Az internet ökoszisztémája és evolúciója

Gyakorlat 2

IP címzés

IP subnetting

- „Valós” (hosztok azonos linken) vagy „logikai” alhálózat (operátor által routing célokra „kreált”)
- **Aggregáció:** sok hoszt azonos prefixen látszik
- CIDR (Classless Interdomain Routing): az IP címek **alhálózati prefixbe (subnet)** gyűjthetők
  - az első X bit az **alhálózat-azonosító**
  - a maradék 32-X bit **hosztazonosító**
  - X-et a **prefix hossz** (pl. `/18`) vagy a **netmask** (pl. `255.255.192.0`) adja meg
- Konvenció: ha az alhálózatról beszélünk, akkor a hosztazonosítót zérus bitekkel töltjük fel

IP subnetting: példa

- **Kérdés:** hány IP címet tartalmaz a `12.130.192.0/21` prefix?
- **Válasz:** a hosztazonosító hossza *32 – 21 = 11*, vagyis a prefixen belül összesen $2^{11}=2048$ IP cím azonosítható egyértelműen
- **Kérdés:** hány hosztazonosító adható ki ebből a prefixből?
- **Válasz:** a prefix első címe nem osztható ki (ez a prefix azonosítására szolgál konvenció szerint), az utolsó sem (multicast), így a válasz 2046

IP subnetting: példa

- **Kérdés:** melyik az első és melyik az utolsó IP cím a `12.130.192.0/21` prefixben?
- **Válasz:** érdemes a bináris reprezentációt használni

CIDR notation	`12.130.192.0/21`
Prefix hossza	21 bit (az MSB-től)
bináris	`00001100 10000010 11000000 00000000`
Subnet mask (bináris)	`11111111 11111111 11111000 00000000`
Subnet mask (dotted)	`255.255.248.0`
Első IP cím	`12.130.192.1`
bináris	`00001100 10000010 11000000 00000001`
Utolsó IP cím	`12.130.199.254` (!!!!)
bináris	`00001100 10000010 11000111 11111110`

IP subnetting: példa

- **Kérdés:** melyik `/19` prefix tartalmazza a `73.38.171.112` IP címet?
- Érdekesség: pontosan 1 ilyen `/19` van!
- **Válasz:** a bináris reprezentációt használva
- Hoszt-azonosító=`000...` (az utolsó 13 bit)

IP cím	`73.38.171.112`
Bináris	`01001001 00100110 10101011 01110000`
Az első 19 bit a prefix azonosító a maradékot kinullázzuk	`01001001 00100110 10100000 00000000`
Dotted decimál notation	`73.38.160.0/19`

IP subnetting: példa

- **Kérdés:** tartalmazza-e a `153.43.255.0/24` prefix a `153.47.255.199` IP címet?
- **Válasz:** mivel `/24`-ről van szó, elég az első 3 decimális számot nézni a címben és prefixben
- Mivel ezek nem egyeznek meg, a válasz nem
- **Kérdés:** tartalmazza-e a `189.208.40.0/22` prefix a `189.208.44.89` IP címet?
- **Válasz:** nem, mert az első 22 bit különbözik

A prefix binárisan	10111101 11010000 00101**0**00 00000000
Az IP cím binárisan	10111101 11010000 00101**1**00 01011001
Az alhálózat azonosítója	10111101 11010000 00101**1**

Legspecifikusabb prefix

- A csomagtovábbítás a FIB alapján történik

Egy router FIB-jének részlete		
IP prefix/hossz	**A prefix binárisan**	**Next-hop IP címe**
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

- Minden IP címre meg kell találni az arra legtöbb biten illeszkedő bejegyzést: **Longest Prefix Match (LPM)**
- Alhálózatokra speciális útválasztási döntések érvényesíthetők

Legspecifikusabb prefix: példa

- **Kérdés:** melyik a legspecifikusabb bejegyzés a `189.111.19.10` IP címre?
- **Válasz:** `189.111.19.10 = 10111101 01101111 00010011 00001010`
- Az első, a második, és a harmadik bejegyzés illeszkedik, így a harmadik a legspecifikusabb
- **Kérdés:** LPM a `189.111.16.110` IP címre?
- **Válasz:** csak az első két bejegyzés illeszkedik
- A `189.111.17.11` IP címre a negyedik a legspecifikusabb bejegyzés

