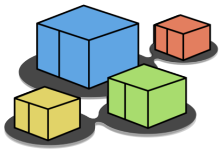
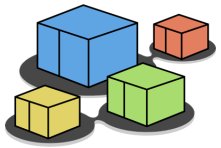


Kathará

kathara lab

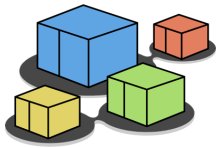
rip with FRRouting

Version	1.1
Author(s)	G. Di Battista, M. Patrignani, M. Pizzonia, L. Ariemma, M. Scazzariello, T. Caiazzi
E-mail	contact@kathara.org
Web	http://www.kathara.org/
Description	experiences with the ripv2 distance vector routing protocol – derives from kathara rip lab ver. 1.2 which, in turns, derives from netkit rip lab ver. 2.4



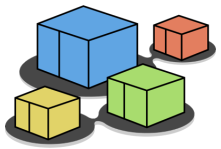
copyright notice

- All the pages/slides in this presentation, including but not limited to, images, photos, animations, videos, sounds, music, and text (hereby referred to as "material") are protected by copyright.
- This material, with the exception of some multimedia elements licensed by other organizations, is property of the authors and/or organizations appearing in the first slide.
- This material, or its parts, can be reproduced and used for didactical purposes within universities and schools, provided that this happens for non-profit purposes.
- Information contained in this material cannot be used within network design projects or other products of any kind.
- Any other use is prohibited, unless explicitly authorized by the authors on the basis of an explicit agreement.
- The authors assume no responsibility about this material and provide this material "as is", with no implicit or explicit warranty about the correctness and completeness of its contents, which may be subject to changes.
- This copyright notice must always be redistributed together with the material, or its portions.



the RIP protocol

- RIP is a routing protocol
- RIPS is a distance-vector routing protocol
  - approach: send all your information to a few
  - update your routing information based on what you hear
- in this lab we will see an example of RIPv2 protocol on the FRRouting Suite

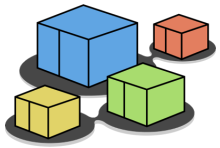


sample `frr.conf` configuration file

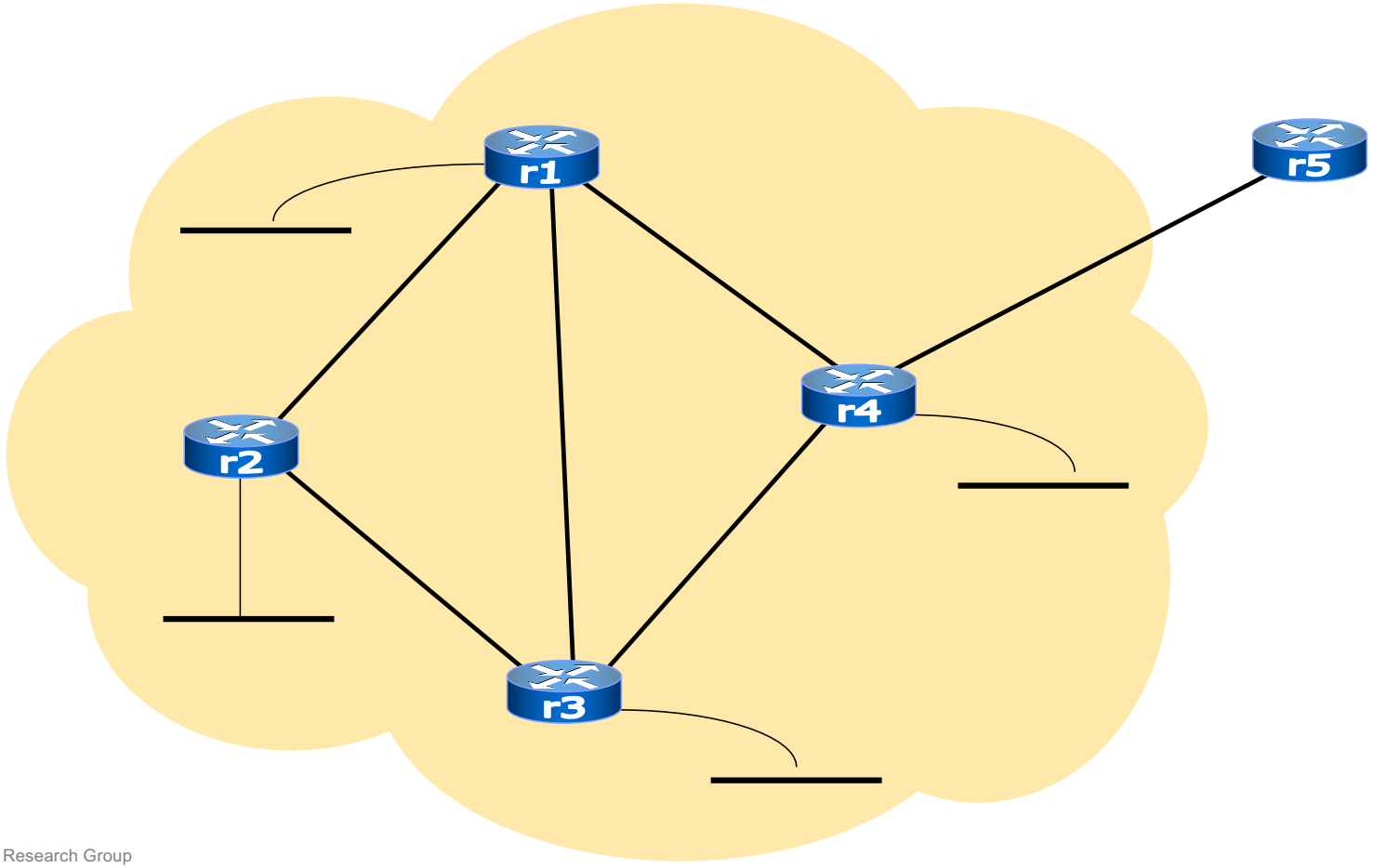
```
root@pc1:~$ cat /etc/frr/frr.conf
!
! FRRouting configuration file
!
!
!   RIP CONFIGURATION
!
router rip
network 100.1.0.0/16
!
log file /var/log/frr/frr.log
```

talk rip on some interface

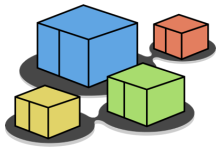
send rip multicast packets to interfaces falling into this prefix



