Carnegie Mellon University

18-441/741 Spring 2021 Project 2 (Recitation 1/2)

Junbo Zhang

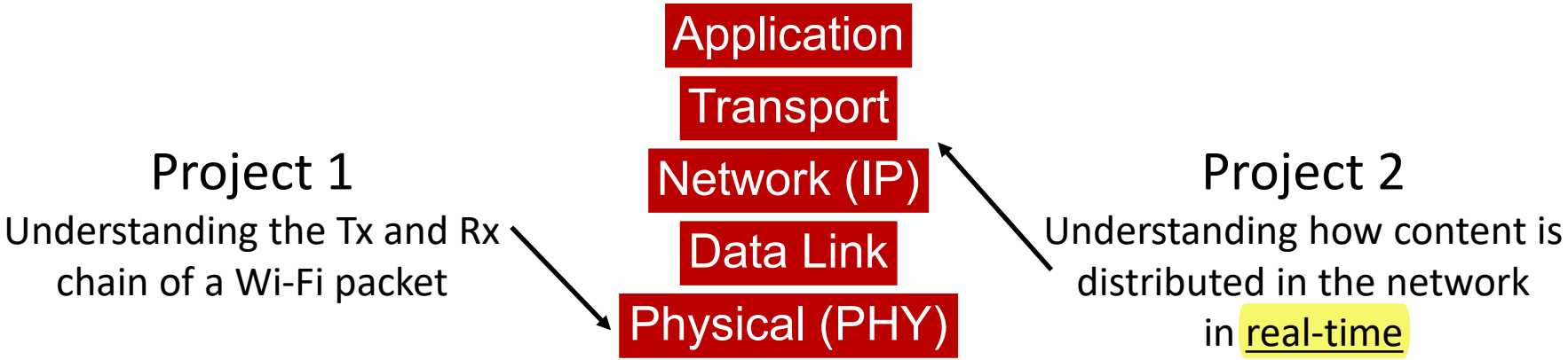
Feb 26

Project 2 Story

- Suppose you are a content provider (YouTube, Netflix, etc.) …
  - All files on a single, super powerful machine?
  - Multiple machines form a network and coordinate with each other? ☑
    - What if a machine is down?
    - What if a new machine is added to the network?
    - What if…

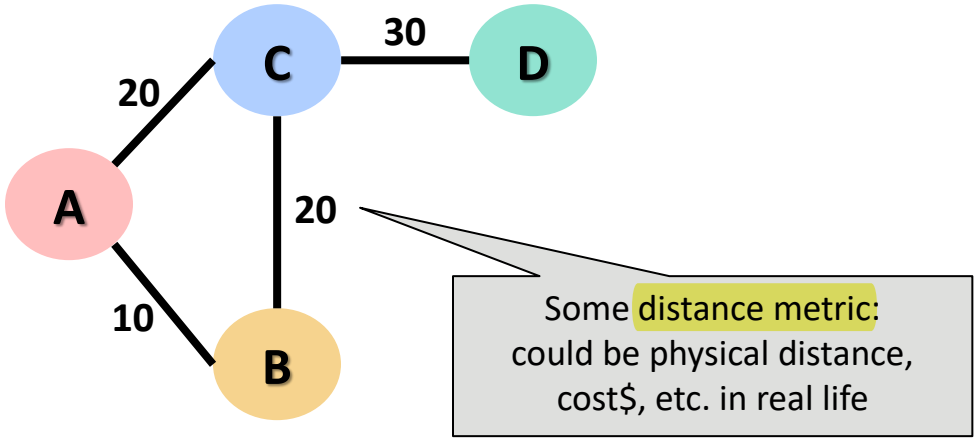
- This project helps you understand how different machines coordinate and communicate with each other in such a scenario
  - Of course, in a simplified manner
    - E.g., no real “content” involved (but will do for project 3)
  - And we add some routing flavor to it (“cost” between nodes, etc.)

Where sits project 2?



