Networking! Ack!

A computer networking zine!

By Julia Evans!

There's so much networking stuff to learn! OMG!

Hey, it's not so bad! Let's learn the ideas one at a time!

Networking isn't magic, but I sure feel like a wizard now!
cast of characters

in your house

your laptop (that you use to look at cats)

your program

operating system (knows how to do networking)

WRT (your home router)

computers you’ll talk to

jvns.ca

DNS server (has cat picture)

(DNS server hosts jvns.ca)

the cat picture we’re downloading

in the middle

intermediate routers on the internet

packets!
What's this?!?

hi! I'm Julia
twitter: @b0rk
blog: http://jvns.ca

I put a picture of a cat on the internet here:

jvns.ca/cat.png

(go look!)

In this zine we'll learn everything (mostly) that needs to happen to get that cat picture from my server to your laptop.

My goal is to help get you from

I've heard about some of these HTTP/DNS/TCP things but I don't understand how they work exactly or how they all fit together

me after I'd been working as a web developer for a year

me now

to...

there's a networking problem! I totally know where to start!
our star: the packet

All data is sent over the internet in packets. A packet is a series of bits (01001011011....) and it's split into sections (or "headers")

Here's what a UDP packet that says "mangotea" looks like. It's 50 bytes in all!

(400 bits)

We are going to work on explaining it!

Julia I don't understand this diagram

---

<table>
<thead>
<tr>
<th>destination MAC</th>
<th>source MAC addr</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>84</td>
</tr>
</tbody>
</table>

84 bits

<table>
<thead>
<tr>
<th>ver</th>
<th>hlen</th>
<th>TOS</th>
<th>packet length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>identification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>flg fragment offset</td>
</tr>
<tr>
<td>TTL</td>
<td>protocol</td>
<td>header checksum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source IP address</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination IP address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source port</th>
<th>destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>UDP checksum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>a</th>
<th>n</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>t</td>
<td>e</td>
<td>a</td>
</tr>
</tbody>
</table>

---

Ethernet frame header (14 bytes)

IP header 160 bits

This tells routers what IP to send the packet to.

UDP header 64 bits

(a TCP packet would have a TCP header instead here)

The packet's "contents" go here. ASCII characters are 1 byte so "mangotea" = 8 bytes 64 bits
steps to get a cat picture
from jvns.ca/cat.png

When you download an image, there are a LOT of networking moving pieces. Here are the basic steps we'll explain in the next few pages.

1. get the IP address for jvns.ca

   where is jvns.ca?
   104.28.7.94

   laptop
   DNS server

2. open a socket

   I want to start talking TCP to 104.28.7.94 port 80
   cool.py
   your program
   OS
   I'll set that up for you

3. open a TCP connection to 104.28.7.94 port 80

   laptop
   SYN
   SYN ACK
   server with cats
   ACK
   this is a "TCP hand shake" we'll explain it in the TCP section

4. request a cat

   GET /cat.png HTTP/1.1
   Host: jvns.ca
   User-Agent: curl
   HTTP request
   laptop

5. get a cat back

   HTTP/1.1 200 OK
   Content-Type: image/png
   Content-Length: 123123
   cat
   <PNG BYTES>

   server with cats

6. clean up

   ➔ close the connection maybe
   ➔ put the bytes for the PNG in a file maybe
   ➔ look at cats definitely
DNS

Step 1: get the IP address for jvns.ca

All networking happens by sending packets. To send a packet to a server on the internet, you need an IP address like 104.28.7.94

jvns.ca and google.com are domain names. DNS (the "Domain Name System") is the protocol we use to get the IP address for a domain name.

The DNS request & response are both usually UDP packets.

When you run `$ curl jvns.ca/cat.png`:

- curl calls the "getaddrinfo" function with "jvns.ca"
- getaddrinfo finds the system DNS server (like 8.8.8.8)
- getaddrinfo makes a request to that server
- IP address: obtained 104.28.7.94

Your system's default DNS server is often configured in /etc/resolv.conf.

8.8.8.8 is Google's DNS server, and lots of people use it. It's a great choice!
There are 2 kinds of **DNS servers**:

**recursive**

I can get you an IP address for ANY website by asking the right authoritative server

**authoritative**

Wanna know where jvns.ca is? Talk to ME!

When you query a recursive DNS server, here's what happens:

I have to talk to THREE authoritative DNS servers? Okay!