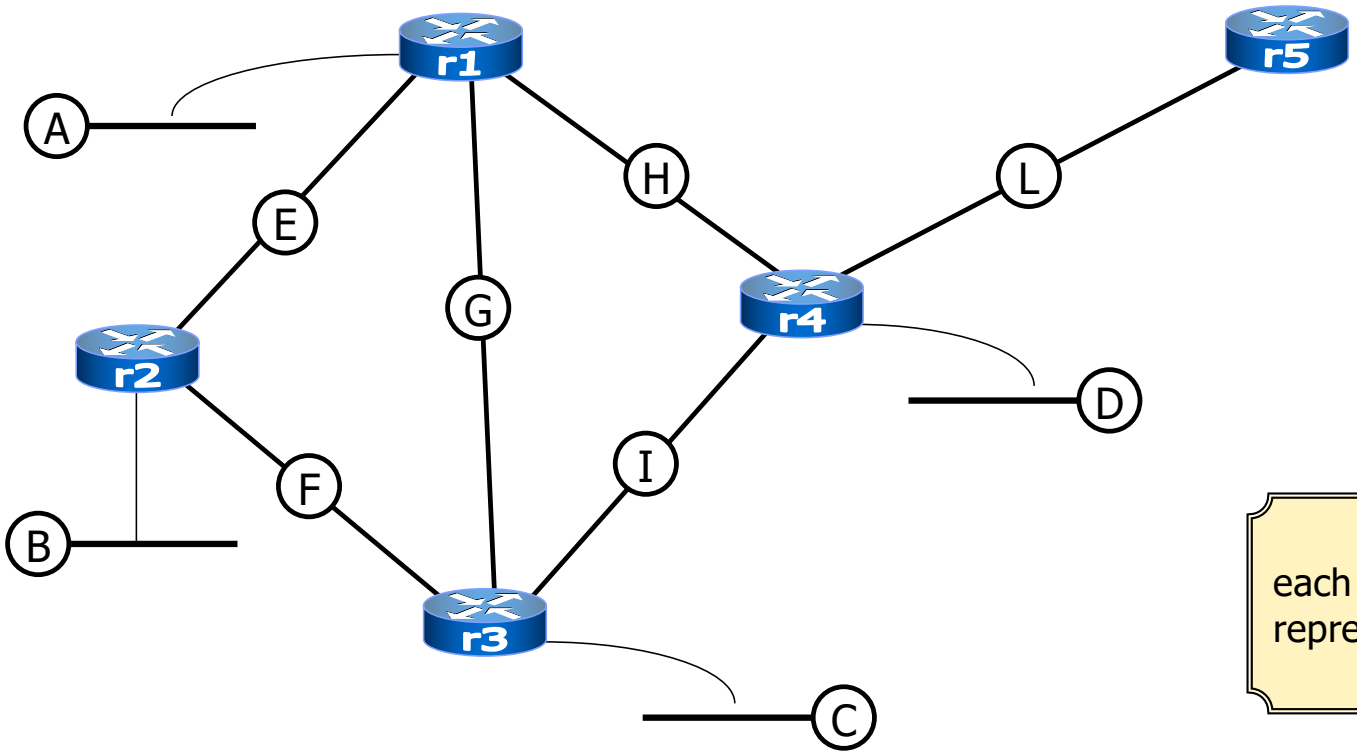
small network connected to Internet



last update: Oct 2023

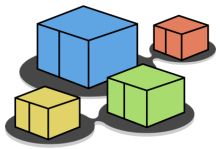


the involved ip subnets

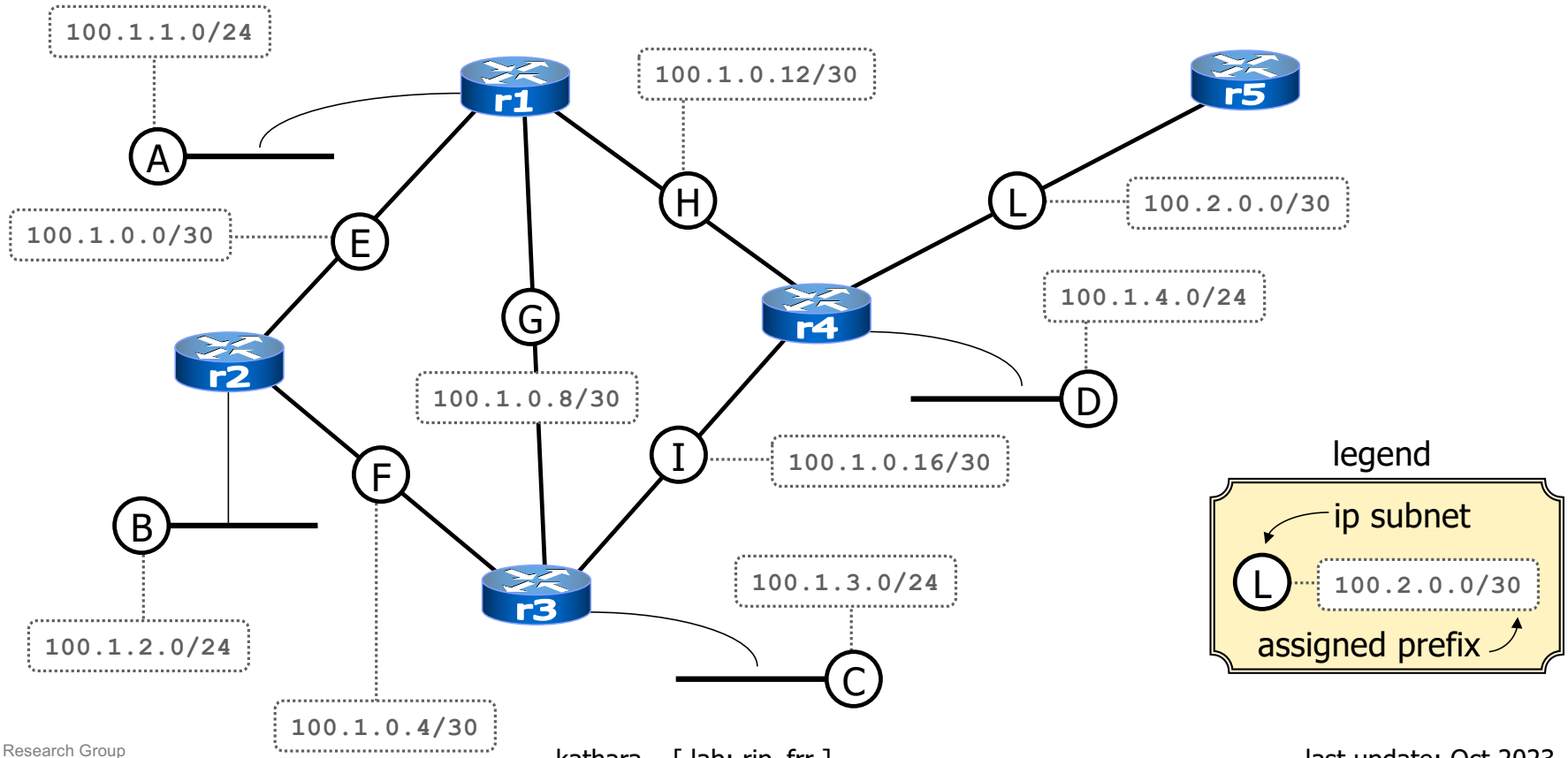


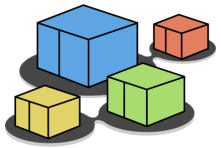
legend

each circle ○ represents a subnet

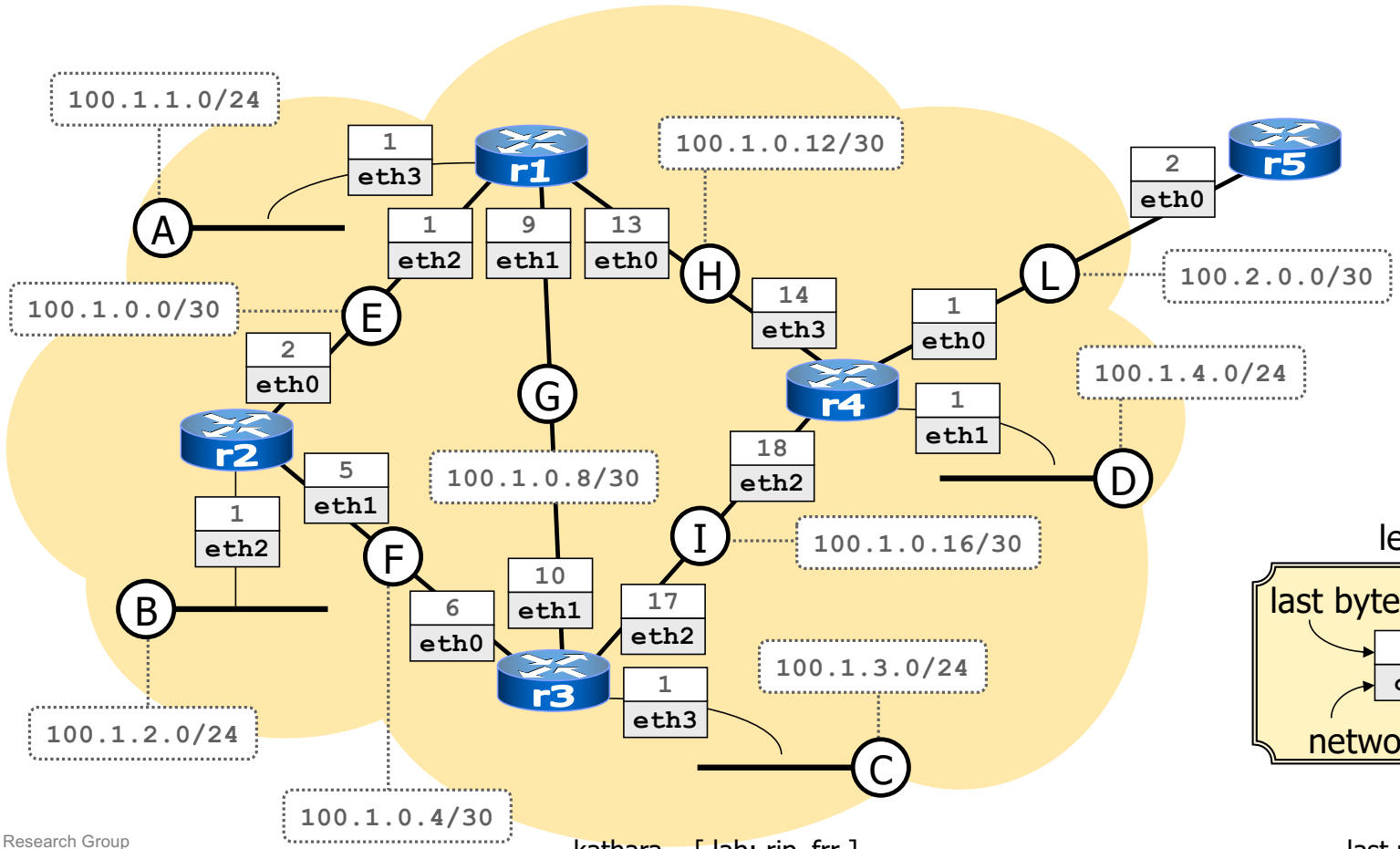


assigning ip numbers to subnets

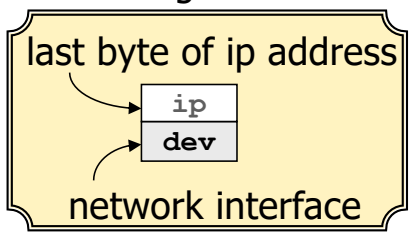


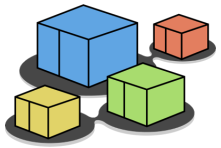


assigning ip numbers to interfaces



legend

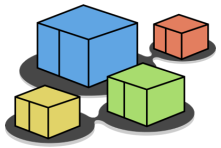




launching the lab

```
root@localhost:~$ cd kathara-lab_rip
root@localhost:~/kathara-lab_rip$ kathara lstart
```

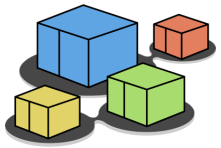
- the lab configuration is such that the frr routing daemon is not automatically started



checking connectivity

- towards a directly connected destination

```
root@r4:~$ ping 100.1.0.13
PING 100.1.0.13 (100.1.0.13) 56(84) bytes of data.
64 bytes from 100.1.0.13: icmp_seq=1 ttl=64 time=1.23 ms
64 bytes from 100.1.0.13: icmp_seq=2 ttl=64 time=0.592 ms
64 bytes from 100.1.0.13: icmp_seq=3 ttl=64 time=0.393 ms

