Snowcast Warmup

GEARUP

A mental model

Snowcast: an "Internet radio station"

- Server: has several "stations" that serve audio data to clients
- Clients: connect to server, ask for a station, receive audio data
  - (Actually two programs, more on this later)

"Radio station"

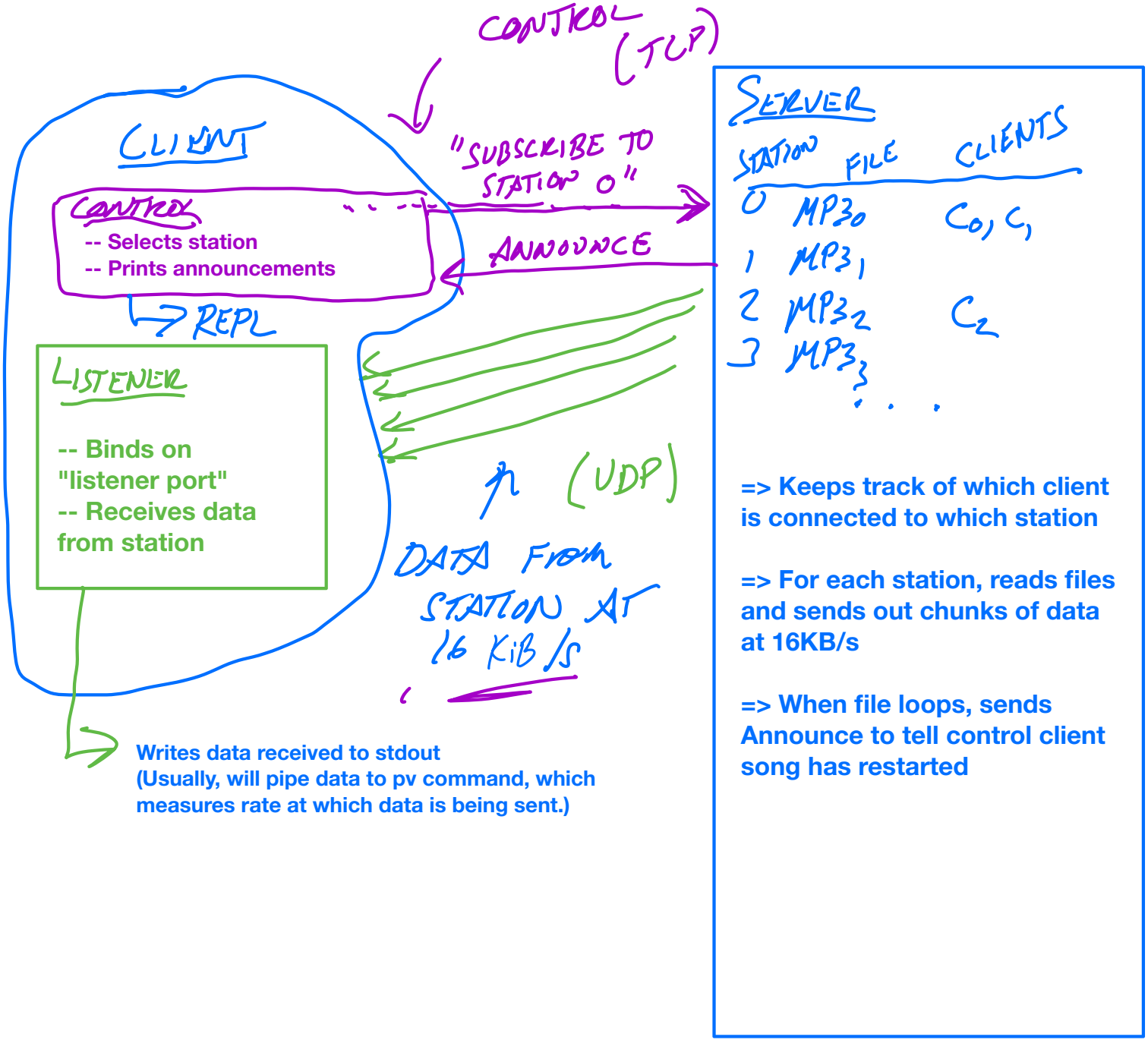
- Server is always "playing" music, even if no one is listening
- Everyone gets the same data at the same time

Not like Spotify. More like Twitch., but for audio.

