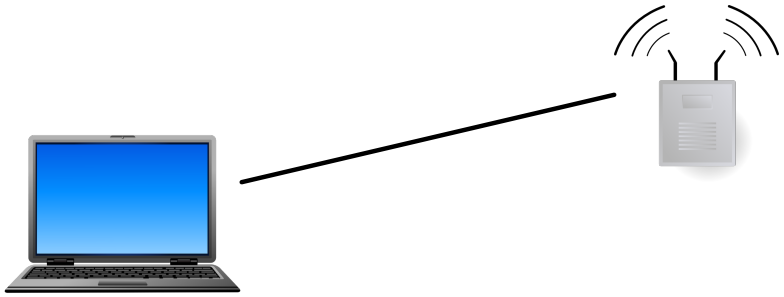
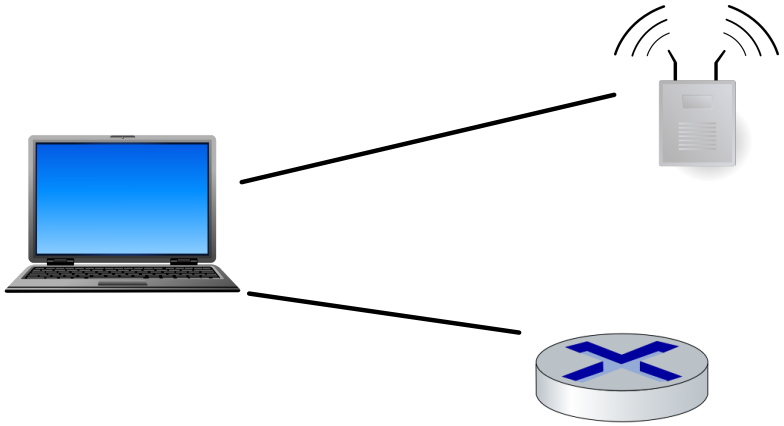
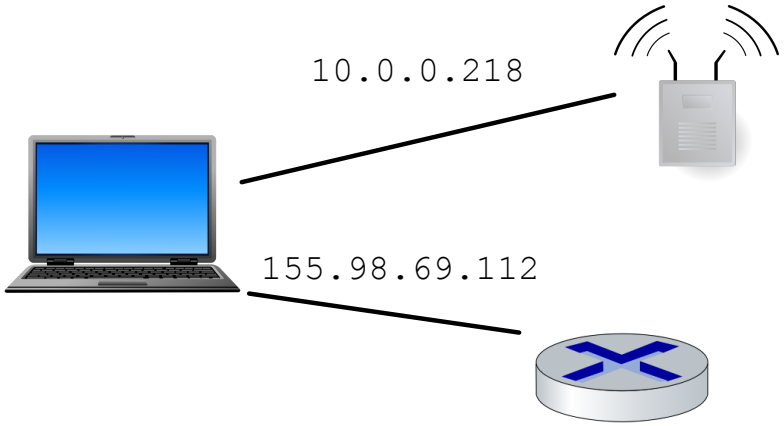
One Machine, Many Networks



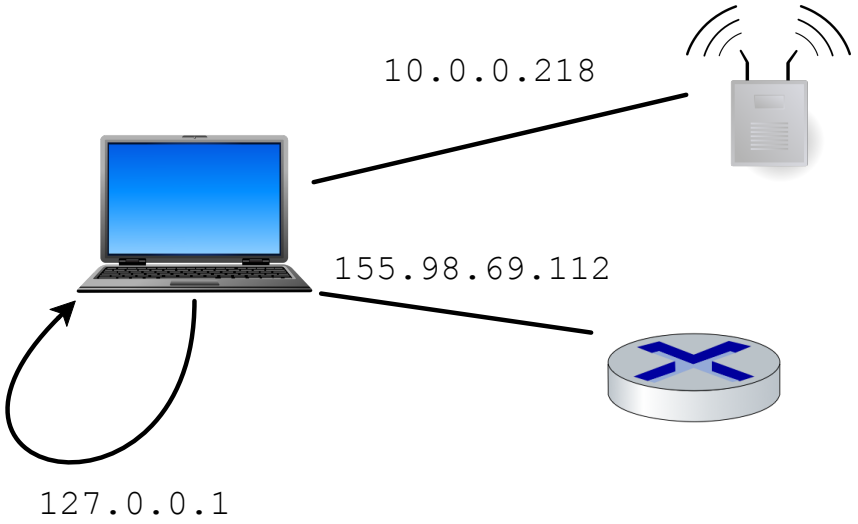
One Machine, Many Networks



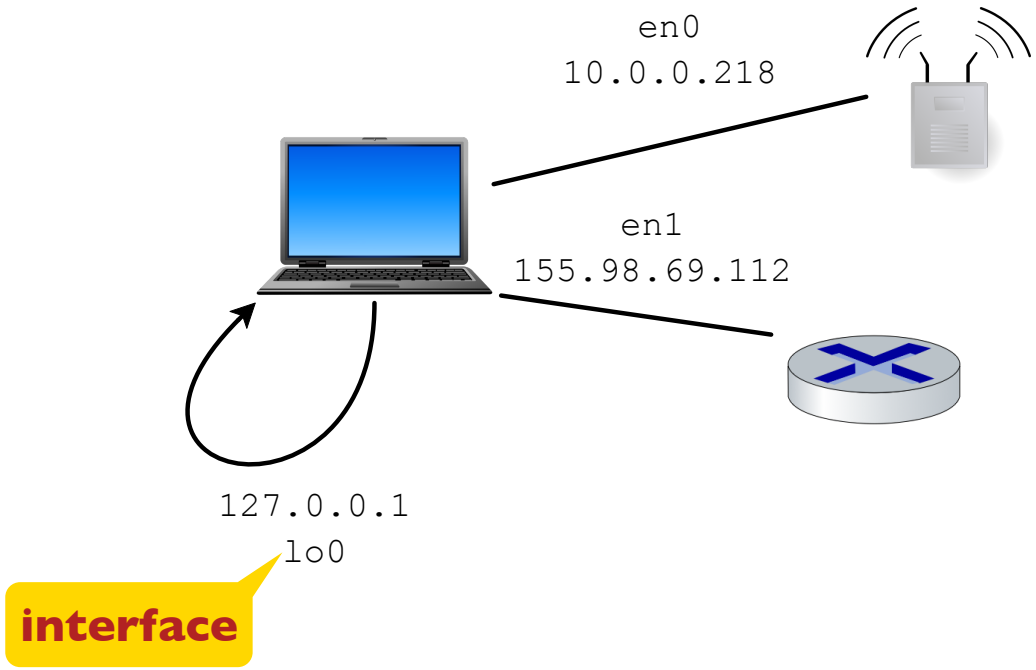
One Machine, Many Networks



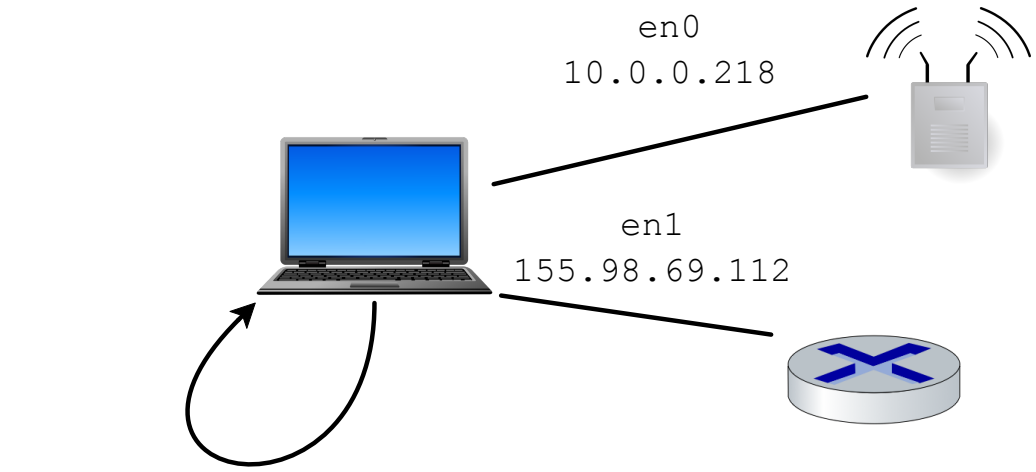
One Machine, Many Networks



One Machine, Many Networks



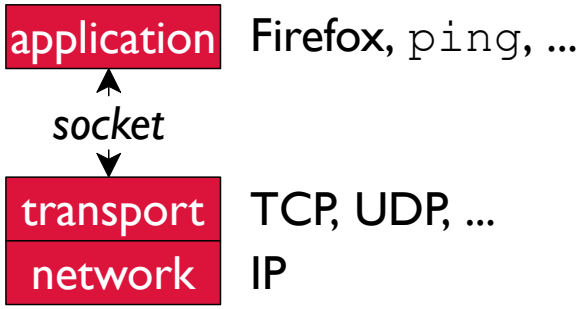
One Machine, Many Networks



View with `ifconfig`

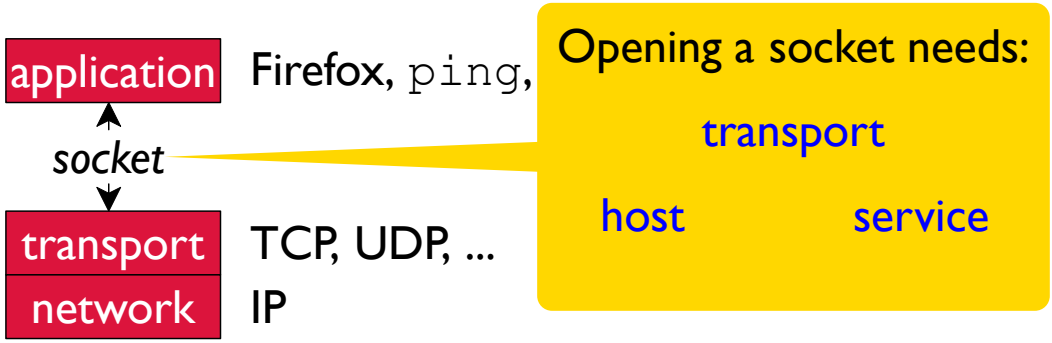
RECAP

Network Layers



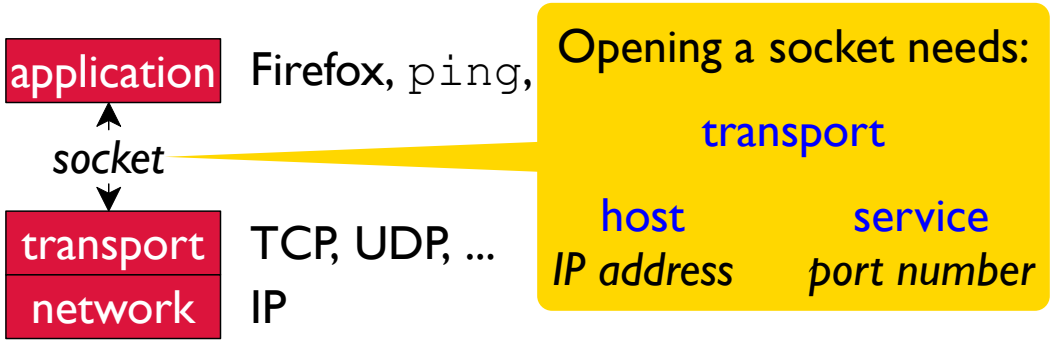
RECAP

Network Layers



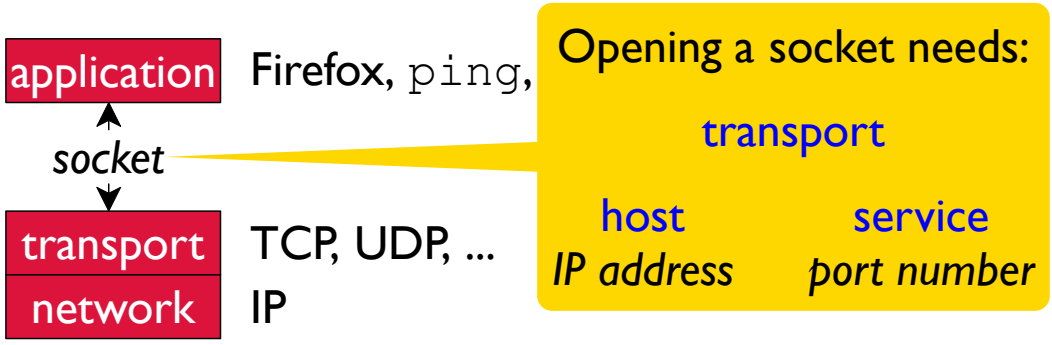
RECAP

Network Layers



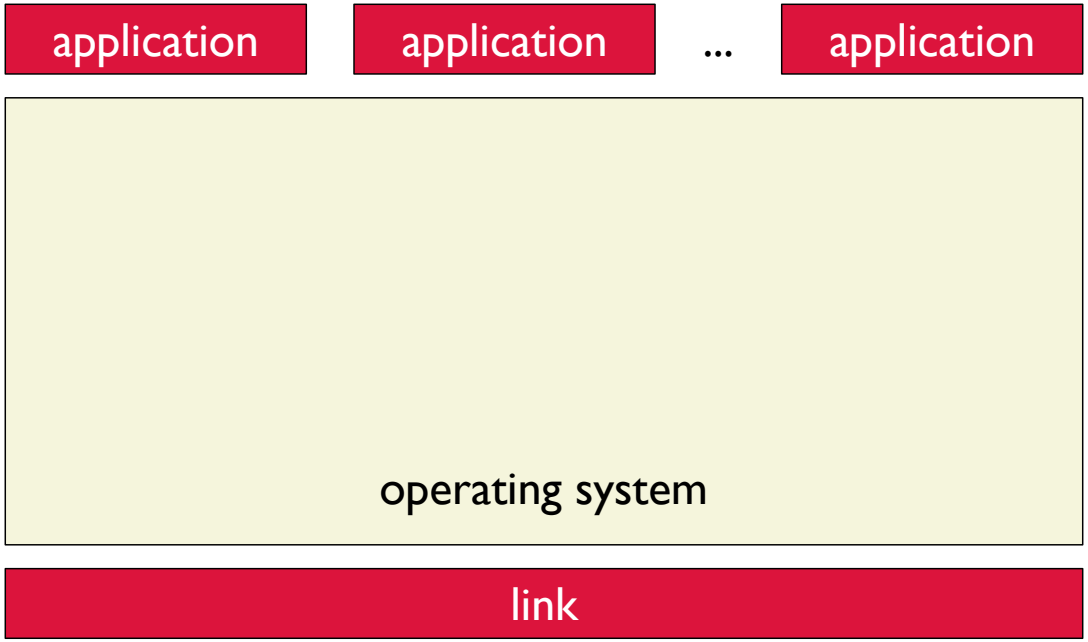
RECAP

Network Layers

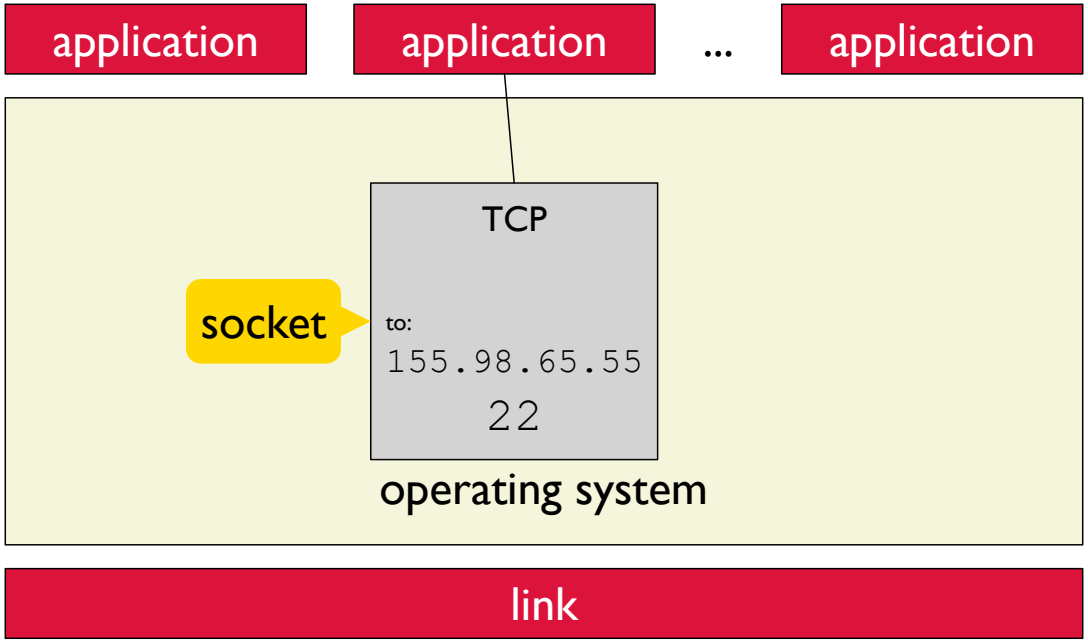


Web server	TCP	`www.cs.utah.edu`	`80`
Modern web server	TCP	`www.cs.utah.edu`	`443`
Mail receiver	TCP	`smtp.cs.utah.edu`	`25`
Name resolver	UDP	`8.8.8.8`	`53`