--- 100.1.0.13 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2032ms
rtt min/avg/max/mdev = 0.393/0.741/1.238/0.360 ms
```

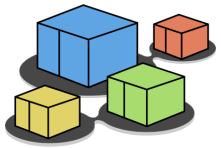


checking connectivity

- towards a remote destination

```
root@r4:~$ ping 100.1.2.1
connect: Network is unreachable
```

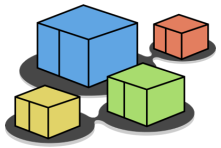
- what's going on?



examining the kernel routing table

```
root@r4:~$ ip route
100.1.0.12/30 dev eth3 proto kernel scope link src 100.1.0.14
100.1.0.16/30 dev eth2 proto kernel scope link src 100.1.0.18
100.1.4.0/24 dev eth1 proto kernel scope link src 100.1.4.1
100.2.0.0/30 dev eth0 proto kernel scope link src 100.2.0.1
```

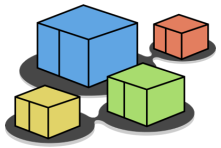
- since no routing daemon is currently running, only directly connected destinations are known to the router



starting the routing daemons

- on each router (but **`r5`**) issue the following command:

```
root@r4:~$ systemctl start frr
```

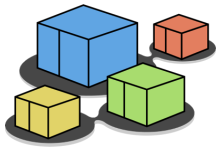


checking connectivity (again)

- towards a remote destination

```
root@r4:~$ ping 100.1.2.1
PING 100.1.2.1 (100.1.2.1) 56(84) bytes of data.
