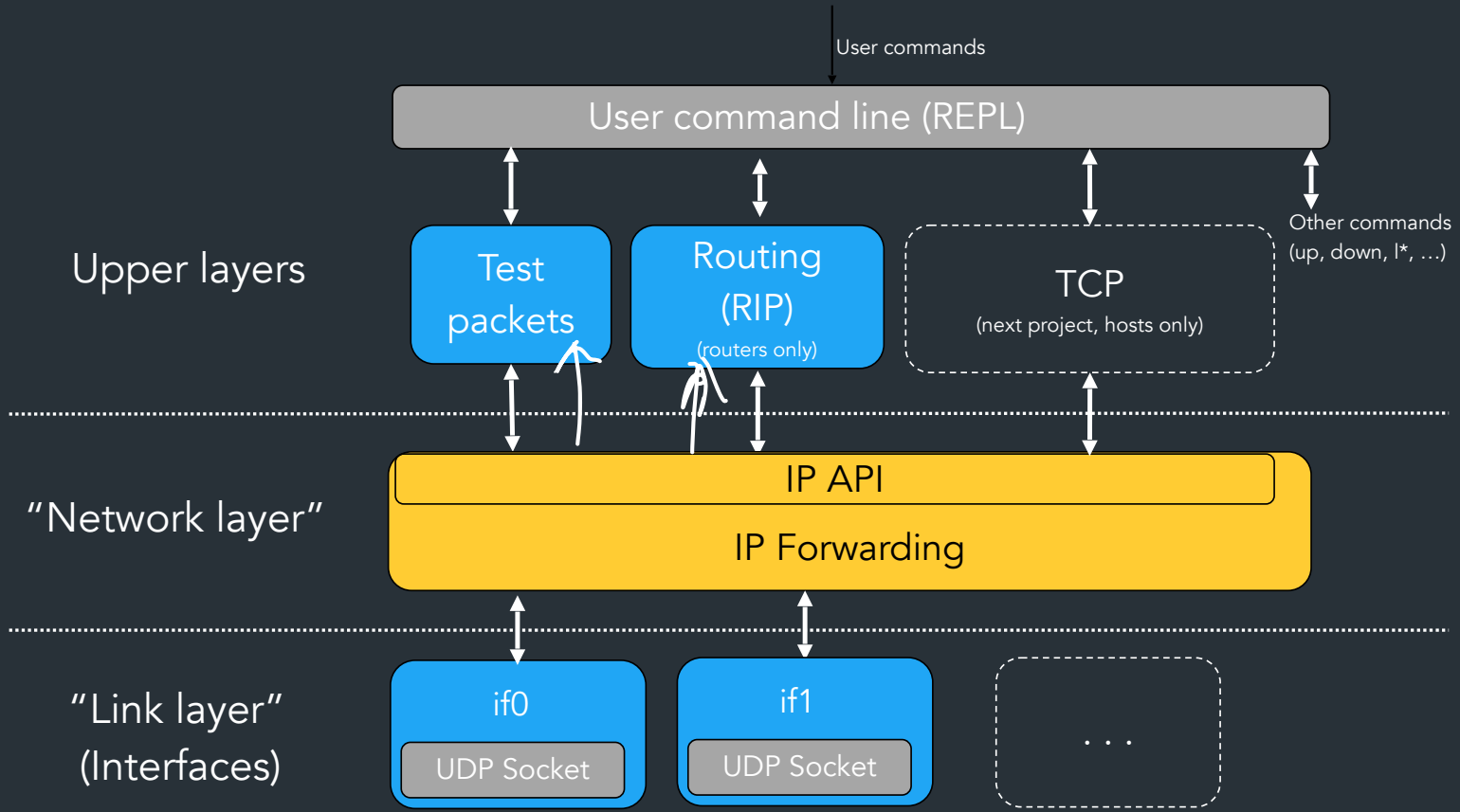
IP Project Gearup II

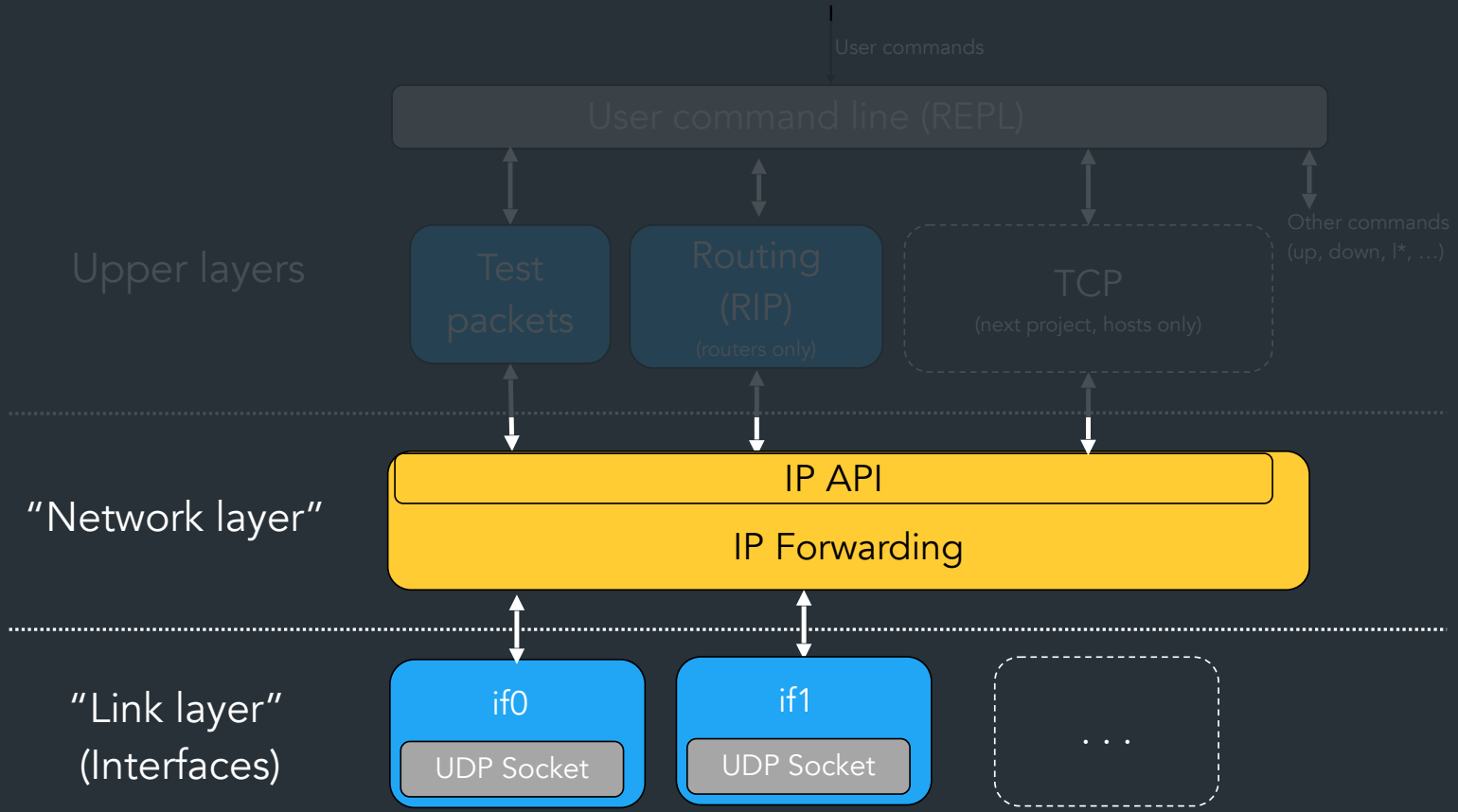
Overview

- How to think about forwarding/link-layer
- How to debug/view in wireshark
- Implementation notes
- Any questions you have

