# Networking

## Making Computers Work Together!

Tim Jackman

BU Summer Challenge

July 10th, 2025

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college
- Every special topic is *deep* and we will only focus on one or two small problems in each of these lectures

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college
- Every special topic is *deep* and we will only focus on one or two small problems in each of these lectures
- Today is about Networking, the part of Computer Science dedicated to multiple computers communicating between one another

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college
- Every special topic is *deep* and we will only focus on one or two small problems in each of these lectures
- Today is about Networking, the part of Computer Science dedicated to multiple computers communicating between one another
- Today, we're going to talk a little about how we using the internet works

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college
- Every special topic is *deep* and we will only focus on one or two small problems in each of these lectures
- Today is about Networking, the part of Computer Science dedicated to multiple computers communicating between one another
- Today, we're going to talk a little about how we using the internet works
  - We won't talk about the actual *infrastructure* (e.g. wifi, cables, etc.), we'll take all that for granted (as a black-box)

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college
- Every special topic is *deep* and we will only focus on one or two small problems in each of these lectures
- Today is about Networking, the part of Computer Science dedicated to multiple computers communicating between one another
- Today, we're going to talk a little about how we using the internet works
  - We won't talk about the actual *infrastructure* (e.g. wifi, cables, etc.), we'll take all that for granted (as a black-box)
- We will answer two questions:

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college
- Every special topic is *deep* and we will only focus on one or two small problems in each of these lectures
- Today is about Networking, the part of Computer Science dedicated to multiple computers communicating between one another
- Today, we're going to talk a little about how we using the internet works
  - We won't talk about the actual *infrastructure* (e.g. wifi, cables, etc.), we'll take all that for granted (as a black-box)
- We will answer two questions:
  - How do computers achieve effective and reliable communication?

# Plan for Today

- This is our first "special topic" lecture, where we learn about an "advanced" CS topic that you might cover in college
- Every special topic is *deep* and we will only focus on one or two small problems in each of these lectures
- Today is about Networking, the part of Computer Science dedicated to multiple computers communicating between one another
- Today, we're going to talk a little about how we using the internet works
  - We won't talk about the actual *infrastructure* (e.g. wifi, cables, etc.), we'll take all that for granted (as a black-box)
- We will answer two questions:
  - How do computers achieve effective and reliable communication?
  - How can we get and use data from other computers over the internet?

How do **you** share information with someone who is far away?

How do **you** share *important* information with someone who is far away?

# Effective Communication Is Reliable

- For communication to be ueful, it needs to be reliable and two-way

# Effective Communication Is Reliable

- For communication to be ueful, it needs to be reliable and two-way
- We need to ensure that whichever computer receives a message is 1) actually *receiving* our message, 2) receiving the *correct* messages, and 3) know when our messages are done

# Effective Communication Is Reliable

- For communication to be ueful, it needs to be reliable and two-way
- We need to ensure that whichever computer receives a message is 1) actually *receiving* our message, 2) receiving the *correct* messages, and 3) know when our messages are done
- To do this, computers use the Transmission Control Protocol

# Making Communication Reliable

- In real life, how might you ensure that someone has *received* your message?

# Making Communication Reliable

- In real life, how might you ensure that someone has *received* your message?
- Two computers use an algorithm to first initially establish communication

# Making Communication Reliable

- In real life, how might you ensure that someone has *received* your message?
- Two computers use an algorithm to first initially establish communication
  - This algorithm is called a *handshake*

# Making Communication Reliable

- In real life, how might you ensure that someone has *received* your message?
- Two computers use an algorithm to first initially establish communication
  - This algorithm is called a *handshake*
- At it's core, the idea is the receiver "replies" to an initial message to establish that messages are successfully arriving

# Making Communication Reliable

- In real life, how might you ensure that someone has *received* your message?
- Two computers use an algorithm to first initially establish communication
  - This algorithm is called a *handshake*
- At it's core, the idea is the receiver "replies" to an initial message to establish that messages are successfully arriving
- Once the sender receives this message, called an ACK (acknowledgement), communication can be begin

# Getting Messages in Order

- Have you ever received text messages out of order?

