0/0 10.1.2.254
OpenWrt(config)# exit
```

- Similarly, set the default gateway on `C2` to `R1` and on `C3` to `R2`!

1) Ping `C2` (`10.1.2.2`) and `R1` (`10.1.2.254`) from `C1` and observe what happens!
2) Ping `C3` (`10.3.4.1`) from `C1` and explain what happens (use `tcpdump` to capture packets at `R1` and `R2`)!

Exercise

3) Add a forwarding table entry at `R1` so that it learn the whereabouts of `Link3`

```
OpenWrt# vtysh
OpenWrt# conf t
OpenWrt(config)# ip rou 10.3.4.0/24 10.2.3.2
OpenWrt(config)# exit
```

– Ping `C3` from `C1` and explain what happens now!

4) Add a “reverse” route to `R2` so that it learn the next-hop to `Link1`

```
OpenWrt# vtysh
OpenWrt# conf t
OpenWrt(config)# ip rou 10.3.4.0/24 10.2.3.2
OpenWrt(config)# exit
```

– Ping `C3` from `C1` now and explain what you see!