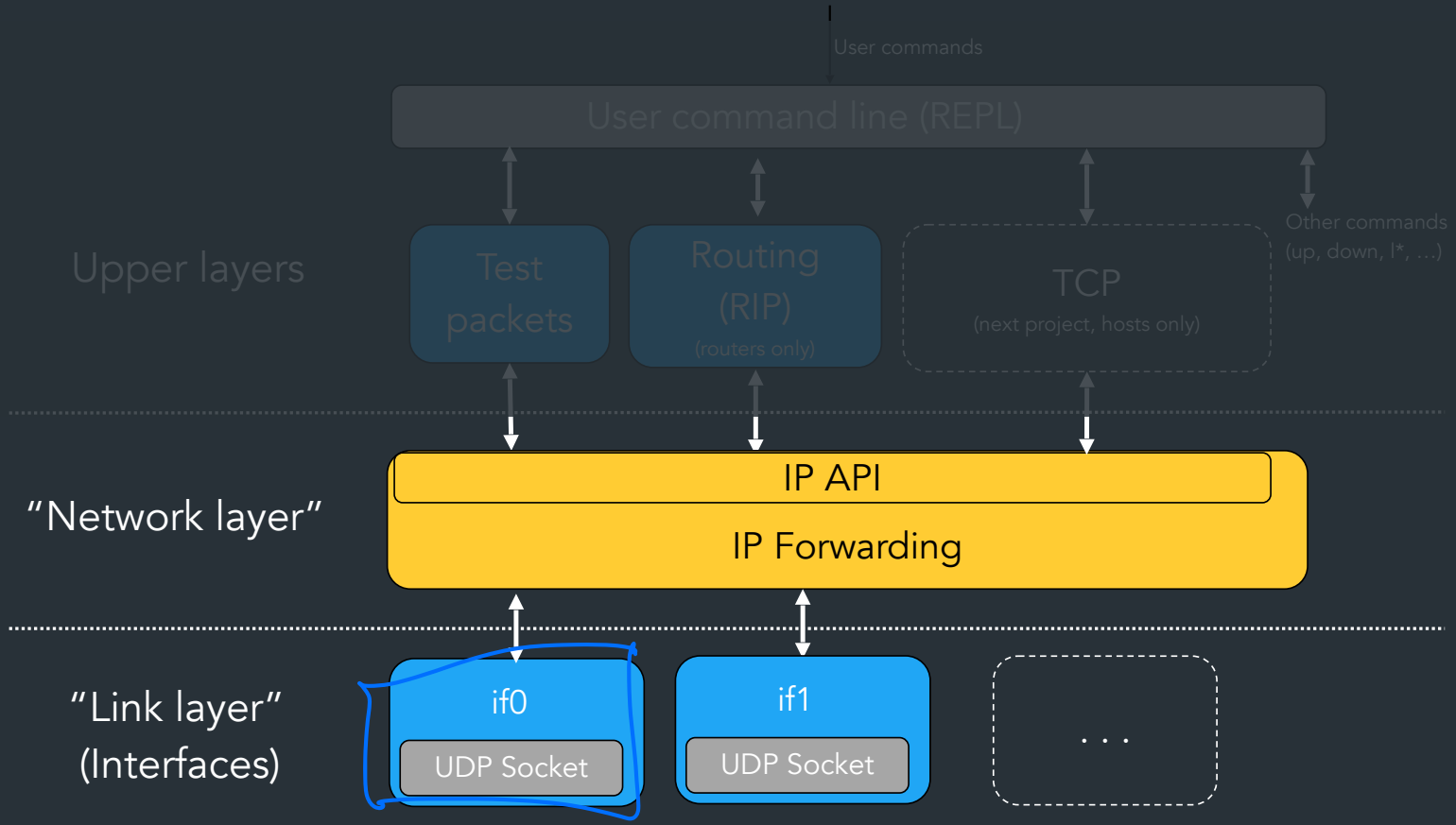
The Big Picture



What you should be focusing on first



What you should be focusing on first



How to receive packets on interfaces, send them back out

How does the link-layer work?

What does it mean to forward vs. send on an interface?

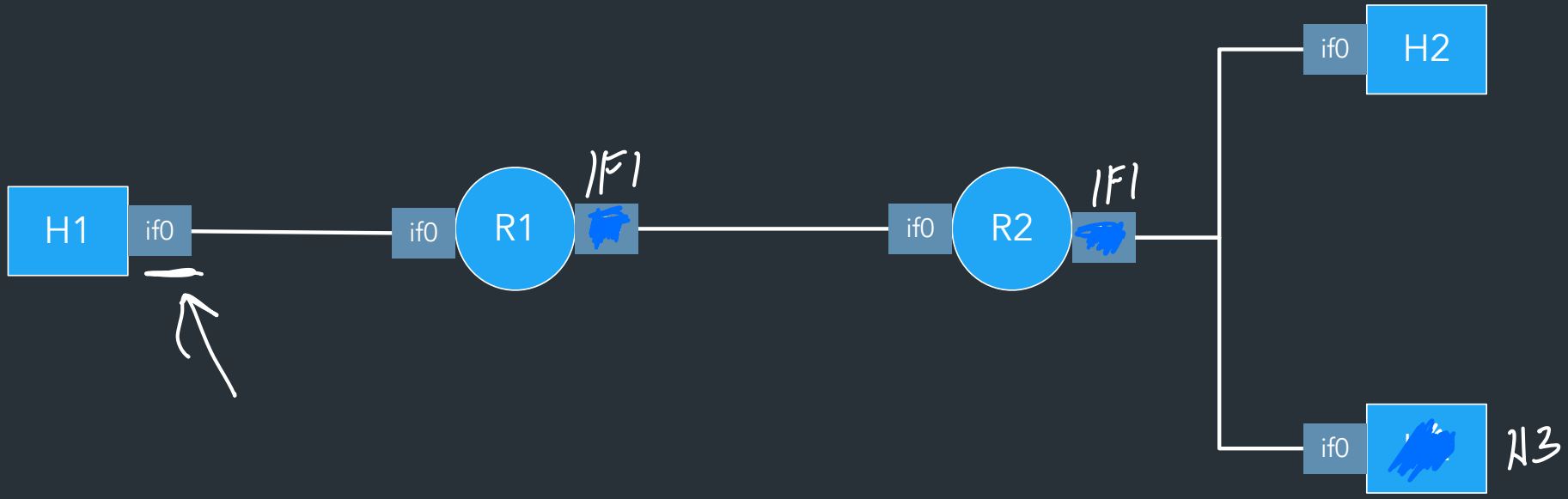
Host

```
> lr
T       Prefix   Next hop Cost
L  10.0.0.0/24  LOCAL:if0    0
S    0.0.0.0/0   10.0.0.2    0
```

Router

```
> lr
T       Prefix   Next hop Cost
R  10.2.0.0/24   10.1.0.2    1
L  10.0.0.0/24  LOCAL:if0    0
L  10.1.0.0/24  LOCAL:if1    0
```

