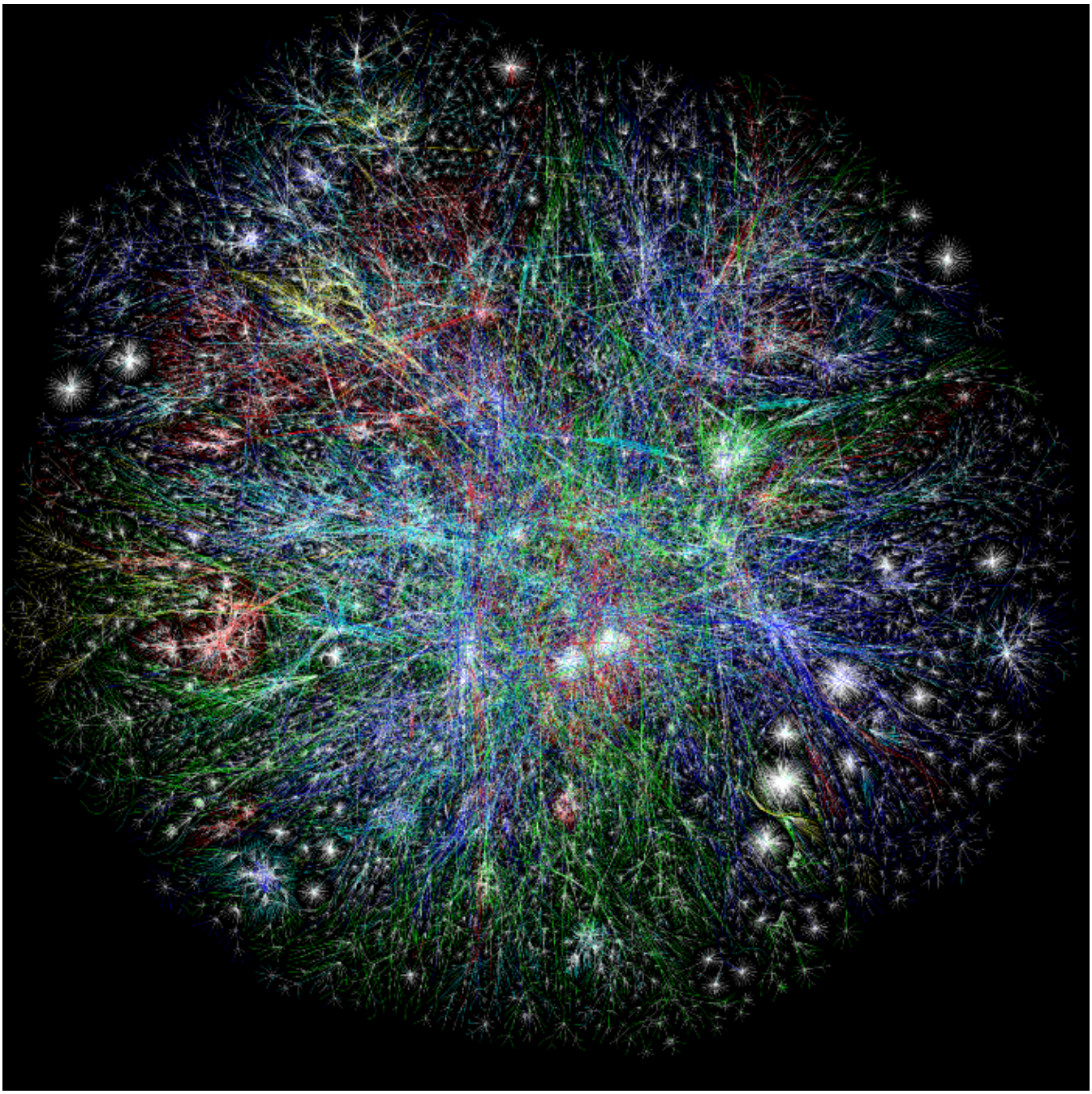
How Does The Internet Work?

TLDR: Routers Moving Packets According To Various Protocols

Steven Li
Aug 1, 2017 · 12 min read



Visualization of the Internet. Source: www.neondystopia.com

How does the Internet Work?

The Internet works through a packet routing network in accordance with the *Internet Protocol (IP)*, the *Transport Control Protocol* (TCP) and other protocols.