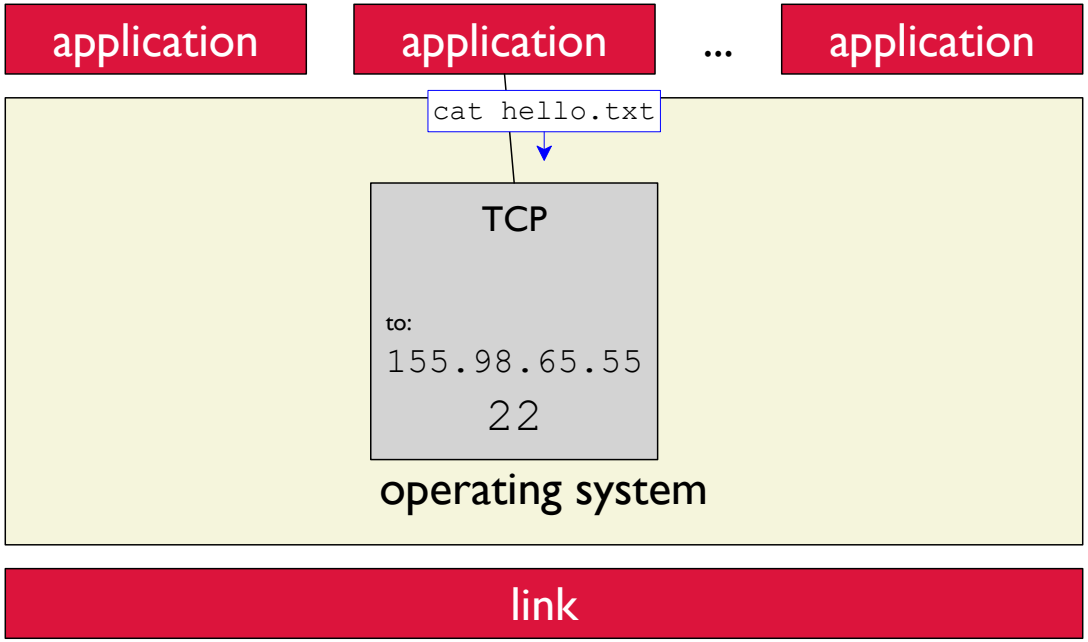
Using Sockets



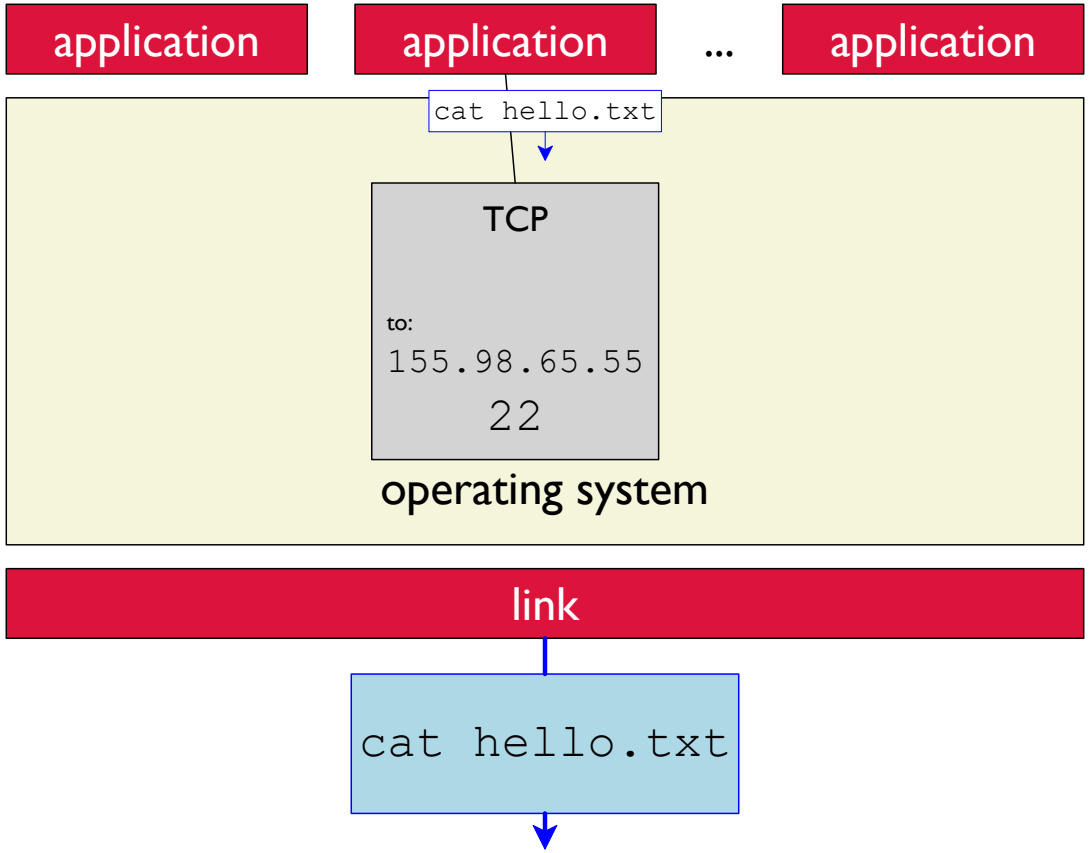
Using Sockets



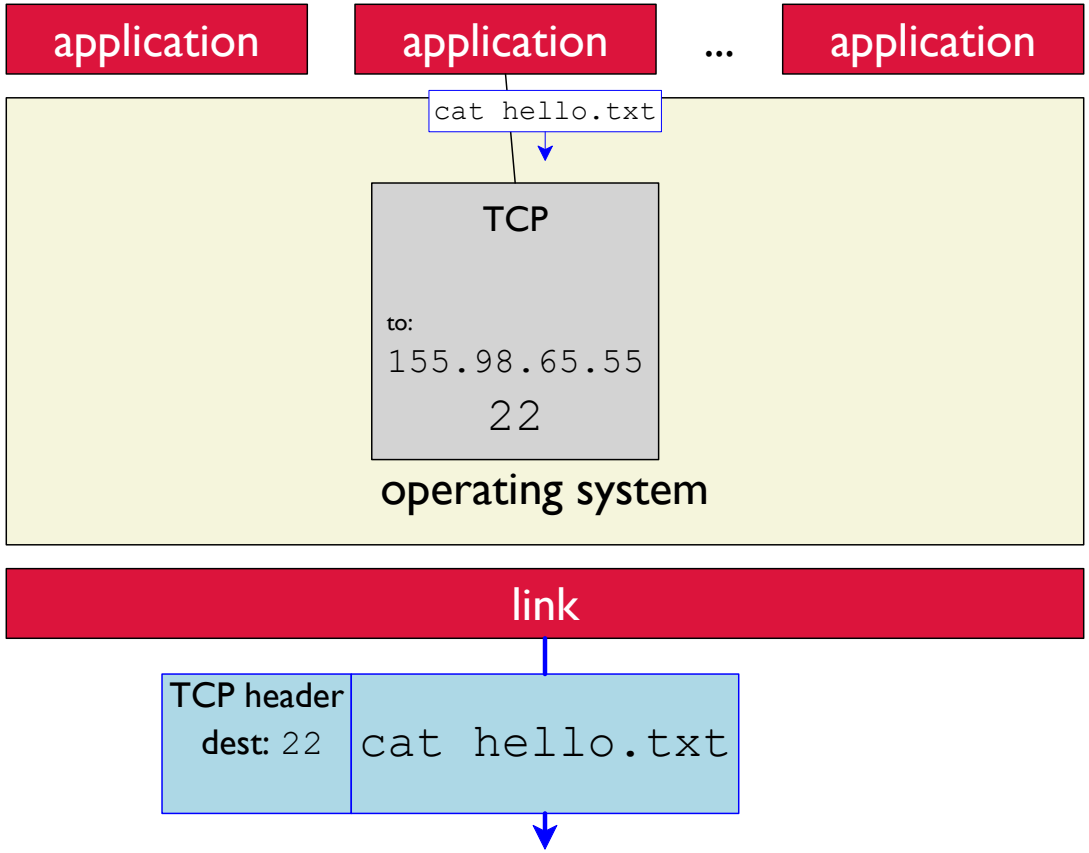
Using Sockets



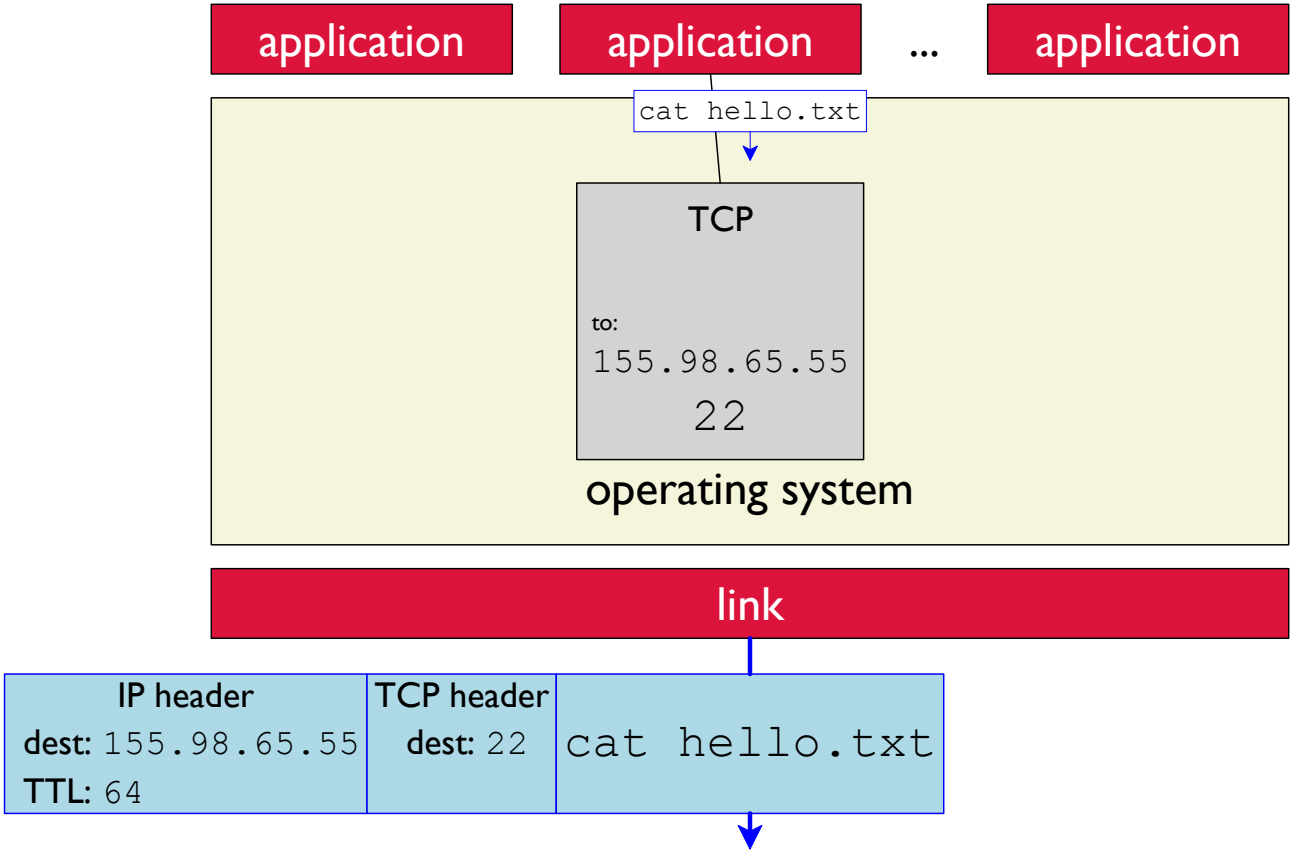
Using Sockets



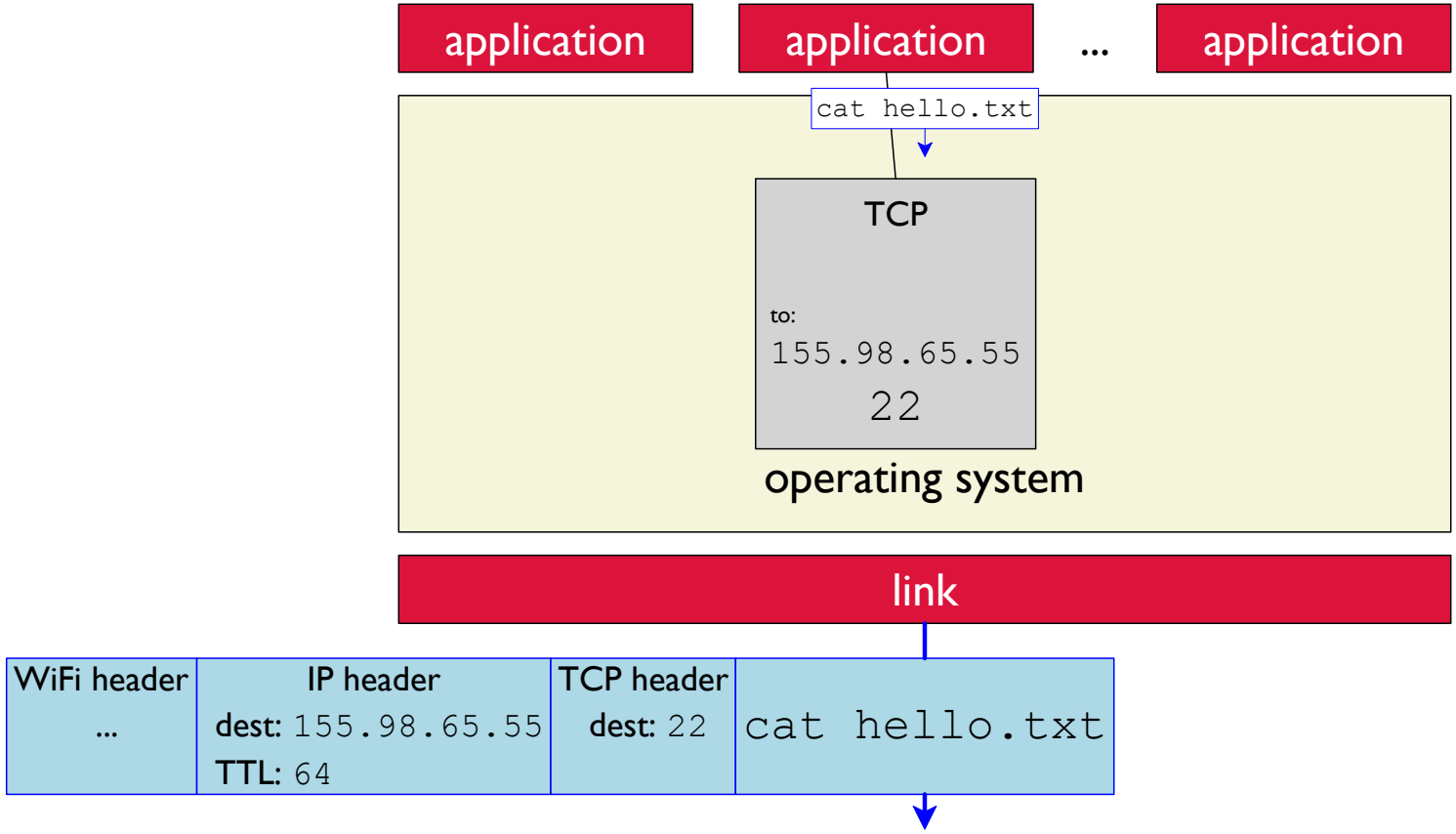
Using Sockets



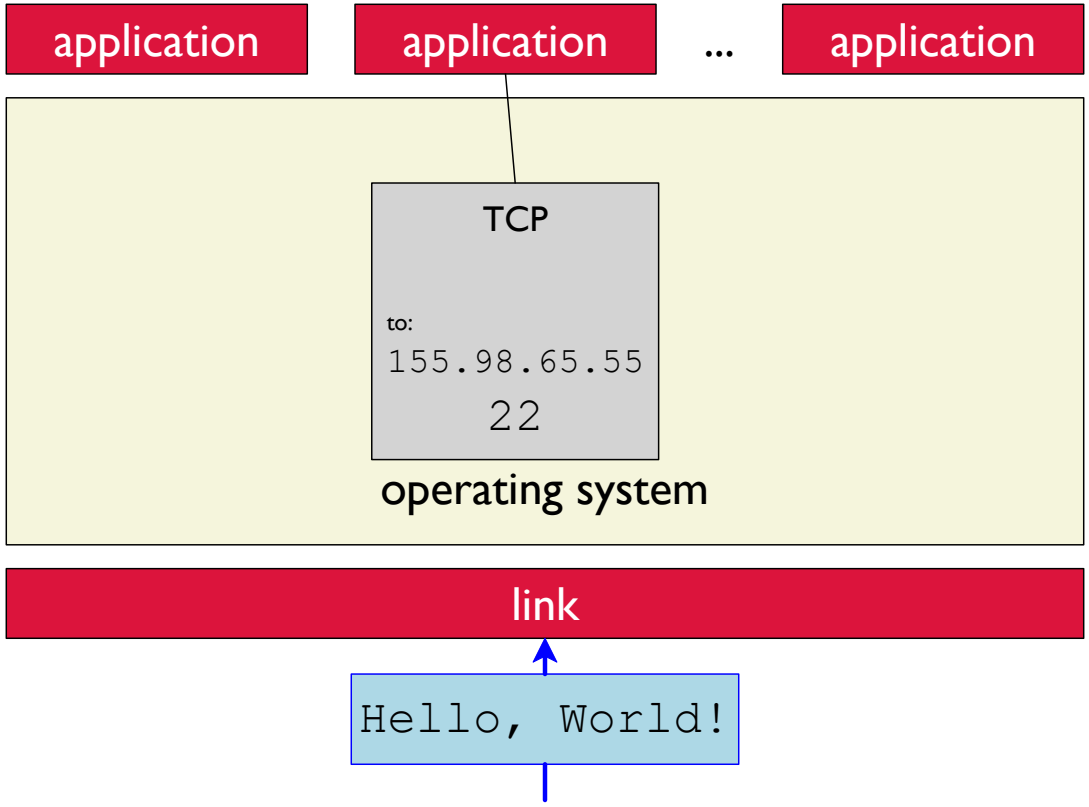
Using Sockets



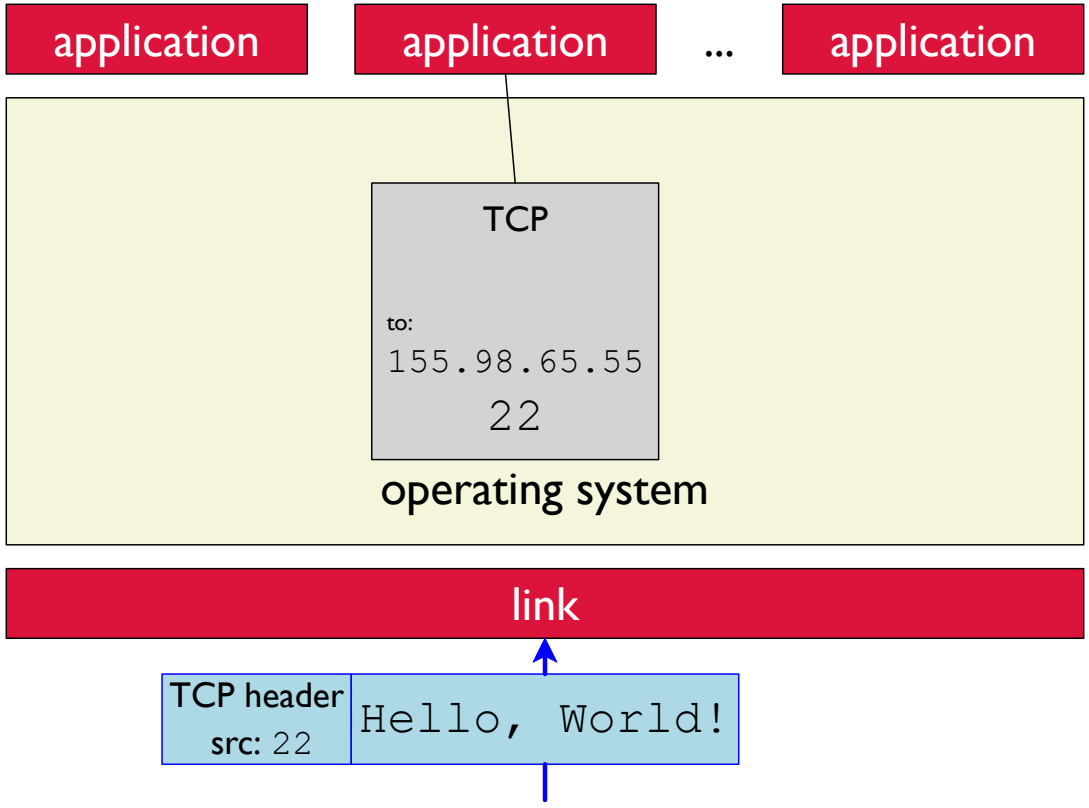
Using Sockets



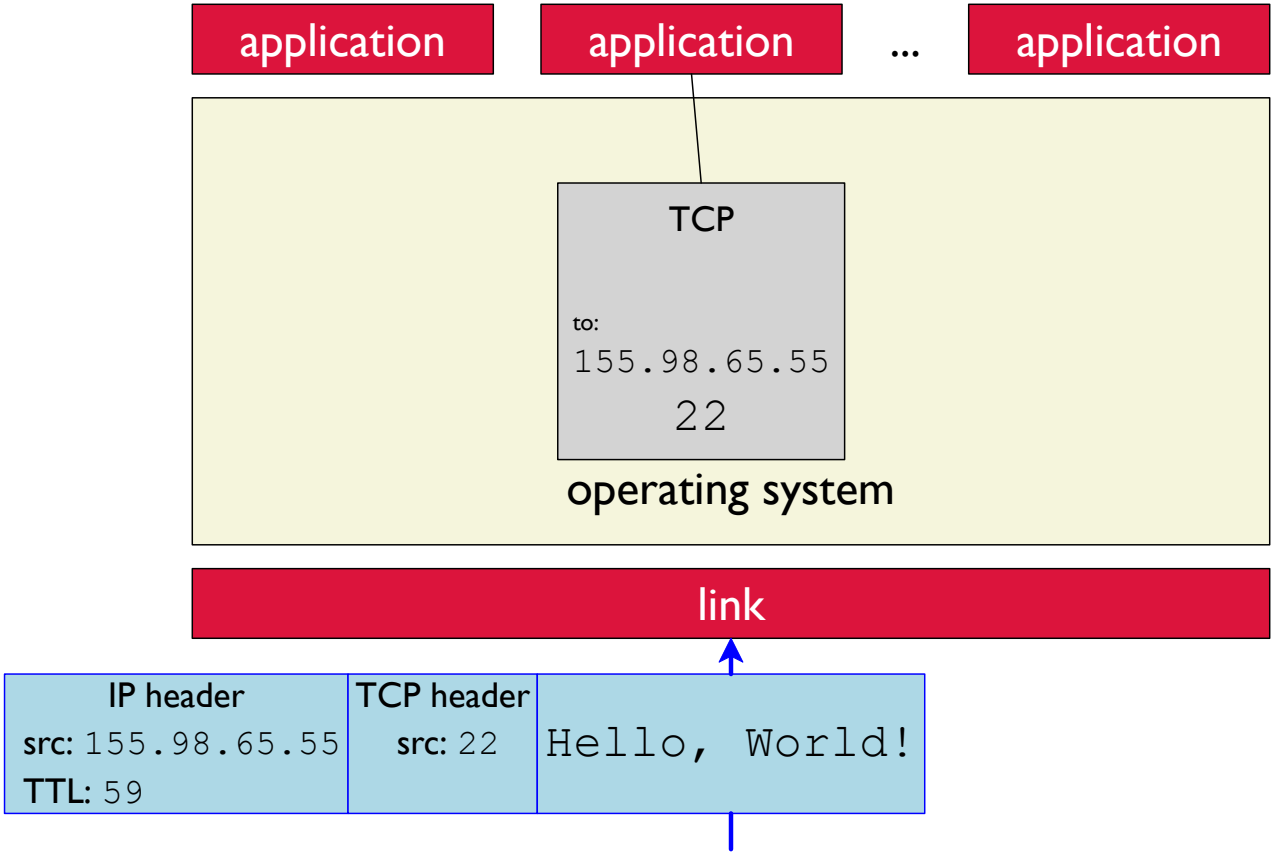
Using Sockets



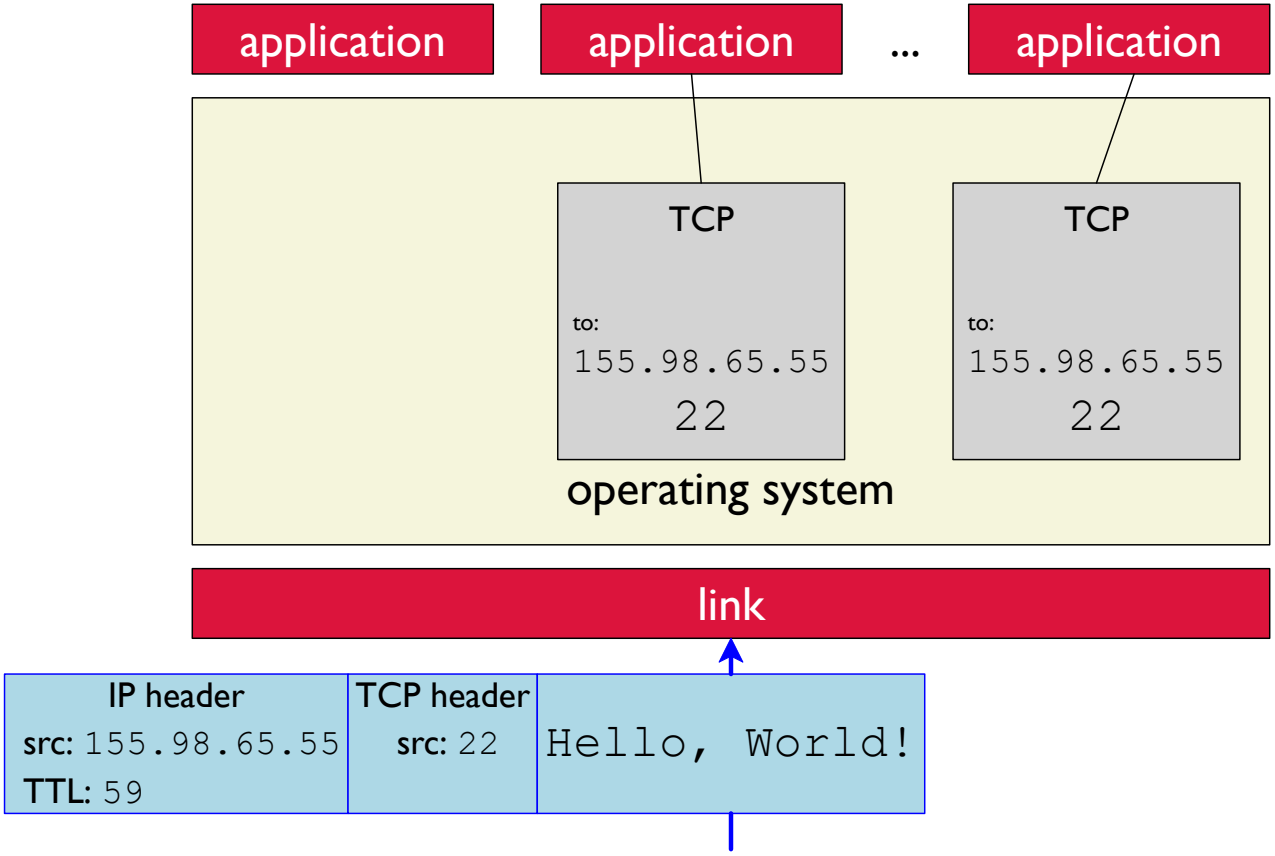
Using Sockets



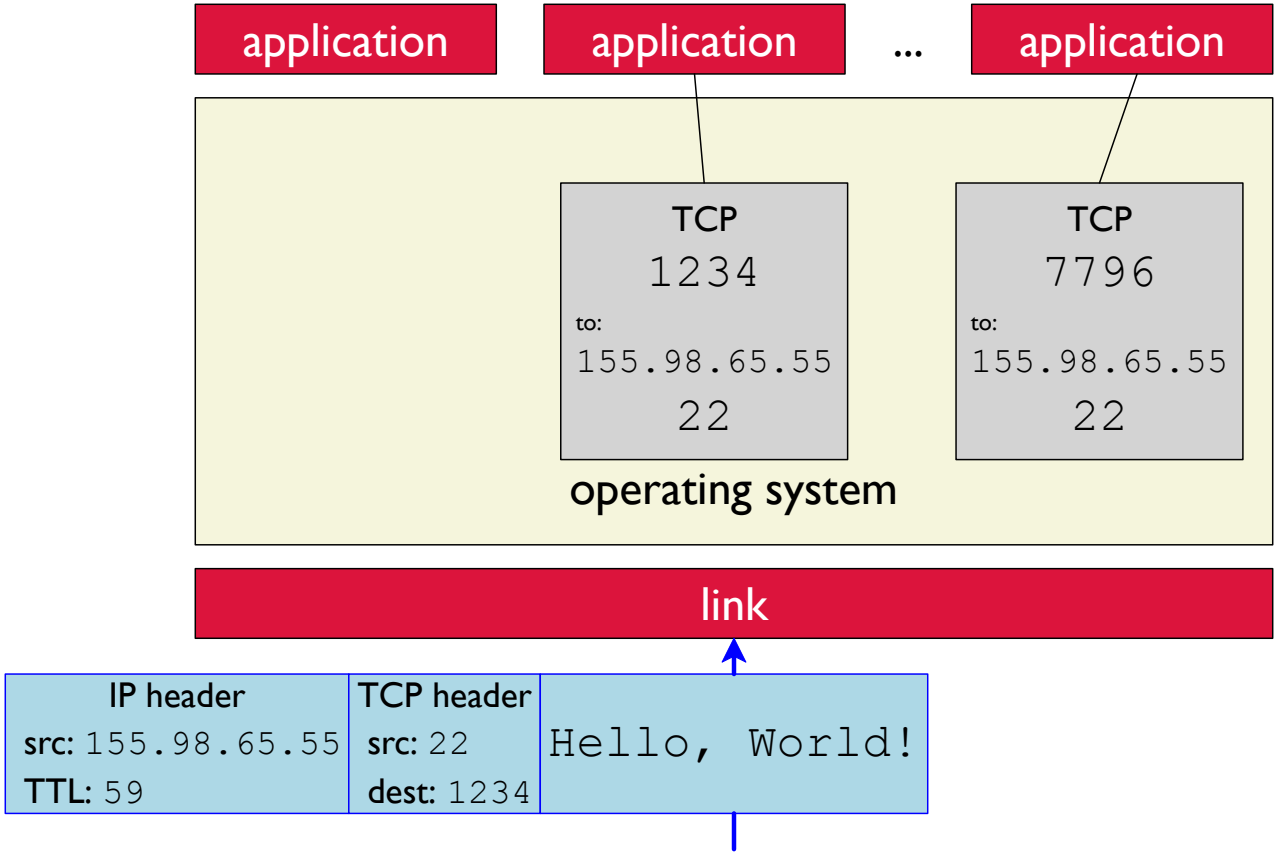
Using Sockets



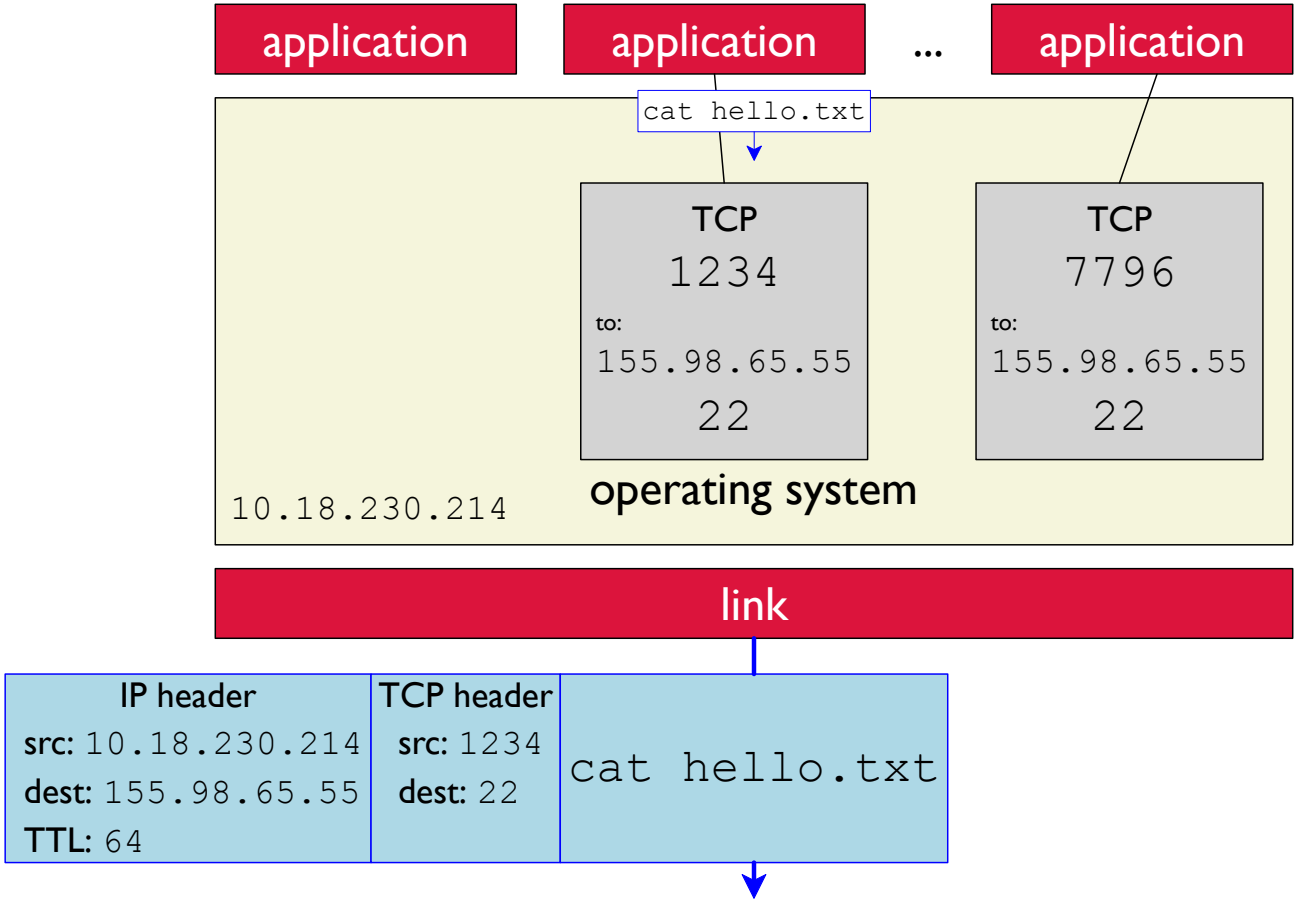
Using Sockets



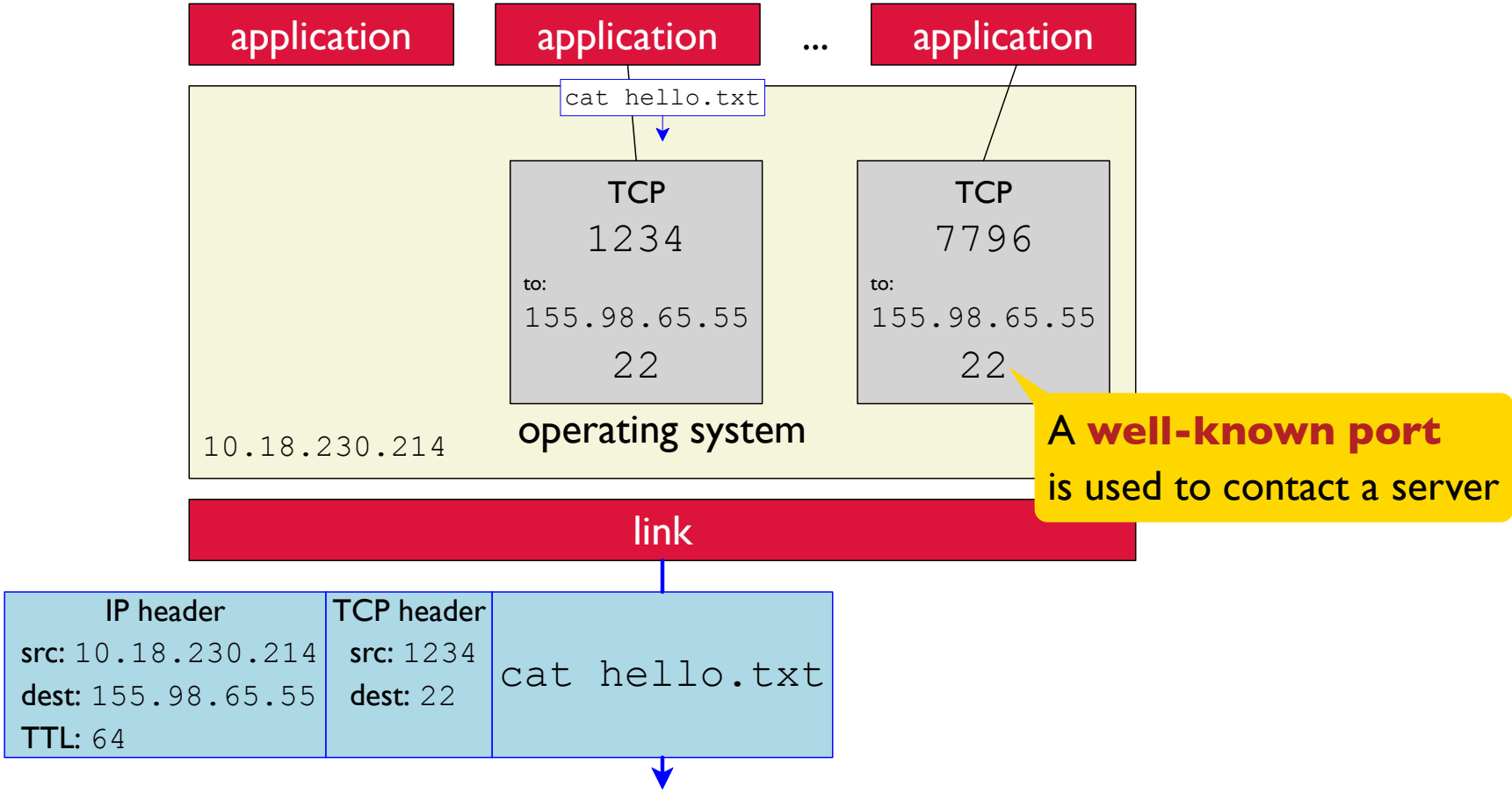
Using Sockets