64 bytes from 100.1.2.1: icmp_seq=1 ttl=63 time=0.743 ms
64 bytes from 100.1.2.1: icmp_seq=2 ttl=63 time=0.875 ms
64 bytes from 100.1.2.1: icmp_seq=3 ttl=63 time=0.685 ms

--- 100.1.2.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.685/0.767/0.875/0.085 ms
```

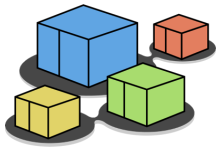
- after a while, all remote destinations are reachable



checking the routing table

- the routing table is now updated

```
root@r4:~$ ip route
100.1.0.0/30 nhid 12 via 100.1.0.13 dev eth3 proto rip metric 20
100.1.0.4/30 nhid 10 via 100.1.0.17 dev eth2 proto rip metric 20
100.1.0.8/30 nhid 10 via 100.1.0.17 dev eth2 proto rip metric 20
100.1.0.12/30 dev eth3 proto kernel scope link src 100.1.0.14
100.1.0.16/30 dev eth2 proto kernel scope link src 100.1.0.18
100.1.1.0/24 nhid 12 via 100.1.0.13 dev eth3 proto rip metric 20
100.1.2.0/24 nhid 12 via 100.1.0.13 dev eth3 proto rip metric 20
100.1.3.0/24 nhid 10 via 100.1.0.17 dev eth2 proto rip metric 20
100.1.4.0/24 dev eth1 proto kernel scope link src 100.1.4.1
100.2.0.0/30 dev eth0 proto kernel scope link src 100.2.0.1
```

