CS 521: Systems Programming

Socket Programming

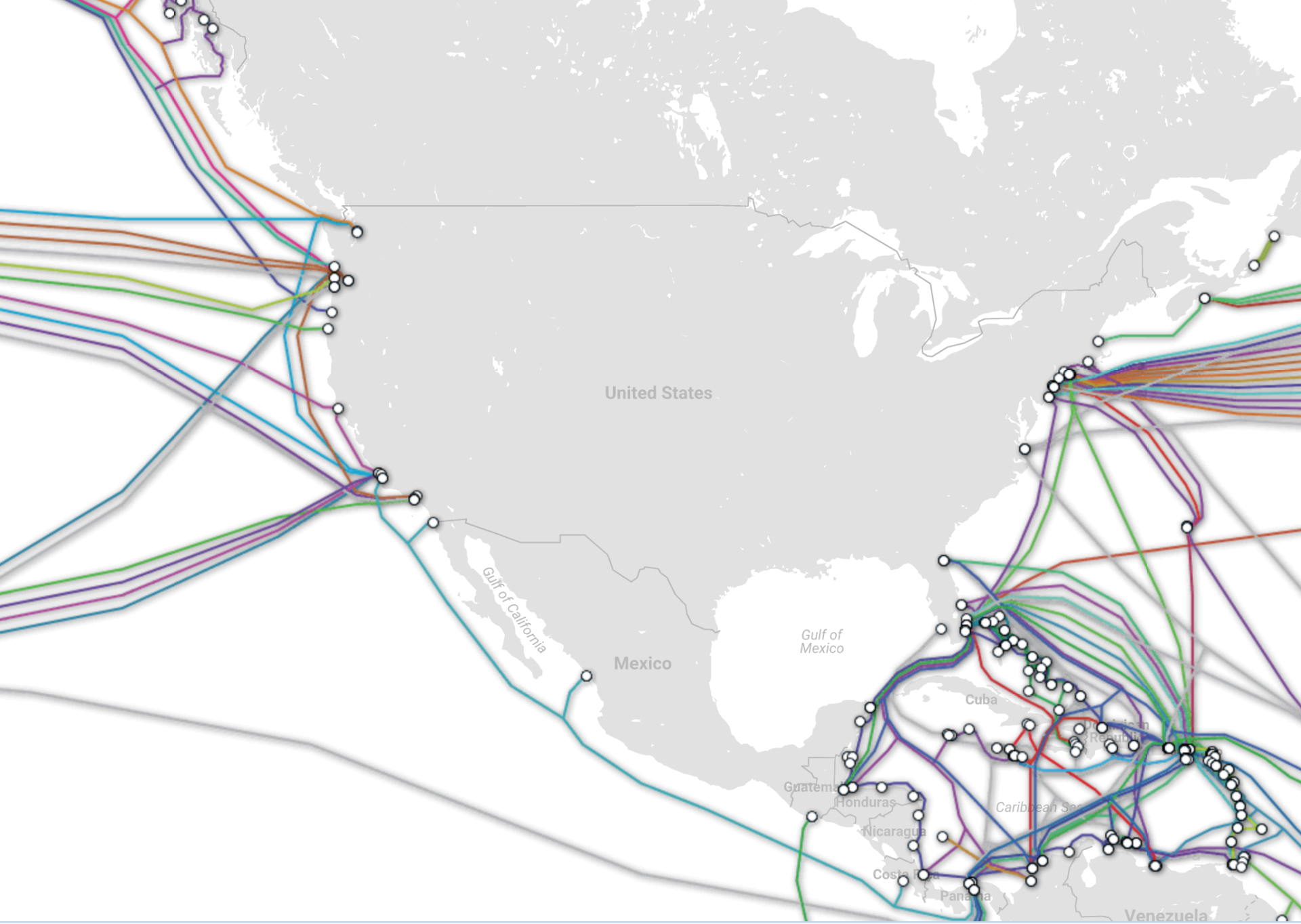
Lecture 22

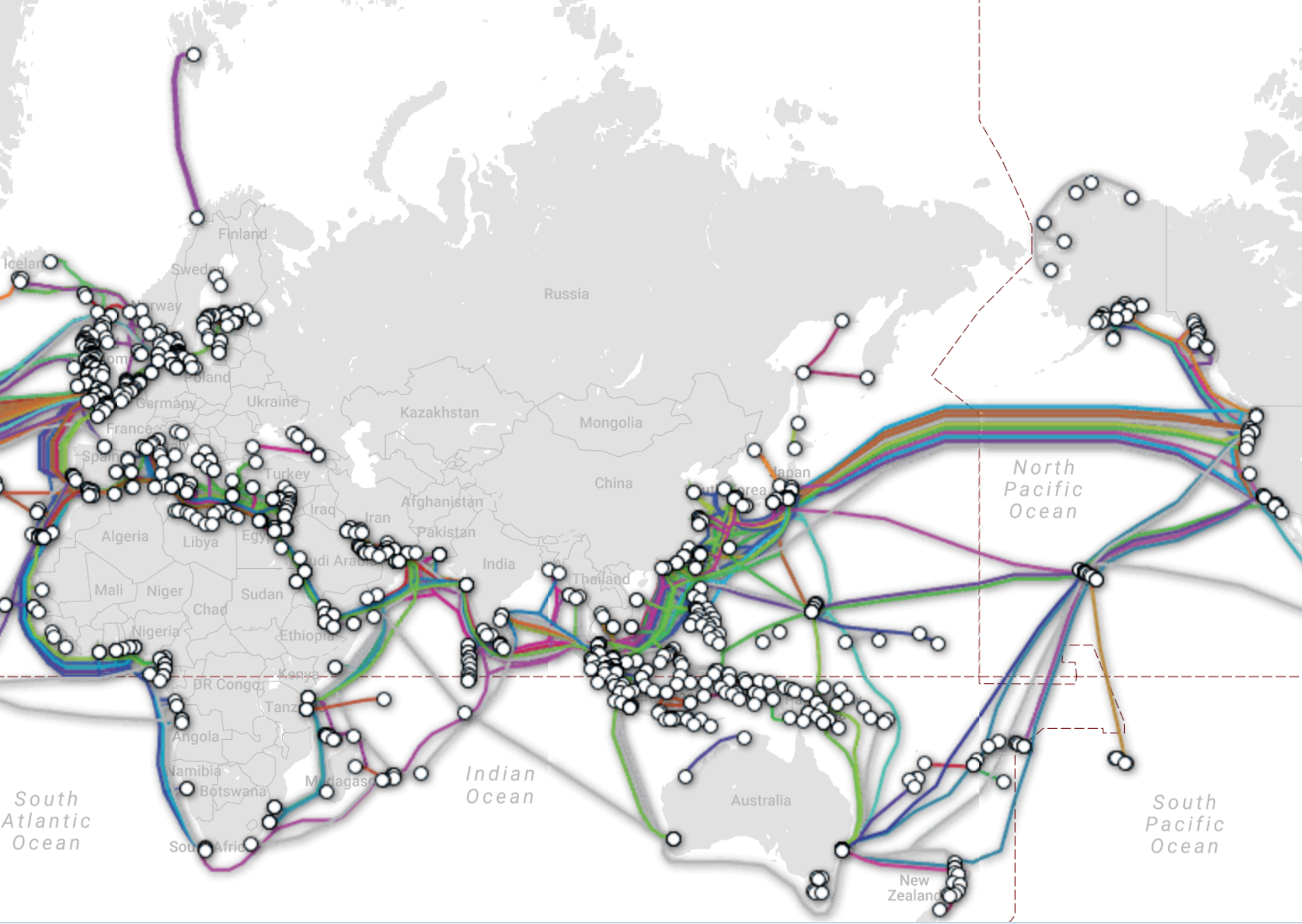
Networking

- Nowadays, you can't do anything without a network connection
- Like, including using your VMs for class…

