Assigned: Mar 28, 2025
Due Date: Apr 14, 2025

CS371 PA2: Error control, Flow control

Problem description: Error and flow control is a fundamental concept in computer networks that allows applications to communicate reliably over an unreliable network. In this assignment, you will implement error control and flow control mechanism on top of the client-server application developed in PA1.

Objective: The objective of this assignment is to implement error control and flow control mechanism in **C** using **UDP sockets**, which only offers unreliable transport, unlike TCP. You will change the PA1 to use UDP socket and observe the packet loss when you increase the number of clients to saturate the UDP server. Next, you will implement SN and ARQ to detect and mitigate the packet loss over an unreliable communication channel.

Note-1: You can use the PA1 code as the skeleton code for PA2.

Note-2: You are required to demo your PA2 during the TA office hour.

Grouping: You can form a group of no more than three students in this class to complete the programming assignment together. You can also choose to complete the programming assignment individually, but the scoring criteria are independent of the group size.

Tasks

Task 1. UDP-based "Stop-and-Wait" Protocol (30 pts)

[Remember to use UDP socket for this task]

In this task, you will analyze packet loss behavior in a UDP-based "Stop-and-Wait" protocol. In PA1, you (in fact) implemented a simple "Stop-and-Wait" protocol between the client and server. You can simply change the TCP socket to UDP socket and you will get a UDP-based "Stop-and-Wait" Protocol.

Even though you are now using unreliable UDP instead of TCP, your PA1 solution already assumes that the server (receiver) has limited buffer capacity and finite processing speed. A key question to consider: If you continue increasing the number of UDP clients until the server becomes overloaded, will you observe packet loss?

For a single client-server pair, the protocol in PA1 explicitly prevents the client (sender) from overwhelming the server (receiver). However, this assumption no longer holds when multiple client threads are introduced. As more clients are added, packet loss will occur. In data center networks, this phenomenon is referred to as "increasing the fan-in degree of traffic."

To measure packet loss, track the number of packets sent by each client thread (`tx_cnt`) and the number of returned packets successfully received by the client thread (`rx_cnt`):

lost_pkt_cnt = tx_cnt - rx_cnt

Since UDP lacks flow control and error control, you will observe packet loss once the server reaches its processing limits – the number of active client threads receives a certain threshold.

*TODOs in Task 1: (1) change to UDP socket (**5 pts**); (2) implement metrics collection mechanism on the client side to track the number of lost packets (**10 pts**); (3) can observe packet loss when you increase the number of client threads (**15 pts**).*

Task 2. UDP-based Sequence Numbers (SN) and Automatic Repeat Request (ARQ) (30 pts)

[Remember to use UDP socket for this task]

As the number of client threads increases, packet loss may occur. To mitigate this, you will implement an error control mechanism using Sequence Numbers (SN) and Automatic Repeat Request (ARQ). This will enable proactive detection of packet loss (via timeout and SN) and retransmission of lost packets through ARQ.

Refer to the pseudo-code from Lecture 18 as a guideline, but note that direct implementation may not be feasible. Keep the following considerations in mind:

1. The pseudo-code in Lecture 18 is designed for a single client-server pair, whereas our case involves multiple concurrent client threads. Therefore, you must include the client ID in the packet header along with SN and other metadata (see the "frame header" structure in Slide 19, Lecture 18). So that the server can differentiate the packets from different clients.

2. You can leverage epoll's built-in timer to handle timeouts. Alternatively, you may implement your own `start_timer` function if preferred.

3. When retransmitting a packet, you do not need to update `tx_cnt`. You can verify the correctness of your protocol by comparing `tx_cnt` and `rx_cnt`—if `tx_cnt == rx_cnt`, the protocol is functioning correctly.

[Task 2 can be built on top of Task 1]

*TODOs in Task 2: (1) implement the timer logic (**5 pts**); (2) implement the mechanism to track packet SN (**5 pts**); (3) implement mechanism to support retransmission (**10 pts**); (4) no packet loss occurs no matter how you increase the number of client threads (**10 pts**).*

Skeleton Code

[!!!You should use your PA1 solution as the skeleton code!!!]