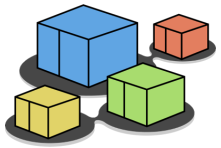


a look at ripv2 packets

- let's sniff ripv2 packets

```
root@r4:~$ tcpdump -tenni eth2 -v
```

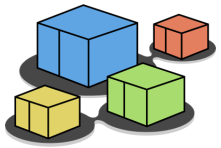
display packet details
(enable full protocol decoding)



a look at ripv2 packets

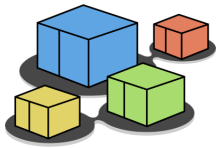
- let's sniff ripv2 packets

```
root@r4:~$ tcpdump -tenni eth2 -v
76:b9:c2:6b:d7:d8 > 01:00:5e:00:00:09, ethertype IPv4 (0x0800), length 166:
(tos 0xc0, ttl 1, id 62062, offset 0, flags [DF], proto UDP (17), length 152)
    100.1.0.17.520 > 224.0.0.9.520:
RIPv2, Response, length: 124, routes: 6 or less
  AFI IPv4,       100.1.0.0/30, tag 0x0000, metric: 2, next-hop: self
  AFI IPv4,       100.1.0.4/30, tag 0x0000, metric: 1, next-hop: self
  AFI IPv4,       100.1.0.8/30, tag 0x0000, metric: 1, next-hop: self
  AFI IPv4,       100.1.1.0/24, tag 0x0000, metric: 2, next-hop: self
  AFI IPv4,       100.1.2.0/24, tag 0x0000, metric: 2, next-hop: self
  AFI IPv4,       100.1.3.0/24, tag 0x0000, metric: 1, next-hop: self
```



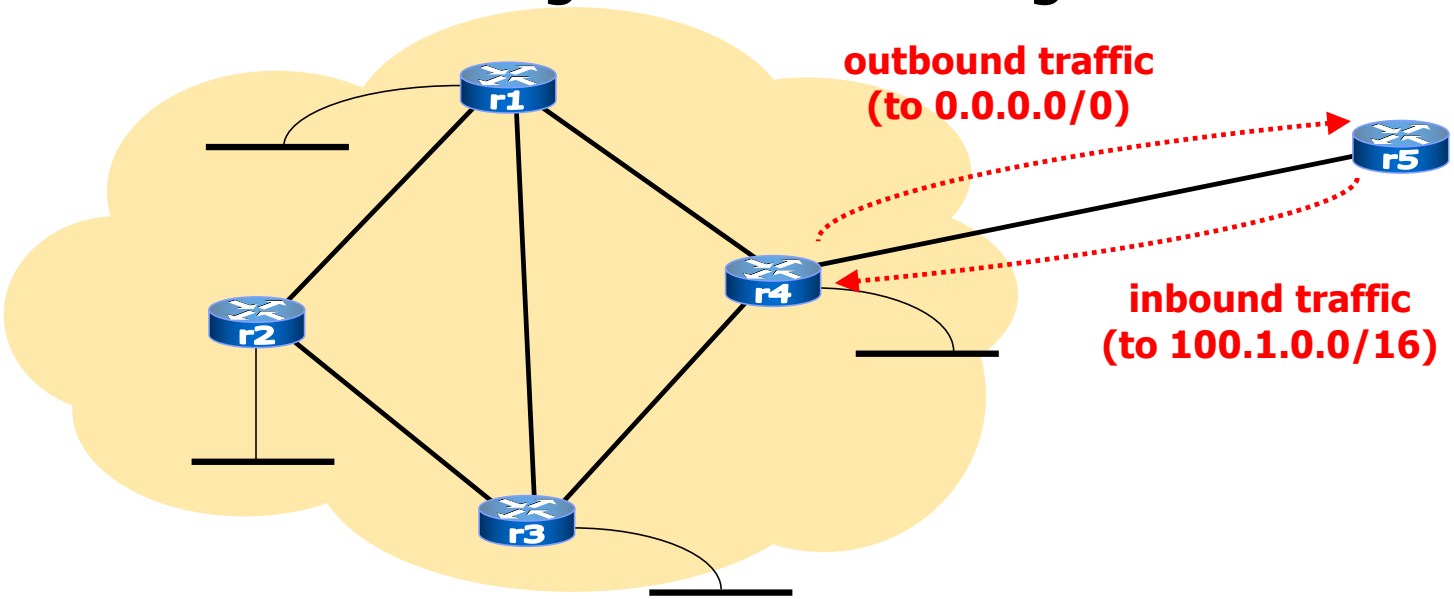
inspecting the rip routing table

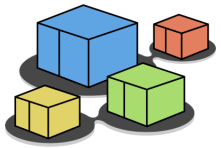
```
root@r4:~$ vtysh
r4# show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

     Network            Next Hop         Metric From            Tag Time
R(n) 100.1.0.0/30       100.1.0.13            2 100.1.0.13        0 02:55
R(n) 100.1.0.4/30       100.1.0.17            2 100.1.0.17        0 02:27
R(n) 100.1.0.8/30       100.1.0.17            2 100.1.0.17        0 02:27
C(i) 100.1.0.12/30      0.0.0.0               1 self              0
C(i) 100.1.0.16/30      0.0.0.0               1 self              0
R(n) 100.1.1.0/24       100.1.0.13            2 100.1.0.13        0 02:55
R(n) 100.1.2.0/24       100.1.0.13            3 100.1.0.13        0 02:55
R(n) 100.1.3.0/24       100.1.0.17            2 100.1.0.17        0 02:27
C(r) 100.1.4.0/24       0.0.0.0               1 self              0
C(r) 100.2.0.0/30       0.0.0.0               1 self              0
```



static routing

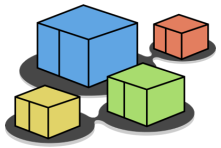
- our network is a stub network (i.e., it has just one connection to an external router, `r5`); hence, static routes are enough for connecting it to the internet