What's a protocol?

A protocol is a set of rules specifying how computers should communicate with each other over a network. For example, the *Transport Control Protocol* has a rule that if one computer sends data to another computer, the destination computer should let the source computer know if any data was missing so the source computer can re-send it. Or the *Internet Protocol* which specifies how computers should route information to other computers by attaching addresses onto the data it sends.

What's a packet?

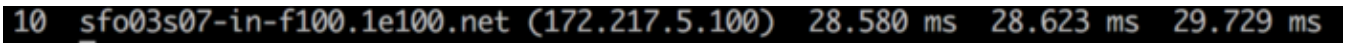
Data sent across the Internet is called a *message*. Before a *message* is sent, it is first split in many fragments called *packets*. These *packets* are sent independently of each other. The typical maximum *packet* size is between 1000 and 3000 characters. The *Internet Protocol* specifies how messages should be packetized.

What's a packet routing network?

It is a network that routes *packets* from a source computer to a destination computer. The Internet is made up of a massive network of specialized computers called *routers*. Each *router's* job is to know how to move *packets* along from their source to their destination. A *packet* will have moved through multiple *routers* during its journey.

When a *packet* moves from one *router* to the next, it's called a *hop*. You can use the command line-tool `traceroute` to see the list of hops packets take between you and a host.

```
traceroute to www.google.com (172.217.5.100), 64 hops max, 52 byte packets
 1  172.20.10.1 (172.20.10.1)  3.609 ms  4.464 ms  3.429 ms
 2  172.26.96.161 (172.26.96.161)  30.308 ms  19.126 ms  31.137 ms
 3  172.16.157.252 (172.16.157.252)  38.053 ms  32.877 ms  29.850 ms
 4  12.249.2.49 (12.249.2.49)  29.553 ms  31.334 ms  27.972 ms
 5  12.83.180.82 (12.83.180.82)  34.983 ms  30.614 ms  36.929 ms
 6  12.122.137.213 (12.122.137.213)  28.439 ms  29.795 ms  29.758 ms
 7  206.121.188.66 (206.121.188.66)  35.760 ms
    206.121.188.62 (206.121.188.62)  32.024 ms
    206.121.188.66 (206.121.188.66)  27.765 ms
 8  108.170.242.225 (108.170.242.225)  30.638 ms  21.460 ms
    108.170.243.1 (108.170.243.1)  30.210 ms
 9  108.170.236.61 (108.170.236.61)  31.080 ms
    108.170.236.63 (108.170.236.63)  110.028 ms
    108.170.236.61 (108.170.236.61)  113.216 ms
```



Command-line utility traceroute showing all the hops between my computer and google's servers

The *Internet Protocol* specifies how network *addresses* should be attached to the *packet's headers,* a designated space in the *packet* containing its meta-data. The *Internet Protocol* also specifies how the *routers* should forward the *packets* based on the *address* in the *header*.

Where did these Internet routers come from? Who owns them?

These *routers* originated in the 1960s as *ARPANET*, a military project whose goal was a computer network that was decentralized so the government could access and distribute information in the case of a catastrophic event. Since then, a number of *Internet Service Providers* (ISP) corporations have added *routers* onto these *ARPANET routers*.

There is no single owner of these Internet *routers*, but rather multiple owners: The government agencies and universities associated with *ARPANET* in the early days and *ISP* corporations like AT&T and Verizon later on.

Asking who owns the Internet is like asking who owns all the telephone lines. No one entity owns them all; many different entities own parts of them.

Do the packets always arrive in order? If not, how is the message re-assembled?

The *packets* may arrive at their destination out of order. This happens when a later *packet* finds a quicker path to the destination than an earlier one. But *packet's header* contains information about the *packet's* order relative to the entire *message*. The *Transport Control Protocol* uses this info for reconstructing the message at the destination.

Do packets always make it to their destination?

The *Internet Protocol* makes no guarantee that *packets* will always arrive at their destinations. When that happens, it's called called a *packet loss*. This typically happens when a *router* receives more *packets* it can process. It has no option other than to drop some *packets*.

