Lab 0 logistics: run `git pull` for small updates

- **Fundamental notion of networking: Three key concepts:**
  - **Datagrams**
    - a short piece of data (aka a “packet”) that includes:
      - a “destination” address with meaning in some context
      - other data (the “payload”)
    - the service abstraction is:
      - the network will make “best efforts” to deliver the datagram to its ultimate destination (what’s marked in the datagram)
      - **no matter where the datagram is currently!**
        - the “destination” address is sufficient to find the **ultimate** destination. It’s not just the “next hop” – it’s the final destination of that datagram.
        - Any part of the network knows how to deal with it to get it closer to its final destination.
  - **Encapsulation**
    - assigning an object to be the **payload of another object**, where the outer object is **oblivious** to the meaning or contents of its payload
    - Examples:
      - The postal service will carry a package you give it [within certain externally visible specifications, e.g. weight and dimensions], without looking inside the contents. The package is opened, and the contents have meaning and are viewed **only by the recipient**.
      - A cargo ship will carry any container you give it, without looking inside the contents [with certain limits. The container is **only opened by the recipient**.
      - The Internet will carry any datagram you give it [within certain externally visible specifications, e.g. length], without looking inside the payload. The payload of an Internet datagram has meaning **only to the endpoints**.
  - **Multiplexing**
    - **sharing** a resource by dispatching according to a **runtime identifier**
    - Examples:
      - the IP address allows different endpoints to share a single communications link. Each router hands off the datagram to a different next hop ***based on the contents of this field***.
      - The “protocol field” of the Internet datagram allows different encapsulated protocols to share the resource of host-to-host Internet datagrams. The kernel hands off the datagram payload to a different internal kernel program (e.g. ping, UDP) ***based on the contents of this field.***
      - The “destination port number” field of the user datagram allows different processes/programs to share the resource of host-to-host

Internet datagrams. The UDP program hands off the payload to a different user program (e.g. Chrome, traceroute) ***based on the contents of this field.***

- **Datagrams**: Service *interface* or *abstraction* **(**ByteStreams are also examples of service abstraction)
    - Datagram is a postcard
        - To: some entity (name of entity with meaning in some context) (*Kelly*)
        - Payload (anything else)
    - Drop it somewhere that the abstraction is meaningful
    - Networks' service contract: send it at "best effort"
- "best effort": what could go wrong
    - intercepted (somebody else could get it) (with or without Kelly)
    - Kelly alone gets it
    - Kelly gets a modified payload
    - Kelly never gets it
    - Kelly gets it after a looooooong time
    - Kelly gets multiple copies
    - Kelly get it after another datagram sent later
    - *Things could get wrong, but this "best effort" is the only contract that networks provide us with*
- Experiment to send things to Bryan and Joanne
    - What Bryan receives: 1: Hello 7: CS 144! 5:seco 4:to the 3:Welcome 5:seco 5:nd lecture of 8:!!!!
    - And Jonanne was not found.
    - But networks still fulfill the contract.
- Each datagram is looked at individually by the networks, and networks try to get it to its destination. Networks do not order the messages, or make sure all messages are sent, or interpret the messages.
- Service interface:
  hosts/endpoints —--- routers/switches —--- hosts/endpoints
  Networks fulfill the "best effort" contracts on datagrams

```C/C++
#include <string>
#include "socket.hh"

class RawSocket: public DatagramSocket {
    public :
        RawSocket() : DatagramSocket( AF_INET, SOCK_RAM<
IPPRQTQ_RAW);
```

```
}

using namespace std;

int main( int argc, char* argv[] ) {
    RawSocket sock;
    string datagram;
    datagram += “Hello from CS144!!”;
    sock.sendto ( Address{“ 1 “}, datagram );
}
```

Datagram needs to contain the destination address, so we need to write Datagram in a special format **Internet datagram** (format/language) (defined in STD 5 (RTC 791): https://www.rfc-editor.org/rfc/rfc791#section-3.1):

- Networks do “best effort” if you send something in this format
- Version: version of format
- Time to live: # of hops to stay alive

C/C++

```
#include "socket.hh"

#include <cstdlib>
#include <iostream>

using namespace std;

// NOLINTBEGIN(*-casting)

class RawSocket : public DatagramSocket
{
public:
  RawSocket() : DatagramSocket( AF_INET, SOCK_RAW, IPPROTO_RAW )
{}
};
```

```
auto zeroes( auto n )
{
  return string( n, 0 );
}

void send_internet_datagram( const string& payload )
{
  RawSocket sock;

  string datagram;
  datagram += char( 0b0100'0101 ); // v4, and headerlength 5
words
  datagram += zeroes( 7 );

  datagram += char( 64 );  // TTL
  datagram += char( 1 );   // protocol
  datagram += zeroes( 6 ); // checksum + src address

  datagram += char( 34 );
  datagram += char( 93 );
  datagram += char( 94 );
  datagram += char( 131 );

  datagram += payload;

  sock.sendto( Address { "1" }, datagram );
}

void send_icmp_message( const string& payload )
{
  send_internet_datagram( "\x08" + payload );
}

void program_body()
{
  string payload;
```

```
  while ( cin.good() ) {
      getline( cin, payload );
      send_icmp_message( payload + "\n" );
  }
}

int main()
{
  try {
      program_body();
  } catch ( const exception& e ) {
      cerr << e.what() << "\n";
      return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}

// NOLINTEND(*-casting)
```

We write something in the **Internet datagram** format, and the networks get it to its destination (with “best effort”).

- Q&As:
  Q: What if we run out of ip addresses?
  A: new version of the format has more than 4 bytes
  Q: Can we quantify “best effort”?
  A: `ping`: special payload for “please reply to the source address” and we can compare how much time it takes to send there and back (roundtrip time/RTT), and also how many packages are lost.
  Q: Why do routers/switches do this work?
  A: Everyone along the path has a bilateral relationship with each other, and we will cover the business model of the Internet later in this class.
  Q: Can we be told whether ‘send’ is successful?
  A: The Internet itself doesn’t tell, the other party can.
  Q: Why ‘\x8’ paddings?
  A: Some of the routers/switches along the network path is nosy (e.g. firewall). ‘\x8’ makes the payload look legitimate.

- TTL (time to live field)
    - Internet protocol: if I speak the Internet datagram language, the Internet sometimes reply an error message
    - If TTL gets to 0, the router that has the packet would send a error message (part of the protocol, not the language)
    - `traceroute` sends datagrams with different TTL to get hops along the path. (`mtr lamp.mit.edu` (man mtr: https://linux.die.net/man/8/mtr))

- **Encapsulation**: looking at the payload is decapsulating the datagram
    - Each person is looking at its own payload in a payload of payload of payload of payload of ….. But only the payload at its level, nothing further
    - Each packet does not know what is in the payload == This is Encapsulation.
    - Encapsulating messages with meaning to us in the payload, and then another abstraction encapsulates this packet as payload.
- **Multiplexing**: share a link across multiple usage/applications, and each single usage/application does not know about it. (e.g. multiple ordinary processes on the same OS)
    - Internet datagram (host-to-host)
        - To: IP address (Protocol: User Datagram *this is multiplexing*)
        - TTL: 64
        - Payload: User datagram
            - To: process (port number) *this is also multiplexing*
            - Payload: “Hello, CS144.”
    - `nc` wraps “Hello via user datagrams!” in User Datagram and then Internet Datagram