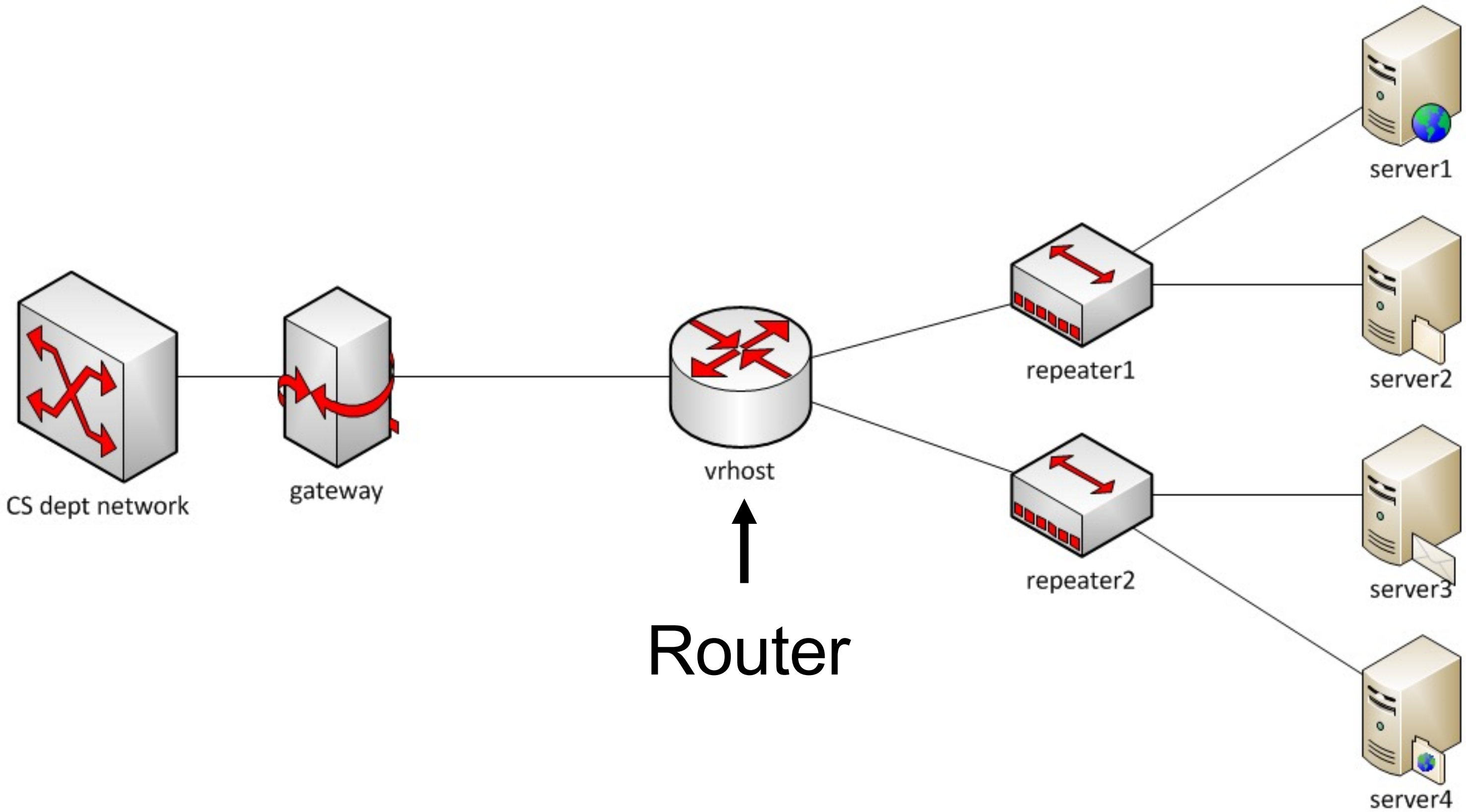
CSC 525:
Computer Networks

build a simple router



- Implement basic packet forwarding functionality at the router
  - ARP, IP, and ICMP
- Will be tested with real traffic: ping and http download
- Ignore the repeaters or treat them as wires.

what you'll start with

- Virtual network topology
  - One per student. Only use your own topology number!
  - Only accessible from within the CS department.
- Stub code
  - It handles the backend setup, and send/receive pkts.
  - Its header files provide some basic data structures
    - Protocol header, interface list, etc.
  - Receive packets: *sr_handlepacket()*
  - Send packets: *sr_send_packet()*
- Your job is to implement the packet processing logic within sr_handlepacket( ).
  - Feel free to add new functions and new files, as long as "make" works.