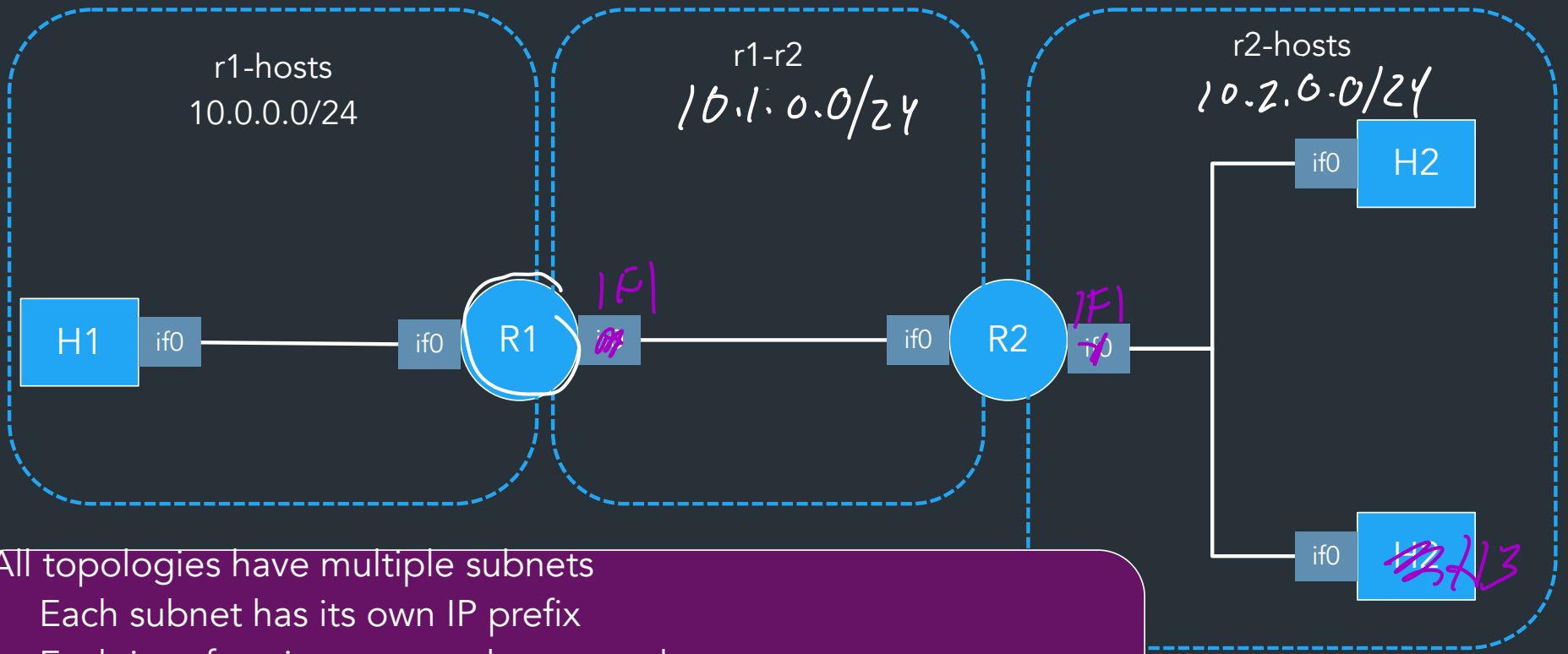
doc-example



Node ::= “host” or “router”
All nodes connect via interfaces
⇒ Hosts have exactly one interface
⇒ Routers have multiple interfaces

```
> lr
T       Prefix    Next hop Cost
R  10.2.0.0/24    10.1.0.2    1
L  10.0.0.0/24   LOCAL:if0    0
L  10.1.0.0/24   LOCAL:if1    0
```

doc-example



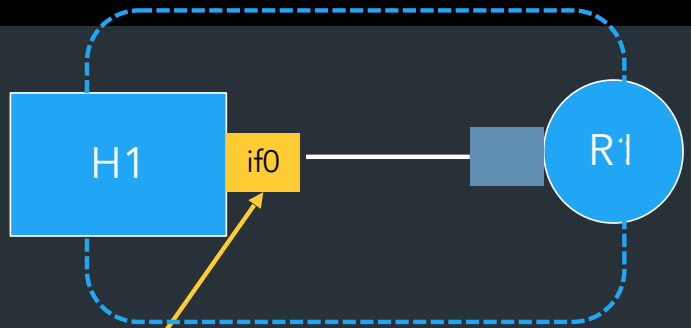
All topologies have multiple subnets

- Each subnet has its own IP prefix
- Each interface is connected to one subnet
- Nodes on the same subnet are neighbors

  => Nodes always know how to send packets to their neighbors

Interface: has a virtual IP, network, "link-layer" UDP port

```
h1.lnx
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2
```



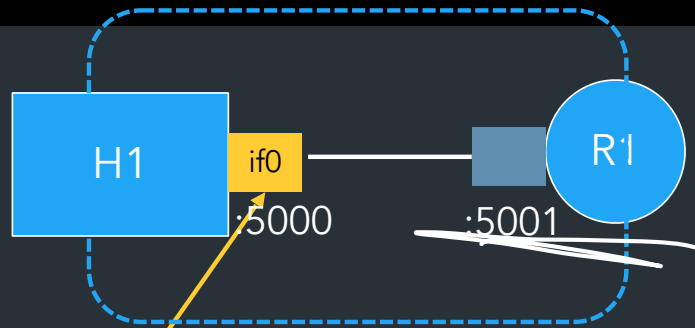
Config for if0

Virtual IP: 10.0.0.1
Network: 10.0.0.0/24
UDP: bind on 127.0.0.1:5000

"LINK LAYER"
ALL INTERFACES LISTEN ON A UDP PORT — NONE :5000

```
h1.lnx
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2
```

ONE FOR EACH OTHER NODE ON THIS SUBNET



EACH INTERFACE HAS A SET OF NEIGHBORS

CAN ALWAYS SEND DIRECTLY TO NEIGHBORS.