adding a static route to `r5`

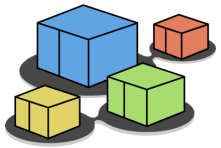
```
root@r5:~$ ip route add 100.1.0.0/16 via 100.2.0.1
root@r5:~$ ping 100.1.2.1
PING 100.1.2.1 (100.1.2.1) 56(84) bytes of data.
64 bytes from 100.1.2.1: icmp_seq=1 ttl=62 time=24.1 ms
64 bytes from 100.1.2.1: icmp_seq=2 ttl=62 time=1.11 ms

--- 100.1.2.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1023ms
rtt min/avg/max/mdev = 1.117/12.634/24.151/11.517 ms
```



configuring r4

- step 1: configuring the default route

```
root@r4:~$ ip route add default via 100.2.0.2
root@r4:~$ ip route
default via 100.2.0.2 dev eth0
100.1.0.0/30 nhid 12 via 100.1.0.13 dev eth3 proto rip metric 20
100.1.0.4/30 nhid 10 via 100.1.0.17 dev eth2 proto rip metric 20
100.1.0.8/30 nhid 10 via 100.1.0.17 dev eth2 proto rip metric 20
100.1.0.12/30 dev eth3 proto kernel scope link src 100.1.0.14
100.1.0.16/30 dev eth2 proto kernel scope link src 100.1.0.18
100.1.1.0/24 nhid 12 via 100.1.0.13 dev eth3 proto rip metric 20
100.1.2.0/24 nhid 12 via 100.1.0.13 dev eth3 proto rip metric 20
100.1.3.0/24 nhid 10 via 100.1.0.17 dev eth2 proto rip metric 20
100.1.4.0/24 dev eth1 proto kernel scope link src 100.1.4.1
100.2.0.0/30 dev eth0 proto kernel scope link src 100.2.0.1
```



configuring r4

- step 2: propagating the default route into rip

```
root@r4:~$ vtysh

Hello, this is FRRouting (version 8.0.1).
Copyright 1996-2005 Kunihiro Ishiguro,

r4# configure
r4(config)# router rip
r4(config-router)# route 0.0.0.0/0
r4(config-router)# quit
r4(config)# quit
r4# exit
```

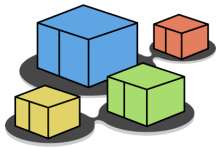
begin configuration

configure the rip protocol

statically configure the default route

end of rip configuration

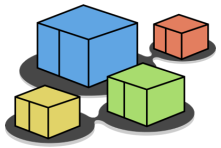
end configuration



the default route

- after a while, the default route has been injected (via rip) into the network

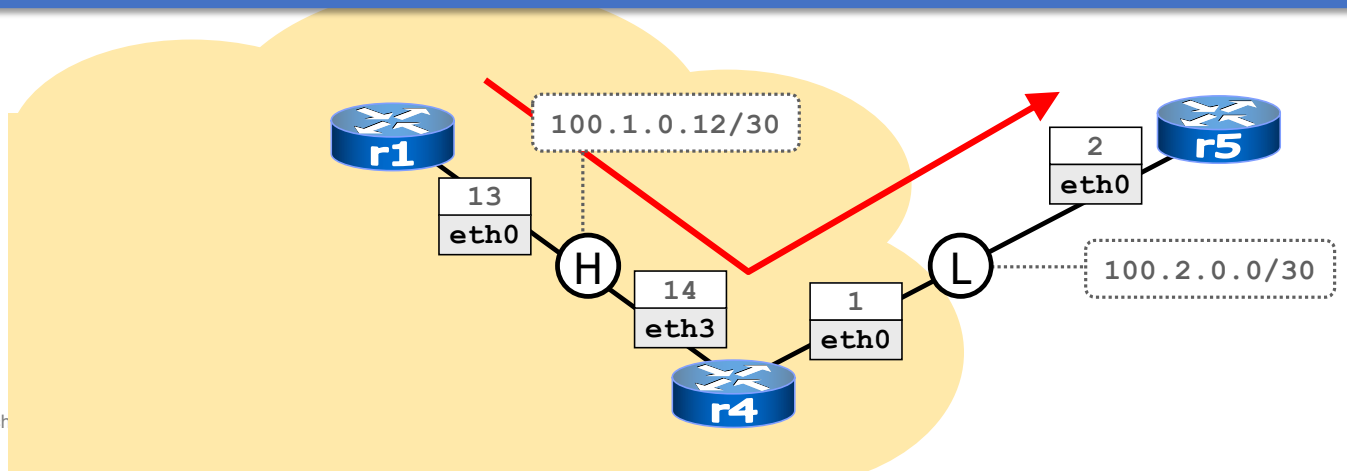
```
root@r1:~$ ip route
default nhid 12 via 100.1.0.14 dev eth0 proto rip metric 20
100.1.0.0/30 dev eth2 proto kernel scope link src 100.1.0.1
100.1.0.4/30 nhid 11 via 100.1.0.10 dev eth1 proto rip metric 20
100.1.0.8/30 dev eth1 proto kernel scope link src 100.1.0.9
100.1.0.12/30 dev eth0 proto kernel scope link src 100.1.0.13
100.1.0.16/30 nhid 12 via 100.1.0.14 dev eth0 proto rip metric 20
100.1.1.0/24 dev eth3 proto kernel scope link src 100.1.1.1
100.1.2.0/24 nhid 14 via 100.1.0.2 dev eth2 proto rip metric 20
100.1.3.0/24 nhid 11 via 100.1.0.10 dev eth1 proto rip metric 20
100.1.4.0/24 nhid 12 via 100.1.0.14 dev eth0 proto rip metric 20
100.2.0.0/30 nhid 12 via 100.1.0.14 dev eth0 proto rip metric 20
```

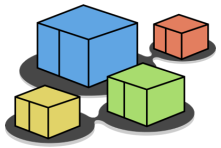


checking connectivity

any (even non-existing) destination

```
root@r1:~$ ping 193.204.161.1
PING 193.204.161.1 (193.204.161.1) 56(84) bytes of data.
From 100.2.0.2 icmp_seq=1 Destination Net Unreachable
From 100.2.0.2 icmp_seq=2 Destination Net Unreachable

--- 193.204.161.1 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss,
time 999ms
```