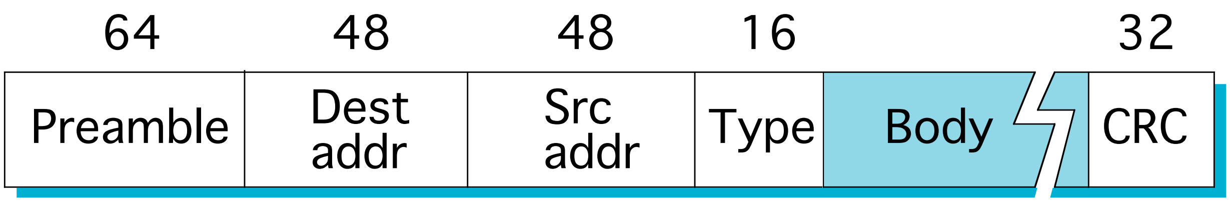
What does a packet look like?

Ethernet	IP or ARP	TCP or ICMP	Payload

A byte pointer to the beginning of the packet

- How to parse the packet header?
  - Cast to a data structure of the protocol header, use pointer to access specific header fields.
  - E.g., (struct sr_ethernet_hdr *) pkt; where pkt is (uint8_t *) that points to the beginning of a packet.
  - Header structures are in the header files of the stub code
- How to know which protocol header to use?
  - Each protocol header contains a field that identifies the next protocol header.

Ethernet



- Ignore preamble and CRC; they're handled by hardware, not passed to router software.
- Addresses
  - 48-bit unicast address assigned to each interface
  - example: **`8:0:2b:e4:b1:2`**
  - Broadcast address: all **`1`**s, i.e., ff:ff:ff:ff:ff:ff
- Type: what next protocol header is (e.g., IP or ARP)
- Body: min 46 bytes, max 1500 bytes

Let's look at a real packet

Start the router with packet logging

```
lectura 171 > ./sr -t 363 -l log
```

ping one of your servers, then stop both ping and router.

tcpdump to display the packets in hexadecimal

```
lectura 174 > tcpdump -vvv -xx -r log
reading from file log, link-type EN10MB (Ethernet)
12:33:33.991796 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.29.15.232
tell 172.29.15.233, length 28
        0x0000:  ffff ffff ffff 5ebf 9475 c872 0806 0001
        0x0010:  0800 0604 0001 5ebf 9475 c872 ac1d 0fe9
        0x0020:  0000 0000 0000 ac1d 0fe8
```

destination source ethertype

Accessing header fields

```
struct sr_ethernet_hdr
{
#ifndef ETHER_ADDR_LEN
#define ETHER_ADDR_LEN 6
#endif
    uint8_t  ether_dhost[ETHER_ADDR_LEN];
    uint8_t  ether_shost[ETHER_ADDR_LEN];
    uint16_t ether_type;
} __attribute__ ((packed)) ;
```

```
uint8_t *p;
struct sr_ethernet_hdr *eth;
struct ip *iphdr
void sr_handlepacket(sr, p, len, iface)
{
   eth = (struct sr_ethernet_hdr *) p;
   eth->ether_type is IP;
   iphdr = (struct ip*)(p + sizeof(struct sr_ethernet_hdr);
   ...
}
```

Address Resolution Protocol

- ARP: Given an IP address, finds the corresponding Ethernet address
  - broadcast the request, i.e., who knows the ethernet address of IP 1.2.3.4?
  - the target machine responds with its MAC address.
  - Cache the reply in a table to avoid asking the same question later.
  - Cache times out after some time.

ARP Packet Format

0	8	16	31
Hardware type = 1		ProtocolType = 0x0800	
HLen = 6	PLen = 4	Operation	
SourceHardwareAddr (bytes 0–3)			
SourceHardwareAddr (bytes 4–5)		SourceProtocolAddr (bytes 0–1)	
SourceProtocolAddr (bytes 2–3)		TargetHardwareAddr (bytes 0–1)	
TargetHardwareAddr (bytes 2–5)			
TargetProtocolAddr (bytes 0–3)			