However, the *Transport Control Protocol* handles *packet loss* by performing re-transmissions. It does this by having the destination computer periodically send

acknowledgement *packets* back to the source computer indicating how much of the message it has received and reconstructed. If the destination computer finds there are missing *packets*, it sends a request to the source computer asking it to resend the missing *packets*.

When two computers are communicating through the *Transport Control Protocol,* we say there is a *TCP connection* between them.

What do these Internet addresses look like?

These *addresses* are called *IP addresses* and there are two standards.

The first address standard is called *IPv4* and it looks like `212.78.1.25` . But because *IPv4* supports only 2^{32} (about 4 billion) possible addresses, the *Internet Task Force* proposed a new address standard called *IPv6*, which look like `3ffe:1893:3452:4:345:f345:f345:42fc` . *IPv6* supports 2^{128} possible addresses, allowing for much more networked devices, which will be plenty more than the as of 2017 current 8+ billion networked devices on the Internet.

As such, there is a one-to-one mapping between *IPv4* and *IPv6* addresses. Note the switch from *IPv4* to *IPv6* is still in progress and will take a long time. As of 2014, Google revealed their *IPv6* traffic was only at 3%.

How can there be over 8 billion networked devices on the Internet if there are only about 4 billion IPv4 addresses?

It's because there are *public* and *private IP addresses*. Multiple devices on a local network connected to the Internet will share the same *public IP address*. Within the local network, these devices are differentiated from each other by *private IP addresses*, typically of the form `192.168.xx` or `172.16.x.x` or `10.x.x.x` where `x` is a number between 1 and 255. These *private IP addresses* are assigned by *Dynamic Host Configuration Protocol (DHCP)*.

For example, if a laptop and a smart phone on the same local network both make a request to www.google.com, before the *packets* leave the modem, it modifies the *packet headers* and assigns one of its ports to that *packet*. When the google server responds to the requests, it sends data back to the modem at this specific port, so the modem will know whether to route the *packets* to the laptop or the smart phone.

In this sense, *IP addresses* aren't specific to a computer, but more the connection which the computer connects to the Internet with. The address that is unique to your computer is the *MAC address*, which never changes throughout the life of the computer.

This protocol of mapping *private IP addresses* to *public IP addresses* is called the *Network Address Translation* (NAT) protocol. It's what makes it possible to support 8+ billion networked devices with only 4 billion possible *IPv4* addresses.

How does the router know where to send a packet? Does it need to know where all the IP addresses are on the Internet?

Every *router* does not need to know where every *IP address* is. It only needs to know which one of its neighbors, called an *outbound link,* to route each packet to. Note that *IP Addresses* can be broken down into two parts, a n*etwork prefix* and a *host identifier*. For example, `129.42.13.69` can be broken down into

```
Network Prefix: 129.42
Host Identifier: 13.69
```

All networked devices that connect to the Internet through a single connection (ie. college campus, a business, or ISP in metro area) will all share the same *network prefix*.

Routers will send all packets of the form `129.42.*.*` to the same location. So instead of keeping track of billions of *IP addresses*, *routers* only need to keep track of less than a million *network prefix*.

But a router still needs to know a lot of network prefixes . If a new router is added to the Internet how does it know how to handle packets for all these network prefixes?

A new *router* may come with a few preconfigured routes. But if it encounters a *packet* it does not know how to route, it queries one of its neighboring *routers*. If the neighbor knows how to route the *packet*, it sends that info back to the requesting *router*. The requesting *router* will save this info for future use. In this way, a new router builds up its own *routing table*, a database of *network prefixes* to *outbound links*. If the neighboring *router* does not know, it queries its neighbors and so on.

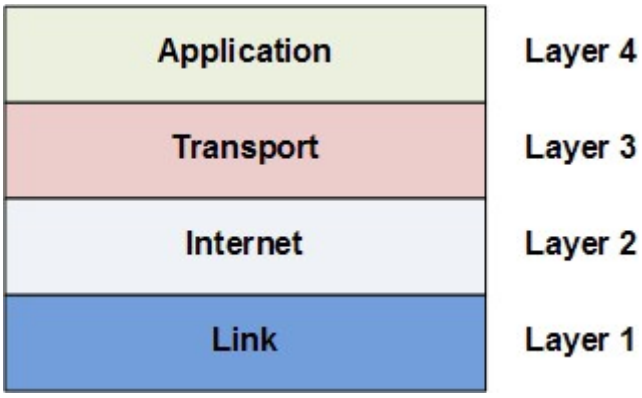
How do networked computers figure out ip addresses based on domain names?