Command-Line Arguments (same as PA1)

It accepts the following 5 command-line arguments:

- "<server|client>" – decide whether the skeleton code will run as a server or as a client program (see example usage below)
- "server_ip" - The IP address of the server.
- "server_port" - The port number of the server.
- "num_client_threads" - The number of client threads to launch.
- "num_requests" - The number requests that each client thread will send.

Note: Please DO NOT change the command-line argument. We will deduct 50 points from your final score if you change the Command-Line Arguments

How to compile (same as PA1)

Note: the instructors of this course stick to Ubuntu 22.04 for development and evaluation of this programming assignment (for details, refer to "Development and Evaluation Environment").

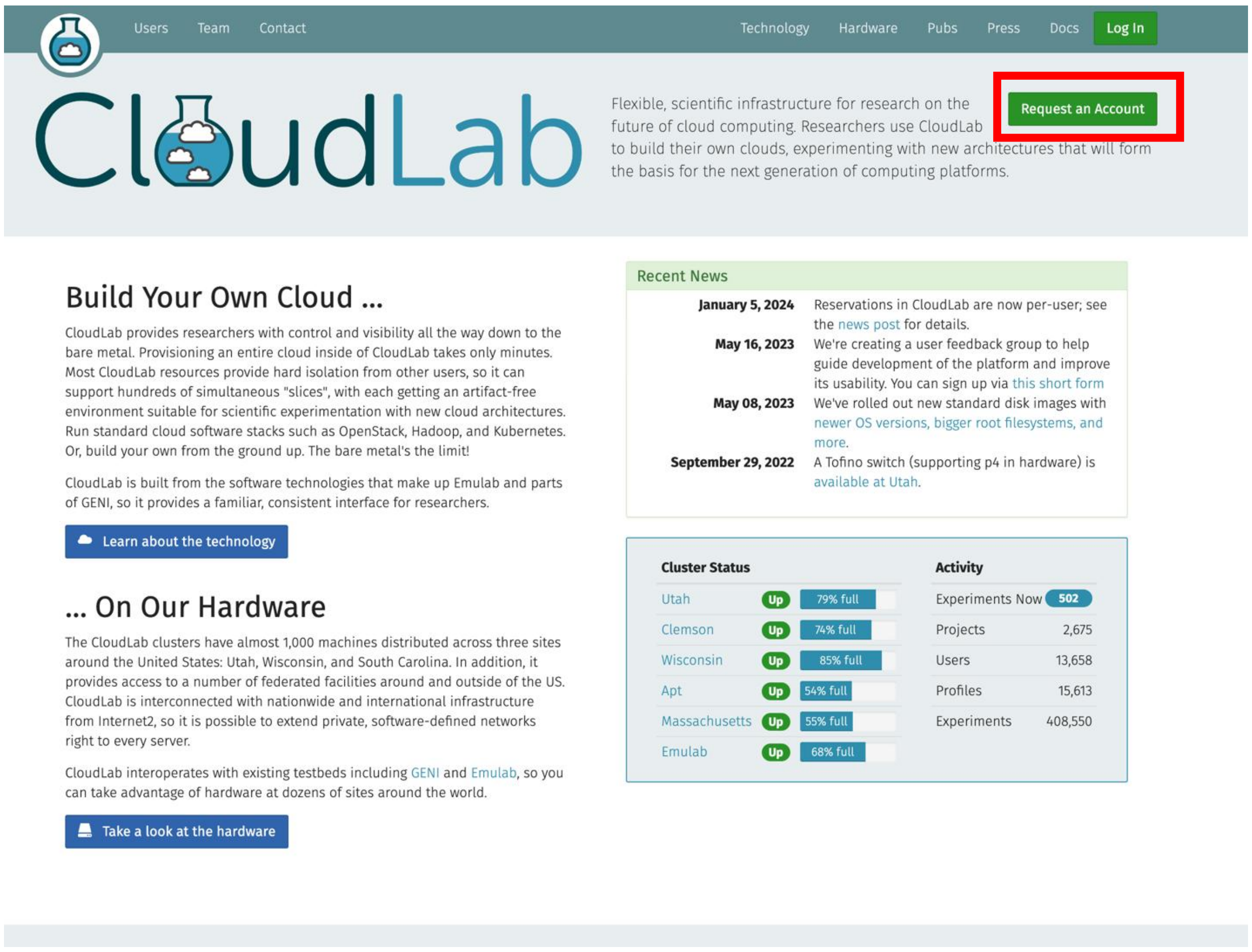
```
gcc -o pa2_binary pa2_code.c -pthread
```

Development and Evaluation Environment (same as PA1)

The skeleton code was developed on Ubuntu 22.04 server, using x86 CPUs. We will stick to this setup, using the x86 Ubuntu 22.04 servers on NSF CloudLab for development and evaluation of this programming assignment.