# Getting Messages in Order

- Have you ever received text messages out of order?
- Effective communication requires that messages be received in order

# Getting Messages in Order

- Have you ever received text messages out of order?
- Effective communication requires that messages be received in order
- Computers add sequence numbers to their messages

# Getting Messages in Order

- Have you ever received text messages out of order?
- Effective communication requires that messages be received in order
- Computers add sequence numbers to their messages
- Computers can be communicating with *many* computers at once which can lead to *chaos*

# Getting Messages in Order

- Have you ever received text messages out of order?
- Effective communication requires that messages be received in order
- Computers add sequence numbers to their messages
- Computers can be communicating with *many* computers at once which can lead to *chaos*
- Messages can sometimes take a *long* time to get from computer to computer

# Getting Messages in Order

- Have you ever received text messages out of order?
- Effective communication requires that messages be received in order
- Computers add sequence numbers to their messages
- Computers can be communicating with *many* computers at once which can lead to *chaos*
- Messages can sometimes take a *long* time to get from computer to computer
  - To minimize confusion, omputers start each chat with different sequence numbers

# The Three-Way Handshake

- In TCP, the initiator sends an initial message called a SYN (synchronize):

# The Three-Way Handshake

- In TCP, the initiator sends an initial message called a SYN (synchronize):
  - "Hey, I want to communicate with you and my messages will be numbered starting at 193424"

# The Three-Way Handshake

- In TCP, the initiator sends an initial message called a SYN (synchronize):
  - "Hey, I want to communicate with you and my messages will be numbered starting at 193424"
- The other computer sends back a message called an ACK-SYN:

# The Three-Way Handshake

- In TCP, the initiator sends an initial message called a SYN (synchronize):
  - "Hey, I want to communicate with you and my messages will be numbered starting at 193424"
- The other computer sends back a message called an ACK-SYN:
  - "I got your message, I'll reply using messages starting at 137210"

# The Three-Way Handshake

- In TCP, the initiator sends an initial message called a SYN (synchronize):
  - "Hey, I want to communicate with you and my messages will be numbered starting at 193424"
- The other computer sends back a message called an ACK-SYN:
  - "I got your message, I'll reply using messages starting at 137210"
- The initiate sends back an ACK:

# The Three-Way Handshake

- In TCP, the initiator sends an initial message called a SYN (synchronize):
  - "Hey, I want to communicate with you and my messages will be numbered starting at 193424"
- The other computer sends back a message called an ACK-SYN:
  - "I got your message, I'll reply using messages starting at 137210"
- The initiate sends back an ACK:
  - "I got your message, let us start sending data back and forth!"

# Did I Hear That Right?

- Imagine trying to talk to someone in a noisy environment, like a club. What might go wrong?

# Did I Hear That Right?

- Imagine trying to talk to someone in a noisy environment, like a club. What might go wrong?
- Errors can happen while a message is being transported across phyiscal wires, wifi, etc.

# Did I Hear That Right?

- Imagine trying to talk to someone in a noisy environment, like a club. What might go wrong?
- Errors can happen while a message is being transported across phyiscal wires, wifi, etc.
- Computers add a *checksum*: the message is put through a function to convert it to a number which is added to

# Did I Hear That Right?

- Imagine trying to talk to someone in a noisy environment, like a club. What might go wrong?
- Errors can happen while a message is being transported across phyiscal wires, wifi, etc.
- Computers add a *checksum*: the message is put through a function to convert it to a number which is added to
- The receiver can compute the checksum and check that it matches

# Stop Talking To Me!

- After establishing a connection, both computers are actively spending resources listening for new messages from the other

# Stop Talking To Me!

- After establishing a connection, both computers are actively spending resources listening for new messages from the other
- If one of them decides it's time to stop talking, how can the other one know?

# Stop Talking To Me!

- After establishing a connection, both computers are actively spending resources listening for new messages from the other
- If one of them decides it's time to stop talking, how can the other one know?
- Someone can send a FIN (final) message...

# Stop Talking To Me!

- After establishing a connection, both computers are actively spending resources listening for new messages from the other
- If one of them decides it's time to stop talking, how can the other one know?
- Someone can send a FIN (final) message...
- But what if THAT message gets lost?

# The Four-way Handshake

- The closer sends a FIN message

# The Four-way Handshake

- The closer sends a FIN message
  - "Hey, I'm not going to send you anymore messages, thanks!"

# The Four-way Handshake

- The closer sends a FIN message
  - "Hey, I'm not going to send you anymore messages, thanks!"
- The other computer replies with an ACK

# The Four-way Handshake

- The closer sends a FIN message
  - "Hey, I'm not going to send you anymore messages, thanks!"
- The other computer replies with an ACK
  - "Ok, I won't look for any more messages from you"

# The Four-way Handshake

- The closer sends a FIN message
  - "Hey, I'm not going to send you anymore messages, thanks!"
- The other computer replies with an ACK
  - "Ok, I won't look for any more messages from you"
- The other computer sends their own FIN message

# The Four-way Handshake

- The closer sends a FIN message
  - "Hey, I'm not going to send you anymore messages, thanks!"
- The other computer replies with an ACK
  - "Ok, I won't look for any more messages from you"
- The other computer sends their own FIN message
  - "Okay, now I also am done sending you messages"

# The Four-way Handshake

- The closer sends a FIN message
  - "Hey, I'm not going to send you anymore messages, thanks!"
- The other computer replies with an ACK
  - "Ok, I won't look for any more messages from you"
- The other computer sends their own FIN message
  - "Okay, now I also am done sending you messages"
- The closer can then sends their own ACK back

# More Protocols

- There are other network protocol that also handle the *sending* aspect of sending messages:

# More Protocols

- There are other network protocol that also handle the *sending* aspect of sending messages:
  - User Datagram Protocol (UDP): faster messages but not guarantee everything is received in the right order

# More Protocols

- There are other network protocol that also handle the *sending* aspect of sending messages:
  - User Datagram Protocol (UDP): faster messages but not guarantee everything is received in the right order
- But what are the messages they are sending?