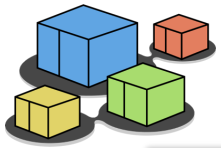




checking connectivity

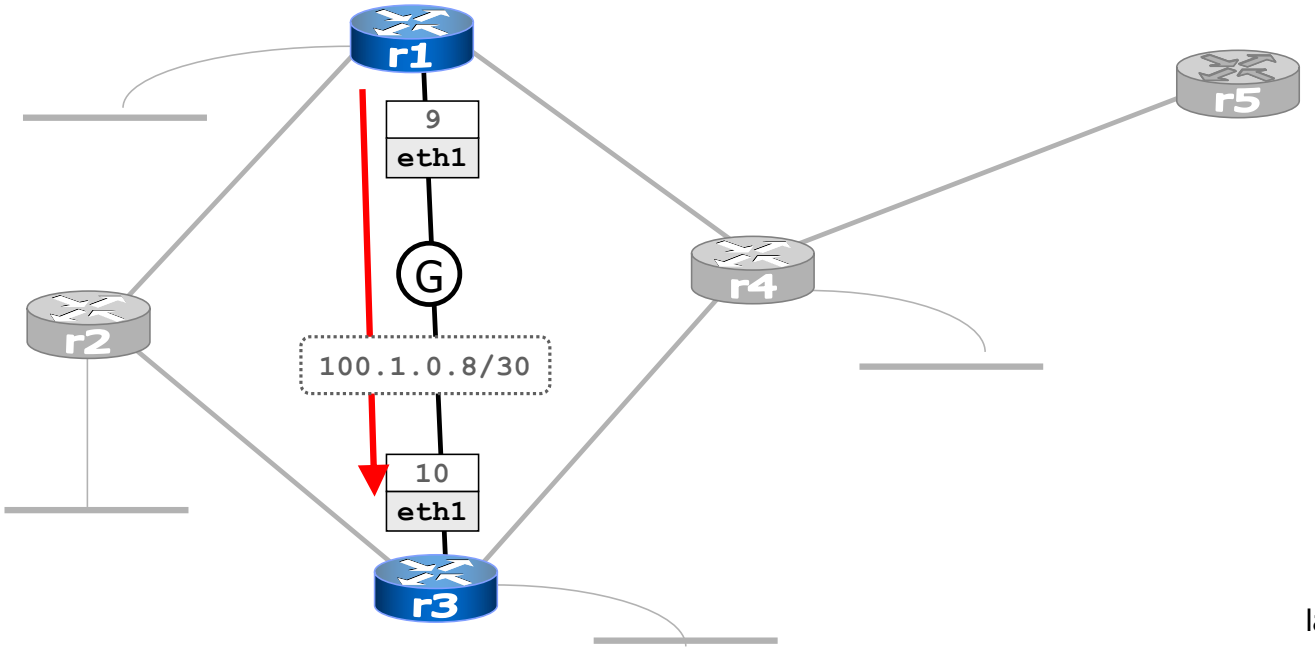
- **r5** is actually receiving echo request packets

```
root@r5:~$ tcpdump -tenni eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 1518 bytes
11:38:43.822503 arp who-has 100.2.0.2 tell 100.2.0.1
11:38:43.824221 arp reply 100.2.0.2 is-at fe:fd:64:02:00:02
11:38:43.825890 IP 100.1.0.13 > 193.204.161.1: icmp 64: echo request seq 1
11:38:43.827139 IP 100.2.0.2 > 100.1.0.13: icmp 92: net 193.204.161.1 unreachable
11:38:44.841566 IP 100.1.0.13 > 193.204.161.1: icmp 64: echo request seq 2
11:38:44.841651 IP 100.2.0.2 > 100.1.0.13: icmp 92: net 193.204.161.1 unreachable

6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

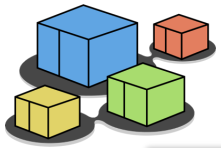


shutting down an interface