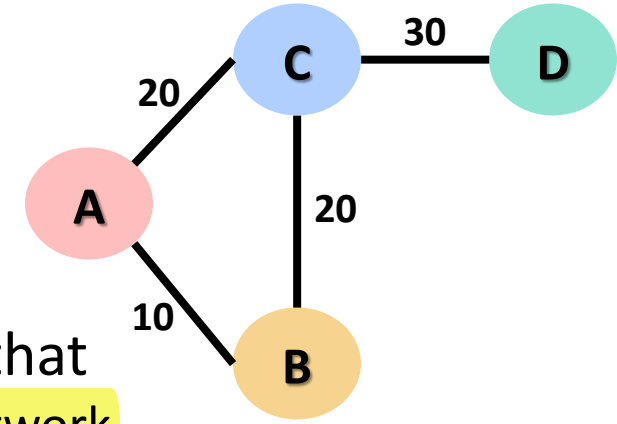
Project 2 in a Nutshell

As the provider, imagine a certain network topology of your machine nodes like this:



Project 2 in a Nutshell

As the provider, imagine a certain network topology of your machine nodes, and then…



The final product – a program **contentserver** that

- can run on any machine to act as a node in your network
- can read a configuration file, in which its neighbor nodes are listed
  - Different nodes have different node.conf, but the same program **contentserver**
- supports adding new neighbors and stopping itself during runtime
- periodically informs its reachability to its neighbors
- automatically detects unreachable neighbors
- can output the whole network topology when queried

Hmm… Where shall I start?

Items	Points (100 – **441** + 110 – **741**)
Setup (submit by checkpoint-1 deadline) - Parse configuration file - Generate UUID (uuid) and save it to config file - Add neighbor (addneighbor) - Node kill (kill) - Preliminary Design Document (1 page) - Successful compilation	30 (Checkpoint-1) March 7 checkpoint: • Basic file operations • Get familiar with socket and thread programming
Peer Routing (submit by final deadline) - Reachability (neighbors) - Link state advertisement (map) - Priority Rank (rank) - Successful submission & compilation - Final Design Document (2-3 pages)	70 (Final) March 28 submission: • Node communication • Link state algorithm (will learn in lectures)
Required for 18-741 (bonus for 441) Active distance metric	**10 (Final)**

will be examined

prepare for yourself

Recitation 1/2

Recitation 2/2

The checkpoint is designed just to make sure you start early!

Carnegie Mellon University

Content

1. Overview ☑

2. **Functions to implement for the checkpoint**
   - Make sure you read the handout carefully for full details

3. Socket programming

4. Thread programming

5. UNIX Machine Resources

Parse Configuration File

```
uuid = f94fc272-5611-4a61-8b27-de7fe233797f
name = node1
backend_port = 18346
peer_count = 2
peer_0 = 24f22a83-16f4-4bd5-af63-9b5c6e979dbb, pi.ece.cmu.edu, 18346, 10
peer_1 = 3d2f4e34-6d21-4dda-aa78-796e3507903c, mu.ece.cmu.edu, 18346, 20
```

A potential example configuration file (**all** fields are optional)

Please read the handout for a comprehensive understanding.

Generate UUID and Save It

```
name = node1
backend_port = 18346
peer_count = 2
peer_0 = 24f22a83-16f4-4bd5-af63-9b5c6e979dbb, pi.ece.cmu.edu, 18346, 10
peer_1 = 3d2f4e34-6d21-4dda-aa78-796e3507903c, mu.ece.cmu.edu, 18346, 20
```

A potential example configuration file (**all** fields are optional)

If this node does not have a UUID specified in its configuration file, you should generate one, and update the configuration file.

In general, you should reasonably update any modification for optional fields.

Add Neighbors

Keyboard input:
addneighbor uuid=e94fc272-5611-4a61-8b27-de7fe233797f host=nu.ece.cmu.edu backend_port=18346 metric=30<newline>

Response: unspecified

Action: Add the given node with the given uuid, host, backend port, and distance metric as your new neighbor

Your node should start performing reachability check on the given node. If the node is active, subsequent calls to /peer/neighbors should contain the information about this node.

Please read the handout for a comprehensive understanding.

Node Kill

Keyboard Input
kill

Response: Program should terminate

Content

1. Overview ☑
2. Functions to implement for the checkpoint ☑

---

3. **Socket programming**
4. Thread programming
5. UNIX Machine Resources

You will need for the 2nd half

What is a socket?

Wiki:

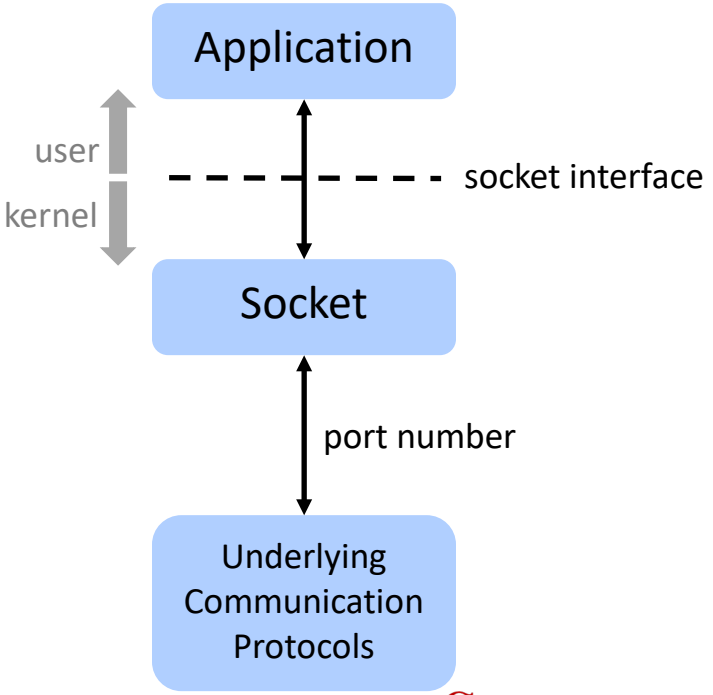
A **network socket** is a software structure within a network node of a computer network that serves as an endpoint for sending and receiving data across the network.

Informally:

- an intermediate role
- provides an interface to the network

We only use sockets to send/receive.

- will learn comprehensively in future lectures



Socket Descriptor

Very similar to:

- File Descriptor (Handle)
  - int open(const char *path, int oflag, .../*,mode_t mode */);

“Everything is a file.”

- Interacting with a socket is very similar to file operations (E.g., I/O)

Socket APIs are available functions that can be called.

- We include some C example functions; feel free to search around Java ones

Socket APIs

SYNOPSIS top

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

DESCRIPTION top

socket() creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

Found in both udpclient.c and udpserver.c.

Reference: https://man7.org/linux/man-pages/man2/socket.2.html

Socket APIs

SYNOPSIS top

```
#include <sys/types.h>          /* See NOTES */
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

DESCRIPTION top

When a socket is created with socket(2), it exists in a name space (address family) but has no address assigned to it. **bind**() assigns the address specified by *addr* to the socket referred to by the file descriptor *sockfd*. *addrlen* specifies the size, in bytes, of the address structure pointed to by *addr*. Traditionally, this operation is called “assigning a name to a socket”.

Found in udpclient.c (it needs to consistently listen).

Reference: https://man7.org/linux/man-pages/

Socket APIs

SYNOPSIS top

```
#include <sys/socket.h>

ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
    int flags, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

DESCRIPTION top

The *recvfrom*() function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

Found in udpclient.c to receive a message (UDP is connectionless).

Reference: https://man7.org/linux/man-pages/

Socket APIs

SYNOPSIS top

```
#include <sys/socket.h>

ssize_t sendto(int socket, const void *message, size_t length,
    int flags, const struct sockaddr *dest_addr,
    socklen_t dest_len);
```

DESCRIPTION top

The *sendto*() function shall send a message through a connection-mode or connectionless-mode socket.

Found in udpserver.c to send a message.

Reference: https://man7.org/linux/man-pages/

Content

1. Overview ☑
2. Functions to implement for the checkpoint ☑
3. Socket programming ☑
4. **Thread programming**
5. UNIX Machine Resources

Thread Programming

You might need multiple threads to handle different tasks in your program.

In C/C++, you can consider the following way to create a new thread.

SYNOPSIS top

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);

Compile and link with -pthread.
```

DESCRIPTION top

The **pthread_create**() function starts a new thread in the calling process. The new thread starts execution by invoking *start_routine*(); *arg* is passed as the sole argument of *start_routine*().

Reference: https://man7.org/linux/man-pages/

Thread Programming

In Java, you might want to refer to the Thread object and its methods.

java.lang

Class Thread

java.lang.Object
    java.lang.Thread

All Implemented Interfaces:

Runnable

Direct Known Subclasses:

ForkJoinWorkerThread

```
public class Thread
extends Object
implements Runnable
```

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Reference: https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

Content

1. Overview ☑
2. Functions to implement for the checkpoint ☑
3. Socket programming ☑
4. Thread programming ☑
5. **UNIX Machine Resources**

ECE Computer Clusters (Multiple Potential Nodes)

Description Page:

https://userguide.its.cit.cmu.edu/research-computing/computer-clusters/

Host Names:

[ece000-ece031].ece.local.cmu.edu example: ece007.ece.local.cmu.edu

You can use any preferred SSH method to log in.

Username and password are corresponding to your Andrew ones.
E.g., Tectia, PuTTY, Xshell or just “Terminal”

CMU UNIX Machine (One Potential Node)

Host Name:

unix.andrew.cmu.edu

Username and password are corresponding to your Andrew ones.

Q & A

You are also welcomed to ask questions later (after chewing on the handout) in office hours.

Don’t wait until deadlines; any of the TAs can answer.