We encourage all students register an NSF CloudLab account and join the "**UKY-CloudNetSys**" project, where you can get exclusive access to an Ubuntu 22.04 bare-metal server for a certain period (it's free!). Please follow the steps below for **CloudLab Registration**:

1. Go to https://cloudlab.us/ and click Request an Account



Flexible, scientific infrastructure for research on the future of cloud computing. Researchers use CloudLab to build their own clouds, experimenting with new architectures that will form the basis for the next generation of computing platforms.

Build Your Own Cloud ...

CloudLab provides researchers with control and visibility all the way down to the bare metal. Provisioning an entire cloud inside of CloudLab takes only minutes. Most CloudLab resources provide hard isolation from other users, so it can support hundreds of simultaneous "slices", with each getting an artifact-free environment suitable for scientific experimentation with new cloud architectures. Run standard cloud software stacks such as OpenStack, Hadoop, and Kubernetes. Or, build your own from the ground up. The bare metal's the limit!

CloudLab is built from the software technologies that make up Emulab and parts of GENI, so it provides a familiar, consistent interface for researchers.

... On Our Hardware

The CloudLab clusters have almost 1,000 machines distributed across three sites around the United States: Utah, Wisconsin, and South Carolina. In addition, it provides access to a number of federated facilities around and outside of the US. CloudLab is interconnected with nationwide and international infrastructure from Internet2, so it is possible to extend private, software-defined networks right to every server.

CloudLab interoperates with existing testbeds including GENI and Emulab, so you can take advantage of hardware at dozens of sites around the world.

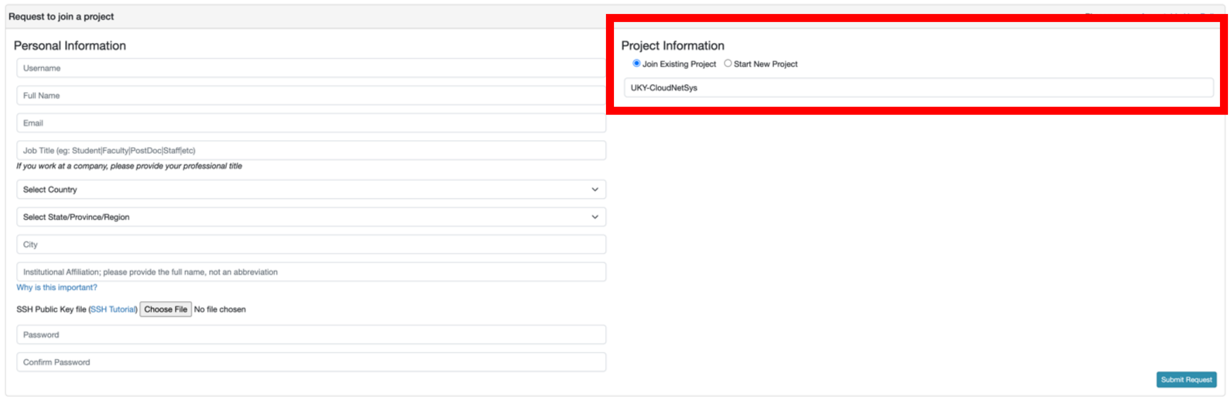
Recent News

January 5, 2024	Reservations in CloudLab are now per-user; see the news post for details.
May 16, 2023	We're creating a user feedback group to help guide development of the platform and improve its usability. You can sign up via this short form
May 08, 2023	We've rolled out new standard disk images with newer OS versions, bigger root filesystems, and more.
September 29, 2022	A Tofino switch (supporting p4 in hardware) is available at Utah.