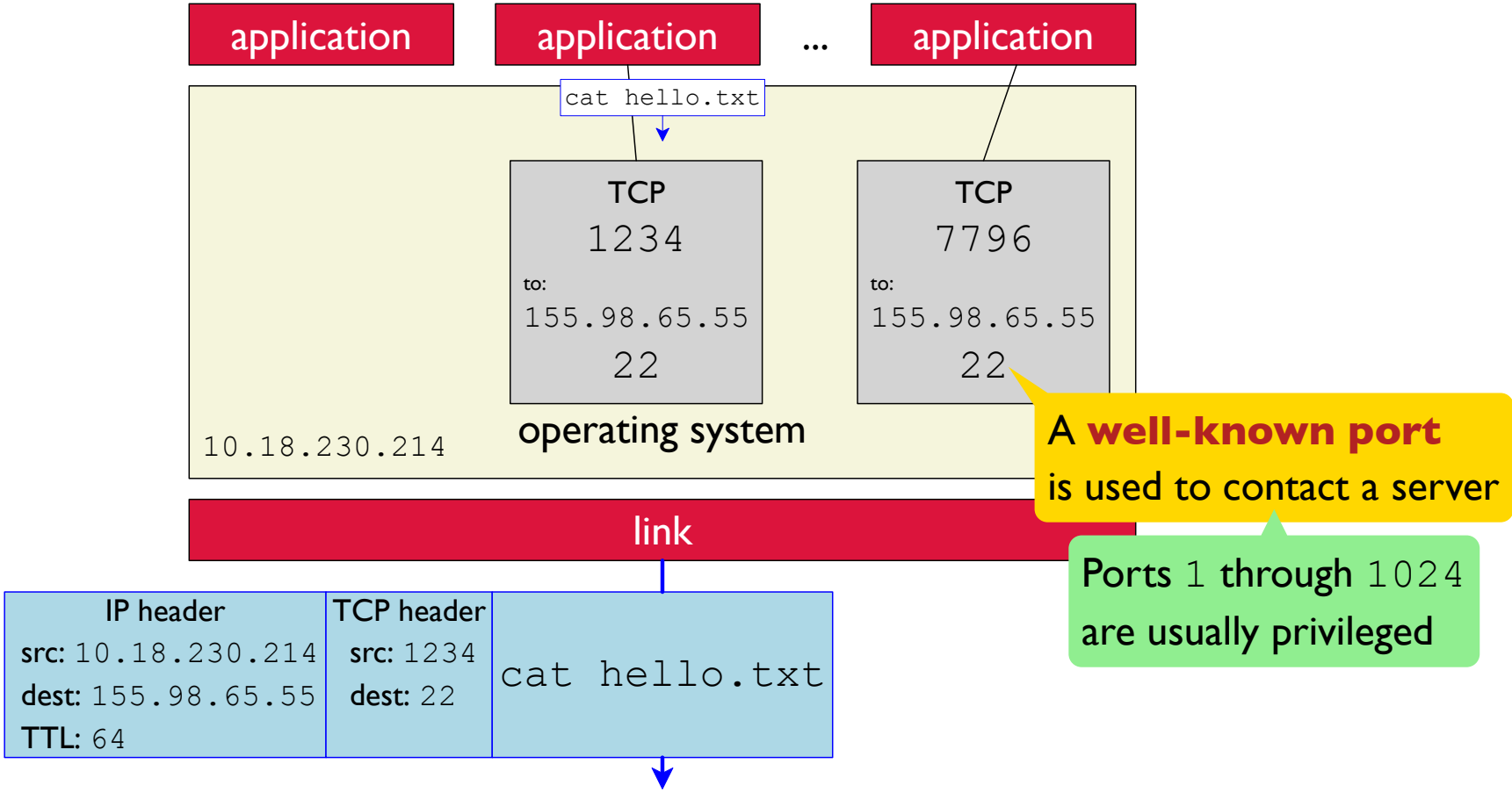
Using Sockets



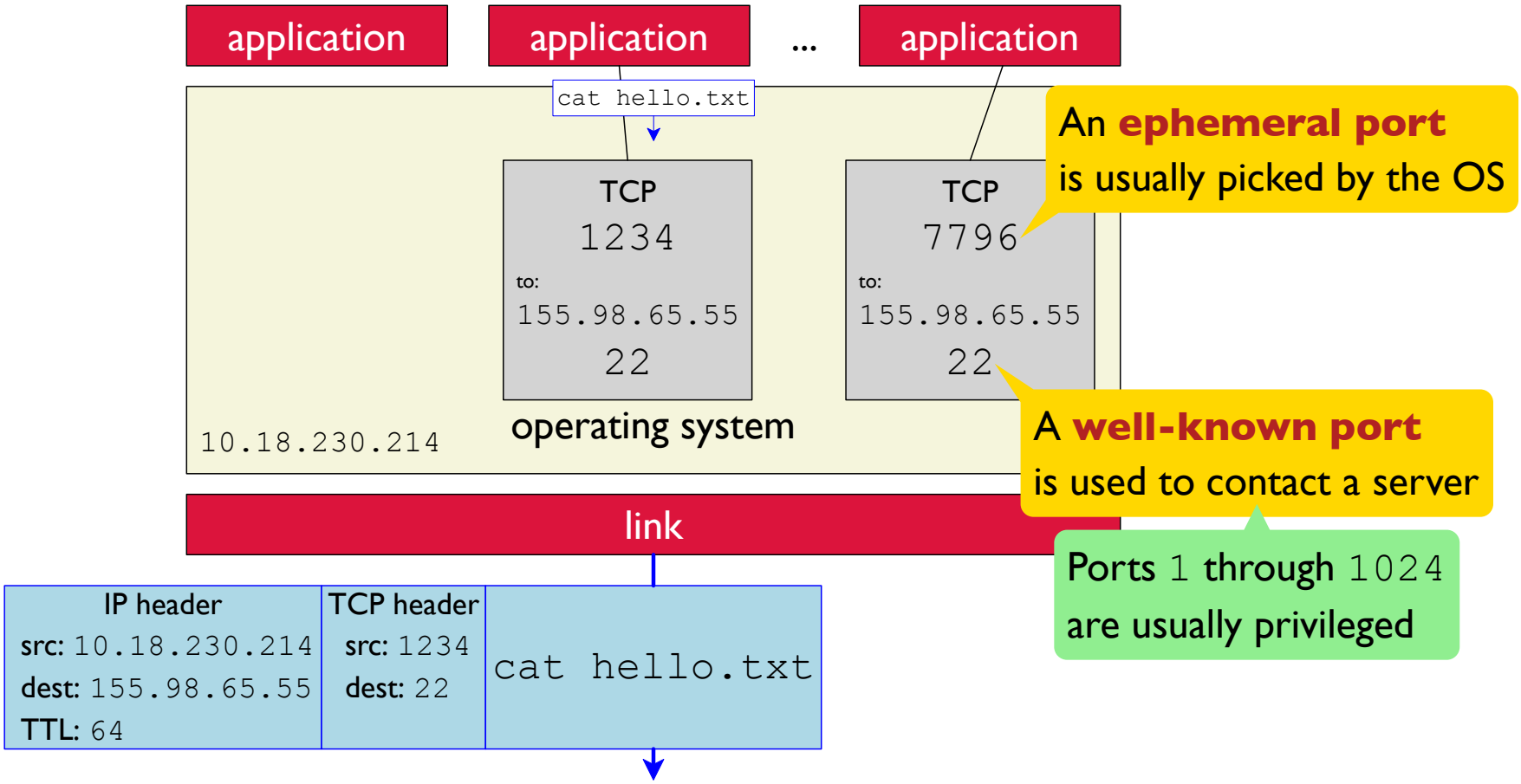
Using Sockets



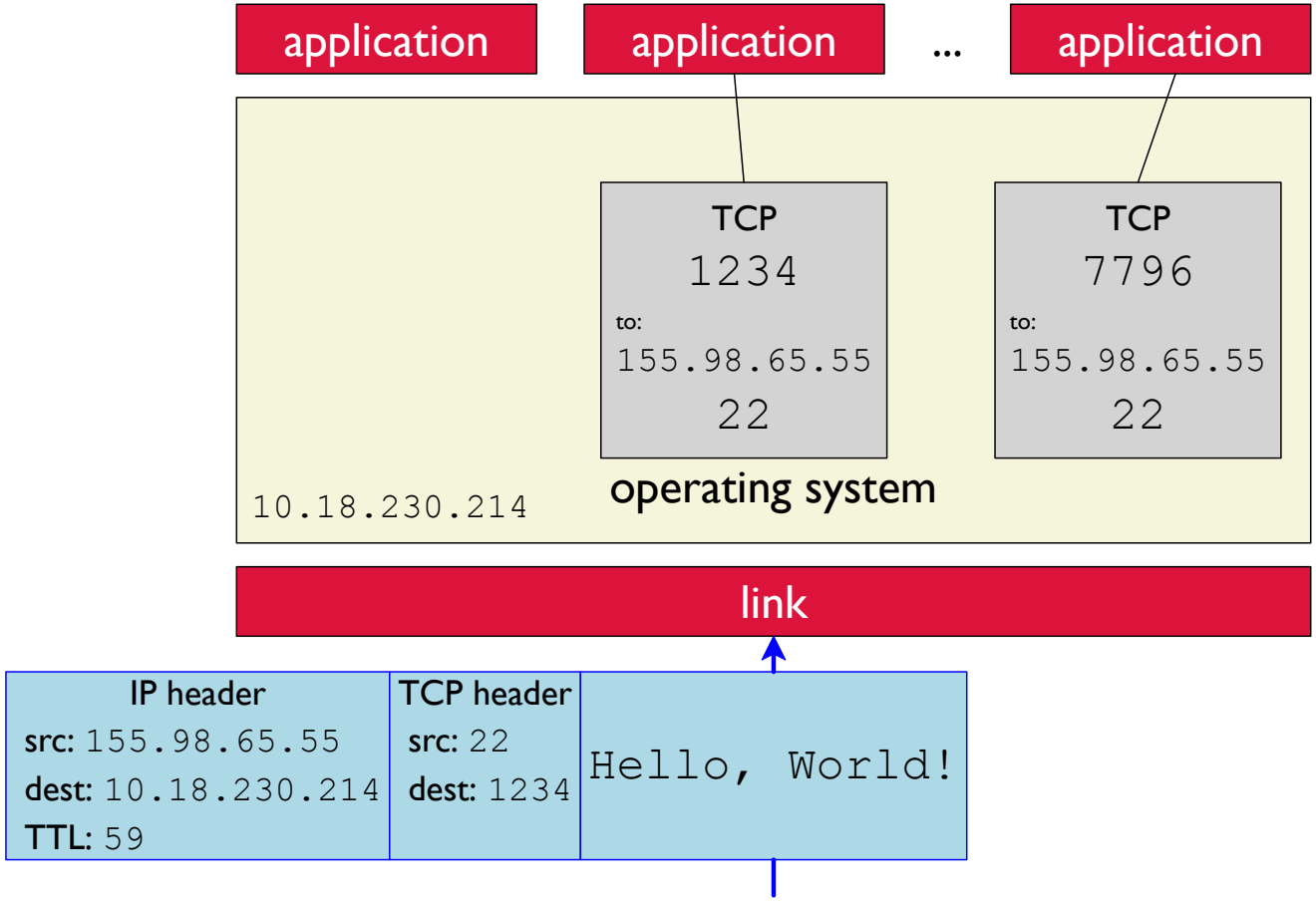
Using Sockets



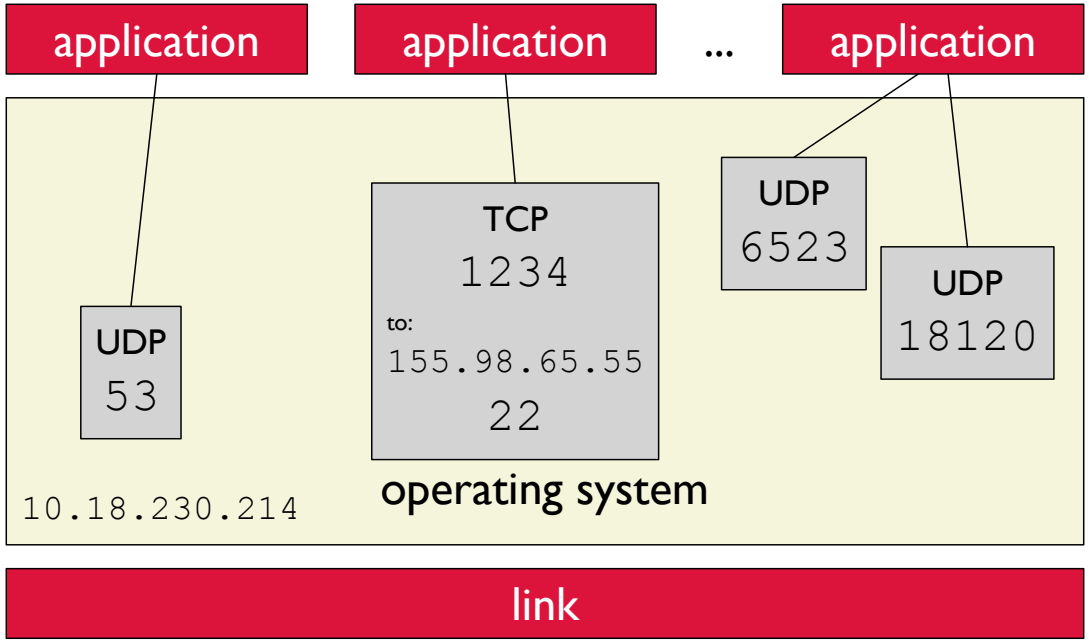
Using Sockets



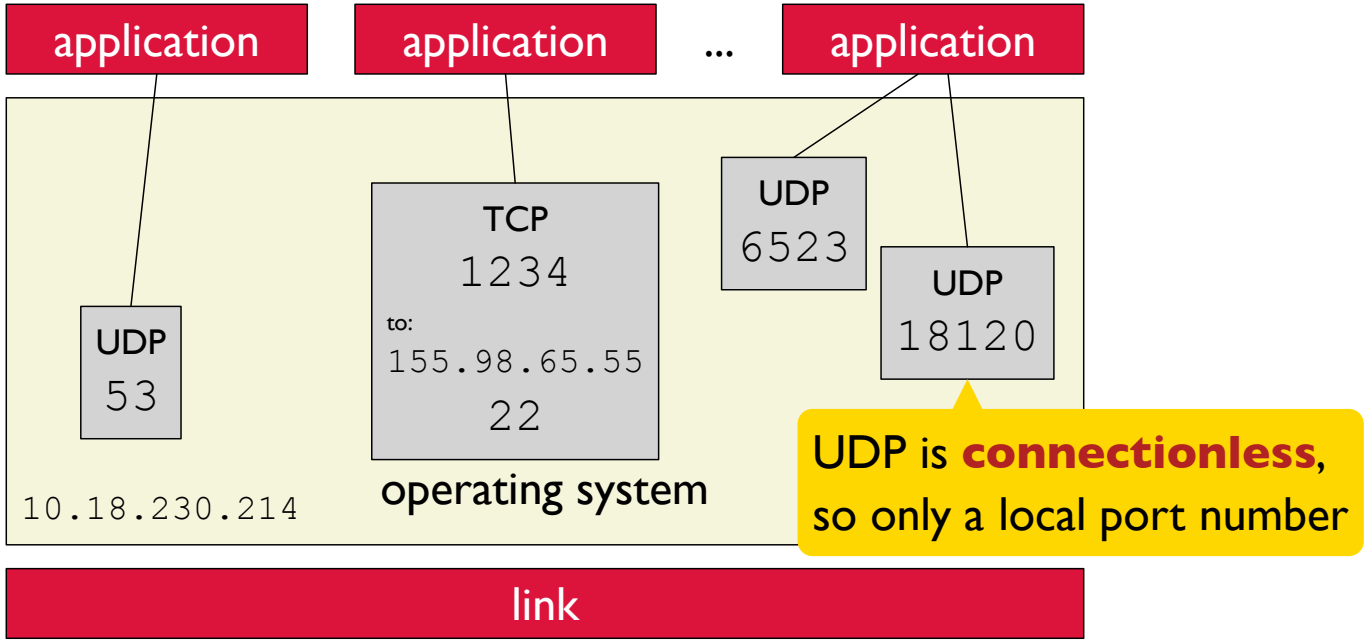
Using Sockets



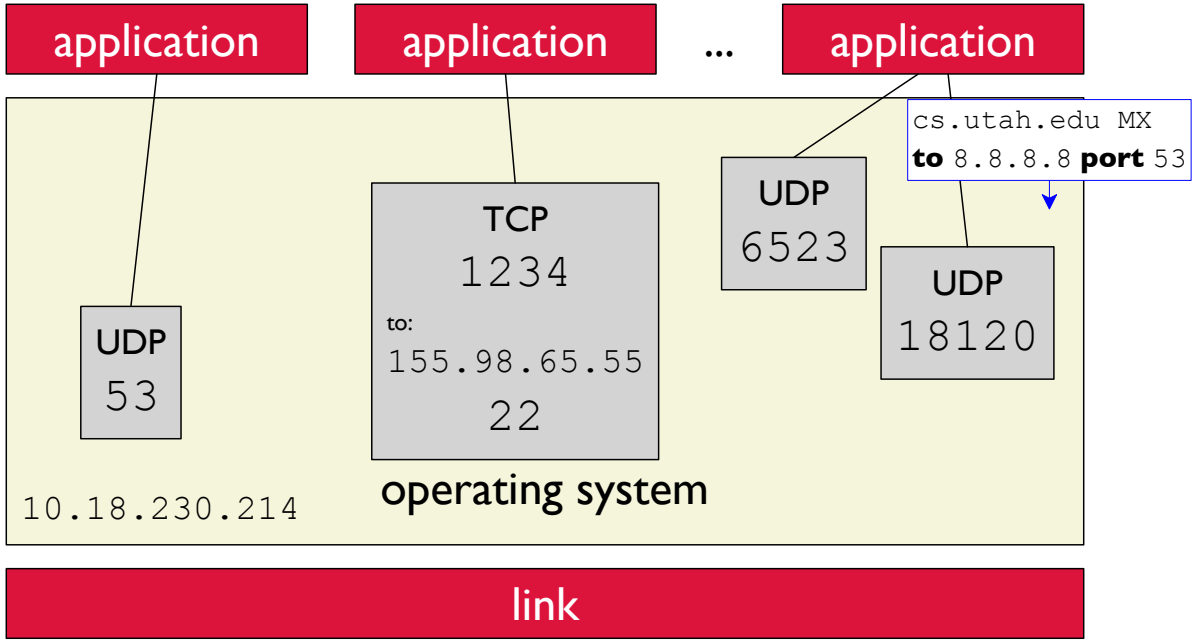
Using Sockets



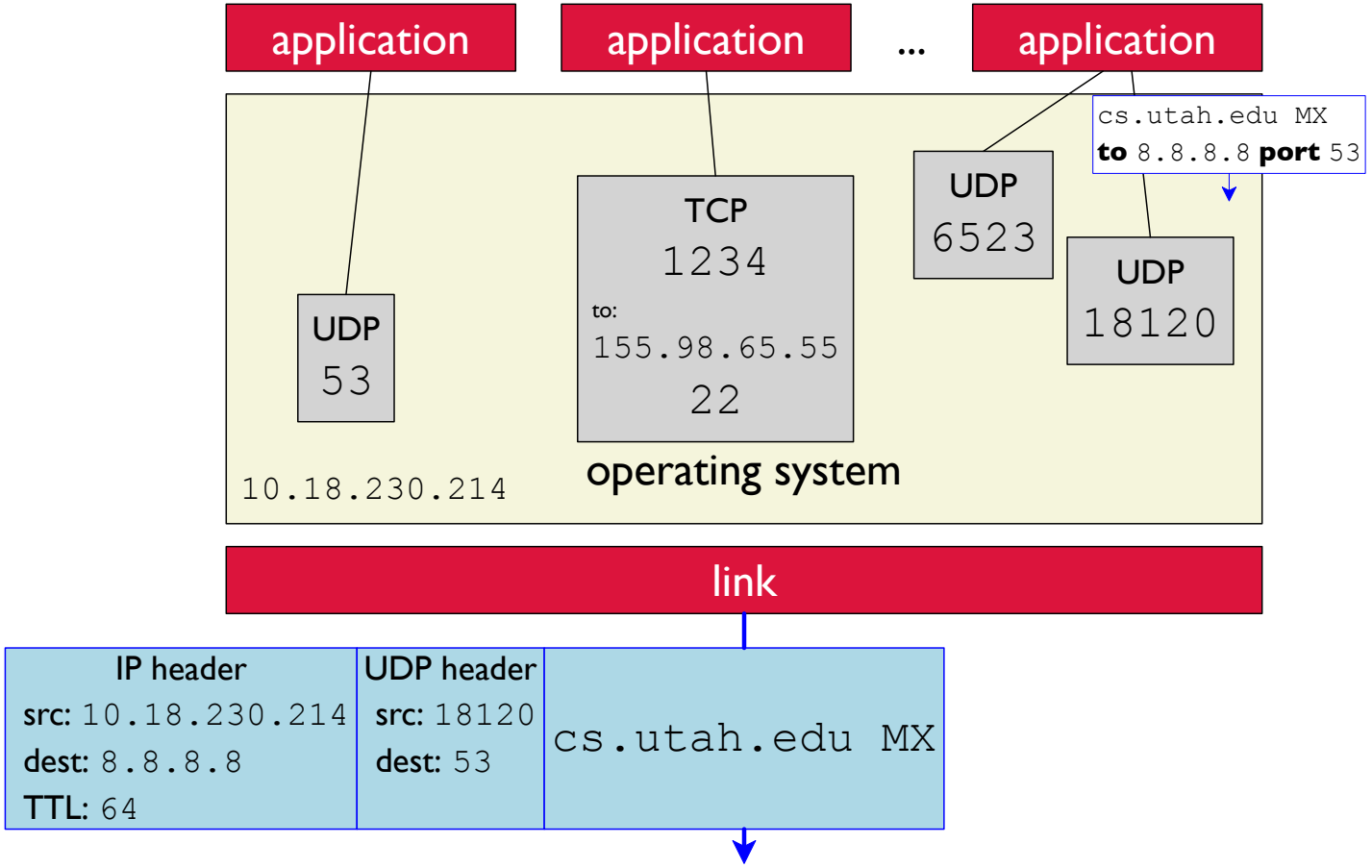
Using Sockets



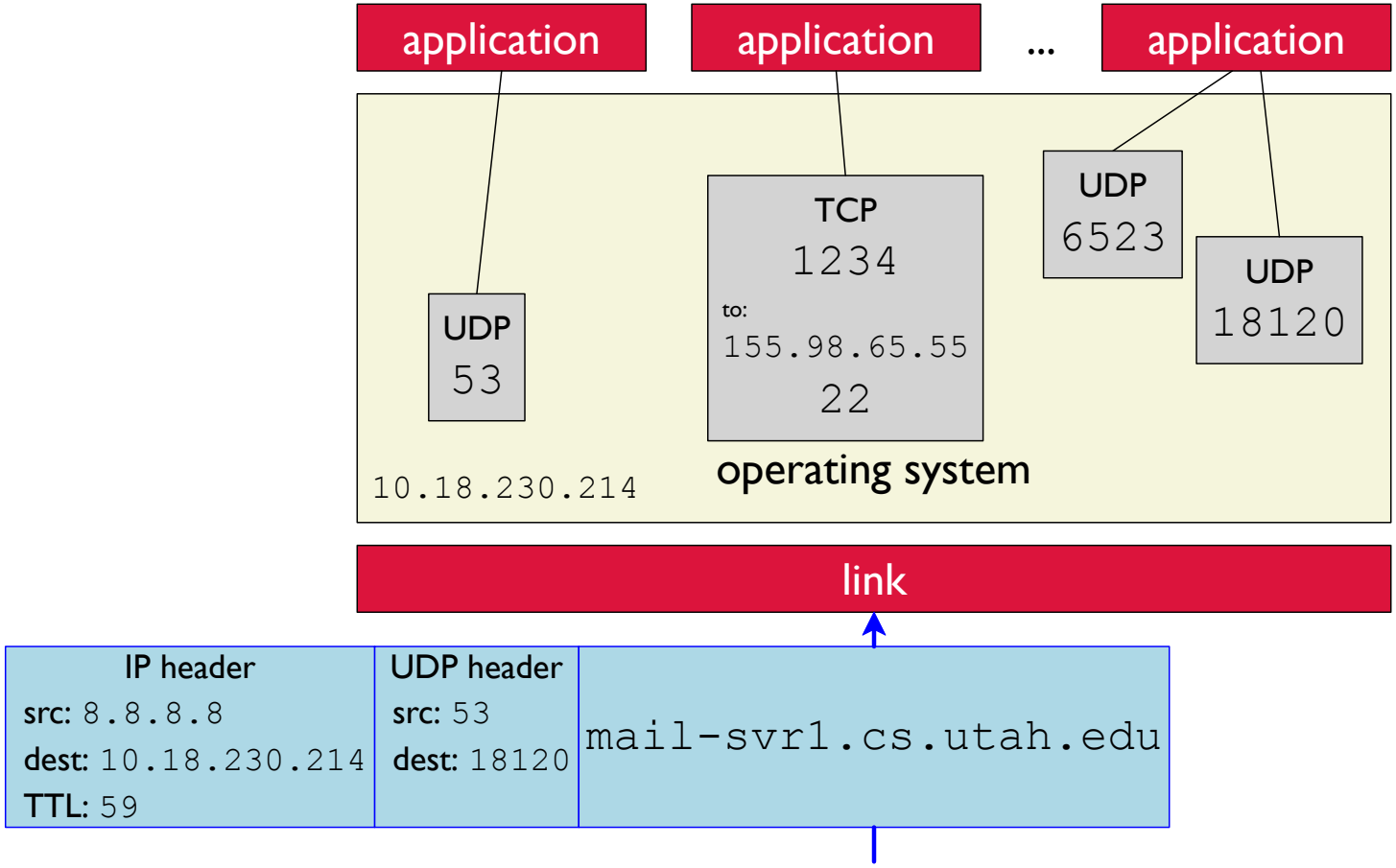
Using Sockets



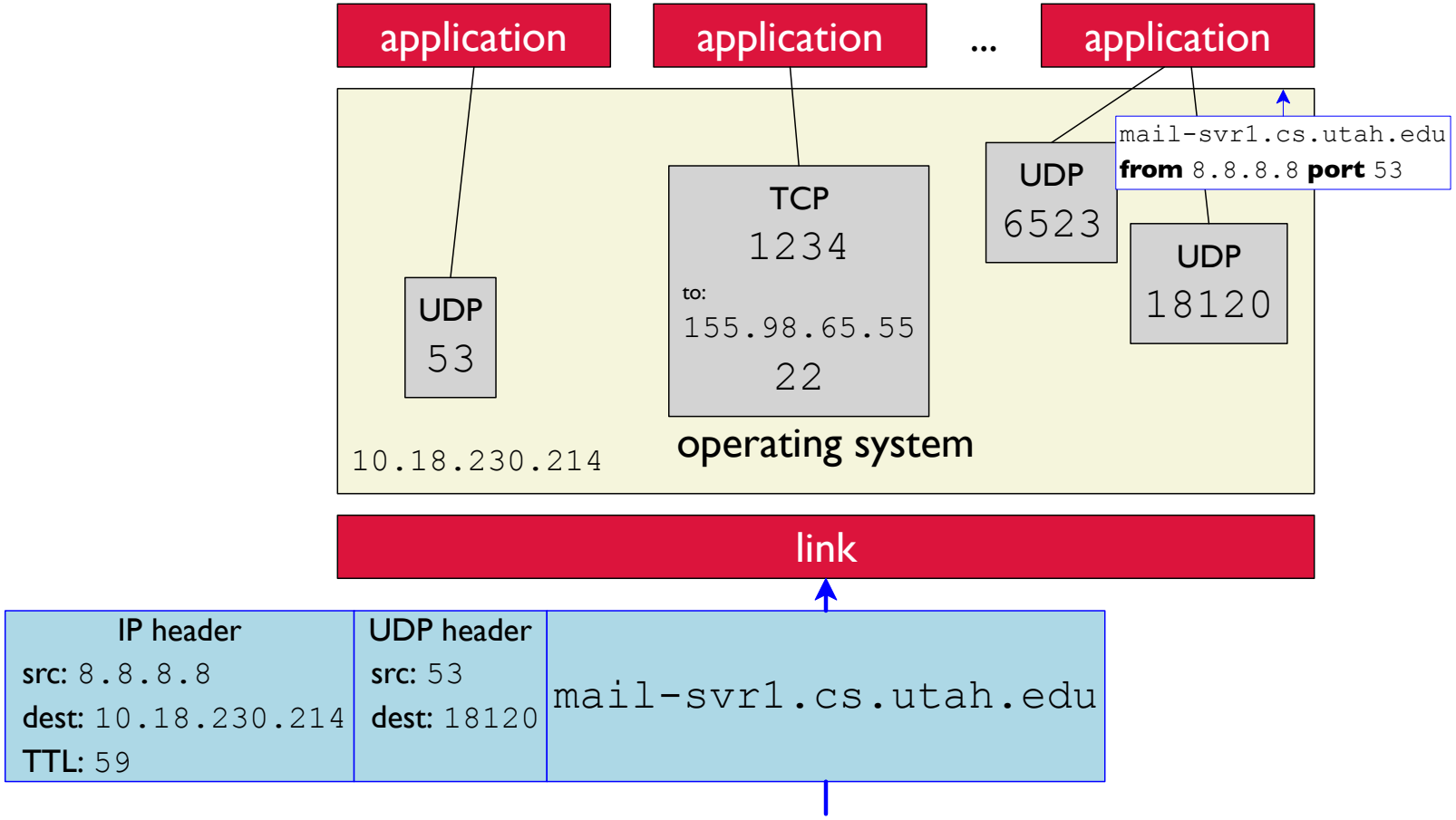
Using Sockets



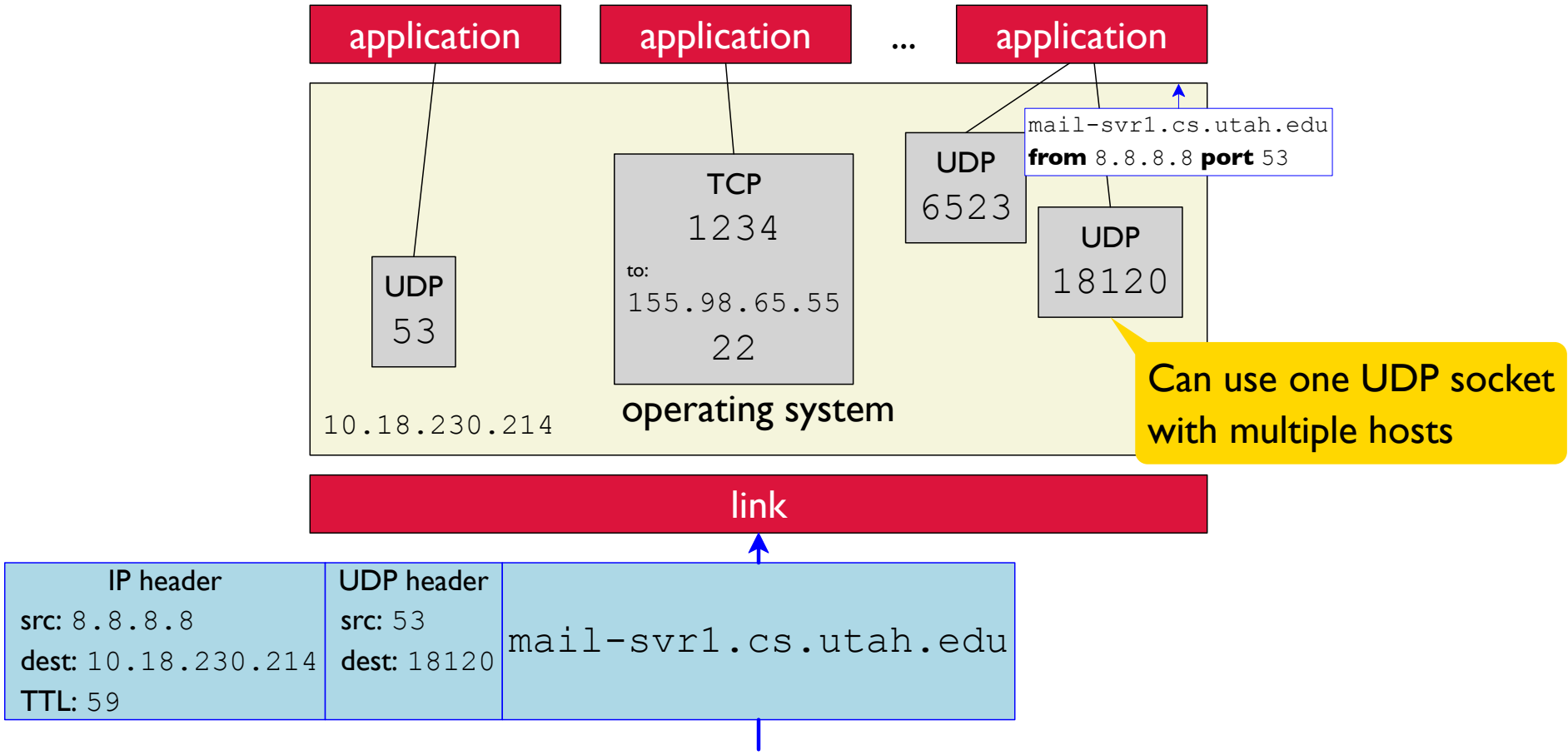
Using Sockets



Using Sockets



Using Sockets



UDP Server

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class Main {