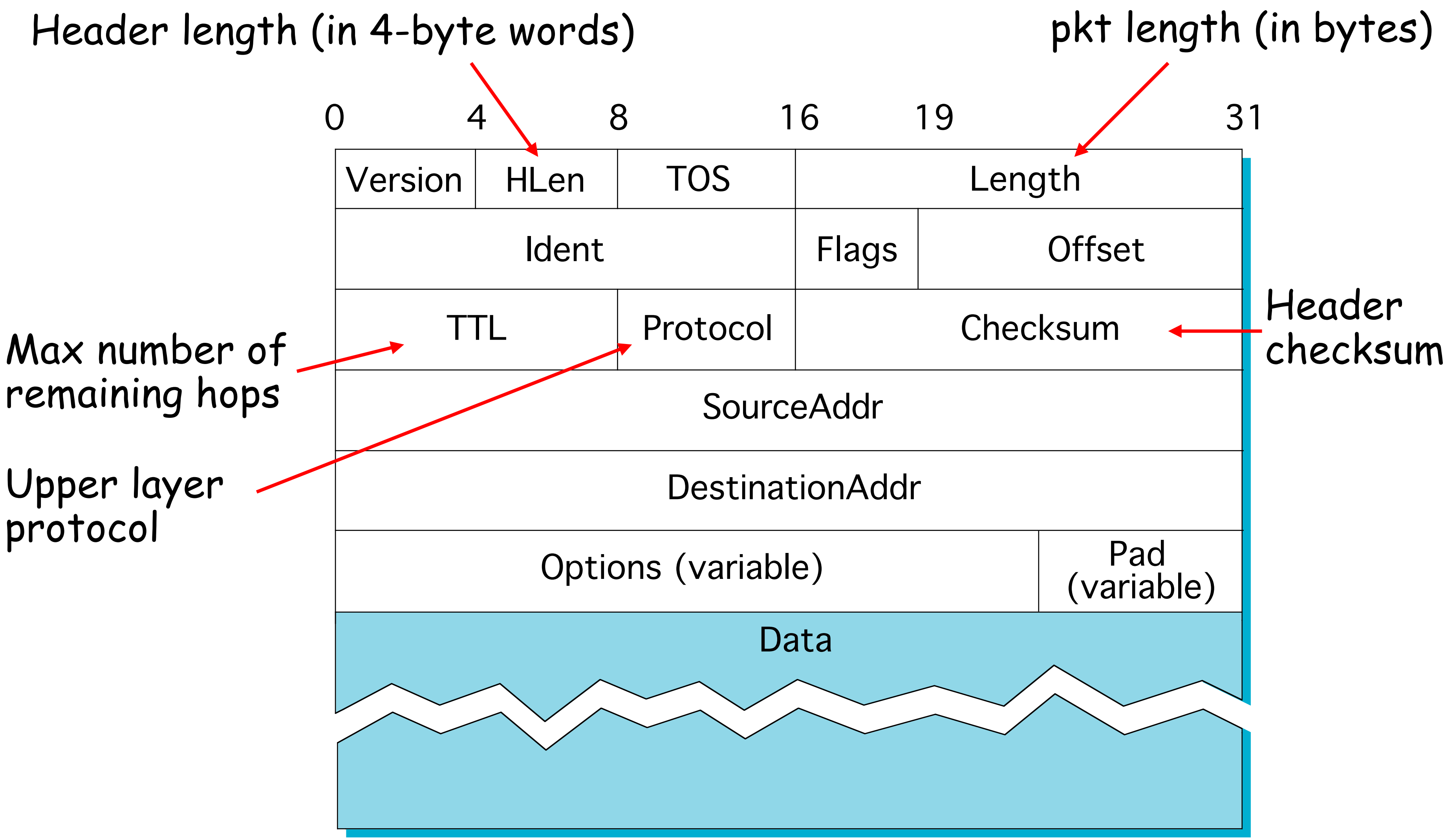
- In our context, Hardware refers to Ethernet, Protocol refers to IP.
- HardwareType: type of physical network (e.g., Ethernet)
- ProtocolType: type of higher layer protocol (e.g., IP)
- HLEN & PLEN: length of physical and protocol addresses in bytes
- Operation: request or response
- Source/Target Physical/Protocol addresses

ARP Details

- The request is broadcasted onto the local link only.
  - Routers don't forward ARP packets.
- The reply is usually unicast to the requester, but can also be broadcasted.
- A host can learn IP/MAC mapping from incoming packets without generating separate ARP requests.
- An entry in ARP table (cache) will time out after a while.

- If you answer the request with a correct reply, you will receive an IP packet next.
  - Otherwise you'll keep getting the same ARP request.