Aggregáció/deaggrregáció

- **Kérdés:** osszuk fel a `1.11.112.0/22` prefixet 2 darab `/24`-re és egy `/23`-ra
- **Válasz:** először osszuk két `/23`-ra, a 23. bit `0` és `1` értéke (helyi érték: 2) generálja a két `/23`-t
  `1.11.112.0/22 =`
  `1.11.112.0/23 ∪ 1.11.114.0/23`
- Majd az első `/23`-at osszuk két `/24`-re
  `1.11.112.0/23 =`
  `1.11.112.0/24 ∪ 1.11.113.0/24`
- Feloszthattuk volna a másodikat is...

IP subnetting: eszközök

- `ipcalc(1)`: konverzió tetszőleges formátumok között: `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 *`

Tipikus ZH feladatok

- Hány IP címet tartalmaz a `120.1.32.0/19` prefix? Hány hosztazonosító adható ki ebből a prefixből? Melyik az első és melyik az utolsó IP cím a `120.1.32.0/19` prefixben?
- Melyik `/14` prefix tartalmazza a `3.41.11.12` IP címet?
- Aggregálható-e a `177.143.96.0/21` és a `177.143.104.0/21`?
- Ossza fel a `107.14.64.0/19` prefixet egy legalább 2000 és két, legalább 1000 címet tartalmazó alhálózatra!

Tipikus ZH feladatok

- Melyik a legspecifikusabb bejegyzés az alábbi FIBben a `10.100.45.1`, `10.100.27.111`, illetve a `10.99.5.5` IP címre?

Egy router FIB-jének részlete	
IP prefix/prefix hossz	**Next-hop IP címe**
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

Csomagok generálása: Scapy

Scapy

- Csomagok összeállítása és küldése tetszőleges tartalommal
- Egyszerű csomagdekódokolás és -kiiratás (akár pdf-be!)
- Szkennelés, traceroute, unit tesztek
- Formátumok támogatása a link rétegtől egészen az alkalmazási rétegig
- Protokollok elleni célzott támadások speciálisan formázott csomagokkal
- Mindez a python szkriptnyelvbe integrálva

Scapy

- A scapy telepítve van az OpenWRT image-ben
- Indítsuk el a múltkori gyakorlaton elmentett topológiát és lépjünk be az `R1`-re

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

- Csomag a `10.0.1.2` címre, maximális TTL-lel

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

Scapy

- Csomag tartalmát érdemes változóba tenni
- Elég a fontos mezőket megadni (többi automatikus)

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

- Összes mező: `p.show()`
- Byte-sorozat: `str(p)`
- Hexa tartalom: `hexdump(p)`
- Csomag küldése: `send(p)` (kitöltött routing tábla kell, különben nem tudjuk elküldeni!)

Scapy

- Protokollok a ″/″ operátorral kombinálhatók
- Küldjünk HTTP csomagot a `10.0.1.2`-re
- A TCP csomagba HTTP tartalmat helyezve a Scapy automatikusan 80-ra állítja a TCP portot

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

- Csomagok elkapása az `R2`-n

```
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
```

Feladatok

- Állítsunk be egy GNS3 hálózatot két routerrel, létesítsünk IP-szintű kapcsolatot (10.5.0.1/24–10.5.0.2/24), generáljunk csomagokat a scapyvel az egyik routeren és figyeljük meg a forgalmat a `tcpdump` programmal a másik oldalon!
- Állapítsuk meg a másik oldali interfész MAC címét!

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

- Küldjünk ICMP „Echo request” csomagot!