Goals

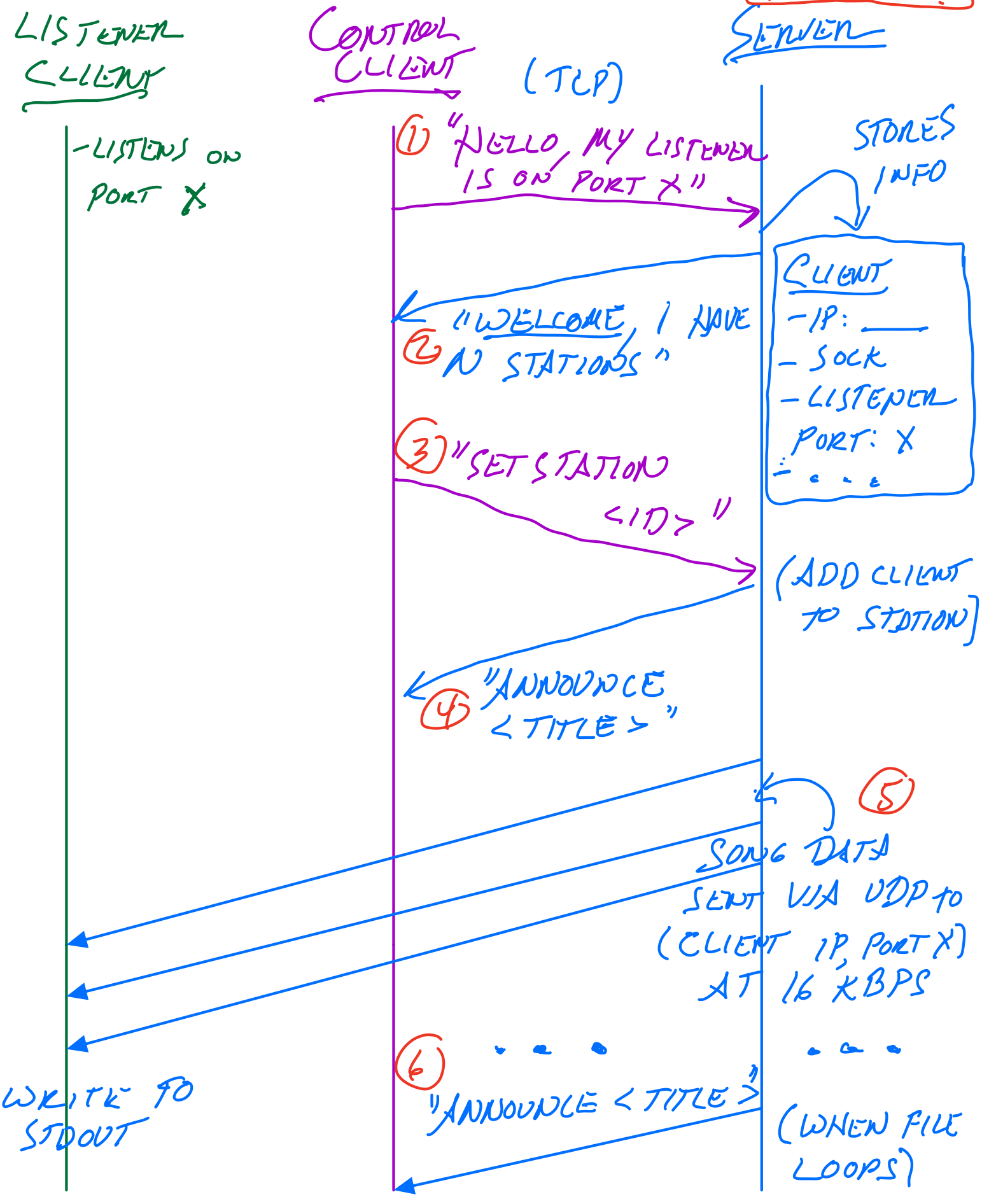
- Intro to socket programming
- Chance to become more comfortable with socket programming, in any language
- Learn how to implement a protocol, design a robust server

Thinking about the architecture for Snowcast



SNOWCAST: PROTOCOL OVERVIEW

ONLY NEED ① + ② FOR MILESTONE!



WHEN BUILDING A NETWORKED APPLICATION

- WHAT STATE DOES THE SERVER STORE?
  SHARED DATA! HOW TO PROTECT IT?
- WHAT STATE DOES THE SERVER NEED PER CLIENT?
- HOW TO HANDLE MULTIPLE CLIENTS?

EG. FOR THE GUESSING GAME EXAMPLE...

- NUMBER WE'RE TRYING TO GUESS
- TOTAL NUMBER OF GUESSES
- ...