Where's jvns.ca? ask there!

Where's jvns.ca? ask there!

where's jvns.ca? ask there!

Recursive DNS servers usually cache DNS records. Every DNS record has a TTL ("time to live") that says how long to cache it for. You often can't force them to update their cache. You just have to wait:

I updated my DNS records, but when I visit the site in my browser I see the old version!!

20 minutes later after the recursive DNS server cache updates...

everything is great now
let's make ❤️ DNS requests ❤️

When you’re setting up DNS for a new domain, often this happens:

I want jvns.ca

I don’t know what that is yet (NXDOMAIN)

Recursive DNS server

Here’s how you can make DNS queries from the command line to understand what’s going on:

```
$ dig jvns.ca
;; ANSWER SECTION
jvns.ca 268 IN A 104.28.6.94
jvns.ca 268 IN A 104.28.7.94
;;SERVER 127.0.1.1 #53
```

- **A record:** is an IP address for one domain.
- **this record expires after 268 seconds:** this information specifies when the record will expire.
- **the DNS server I’m using:** 127.0.1.1

```
$ dig @8.8.8.8 jvns.ca
```

- **8.8.8.8 is Google’s recursive DNS server. @8.8.8.8 queries that instead of the default.**

```
$ dig +trace jvns.ca
```

- **dig +trace basically does the same thing a recursive DNS server would do to find your domain’s IP**

These are the 3 authoritative servers a recursive server has to query to get an IP for jvns.ca:
Sockets

Step 2: now that we have an IP address, the next step is to open a socket! Let's learn what that is.

Your program doesn't know how to do TCP

I don't know what "TCP" is, I just want to get a webpage.

What using sockets is like

**Step 1**: ask the OS for a socket
**Step 2**: connect the socket to an IP address and port
**Step 3**: write to the socket to send data

4 common socket types

- **TCP**: to use TCP
- **UDP**: to use UDP
- **RAW**: for ultimate power, ping uses this to send ICMP packets
- **UNIX**: to talk to programs on the same computer

When you connect with a TCP socket

OS

SYN

SYN

ACK

Server

(We'll explain this SYN ACK thing soon)

When you write to a socket

```
code.py
```

writes lots of data

splits it up into packets

→ to send it

This socket interface is great! The operating system does so much for me!
TCP: how to reliably get a cat

Step 3 in our plan is “open a TCP connection!”
Let’s learn what this “TCP” thing even is:

When you send a packet on the internet sometimes it gets lost.

TCP lets you send a stream of data reliably even if packets get lost or sent in the wrong order.

How does TCP work, you ask? WELL!

How to know what order the packets should go in:
Every packet says what range of bytes it has
Like this:

- once upon a time ➔ bytes 0-13
- magical oyster ➔ bytes 30-42
- me there was a man ➔ bytes 14-29

Then the client can assemble all the pieces into:
“Once upon a time there was a magical oyster”
The position of the first byte (0, 11, 30 in our example) is called the “sequence number”

How to deal with lost packets:
When you get TCP data, you have to acknowledge it: (ACK)

If the server doesn’t get an ACKnowledgement, it will retry sending the data.
The **TCP Handshake**

This is what a TCP header looks like:

- the "sequence number" lets you assemble packets in the right order.
- this is the **SYN** bit.

Every TCP connection starts with a "handshake". This makes sure both sides of the connection can communicate with each other.

But what do "**SYN**" and "**ACK**" mean? Well! TCP headers have 6 bit flags (SYN, ACK, RST, FIN, PSH, URG) that you can set (you can see them in the diagram). A **SYN** packet is a packet with the **SYN** flag set to 1.

When you see "connection refused" or "connection timeout" errors, that means the TCP handshake didn’t finish!

I ran **Sudo tcpdump host jvns.ca** in one and **curl jvns.ca** in another. This is some of the output:

```
localhost:51104 > 104.28.6.94:80  Flags [S]  
104.28.6.94:80 > localhost:51104  Flags [S.]  
localhost:51104 > 104.28.6.94:80  Flags [.]  
```

- **S** is for **SYN**
- **.** is for **ACK**

**jvns.ca IP address**
HTTP

Step 6: Finally, we can request cat.png!

Every time you get a webpage or see an image online, you're using HTTP.

HTTP is a pretty simple plaintext protocol. In fact, it's so simple that you can make a HTTP request by hand right now. Let's do it!!!