    public static void main(String[] args) throws IOException {
        int server_port = 5678;
        System.out.println("Listening at " + server_port);
        DatagramSocket socket = new DatagramSocket(server_port);

        byte[] buffer = new byte[512];
        DatagramPacket pkt = new DatagramPacket(buffer, buffer.length);
        for (int count = 1; true; count++) {
            socket.receive(pkt); // <----- waits here
            System.out.println(count + " Heard from " + pkt.getAddress() + " " + pkt.getPort());
            for (int i = 0; i < pkt.getLength(); i++)
                System.out.printf(" %x", (int)buffer[i] & 0xFF);
            System.out.print("\n");
        }
    }
}
```

UDP Client

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class Main {
    public static void main(String[] args) throws IOException {
        int server_port = 5678;
        InetAddress server_host = InetAddress.getByName("localhost");
        System.out.println("Sending to " + server_host + " " + server_port);
        DatagramSocket socket = new DatagramSocket();

        System.out.println("I am " + socket.getLocalPort());

        byte[] data = new byte[3];
        data[0] = 10;
        data[1] = 20;
        data[2] = 30;
        DatagramPacket pkt = new DatagramPacket(data, data.length, server_host, server_port);
        socket.send(pkt);
    }
}
```

Demo

Show server code and run, but initially with
`System.out.printf(" %d", buffer[i]);`

Show client code and run

Confirm that outputs make sense

Run client again, and see a fresh ephemeral address

Demo

Client on remote machine is a variant that iterates `1` time

Run client on remote machine

- Show address obtained from `ifconfig` on server
- Confirm that client address matches expected

Adjust client on remote machine to iterate more:

- Send `1000` messages, probably all received
- Send `10000` messages, probably some dropped

Demo

Point `dig` at server:

```
dig @localhost -p 5678 www.cs.utah.edu
```

Note multiple requests, including from IPv6

Can we make sense of these bytes?