- CLIENT SOCKET
MAYBE:
  - UNIQUE ID, GUESS HISTORY...

- ONE GOROUTINE ON SERVER FOR EACH CLIENT
- SERVER KEEPS LIST OF CLIENTS TO NOTIFY WHEN GAME RESETS

FOR YOUR DESIGN DOCUMENT, THINK ABOUT HOW YOU WOULD DO THIS FOR SNOWCAST!

Thinking about your system's design

When building a networked application, consider the following

E.g. for the guessing game example...

=> What state does the server store?

- NUMBER WE'RE TRYING TO GUESS
- TOTAL NUMBER OF GUESSES
- ...

SHARED DATA! HOW TO PROTECT IT??

What state does the server need per client?

- CLIENT SOCKET

MAYBE:

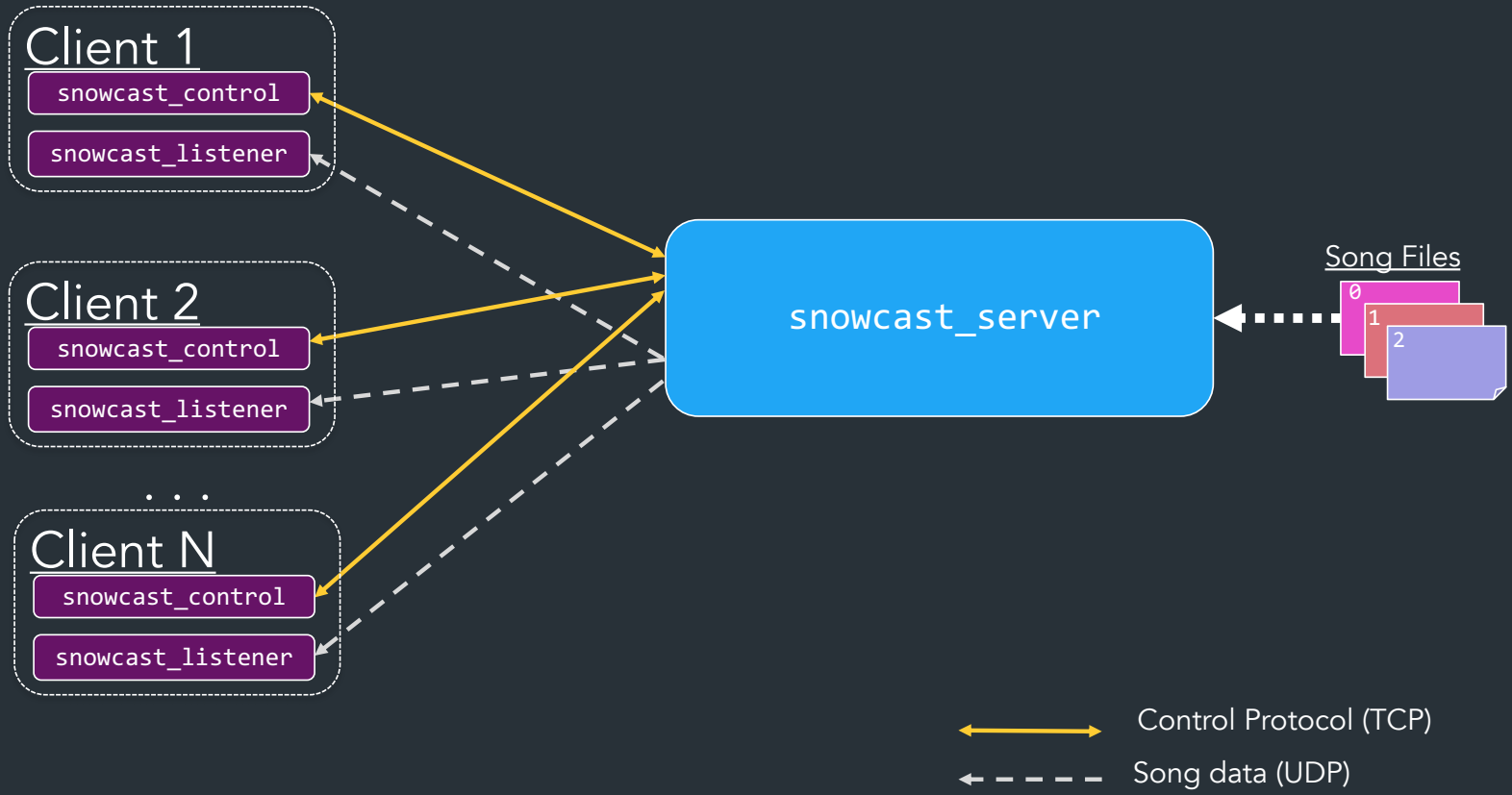
- UNIQUE ID, GUESS HISTORY...