First, let's make a file called request.txt:

```
GET / HTTP/1.1
Host: ask.metafilter.com
User-Agent: zine
```

(put 2 newlines at the end)

Then:

```
cat request.txt | nc metafilter.com 80
```

The `nc` command ("netcat") sets up a TCP connection to metafilter.com and sends the HTTP request you wrote! The response we get back looks like:

```
200 OK
Content-Length: 120321
... headers...
... a bunch of HTML
```

HTTP/2 is the next version of HTTP. It's very different but we're out of space.
important HTTP headers

This is a HTTP request:
GET /cat.png HTTP/1.1
Host: jvns.ca
User-Agent: zine

The User-Agent: and Host: lines are called "headers". They give the webserver extra information about what webpage you want!

The Host header - my favorite!

Most servers serve lots of different websites. The Host header lets you pick the one you want!

Servers also send response headers with extra information about the response.

More useful headers:

- **User-Agent**: Lots of servers use this to check if you're using an old browser or if you're a bot.
- **Accept-Encoding**: Want to save bandwidth? Set this to "gzip" and the server might compress your response.
- **Cookie**: When you're logged into a website, your browser sends data in this header! This is how the server knows you're logged in.
... and now for even MORE

We’ve covered the basics of how to download a cat picture now! But there’s a lot more to know! Let’s talk about a few more topics.

We’ll explain a little more about networking protocols:

- what a port actually is
- how a packet is put together
- security: how SSL works
- the different networking layers
- UDP and why it’s amazing

and how packets get sent from place to place:

- how packets get sent in a local network
- and how packets get from your house to jvns.ca
- networking notation

let’s learn MORE!
Networking layers mostly correspond to different sections of a packet.

Layer 1: wires + radio waves

Layer 2: Ethernet/wifi protocol.
Your network card understands it.

Layer 3: IP addresses
Routers look at this a lot to decide where to send the packet next.

Layer 4: TCP or UDP
Where you get your ports!

Layer 5+6: don’t really exist here (though people call SSL “layer 5”)

Layer 7: HTTP and friends
Routers ignore this layer mostly. DNS queries, emails, etc. go here.
Your home router looks at layers 2+3+4

Your applications mostly worry about layer 7 but they get to tell the operating system what IP and port to use.

The network card in your computer only cares about layers 1+2.

The cool thing is that the layers are mostly independent of each other — you can change the IP address (layer 3) and not worry about layers 4+7.
What's a port?

Ports are part of the TCP and UDP protocols. (TCP port 999 and UDP port 999 are different.)

When you send a TCP message, you want to talk to a specific kind of program. This would be bad:

hi I want to get a webpage

uh I’m a mail server sorry

We want to have different kinds of programs on the same server: minecraft DNS email

So every TCP packet has a port number between 1 and 65535 on it:

I’m listening on TCP port 80

Oooh! That’s for ME!

Here’s a TCP packet with port 80 on it!

Netstat and lsof can tell you which ports are in use on your computer.

Some common ports:

DNS: UDP port 53
HTTP: TCP port 80
HTTPS: TCP port 443
SMTP: TCP port 25
Minecraft: TCP+UDP 25565
UDP
user datagram protocol

DNS sends requests using UDP. UDP is a really simple protocol. The packets look like this:

```
  UDP header
    ~ IP stuff ~
    source port  destination port
    ~ packet contents ~
  length        UDP checksum
```

"unreliable data protocol" (not what it really stands for)
When you send UDP packets, they might arrive
- out of order
- never
any packet can actually get lost, but UDP won't do anything to help you.

Packet sizes are limited
I'm gonna put 3000 characters in this packet
nope that won't fit. 1500 bytes is probably a better size.

* packet sizes are actually a super interesting topic. Search "MTU"

you need to decide how to organize your data into packets manually

ok, 623 bytes in this packet, 747 bytes in that one...

VPNs use UDP
hi I want to talk to 12.12.12.12
Ok stuff all your data into a UDP packet, send it to me, I'll pass it along.

Streaming video often uses UDP
Read http://hpbn.co/webrtc for a GREAT discussion of using UDP in a real time protocol.
Local networking
how to talk to a computer in the same room

Every computer is in a subnet. Your subnet is the list of computers you can talk to directly.

What does it mean to talk “directly” to another computer? Well, every computer on the internet has a network card with a MAC address.