We call looking up the *IP address* of a human-readable domain name like `www.google.com` "resolving the IP address". Computers resolve IP addresses through the *Domain Name System* (*DNS*), a decentralized database of mappings from *domain names* to *IP addresses*.

To resolve an IP address, the computer first checks its local *DNS* cache, which stores the *IP address* of web sites it has visited recently. If it can't find the *IP address* there or that *IP address* record has expired, it queries the *ISP's DNS* servers which are dedicated to resolving IP addresses. If the *ISP's DNS* servers can't find resolve the *IP address*, they query the *root name servers*, which can resolve every domain name for a given *top-level domain* . *Top-level domains* are the words to the right of the right-most period in a domain name. `.com` `.net` `.org` are some examples of *top-level domains*.

How do applications communicate over the Internet?

Like many other complex engineering projects, the Internet is broken down into smaller independent components, which work together through well-defined interfaces. These components are called the *Internet Network Layers* and they consist of *Link Layer*, *Internet Layer*, *Transport Layer*, and *Application Layer*. These are called layers because they are built on top of each other; each layer uses the capabilities of the layers beneath it without worrying about its implementation details.



Internet applications work at the *Application Layer* and don't need to worry about the details in the underlying layers. For example, an application connects to another application on the network via TCP using a construct called a *socket*, which abstracts away the gritty details of routing *packets* and re-assembling *packets* into *messages*.

What do each of these Internet layers do?

At the lowest level is the *Link Layer* which is the "physical layer" of the Internet. The *Link Layer* is concerned with transmitting data bits through some physical medium like

fiber-optic cables or wifi radio signals.

On top of the *Link Layer* is the *Internet Layer*. The *Internet Layer* is concerned with routing packets to their destinations. The *Internet Protocol* mentioned earlier lives in this layer (hence the same name). The *Internet Protocol* dynamically adjusts and reroutes *packets* based on network load or outages. Note it does not guarantee *packets* always make it to their destination, it just tries the best it can.

On top of the *Internet Layer* is the *Transport Layer*. This layer is to compensate for the fact that data can be loss in the *Internet* and *Link* layers below. The *Transport Control Protocol* mentioned earlier lives at this layer, and it works primarily to re-assembly packets into their original *messages* and also re-transmit *packets* that were loss.

The *Application Layer* sits on top. This layer uses all the layers below to handle the complex details of moving the packets across the Internet. It lets applications easily make connections with other applications on the Internet with simple abstractions like sockets. The HTTP protocol which specifies how web browsers and web servers should interact lives in the *Application Layer*. The IMAP protocol which specifies how email clients should retrieve email lives in the *Application Layer*. The FTP protocol which specifies a file-transferring protocol between file-downloading clients and file-hosting servers lives in the *Application Layer*.

What's a client versus a server?

While *clients* and *servers* are both applications that communicate over the Internet, *clients* are "closer to the user" in that they are more user-facing applications like web browsers, email clients, or smart phone apps. *Servers* are applications running on a remote computer which the *client* communicates over the Internet when it needs to.

A more formal definition is that the application that initiates a *TCP connection* is the *client*, while the application that receives the *TCP connection* is the *server*.

How can sensitive data like credit cards be transmitted securely over the Internet?

In the early days of the Internet, it was enough to ensure that the network *routers* and links are in physically secure locations. But as the Internet grew in size, more *routers* meant more points of vulnerability. Furthermore, with the advent of wireless technologies like WiFi, hackers could intercept *packets* in the air; it was not enough to

just ensure the network hardware was physically safe. The solution to this was *encryption* and *authentication* through *SSL/TLS*.

What is SSL/TLS?