```
root@r1:~$ traceroute 100.1.0.10
traceroute to 100.1.0.10 (100.1.0.10), 64 hops max, 40 byte packets
 1  100.1.0.10 (100.1.0.10)  24 ms  1 ms  1 ms
root@r1:~$ ip link set eth1 down
```

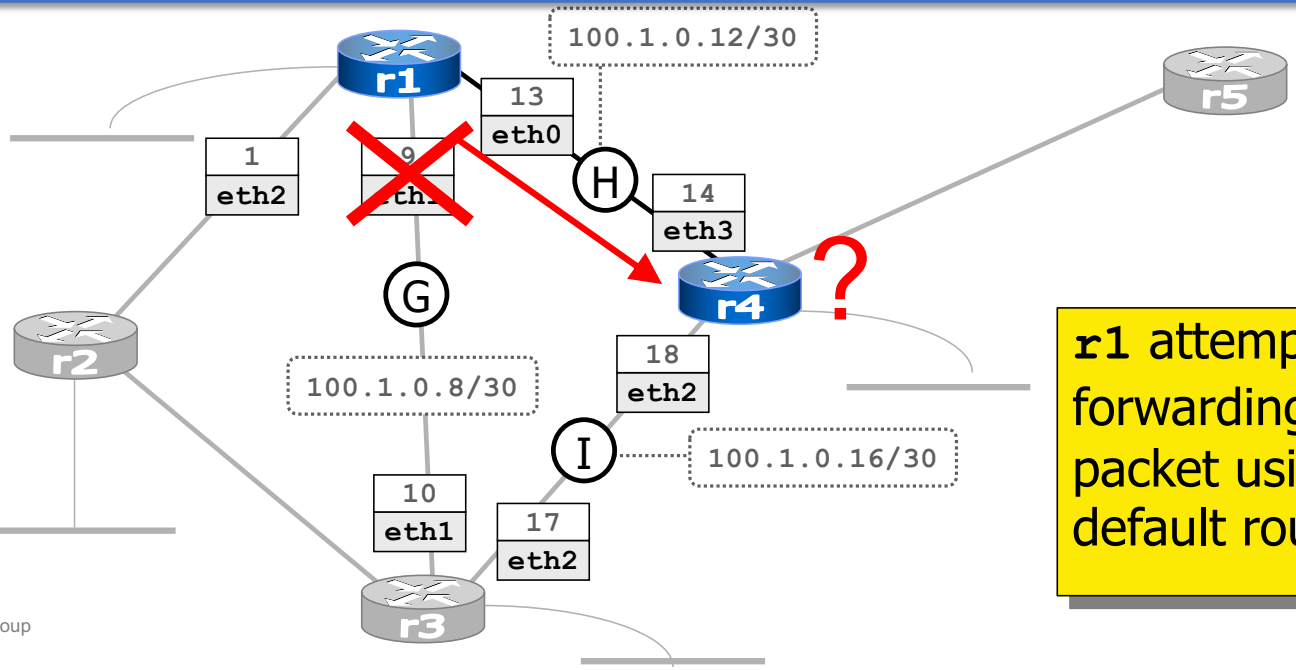


last update: Oct 2023



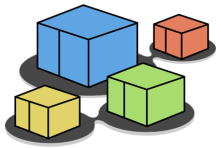
shutting down an interface

```
root@r1:~$ traceroute 100.1.0.10
traceroute to 100.1.0.10 (100.1.0.10), 64 hops max, 40 byte packets
 1  100.1.0.14 (100.1.0.10)  1 ms  1 ms  1 ms
 2  * * *
 3  * * *
```



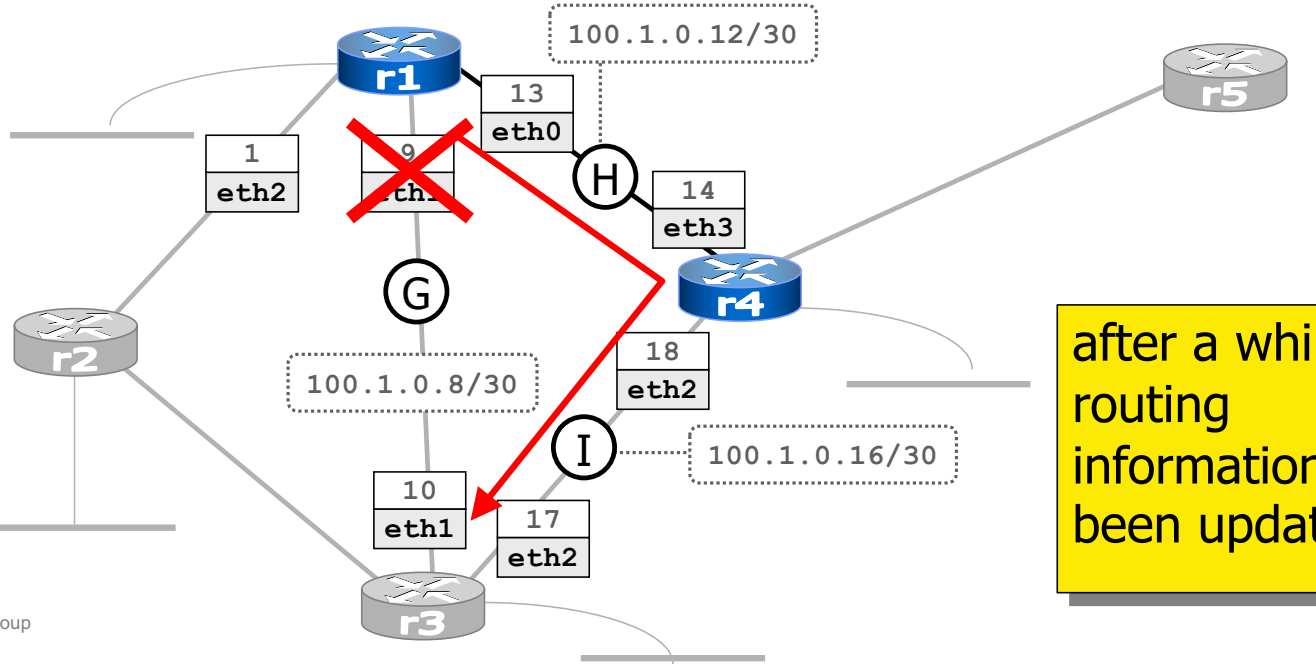
r1 attempts forwarding the packet using the default route

last update: Oct 2023



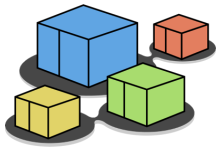
shutting down an interface

```
root@r1:~$ traceroute 100.1.0.10
traceroute to 100.1.0.10 (100.1.0.10), 64 hops max, 40 byte packets
 1  100.1.0.14 (100.1.0.14)  1 ms  1 ms  1 ms
 2  100.1.0.10 (100.1.0.10)  5 ms  2 ms  1 ms
```



after a while, routing information has been updated

last update: Oct 2023



about `redistribute connected`

- by default (i.e., without further configuration) RIP already propagates information about all directly connected subnets attached to RIP-speaking interfaces
- **`redistribute connected`** forces RIP to propagate information about all connected subnets
- the semantic of **`redistribute connected`** applies to all routing protocols
- the default behavior does not
  - some protocols (e.g., bgp) are lazier, and do not propagate anything unless explicitly told to do so