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

- Protokoll „fuzzing”: protokoll security tesztelés érvénytelen csomagokkal

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

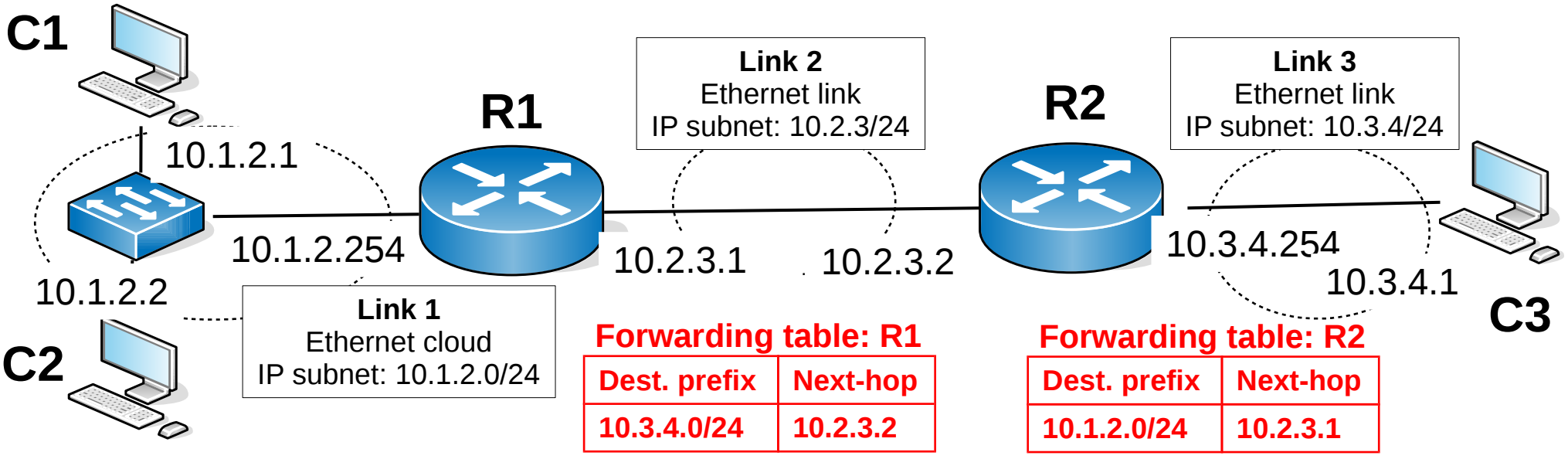
- BGP fuzz testing: (R2-n fusson a BGP: vtysh/”conf t”/”router bgp 10”/exit/exit/exit)

```
>>> sck=socket.socket()                    # python socket
>>> sck.connect(("10.5.0.2",179))          # kapcsolódás a BGP daemon-hoz
>>> str=StreamSocket(sck)                  # scapy stream, csomagküldéshez
>>> p=IP(dst="10.5.0.2")/TCP(dport=179)/fuzz(Raw()) # „fuzz” BGP pkt
>>> str.send(p)                            # csomag elküldése
273                                        # a BGP bont: RST-t küld
>>> sck.close()                            # reset-elt kapcsolat bontása
```

IP csomagtovábbítás

IP csomagtovábbítás

- Az IP csomagtovábbítás **linken belül** (`C1` ↔ `C2`) és **közvetlenül szomszédos linkek között** (`R1`: `Link1` ↔ `Link2`, `R2`: `Link2` ↔ `Link3`) automatikus
- A csomagtovábbítás **távoli linkek között** (`Link1` ↔ `Link3`) csomagtovábbítási tábla (FIB) alapján történik, amit a közbülső routereken (`R1` és `R2`) fel kell konfigurálni