Config for if0

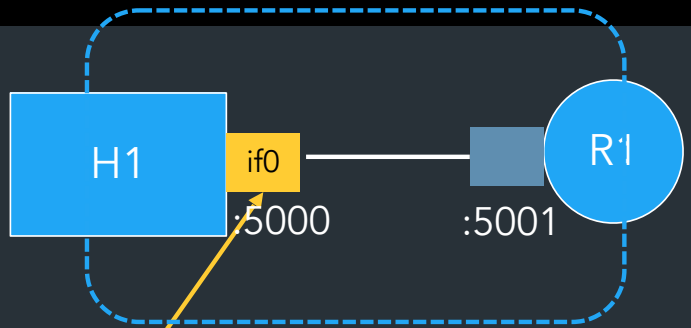
Virtual IP: 10.0.0.1

Network: 10.0.0.0/24

UDP: bind on 127.0.0.1:5000

neighbors: { 10.0.0.2 => 127.0.0.1:5001 }

```
h1.lnx
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2
```



Config for if0

Virtual IP: 10.0.0.1

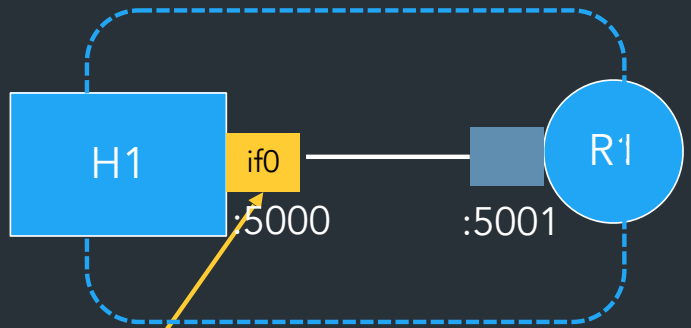
Network: 10.0.0.0/24

UDP: bind on 127.0.0.1:5000

neighbors: { 10.0.0.2 => 127.0.0.1:5001 }

Each interface has a list of neighbors: mapping of IPs to UDP ports
=> Like an ARP table, but always known ahead of time

```
h1.lnx
interface if0 10.0.0.1/24 127.0.0.1:5000 # to network r1-hosts
neighbor 10.0.0.2 at 127.0.0.1:5001 via if0 # r1
route 0.0.0.0/0 via 10.0.0.2
```



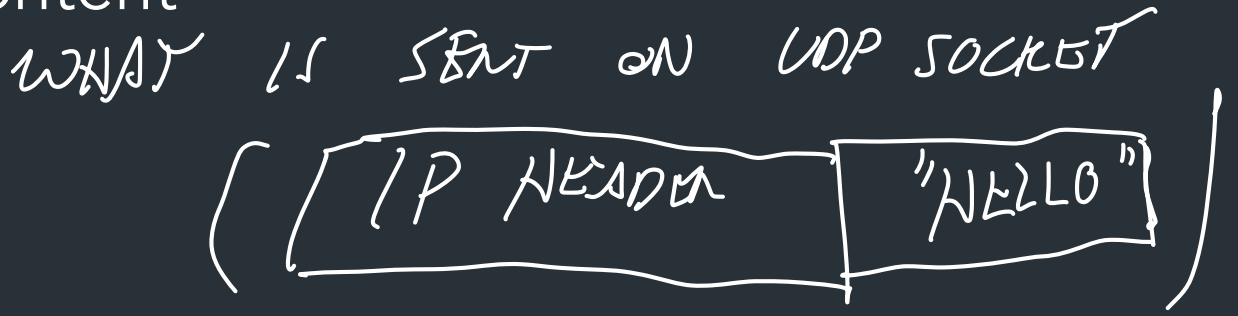
=> H1 can reach 10.0.0.2
by sending to UDP port 5001

Config for if0
Virtual IP: 10.0.0.1
Network: 10.0.0.0/24
UDP: bind on 127.0.0.1:5000
neighbors: { 10.0.0.2 => 127.0.0.1:5001 }

So if we want to send from H1 to R1,
we need to send something to UDP port 5001 => but what?

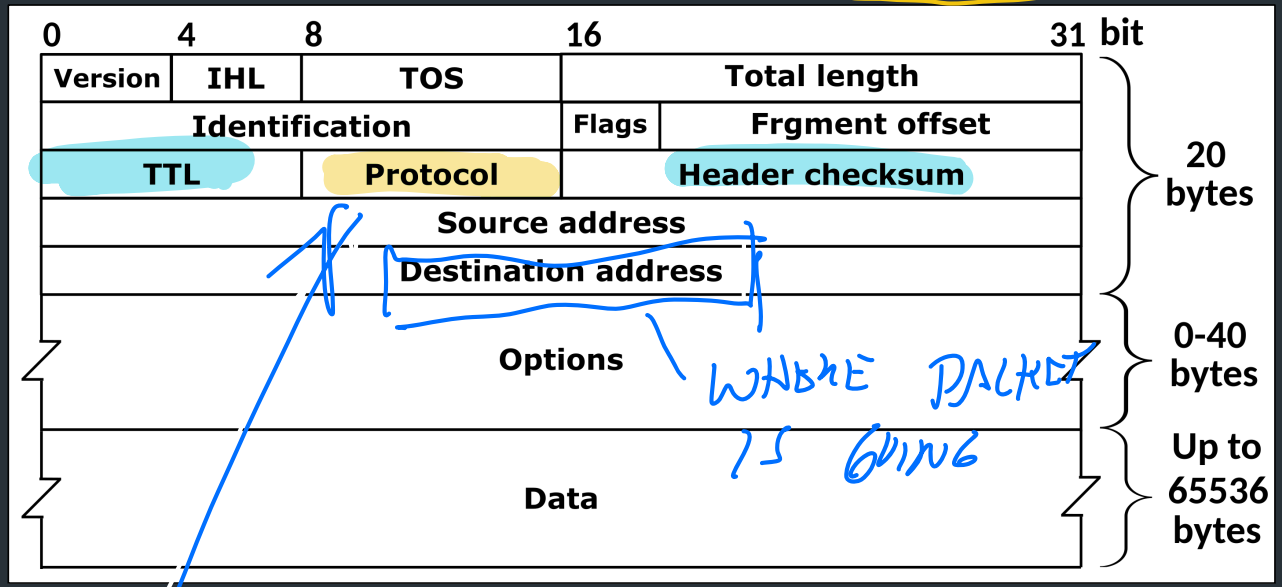
How to think about encapsulation

- Each interface: thread/goroutine/etc listening on a UDP port
- Each packet contains an IP header + whatever message content



FOR MORE INFO, SEE LECTURE 7

IP Header



WHERE PACKET IS GOING

HOW WE "UNWRAP" THE PACKET.