IP Packet Header



The basic header (without options) is 20 bytes long

Basic IP packet processing

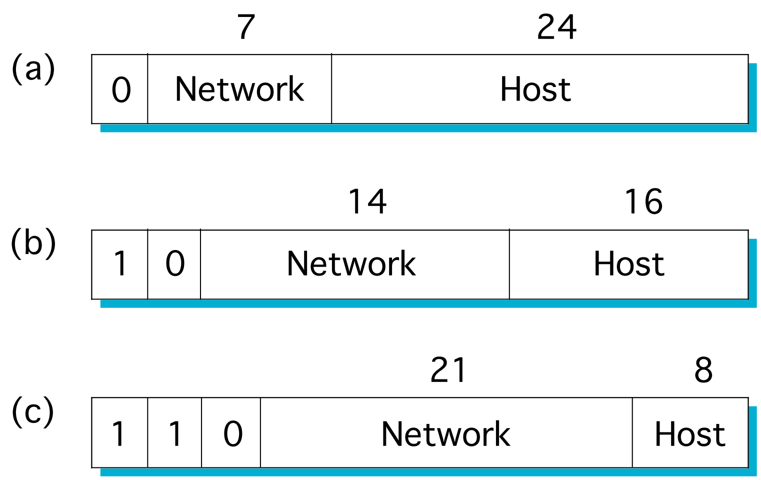
- Is the destination myself?
  - Yes. Keep processing if it's an ICMP ping, otherwise drop it.
  - No. forward the packet as follows.
- Decrement TTL by 1,
  - if the result TTL == 0, discard the packet
  - Otherwise update the checksum
    - Set checksum field to be 0, recalculate the checksum over the header, and record the result in the field.
- Use the destination address to look up the routing table to find a matching entry, which tells the next-hop for this packet.
- Send the packet to the next hop.

Network and Prefix

- In order to scale, IP routing tables record one entry per *destination network*.
  - all nodes in same network share the same IP address prefix.
- A new node gets an IP address from the network operator to ensure the address has the network's prefix.
- When a node moves from one network to another, it has to change its IP address.

How many bits in the prefix?

- Original IP design: class-based address



1.0.0.0 to
127.255.255.255

128.0.0.0 to
191.255.255.255

192.0.0.0 to
223.255.255.255

The good: given an IP, we know its prefix.

The bad: class-B addresses are in great demand. Lots of unused addresses in larger network (i.e., shorter prefixes)

Classless Addressing

- Today we use the so-called *classless addressing*.
  - The network part or the prefix can have any number of bits, which must be explicitly specified by either a network mask or prefix length.
  - e.g., the old class B address has a prefix length /16, or equivalently, a network mask of 255.255.0.0
  - Address & mask => prefix
  - 1.2.3.4/16 or 1.2.3.4 with mask 255.255.0.0

Take a look at the interface

```
lectura 136 > ifconfig
eth0      Link encap:Ethernet  HWaddr 00:E0:81:31:73:50
          inet addr:192.12.69.186  Bcast:192.12.69.255  Mask:255.255.255.0
```

- Mask indicates how long the network part is
  - addr & mask = network
- Broadcast address has all ones in the host part (non prefix).

Routing Table Lookup

dest	nexthop (gw)	mask	interface
`172.24.74.64`	`0.0.0.0`	`255.255.255.248`	`eth1`
`172.24.74.80`	`0.0.0.0`	`255.255.255.248`	`eth2`
`0.0.0.0`	`172.24.74.17`	`0.0.0.0`	`eth0`

- Lookup match if (packet's dest & mask) == (table's dest & mask)
- Longest prefix match: if there're multiple matches in the table, pick the one that has the longest prefix.
- 0.0.0.0 as the dest means that this is the default route. It matches every address, but has the shortest prefix. Used only if no other match.
- Nexthop and interface is for where to forward the packet.
- 0.0.0.0 as the nexthop means that the nexthop IP is the same as the destination IP of the packet.

How to send to next-hop?

- After looking up the routing table, we know the nexthop's IP address and the interface.
- Now need to send the packet to the nexthop via the interface.
- But how to target the packet to the nexthop?
  - Cannot modify dest address in IP header since it has to remain the same for final destination.

- Use next-hop's MAC address as the destination in Ethernet header!
  - ***IP destination remain the same the entire journey, but Ethernet destination address changes at every hop.***

- But, how to get next-hop's MAC address?
  - We've already covered it.