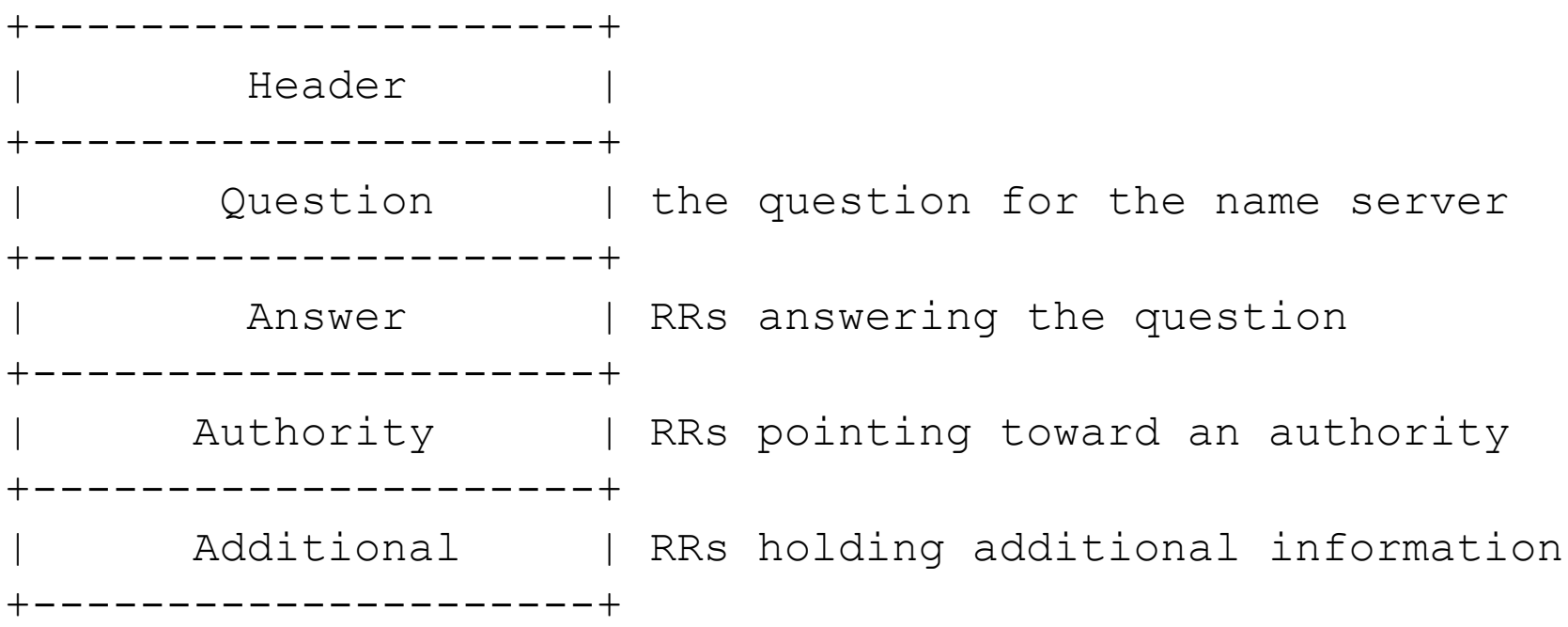
RFCs

Many internet protocols are defined by standards called **Requests for Comments** or **RFCs**

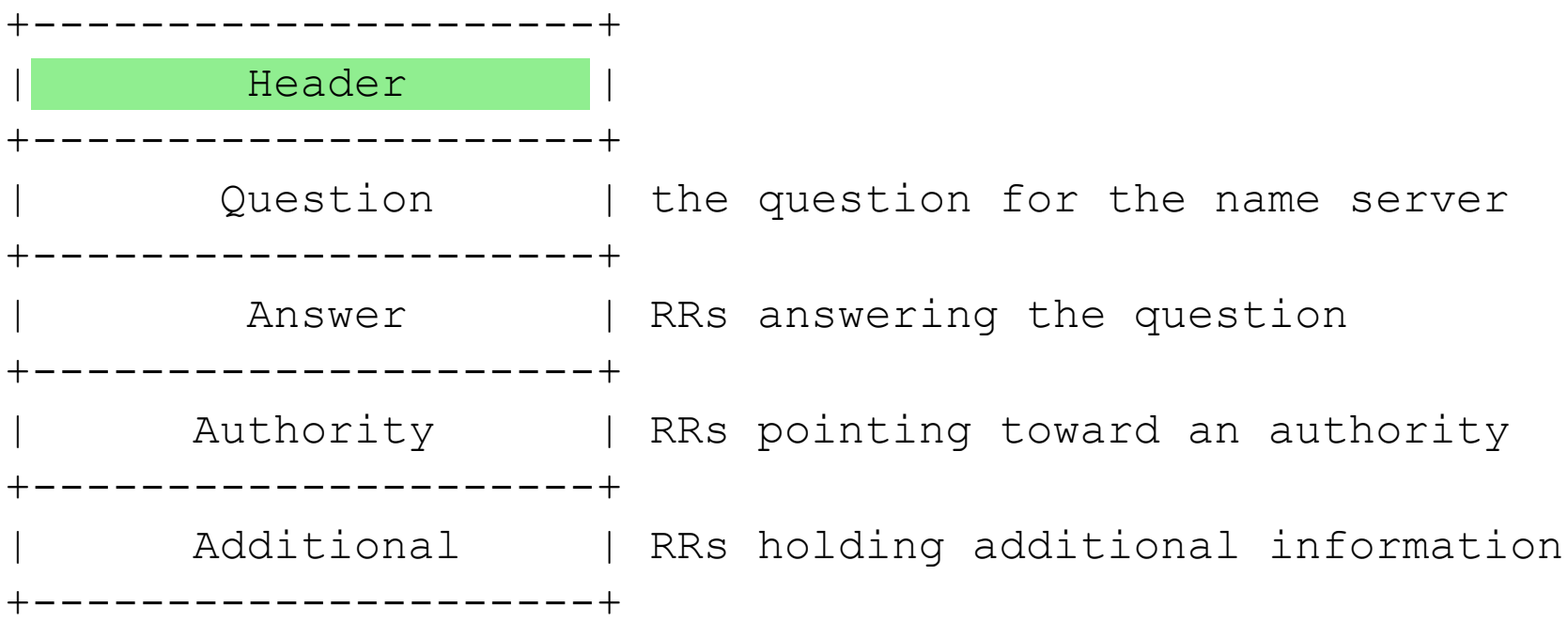
Sometimes, an initial RFC is refined or superceded by later RFCs

DNS is defined by RFCs 1035, 2929, and more

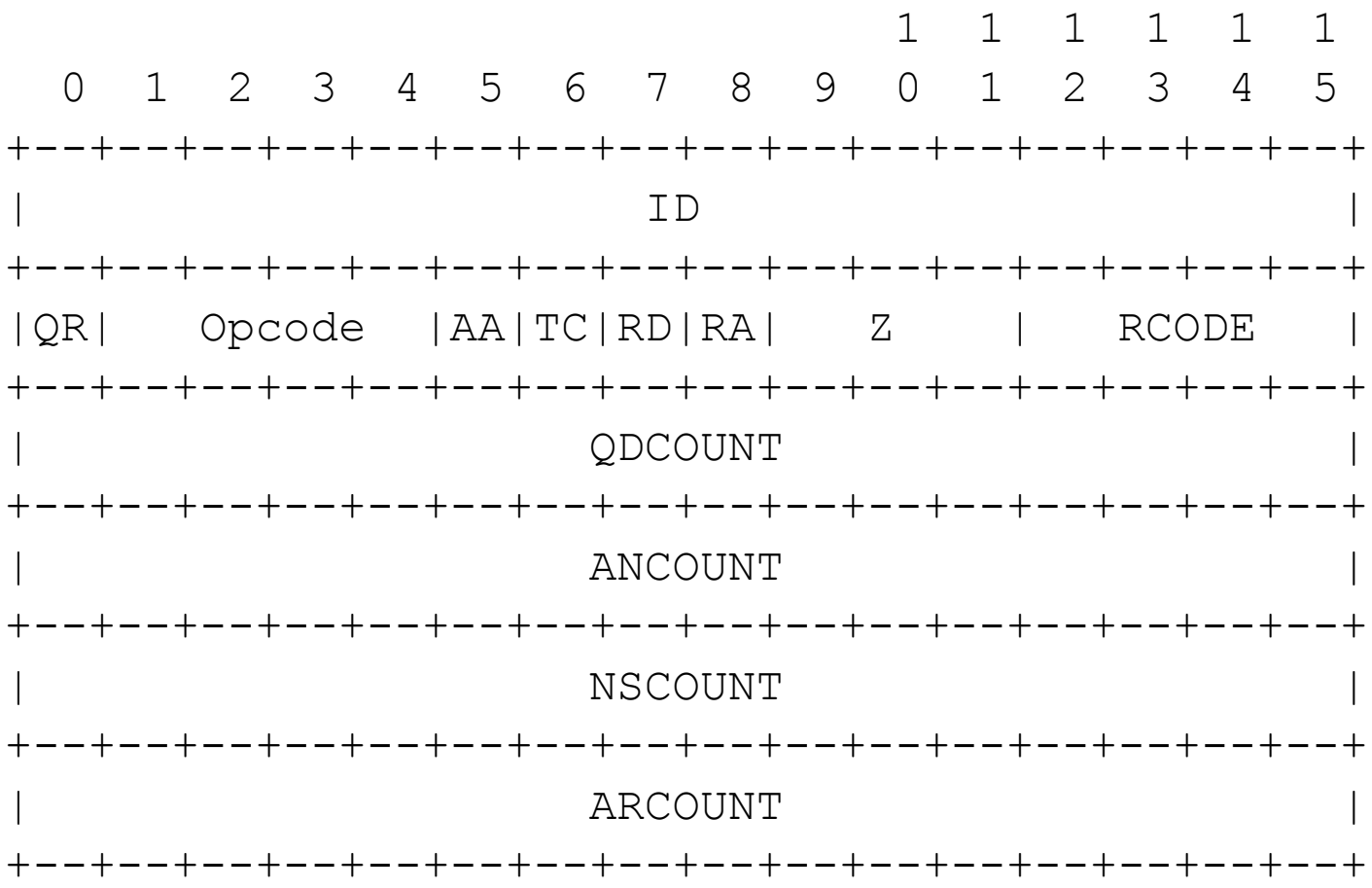
DNS Messages (RFC 1035)



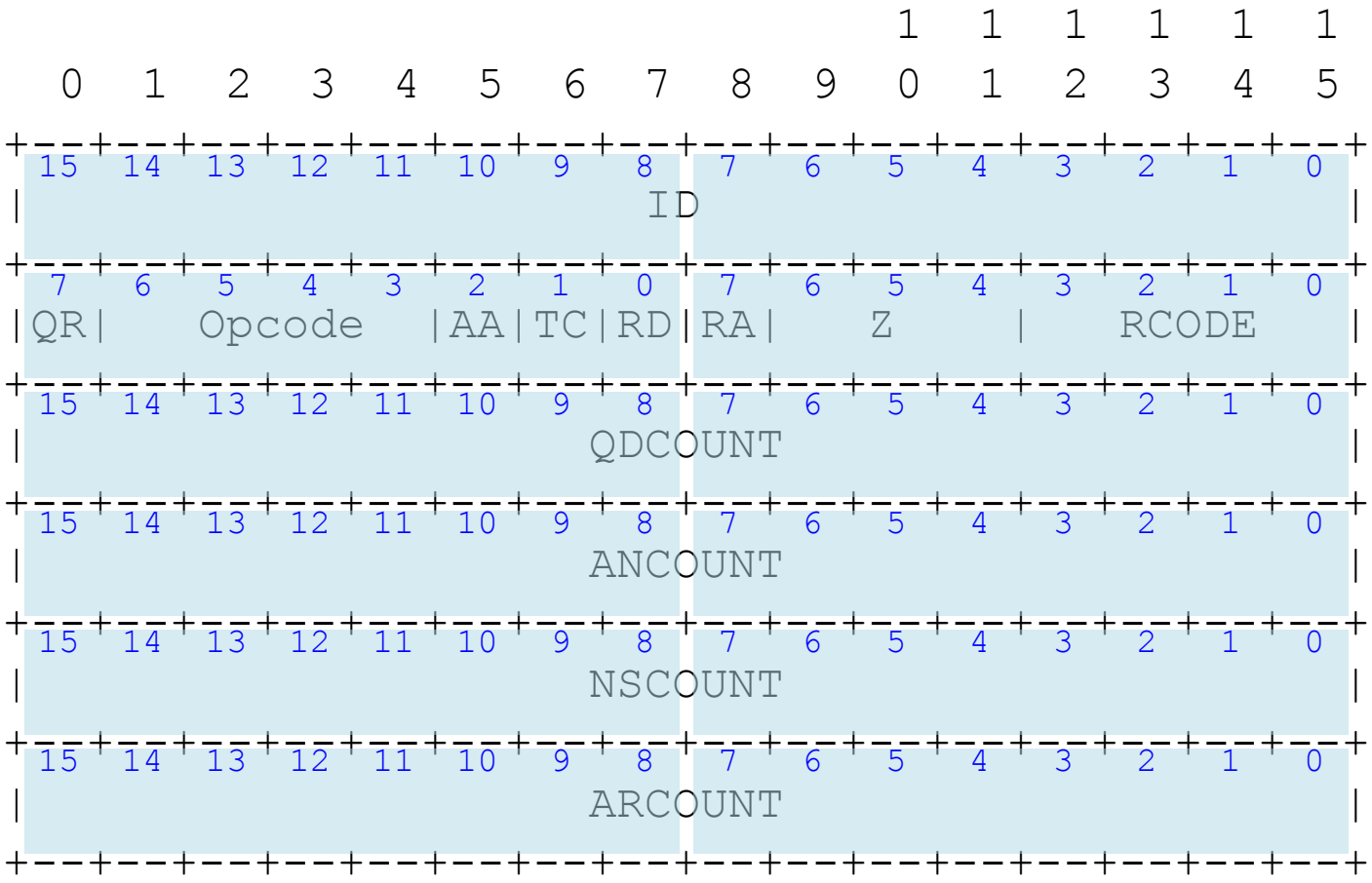
DNS Messages (RFC 1035)