Cluster Status		
Utah	Up	79% full
Clemson	Up	74% full
Wisconsin	Up	85% full
Apt	Up	54% full
Massachusetts	Up	55% full
Emulab	Up	68% full

Activity	
Experiments Now	502
Projects	2,675
Users	13,658
Profiles	15,613
Experiments	408,550

2. Fill in your personal information, select **Join Existing Project** and enter **UKY-CloudNetSys** then Submit and wait for approval**.**



Request to join a project

Personal Information

Username

Full Name

Email

Job Title (eg: Student|Faculty|PostDoc|Staff|etc)

If you work at a company, please provide your professional title

Select Country

Select State/Province/Region

City

Institutional Affiliation; please provide the full name, not an abbreviation

Why is this important?

SSH Public Key file (SSH Tutorial) Choose File No file chosen

Password

Confirm Password

Project Information

[✓ Join Existing Project] Start New Project

UKY-CloudNetSys

Submit Request

3) This video provides an introduction to cloud lab:

Creating a New Experiment in Cloudlab and How to Extend an Experiment

https://uky.zoom.us/rec/share/DXfu7qXihjnpYJgGM3sicsEwwZgN6qHGKewR_9sINU7w4PrMNXewakfRkgjuwoA6.tnzG03d22xZkRtTA

Useful references for Cloudlab usage:

- Resource availability on Cloudlab: https://www.cloudlab.us/resinfo.php
- Hardware specs on Cloudlab: https://docs.cloudlab.us/hardware.html

Important Note: You can also use the Ubuntu virtual machine to develop work on your own laptop, but please test the code on NSF CloudLab before submitting. You are responsible for ensuring that the submitted code can run on the x86 Ubuntu 22.04 machine on NSF CloudLab.

If you do not comply with this policy, your submission will be counted as 0 points because we cannot evaluate your submission.

Submission Instructions

Please upload your programming assignment to Github and submit the link to your Github repository for your programming assignment to Canvas.

Other submission methods will not be considered and will be scored **zero**.

Please follow the requirements below when submitting your programming assignment. Your Github repository for your programming assignment must contain the following **TWO** files:

- **Source code of Task 1.**
- **Source code of Task 2.**

Please note that we will use the last commit before the due date to evaluate the programming assignment. Any commits after the due date will not be considered.

Grading Policy

Evaluation Criteria

The total score is 100 points. You need to ensure that the client and server can run seamlessly. If neither the client nor the server can run, the submitted score will be **zero**. The following score breakdown is based on the client and server being able to run, that is, the client can send messages to the server, and the server can send messages back to the client without any errors or crashes:

- Task 1 (**30 pts**)
  - (1) change to UDP socket (**5 pts**);
  - (2) implement metrics collection mechanism on the client side to track the number of lost packets (**10 pts**);
  - (3) can observe packet loss when you increase the number of client threads (**15 pts**).
- Task 2 (**30 pts**)
  - (1) implement the timer logic (**5 pts**);
  - (2) implement the mechanism to track packet SN (**5 pts**);
  - (3) implement mechanism to support retransmission (**10 pts**);
  - (4) no packet loss occurs no matter how you increase the number of client threads (**10 pts**).
- Demo (**25 pts**)
  - Demo that you can observe packet loss when you increase the number of client threads (**5 pts**).
  - Demo that no packet loss occurs no matter how you increase the number of client threads (**10 pts**).
  - TA will ask some implementation details regarding your PA2 code (**10 pts**).
- Properly close the file descriptors of sockets and epoll (**5 pts**)
- Handle error cases gracefully (**5 pts**)
  - e.g., failed socket connections, epoll failures, etc.
- Code readability (**5 pts**)
  - Use consistent indentation (e.g., 4 spaces) to visually structure code block
  - Use consistent spacing around operators, keywords, and other code elements

Important Notes:

- Please use the correct socket type based the requirement of each task
  - Your submission will be counted as 0 points if you use wrong socket types.
- Please DO NOT change the command-line argument.
  - We will deduct 50 points from your final score if you change the Command-Line Arguments
- Please follow Submission Instructions on page 6.
  - Other submission methods will not be considered and will be scored zero.
- If you do not comply with the policy of Development and Evaluation Environment, your submission will be counted as 0 points because we cannot evaluate your submission.

References

- Practical TCP/IP Sockets in C, Second Edition
    - https://cs.baylor.edu/~donahoo/practical/CSockets2/
- Linux epoll documentation
    - https://man7.org/linux/man-pages/man7/epoll.7.html
    - https://linux.die.net/man/4/epoll
- Lecture 18