# More Protocols

- There are other network protocol that also handle the *sending* aspect of sending messages:
  - User Datagram Protocol (UDP): faster messages but not guarantee everything is received in the right order
- But what are the messages they are sending?
- This depends on what *applications* (programs) on the two computers are trying to communicate.

# More Protocols

- There are other network protocol that also handle the *sending* aspect of sending messages:
  - User Datagram Protocol (UDP): faster messages but not guarantee everything is received in the right order
- But what are the messages they are sending?
- This depends on what *applications* (programs) on the two computers are trying to communicate.
  - These have their own protocols

# How Does The Web Work?

- When you visit a webpage (like the course website), you computer communicates with a server and receives the webpage as data

# How Does The Web Work?

- When you visit a webpage (like the course website), you computer communicates with a server and receives the webpage as data
- These messages follow the HyperText Transfer Protocol (HTTP)

# How Does The Web Work?

- When you visit a webpage (like the course website), you computer communicates with a server and receives the webpage as data
- These messages follow the HyperText Transfer Protocol (HTTP)
- Your computer sends a GET message, requesting the content of the page:

# How Does The Web Work?

- When you visit a webpage (like the course website), you computer communicates with a server and receives the webpage as data
- These messages follow the HyperText Transfer Protocol (HTTP)
- Your computer sends a GET message, requesting the content of the page:

```
GET /tjackman/busc.html HTTP/1.1
host:  cs-people.bu.edu
User-Agent:  Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

# How Does The Web Work?

- When you visit a webpage (like the course website), you computer communicates with a server and receives the webpage as data
- These messages follow the HyperText Transfer Protocol (HTTP)
- Your computer sends a GET message, requesting the content of the page:

```
GET /tjackman/busc.html HTTP/1.1
host:  cs-people.bu.edu
User-Agent:  Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

- The first line is the request itself

# How Does The Web Work?

- When you visit a webpage (like the course website), you computer communicates with a server and receives the webpage as data
- These messages follow the HyperText Transfer Protocol (HTTP)
- Your computer sends a GET message, requesting the content of the page:

```
GET /tjackman/busc.html HTTP/1.1
host:  cs-people.bu.edu
User-Agent:  Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

- The first line is the request itself
- The rest of the messages are called *headers*, and they're extra information that the receiver might use to process the message

# How Does The Web Work?

- When you visit a webpage (like the course website), you computer communicates with a server and receives the webpage as data
- These messages follow the HyperText Transfer Protocol (HTTP)
- Your computer sends a GET message, requesting the content of the page:
- The first line is the request itself
- The rest of the messages are called *headers*, and they're extra information that the receiver might use to process the message
- Other kinds of HTTP message actions include: POST, PUT, DELETE

# A Server Reply

```
HTTP/1.1 200 OK
Date:  Tue, 08 Jul 2025 19:05:00 GMT
Server:  Apache/2.4.41 (Ubuntu)
Content-Type:  text/html; charset=UTF-8
Content-Length:  800

[the actual content of the website]
```

# HTML

- Your browser receives an html file and then uses it to generate the visual webpage

# HTML

- Your browser receives an html file and then uses it to generate the visual webpage
- HTML stands for Hypertext Markup Language

# HTML

- Your browser receives an html file and then uses it to generate the visual webpage
- HTML stands for Hypertext Markup Language
  - A markup language is a way to encode a document's content and structure

# HTML

- Your browser receives an html file and then uses it to generate the visual webpage
- HTML stands for Hypertext Markup Language
  - A markup language is a way to encode a document's content and structure
  - HTML uses tags to represent the structure of the document:

# HTML

- Your browser receives an html file and then uses it to generate the visual webpage
- HTML stands for Hypertext Markup Language
  - A markup language is a way to encode a document's content and structure
  - HTML uses tags to represent the structure of the document:

```
<h1> BU Summer Challenge </h1>
```

# HTML

- Your browser receives an html file and then uses it to generate the visual webpage
- HTML stands for Hypertext Markup Language
  - A markup language is a way to encode a document's content and structure
  - HTML uses tags to represent the structure of the document:

```
<h1> BU Summer Challenge </h1>
```

- The html often only gives limited instructions on how to style the page, rather focusing on just what content and how it should be arranged

# Cascading Style Sheets

- To *style* a webpage, typically a style.css file is also sent

# Cascading Style Sheets

- To *style* a webpage, typically a style.css file is also sent
- The .css file tells your browser more about how it should *look*

# Cascading Style Sheets

- To *style* a webpage, typically a style.css file is also sent
- The .css file tells your browser more about how it should *look*

```
.h1 {
color:  purple;
}
```

# JavaScript

- Neither HTML or CSS are programming languages and interaction is limited without some way to code

# JavaScript

- Neither HTML or CSS are programming languages and interaction is limited without some way to code
- Web development uses JavaScript

# JavaScript

- Neither HTML or CSS are programming languages and interaction is limited without some way to code
- Web development uses JavaScript
- JavaScript has nothing to do with Java, it's named that way for purely marketing

# JavaScript

- Neither HTML or CSS are programming languages and interaction is limited without some way to code
- Web development uses JavaScript
- JavaScript has nothing to do with Java, it's named that way for purely marketing
    - Java is popular, if we call it JavaScript we get that brand synergy!!

# JavaScript

- Neither HTML or CSS are programming languages and interaction is limited without some way to code
- Web development uses JavaScript
- JavaScript has nothing to do with Java, it's named that way for purely marketing
  - Java is popular, if we call it JavaScript we get that brand synergy!!
- Java loves to throw errors but websites should avoid crashing at all costs!

# APIs

- Communicating with other computers is useful: other people have useful data or functions

# APIs

- Communicating with other computers is useful: other people have useful data or functions
- We can request data from servers over the internet using RESTful API

# APIs

- Communicating with other computers is useful: other people have useful data or functions
- We can request data from servers over the internet using RESTful API
    - An API (application programming interface) is software that provides tools to other pieces of software

# APIs

- Communicating with other computers is useful: other people have useful data or functions
- We can request data from servers over the internet using RESTful API
  - An API (application programming interface) is software that provides tools to other pieces of software
  - Using a specific API depends on it's implementation

# APIs

- Communicating with other computers is useful: other people have useful data or functions
- We can request data from servers over the internet using RESTful API
  - An API (application programming interface) is software that provides tools to other pieces of software
  - Using a specific API depends on it's implementation
  - RESTful APIs use a standardized format consistent with HTTP messages

# APIs

- Communicating with other computers is useful: other people have useful data or functions
- We can request data from servers over the internet using RESTful API
  - An API (application programming interface) is software that provides tools to other pieces of software
  - Using a specific API depends on it's implementation
  - RESTful APIs use a standardized format consistent with HTTP messages
- RESTful APIs return the requested data back, usually in a format called JSON

# JSON

- JSON stands for JavaScript Object Notation, and it's a data file format

# JSON

- JSON stands for JavaScript Object Notation, and it's a data file format

```
{
  "first_name":  "Tim",
  "last_name":  "Jackman",
  "is_alive":  true,
  "age":  27,
  "address":  {
    "city":  "Boston",
    "state":  "MA",
    "postal_code":  "02130"
  }
}
```

- JSON is an example of a *dictionary* data structure

- JSON is an example of a *dictionary* data structure
- It consists of a list of *key-value* pairs, data is associated with a specific *key*

- JSON is an example of a *dictionary* data structure
- It consists of a list of *key-value* pairs, data is associated with a specific *key*
    - Like how a dictionary associates words (keys) to their definitions (values)

- JSON is an example of a *dictionary* data structure
- It consists of a list of *key-value* pairs, data is associated with a specific *key*
    - Like how a dictionary associates words (keys) to their definitions (values)
- The value itself can be another dictionary (like how an array can contain an array)

- JSON is an example of a *dictionary* data structure
- It consists of a list of *key-value* pairs, data is associated with a specific *key*
    - Like how a dictionary associates words (keys) to their definitions (values)
- The value itself can be another dictionary (like how an array can contain an array)
- APIs may return JSON files with requested data stored in specified keys

- JSON is an example of a *dictionary* data structure
- It consists of a list of *key-value* pairs, data is associated with a specific *key*
  - Like how a dictionary associates words (keys) to their definitions (values)
- The value itself can be another dictionary (like how an array can contain an array)
- APIs may return JSON files with requested data stored in specified keys
  - You might call a Weather API with a specific city, and it might return a Weather Data JSON file that stores the temperature under the key ``temp''.

- JSON is an example of a *dictionary* data structure
- It consists of a list of *key-value* pairs, data is associated with a specific *key*
  - Like how a dictionary associates words (keys) to their definitions (values)
- The value itself can be another dictionary (like how an array can contain an array)
- APIs may return JSON files with requested data stored in specified keys
  - You might call a Weather API with a specific city, and it might return a Weather Data JSON file that stores the temperature under the key ``temp''.
- While you could parse JSON files with a Scanner, there are tools to make parsing easier