UDP-in-IP example

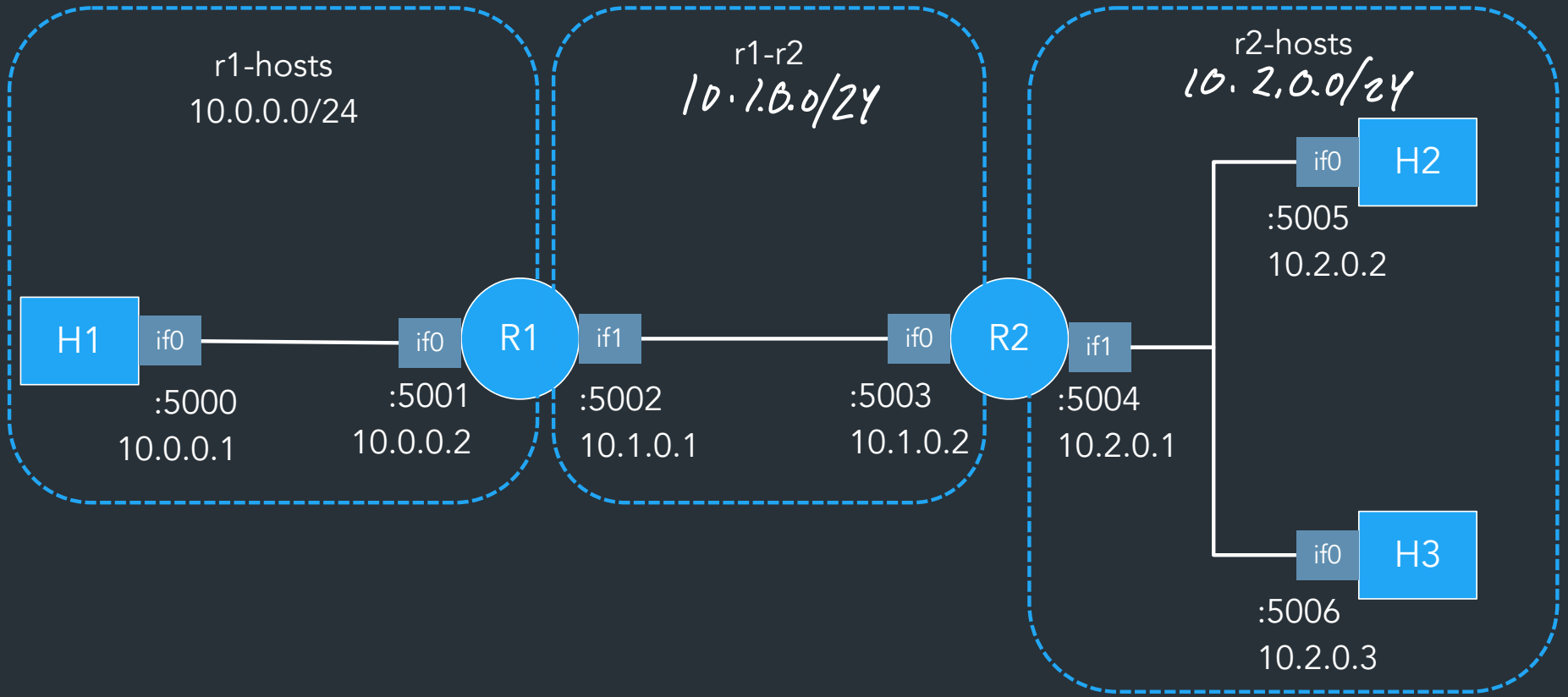
- Complete code example for building an IP header, adding it to a packet, and sending it via UDP
  - Also computes/validates checksum!