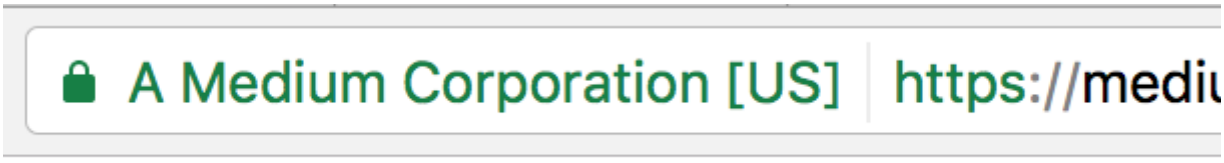
SSL stands for *Secured Sockets Layer. TLS* stands for *Transport Layer Security*. *SSL* was first developed by Netscape in 1994 but a later more secure version was devised and renamed *TLS*. We will refer to them together as *SSL/TLS*.

SSL/TLS is an optional layer that sits between the *Transport Layer* and the *Application Layer*. It allows secure Internet communication of sensitive information through *encryption* and *authentication*.

Encryption means the *client* can request that the *TCP connection* to the *server* be encrypted. This means all *messages* sent between *client* and *server* will be encrypted before breaking it into *packets*. If hackers intercept these *packets*, they would not be able to reconstruct the original *message*.

Authentication means the *client* can trust that the *server* is who it claims to be. This protects against man-in-the-middle attacks, which is when a malicious party intercepts the connection between *client* and *server* to eavesdrop and tamper with their communication.

We see *SSL* in action whenever we visit SSL-enabled websites on modern browsers. When the browser requests a web site using the `https` protocol instead of `http`, it's telling the web server it wants an *SSL* encrypted connection. If the web server supports *SSL*, a secure encrypted connection is made and we would see a lock icon next to the address bar on the browser.



The medium.com web server is SSL-enabled. The browser can connect to it over https to ensure that communication is encrypted. The browser is also confident it is communicating with a real medium.com server, and not a man-in-the-middle.

How does SSL authenticate the identity of a server and encrypt their communication?

It uses *asymmetric encryption* and *SSL certificates*.

Asymmetric encryption is an encryption scheme which uses a *public key* and a *private key*. These keys are basically just numbers derived from large primes. The *private key* is used to decrypt data and sign documents. The *public key* is used to encrypt data and verify signed documents. Unlike *symmetric encryption*, *asymmetric encryption* means the ability to encrypt does not automatically confer the ability to decrypt. It does this by using principles in a mathematical branch called number theory.

An *SSL certificate* is a digital document that consists of a *public key* assigned to a web server. These *SSL certificates* are issued to the *server* by *certificate authorities*. Operating systems, mobile devices, and browsers come with a database of some *certificate authorities* so it can verify *SSL certificates*.

When a *client* requests an SSL-encrypted connection with a *server*, the *server* sends back its *SSL certificate*. The *client* checks that the *SSL certificate*

- is issued to this *server*
- is signed by a trusted *certificate authority*
- has not expired.

The *client* then uses the *SSL certificate's public key* to encrypt a randomly generated *temporary secret key* and send it back to the *server*. Because the *server* has the corresponding *private key*, it can decrypt the client's *temporary secret key*. Now both *client* and *server* know this *temporary secret key*, so they can both use it to symmetrically encrypt the *messages* they send to each other. They will discard this *temporary secret key* after their session is over.

What happens if a hacker intercepts an SSL-encrypted session?

Suppose a hacker intercepted every *message* sent between the *client* and the *server*. The hacker sees the *SSL certificate* the *server* sends as well as the *client's* encrypted *temporary secret key*. But because the hacker doesn't have the *private key* it can't decrypt the *temporarily secret key*. And because it doesn't have the *temporary secret key*, it can't decrypt any of the *messages* between the *client* and *server*.

Summary

- The Internet started as ARPANET in the 1960s with the goal of a decentralized computer network.

- Physically, the Internet is a collection of computers moving bits to each other over wires, cables, and radio signals.
- Like many complex engineering projects, the Internet is broken up into various layers, each concerned with solving only a smaller problem. These layers connect to each other in well-defined interfaces.
- There are many protocols that define how the Internet and its applications should work at the different layers: HTTP, IMAP, SSH, TCP, UDP, IP, etc. In this sense, the Internet is as much a collection of rules for how computers and programs should behave as it is a physical network of computers.
- With the growth of the Internet, advent of WIFI, and e-commerce needs, *SSL/TLS* was developed to address security concerns.

Thanks for reading. Comments/corrections/questions are welcome. Feel free to leave them below.

Internet IP Tcp Ssl Tls