Put it together

- IP packet processing
- IP packet forwarding
  - Routing table lookup to find out nexthop
  - ARP to find out MAC address of nexthop
    - Maintain ARP cache
  - Send packet to nexthop using its MAC address.

Internet Control Message Protocol

- ICMP: used by hosts & routers for network-level error reporting and diagnosis
  - unreachable host, network, port, protocol, echo request/reply, and many more.
- ICMP msgs are carried in IP packets
- ICMP message format

<table>
<tr><td colspan="3">IP header</td></tr>
<tr><td>type</td><td>code</td><td>checksum</td></tr>
<tr><td colspan="3">unused (or used by certain ICMP types)</td></tr>
<tr><td colspan="3">IP header and first 64bits of data
Or
data (according to ICMP types)</td></tr>
</table>

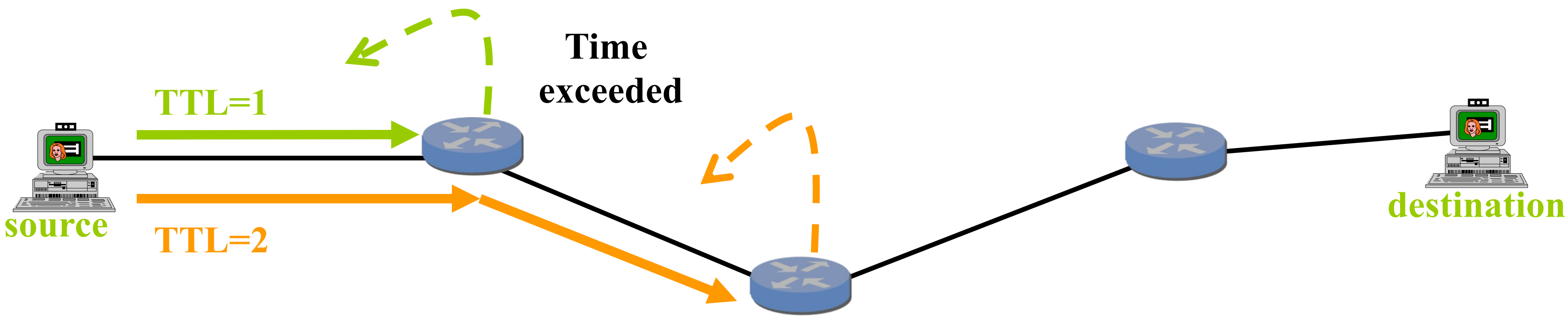
Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

ping

- If received an ICMP echo request whose IP destination is the router itself, reply with an ICMP echo reply.
- Don't need to do anything to echo requests destined to other hosts, just normal IP packet forwarding.

Traceroute

- Not required for this project
- Measuring the forwarding path
- Time-To-Live field in IP packet header
  - Source sends a packet with a TTL of *n*
  - Each router along the path decrements the TTL
  - "TTL exceeded" sent when TTL reaches *0*
- Traceroute exploits this behavior by sending a sequence of packets with increasing initial TTL.



Send packets with TTL=1, 2, 3, … and record source of "TTL exceeded" mesg

Traceroute Example

```
bobcat2 117 > traceroute www.google.com
 1  * * *
 2  tcsn-dsl-gw04-196.tcsn.qwest.net (168.103.240.196) 52.629 ms
 3  * tcsn-agw1.inet.qwest.net (168.103.240.93)  52.172 ms
 4  tus-core-01.inet.qwest.net (205.171.149.33)  51.762 ms
 5  svl-core-01.inet.qwest.net (67.14.12.6)  74.450 ms
 6  pax-edge-01.inet.qwest.net (205.171.214.30)  76.415 ms
 7  * * *
 8  209.85.130.6 (209.85.130.6)  76.325 ms
 9  66.249.94.226 (66.249.94.226)  77.372 ms
10  216.239.49.66 (216.239.49.66)  76.611 ms
11  mc-in-f99.google.com (66.102.7.99)  76.144 ms
```

The Implementation

- ARP implementation
  - Be able to respond to ARP request, send ARP request when needed, and maintain ARP cache.
- IP forwarding: given an incoming packet
  - Decrement TTL and update checksum.
  - Lookup the routing table to find the nexthop's IP.
  - Invoke ARP to find out the nexthop's MAC address.
  - Send the packet to the nexthop.
- ICMP
  - Send echo reply in response to echo request.

Four steps

- Test the stub code
- Implement ARP
- Implement IP forwarding
- Implement ICMP ping

- Test the code along the way; don't test a big chunk of code all at once.