- Let's break down how this works…

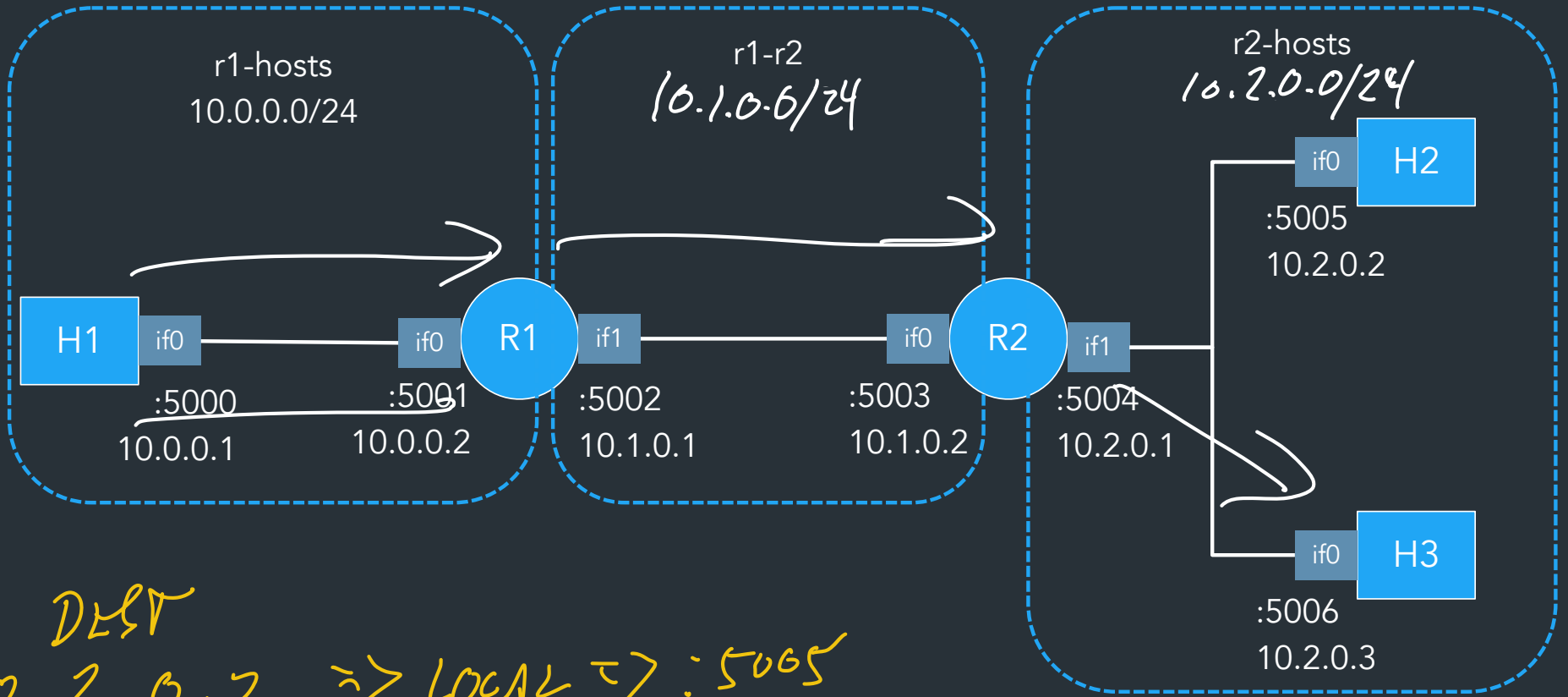
To send some data

- Build an IP header
  - Fill in all header fields as appropriate (source, dest IP, etc.)
  - Compute the checksum
- UDP Packet: IP header + data you want to send
- Send packet via socket for that interface