DNS Header (RFC 1035)



DNS Header (RFC 1035)



Demo

Maybe change server back to hex mode:

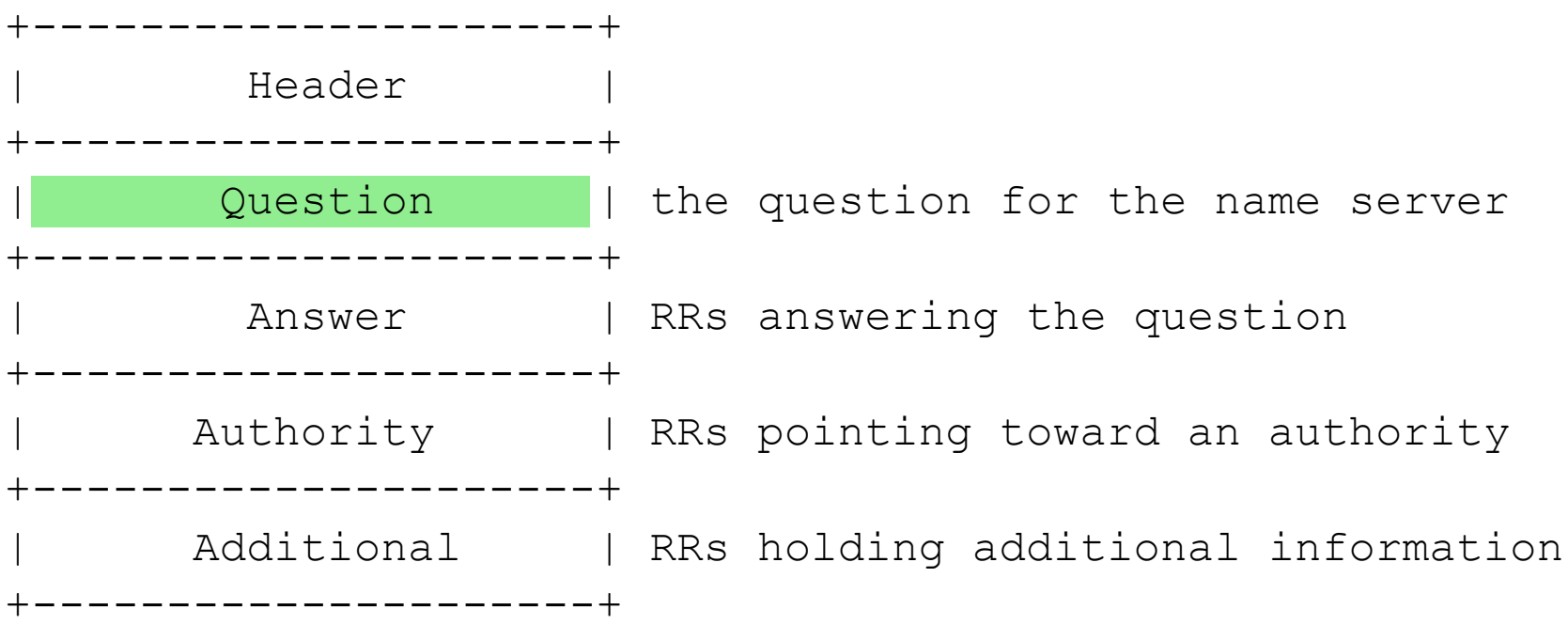
```
System.out.printf(" %x", (int)buffer[i] & 0xFF);
```

First two bytes from `dig` are meant to change every time

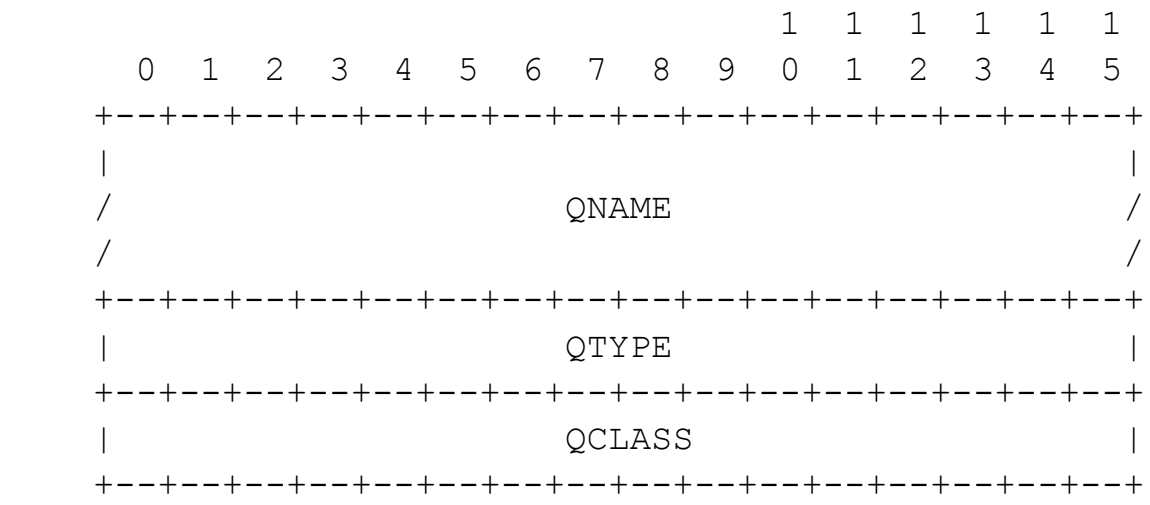
Next few bytes are always the same

- flags indicate `RD`, for example
- next is question count of `1`

DNS Messages (RFC 1035)



DNS Question (RFC 1035)



where:

QNAME a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.

Demo

At `QNAME`, our server's output has a notable 3-2-4-3 pattern

`www cs utah edu`

See the `QNAME` description again

Wireshark can help us more...

Demo

Start Wireshark

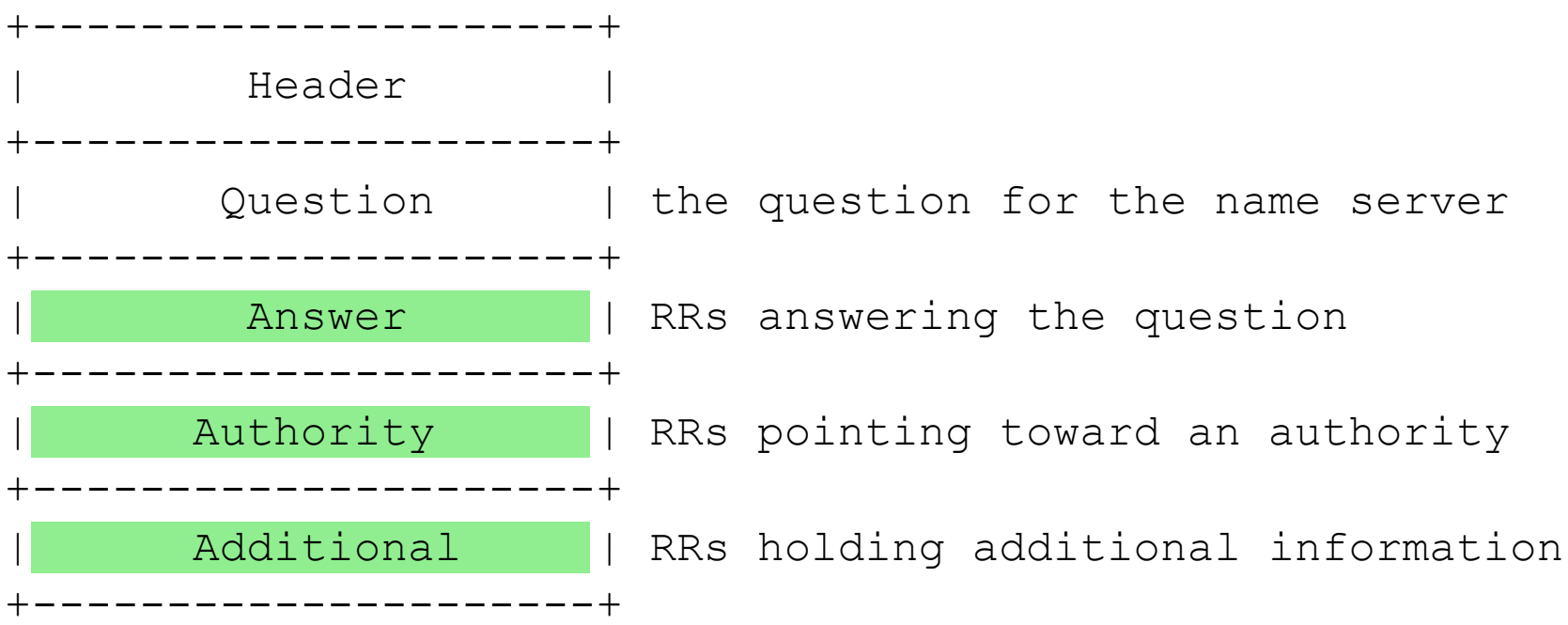
- select interface `Loopback: lo0`
- set filter to `udp && udp.port == 5678`

Run local client

Wireshark shows sent packet, but describes as "malformed," because port 5678 not standard

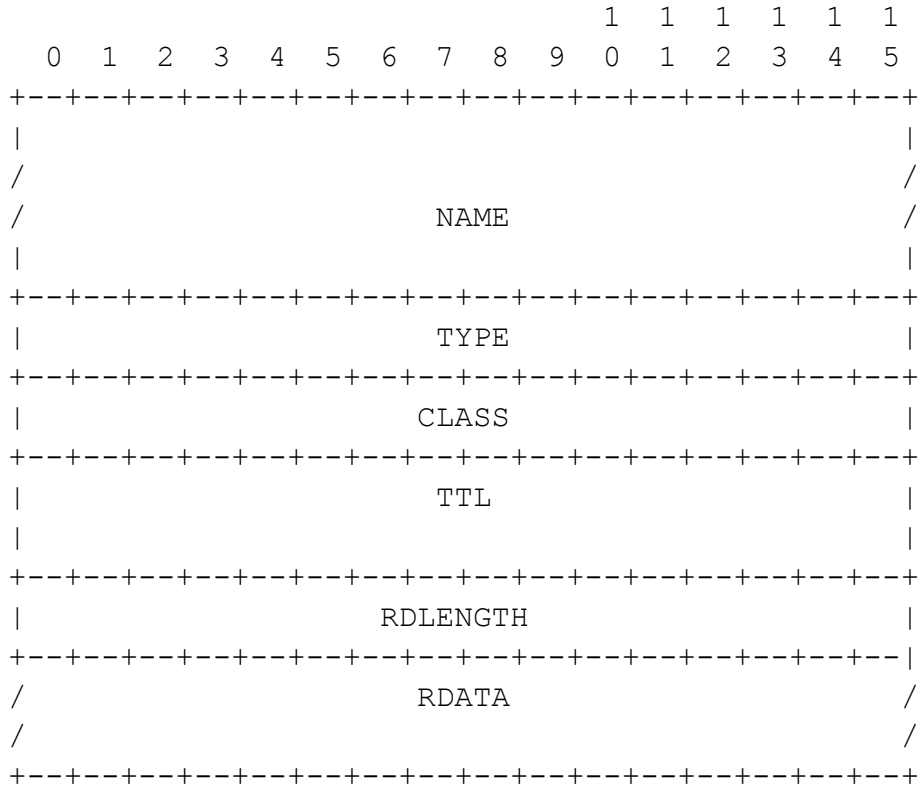
- Right-click (or control-click) line in top panel
- Pick **Decode As...**
- Put `DNS` in **Current** column
- Bottom-right panel now says "DNS"

DNS Messages (RFC 1035)



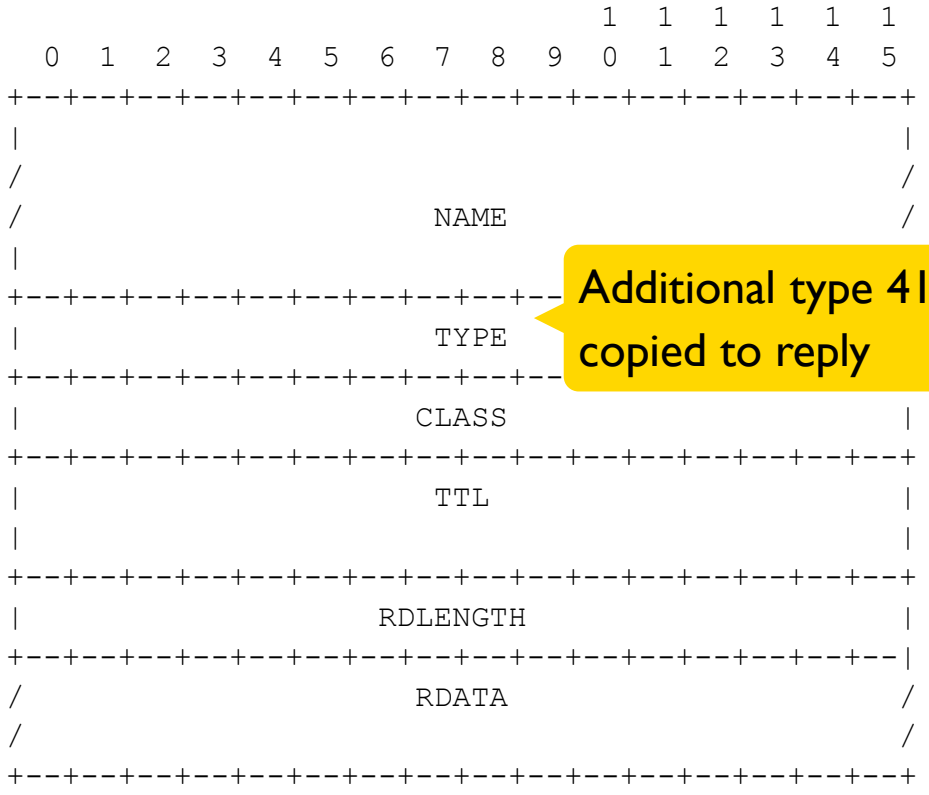
DNS Resource Record (RFC 1035)

Used for Answer, Authority, and Additional:



DNS Resource Record (RFC 1035)

Used for Answer, Authority, and Additional:



DNS Names (RFC 1035)

Relevant to names, whenever they appear:

4.1.4. Message compression

In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurance of the same name.

The pointer takes the form of a two octet sequence:

The first two bits are ones. This allows a pointer to be distinguished from a label, since the label must begin with two zero bits because labels are restricted to 63 octets or less.

Resources

https://datatracker.ietf.org/doc/html/rfc1035

https://datatracker.ietf.org/doc/html/rfc2929

https://docs.oracle.com/javase/8/docs/api/java/net/DatagramPacket.html

https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html

https://docs.oracle.com/javase/8/docs/api/java/nio/ByteBuffer.html

Summary

Message exchange via UDP involves

- receiver *address* and *port*
- sender *address* and *port*

Each packet has all four

Server needs a known address and port

A client tends to use an **ephemeral port**

Adopt a byte-level view of protocols when reading **RFCs** and using Wireshark