Network Terminology

- Host – computer/device connected to a network. Communicate via **packets**
  - How do they connect? https://www.submarinecablemap.com
- Packet – a unit of data, generally represented as a sequence of bytes
- Router – device that forwards packets through the network until they reach their destination host
- Protocol – defines how packets are laid out


United States
Mexico
Gulf of California
Gulf of Mexico
Cuba
Honduras
Nicaragua
Venezuela


Russia
Finland
Sweden
Norway
Germany
Ukraine
France
Spain
Turkey
Kazakhstan
Mongolia
China
Iraq
Iran
Afghanistan
Pakistan
India
Thailand
Japan
Algeria
Libya
Egypt
Mali
Niger
Sudan
Chad
Nigeria
Ethiopia
Kenya
DR Congo
Tanzania
Angola
Namibia
Botswana
Madagascar
South Africa
Australia
New Zealand
North Pacific Ocean
South Pacific Ocean
Indian Ocean
South Atlantic Ocean

TCP

- We use the **Internet Protocol (IP)** Suite for a majority of our communications
- For reliable delivery, we use the **Transmission Control Protocol (TCP)**
  - Modeled as a *stream* of bytes
  - Packets will reach their destination (eventually…) and the contents are verified
    - Retransmit when a failure/corruption occurs
  - Packets are received **in order**

UDP

- **User Datagram Protocol (UDP)** on the other hand, is *connectionless*
  - Rather than a stream of bytes, we deal with discrete messages
- No acknowledgment or retransmission
- Ordering is not guaranteed
- Used for games, video streaming, and other applications where delivery needn't be guaranteed

Sockets

- The standard API for network communication is **sockets**
  - Introduced in 1983, 4.2 BSD UNIX
- Sockets follow the Unix philosophy
  - everything is a file!
- When we open a connection, we get a file descriptor that we can `read()` and `write()` from
  - Just like a file… or pipe

Ports

- Each outgoing and incoming socket connection is assigned a **port number**
  - 16-bit unsigned integer (max: 65535)
- Want to talk to a web server via (HTTP)? Port 80!
- How about SSH? Port 22!
- Processes **bind** to these ports and listen on them
- When initiating an outgoing connection (e.g., from your browser) you can assign any available port

Clients and Servers

- Clients connect to a remote host for some type of service
  - ssh, http, etc.
- A server listens for these incoming connections and responds to them

Basic Server Workflow

1. Create a socket
2. Bind to a port
3. Listen for connections
4. Accept incoming connections:
   - This wraps a usual Unix file descriptor internally
     - Note: TCP is bidirectional: you can send and receive from the same FD

Basic Client Workflow

1. Create a socket
2. Connect to the remote host
3. Write/read data to/from the socket
   - Again, TCP is bidirectional: you can send and receive from the same connection

TCP Weirdness

- The first unintuitive thing about (TCP) sockets is there is no concept of a "message"
- Instead, everything gets read/written as byte arrays (streams of bytes)
  - Not all the bytes will come in at the same time, although order is guaranteed with TCP
- We generally need to use fixed-size messages or prefix them with a length to know what to expect

Simple Messaging [1/2]

- A common message format:
  - `[ MESSAGE SIZE ][ MESSAGE PAYLOAD ]`
- Once you've unpacked the message payload, it can contain more fields
  - For example: message **type**, version number, flags, etc.
- This allows for a layered approach:
  - Network code
  - Message creation code
  - Pass through a chain of handlers

Simple Messaging [2/2]

- If you don't need advanced features, size-prefixed messages work well
- Exceptions:
  - You'd like to avoid reading the entire message before you start processing it
  - You don't even need to process the whole message (perhaps you are forwarding it somewhere else)
- Network **wire formats** have a huge range of features and complexity

Serialization

- **Serialization** transforms an object, structure, or application state into a format for transmission
  - (and often storage to disk)
- Most common: **binary** formats
  - Better performance
- When you receive a serialized message, transforming it back into its original representation is called **deserialization**

Trying it out

- Let's build a simple chat program…