doc-example

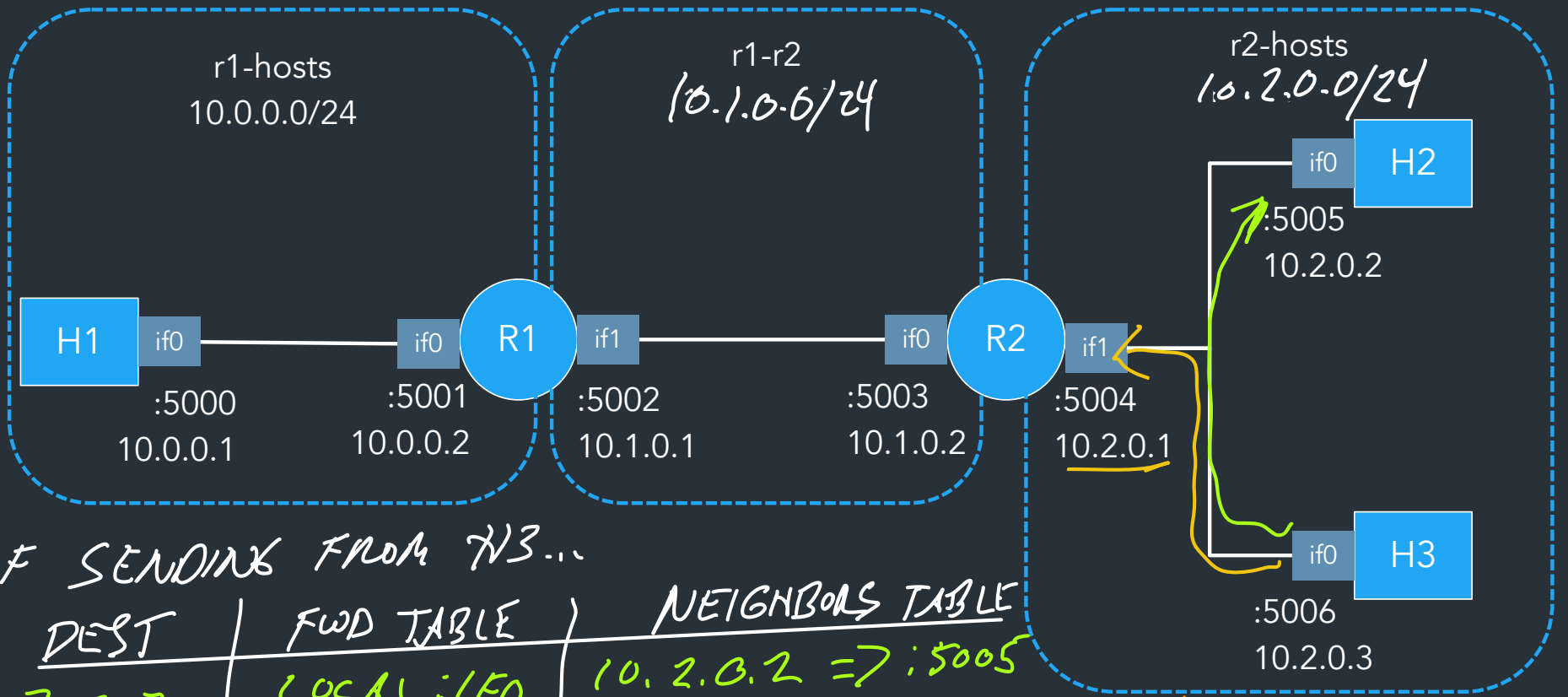

r1-hosts
10.0.0.0/24
r1-r2
10.1.0.0/24
r2-hosts
10.2.0.0/24
H1
if0
:5000
10.0.0.1
if0
:5001
10.0.0.2
R1
if1
:5002
10.1.0.1
if0
:5003
10.1.0.2
R2
if1
:5004
10.2.0.1
if0
H2
:5005
10.2.0.2
if0
H3
:5006
10.2.0.3

What would it look like to send from h1 -> h3?



DEST
10.2.0.2 => LOCAL => :5005
10.0.0.1 => NH 10.2.0.1 => :5004

What happens if h2 sends to h3?



IF SENDING FROM N3...

DEST	FWD TABLE	NEIGHBORS TABLE
10.2.0.2	LOCAL:IF0	10.2.0.2 => :5005
10.0.0.1	10.2.0.1 (DEFAULT)	10.2.0.1 => :5004

N2 IS NEIGHBOR, SO CAN SEND DIRECTLY TO HOST!

Receiving packets

- Receive packet from link layer
- Parse IP header and determine if packet is valid
  - TTL, checksum, etc…
- Check destination IP => NEXT PAGE
  - If destination is your IP: deliver locally
  - If not, consult forwarding table

FORWARDING STEPS:

CONSIDER PACKET WITH DESTINATION IP D

If destination IP D matches one of this node's assigned IPs
=> Packet is for this node => Send "up" (more on this later)

Otherwise, check forwarding table to look for a match
(If multiple matches, take the most specific prefix (lecture 7, 9)

If the result is a local route (ie, maps to some ifX)

=> Look up UDP port for D in neighbors table for ifX
Send packet to this port

(SEND DIRECTLY
EG. $N_2 \rightarrow N_3$)

If the result is not a local route (ie, has next hop IP G)
=> Need to send packet to G instead:
Look up G in forwarding table
=> maps to some local route on some interface ifY
Look up UDP port for G in ifY's neighbor's table
Send packet to this port

(SEND VIA GATEWAY
EG. $N_2 \rightarrow R_2 \rightarrow \ldots$)

CHOOSING NEXT HOP DESTINATION!

How to send "up"?

OUR NODES CAN DO 20 DIFFERENT THINGS W/ PACKETS:

PROTOCOL NUM

HOSTS:
- TEST PACKETS (0)
- TCP (6)

ROUTERS
- TEST PACKETS (0)
- RIP PACKETS (200)

⇒ LOOK UP A HANDLER FOR THE PACKET BASED ON PROTOCOL NUM

REGISTER_HANDLER(NUM, SOMEFUNC)

DO THIS AT STARTUP — TELL IP STACK TO CALL SOMEFUNC WHEN RECEIVING A PACKET W/ THIS PROTOCOL.