Forwarding table: R1

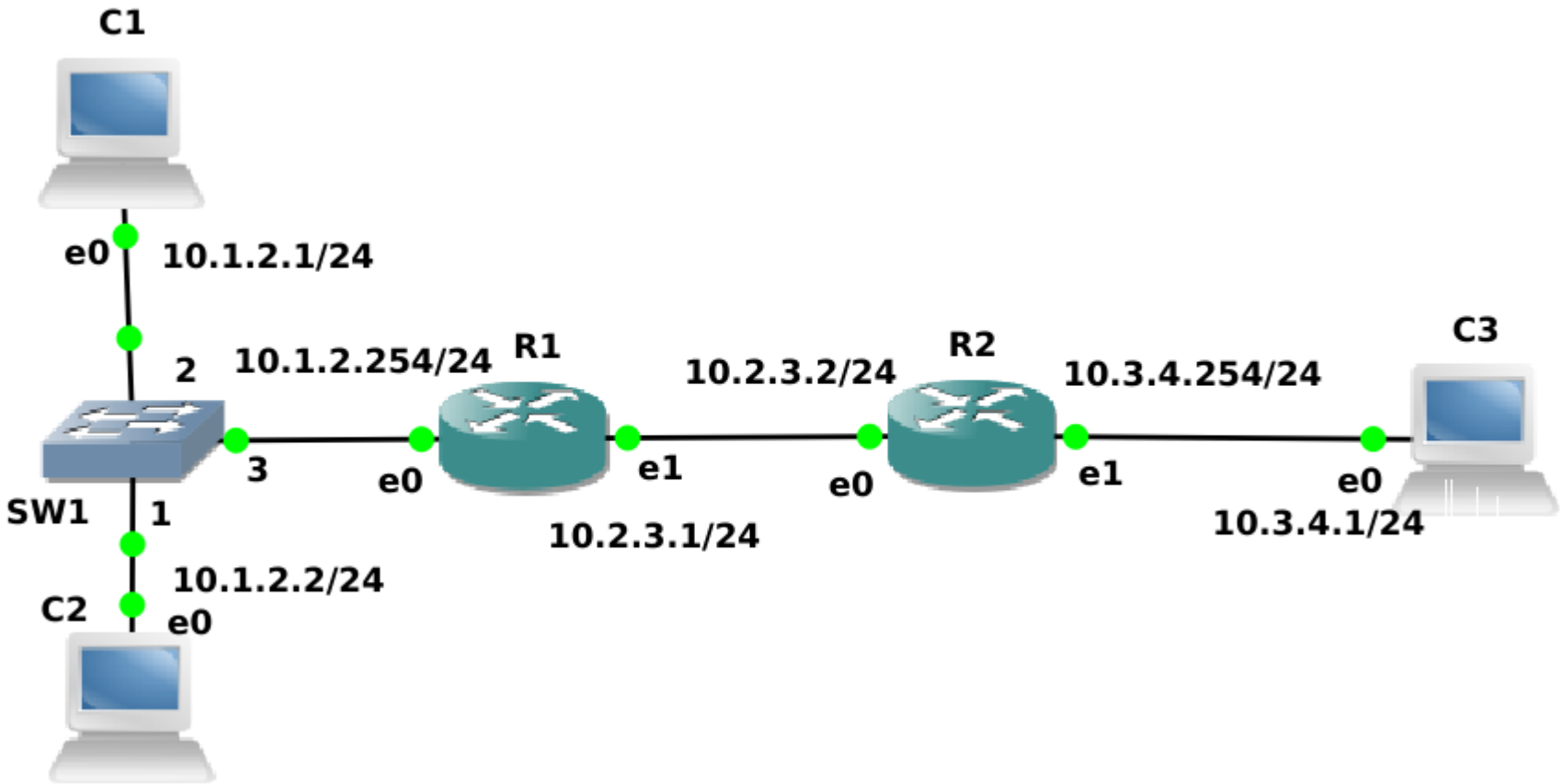
Dest. prefix	Next-hop
10.3.4.0/24	10.2.3.2

Forwarding table: R2

Dest. prefix	Next-hop
10.1.2.0/24	10.2.3.1

Feladat

- Konfiguráljuk az alábbi topológiát a GNS3-ban, a hosztok és a routerek futtassák az OpenWRT image-et (használjuk értelemszerűen a “Change hostname” és a “Change symbol” menüket) és osszunk ki interfész IP címeket az ábra szerint!



Feladat

- Legyen `R1` a default gateway `C1`-en

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

- Hasonlóan, legyen a default gateway a `C2`-n az `R1` és a `C3`-on az `R2`!

1) Pingeljük a `C2` hosztot (`10.1.2.2`) és az `R1` routert (`10.1.2.254`) `C1`-ről és figyeljük meg, mi történik!

2) Pingeljük a `C3` hosztot (`10.3.4.1`) a `C1`-ről és a `tcpdump` segítségével kapjuk el a csomagokat `R1` és `R2` routereken! Mit látunk?

Feladat

3) Adjunk egy bejegyzést az `R1` csomagtovábbítási táblájához, hogy tudja, merre található a `Link3`

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

- Pingeljük ismét `C3` hosztot `C1` hosztról és magyarázzuk meg, mit látunk!

4) Adjunk egy “reverse” route-ot `R2` router FIBjéhez!

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

- Újra pingeljük `C3`-at `C1`-ről. Mi történik?