How to handle multiple clients?

- ONE GOROUTINE ON SERVER FOR EACH CLIENT
- SERVER KEEPS LIST OF CLIENTS TO NOTIFY WHEN GAME RESETS

FOR YOUR DESIGN DOCUMENT, THINK ABOUT HOW YOU WOULD DO THIS FOR SHOWCASE!

Concretely: how Snowcast works



Demo

What you will implement

You will implement all three programs

- `snowcast_server`: the server
- The client (two programs):
  - `snowcast_control`: Control client
  - `snowcast_listener`: Listener client

- We give you the specification for the protocol, and how the programs should behave
- You decide *how* to implement them

Need to be able to interoperate with our reference version (and tests)!

Roadmap

- Setup <--- you are here
- Milestone: Sending initial messages (welcome/hello)
- Building your server (where to put state, etc.)
  - Subscribing to stations
  - Listing clients
- Listener + streaming
- Announcements while streaming
- Error handling/timeouts/etc

What we will test

- Your programs must interoperate with ours (ie, speak the same protocol)
- Don’t need to stream music per se—we just measure for a streaming rate of ~16KiB/s

- Some server design guidelines (see spec)
  - Must support multiple clients, protect shared data
  - Reasonable error handling (+timeouts)

Languages

You can work in any of the following languages:

- Go
- C/C++
- Rust

We recommend Go, even if it is new to you.

The time to learn go will be less than the time you'd otherwise spend debugging stuff in C.

Essential Resources

- The handout: high-level overview, grading details
- Setup guide and warmup: Warmup tutorial, Implementation-level resources, FAQs
- The specification: all the details on the Snowcast protocol
  - Implementation spec: how your programs should behave (arguments, etc.)
  - Protocol spec: how your messages should be formatted, etc.
- Lecture examples: don't copy, but look at them side by side
- Reference version: complete working version you can try
- Test suites: you can run our tests!

See the FAQ/Reading list post on Ed!

Implementation resources

- Language resources on website
- Setup/warmup guide: LOTS of tutorials re: testing, debugging, common things that go wrong...
- Some utilities for C (linked list, hash table)

Libraries

- You can use libraries you find online (go packages, rust crates, etc), as long as it doesn't trivialize the assignment
- You must manually parse packets on your own
- Easy examples: argument parsing, logging, …

If you're unsure (especially networking-related stuff), please ask!

How to get started

Dev environment

- You should be working in the container environment
- Be sure to clone your repo where you can access it from the container

```
- ...
|--DEV-ENVIRONMENT
| |--docker/
| |--home/
| | |--snowcast-yourname/
| |--run-container
| |-- ...
...
```

How to start

- Watch Lecture 3, if you have not done so already

- Follow the steps in the warmup, which guides you through building/sending messages

Take a look at the "guessing game" example code (from lecture 3, also full version) for more details

How to start a go project

For details, please see recording and sockets example code from Lecture 3

```
-snowcast-yourname
|
|-cmd/
|   |
|   - some-program/
|   |    - main.go
|   - another-program
|   |    - main.go
|-pkg/
|   - somelib/
|   |    - somelib.go
```

For each program you want to build, create a dir and main.go under cmd/

Place shared code here, can be imported by any program in cmd

The reference implementation

A complete implementation of Snowcast you can run

- Try it out! See how your program should operate!

- Your implementation should act like this one

Wireshark

This is the best way to test early-on.

Ask yourself:

- “Does this packet match the specification?”
- “Any warnings from wireshark?”

The tester and autograder

We have provided a test suite with all of our tests

- Check your work as you go, see it in Wireshark
- We'll have the same test suite available in gradescope soon

  ⇒ Gradescope is generally more authoritative/reliable than your own system

- Want to know what a test does? See the list of tests!

Note: please test your project manually first (similar to the demo in this recording) before using the tester, or as soon as you have failing tests
=> This is often the best way to understand what's happening.

If you are failing tests

- Run manually => observe output in Wireshark
- Run the test on its own (see setup guide)

- Is Gradescope any different?

If you are convinced you should be passing a test, but it's failing, note it in your readme—we'll consider this when grading.

Recommendations

- Start early, please ask questions
- Use tests/wireshark to help you debug!