How to table lookup?

Dest IP == 10.0.0.5, where to send packet?

```
r1:

> lr
T       Prefix      Next hop  Cost
L   10.0.0.0/24   LOCAL:if0     0
L   10.1.0.0/24   LOCAL:if1     0
R   10.2.0.0/24    10.1.0.2     1
```

```
h1:
 > lr
T       Prefix      Next hop  Cost
L   10.0.0.0/24   LOCAL:if0     0
S    0.0.0.0/0     10.0.0.2     0
```

10.0.0.1

10.2.5.7

- You can decide how to store the table
- Need to find the most specific matching prefix
- Use built-in datatypes to help you!
  Go: prefix.Contains() (netip.Prefix)

You do NOT need to be particularly efficient about this step!

Implementation: key resources

- Use an external library for parsing IP header (don't do this yourself)
  - For Go/C, see UDP-in-IP examples
  - Rust: etherparse library
- We provide parsers for the lnx files—don't bother writing these yourself
- You're welcome to use third-party libraries, so long as they don't trivialize the assignment (ask if you're concerned)
  - Data structures, argument parsing, are fine

IP types and go

Go has two IP types, net.IP and (newer) netip.Addr

- netip.Addr and netip.Prefix the one you want

⇒ These libraries have useful helper functions, use them!

Testing your IP

vnet_run: Run all nodes in a network automatically

- Can run on your node, or the reference
- Uses tmux: see getting started guide for details
- Can run some nodes as reference, some nodes as yours

⇒ See getting started guide for details, more soon!

Viewing packets in wireshark

Sample Topologies

Some example networks you can test with…

See “sample networks” page for more info, including what kinds of things you can test with each network

Roadmap

Once you can send across one router, start thinking about RIP

3. Make sure you can share routes and update the forwarding table
   - Eg. linear-r2h2: H1 -> R1 -> R2 -> H2

4. Try disabling/enabling links, make routes expire

5. Loop network: finding best path, updating routes as topology changes