When you send a packet to a computer in your subnet, you put the computer’s MAC address on it. To get the right MAC, your computer uses a protocol called ARP: (Address Resolution Protocol)

You can run `arp -na` to see the contents of the ARP table on your computer. It should look like this:

```
$ arp -na
? (192.168.1.120) at 94:53:30:91:98:c8 [ether] on wlp3s0
```
How packets get sent across the ocean

I have a packet for 104.28.7.9 but how do I get through all this INTERNET?

we got it!

your laptop → home router → intermediate routers → cable under the OCEAN → 104.28.7.94 server

When a packet arrives at a router

destination: 104.28.7.94

possible next steps, where will the packet go NEXT?!

cisco or avnu router

Routers use a protocol called **BGP** to decide what router the packet should go to next:

A packet can take a lot of different routes to get to the same destination!

The route it takes to get from A→B might be different from B→A.

Exercise: Run `traceroute google.com` to see what steps your packet takes to get to google.com.
Notation time!

People describe groups of IP addresses using CIDR notation.

**Example CIDRs**

- **CIDR**  range of IPs
  - 10.0.0.0/8    10.0.0.0/8
  - 10.9.0.0/16   10.9.0.0
  - 10.9.8.0/24   10.9.8.0

**Important examples**

- 10.0.0.0/8 and 192.168.0.0/16
- 172.16.0.0/12

are reserved for local networking.

In CIDR notation, a /n gives you $2^{32-n}$ IP addresses. So a /24 is $2^8 = 256$ IPs.

It's important to represent groups of IP addresses efficiently because routers have LOTS TO DO.

/router

is 192.168.3.2 in the subnet 192.168.0.0/16? I can do some really fast bit arithmetic and find out!

10.9.0.0 is this in binary:

00001010 00001001 00000000 00000000

first 24 bits

10.9.0.0/24 is all the IP addresses which have the same first 24 bits as 10.9.0.0!
SSL / TLS
(TLS: newer version of SSL)

When you send a packet on the internet, LOTS of people can potentially read it.

SSL encrypts your packets:

old packet

| to: 9.9.32.94:443 |
| from: 31.99.1.2:999 |

new packet

| to: 9.9.32.94:443 | 443 is the usual SSL part |
| from: 31.99.1.2:999 |

here is my secret lemon pie recipe

x 8; fae94ae
jjb43, 8b"5jkk

nobody’s gonna know the secret pie recipe NOW!

What happens when you go to https://jvns.ca:

(client) hello

(server) here’s my SSL certificate

(my half of the key exchange)

(client) here’s my half of the key exchange

(server) sweet.

(very simplified)

Once the client and server agree on a key for the session, they can encrypt all the communication they want.

To see the certificate for jvns.ca, run:

$ openssl s_client -connect jvns.ca:443 -servername jvns.ca

TLS is really complicated. You can use a tool like SSL Labs to check the security of your site.
Wireshark

Wireshark is an amazing tool for packet analysis. Here's an exercise to learn it! Run this:

```
sudo tcpdump port 80 -w http.pcap
```

While that’s running, open metafilter.com in your browser. Then press Ctrl+C to stop tcpdump. Now we have a pcap!

Open http.pcap with Wireshark.

Some questions you can try to answer:

1. What HTTP headers did your browser send to metafilter.com?  
   (hint: search `frame contains "GET"`)

2. How many packets were exchanged with metafilter.com’s server?  
   (hint: search `ip.dst == 54.1.2.3` put the IP from `ping metafilter.com` here)

Wireshark makes it easy to look at:

- IP addresses and ports
- SYNs and Acks for TCP traffic
- exactly what’s happening with DNS requests
- and so much more. It’s a great way to poke around and learn.
If you want to know more about networking:

→ make network requests! play with
  
  **dig**  **traceroute**  **tcpdump**  **ifconfig**
  **netcat**  **wireshark**  **netstat**

→ beej’s guide to network programming is a useful +funny guide to the socket API on Unix systems.
  
  → beej.us/guide/bgnet

→ High Performance Browser Networking is a ★fantastic★ and practical guide to what you need to know about networking to make fast websites.
  You can read it for free at:
  
  → hpbn.co

Thanks to Kamal Marhubi, Chris Kanich, and Ada Munroe for reviewing this!

Cover art by the amazing Liz Baillie
like this?
you can print more!
for free!
http://jvns.ca/zines

CC-BY-NC-SA  wizard industries 2017