ASSIGNMENT 1

DEVELOP A NETWORK APPLICATION

COURSE: COMPUTER NETWORKS, SEMESTER 1, 2024-2025

OBJECTIVES

Build a **simple segment chat application** (Discord-like) with application protocols defined by each group, using the TCP/IP protocol stack.

APPLICATION DESCRIPTION

Application overview

- Hybrid paradigm: This application uses both client-server paradigm and peer-to-peer paradigm.
- The application performs the client-server during the initialization time to submit the information of upcoming new peers.
- The application leverages peer-to-peer to broadcast the content from one peer to all other peers (as a live streaming session).
- The application supports client-server when the live streamer is offline which is in low traffic conditions.
- Hosts: there are two types of hosts in this system: a centralized server and several normal PCs.

System infrastructure in hybrid paradigm

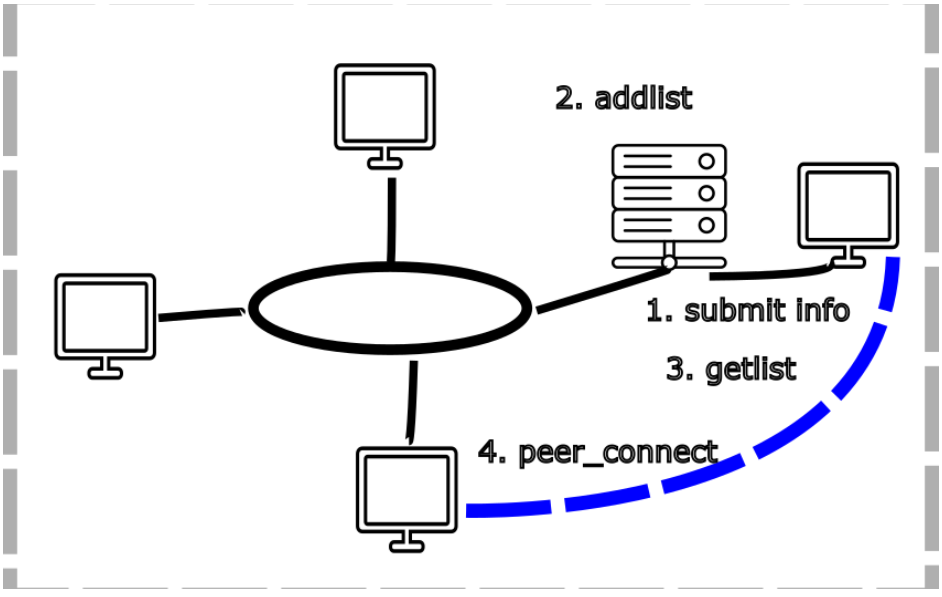

Figure 1. Illustration of segment chat system.

- A peer is an application instance running on a specific host. Three different instances of the application running on a single host are described as 3 peers.
- A centralized server has a tracker that keeps the tracking list of the peers that are connected.
- A peer who joins the system will submit its information to the centralized server and this information is updated to the centralized server tracking list.
- A peer can request the current list of systems.
- The detailed connection steps are:
    - 1. submit_info: new peer submit its host IP and its port
    - 2. add_list: tracker process on the centralized server add the new information to the tracking list
    - 3. get_list: the centralized server response with the tracking list on peer request
    - 4. peer_connect based on the obtained tracking list, the peer connects directly to another peer.

System components

The components of the **simple segment chat application** system are listed below:

Authentication

Visitor mode: users can connect to the system to retrieve the channel content but they are forbidden to perform any content modification. Besides, the channel can set the permission to not allow visitor viewing operation.

- *The detailed specifications of this mode are:*
    - Visitors do not need to login (no authentication required)
    - Visitors need to give/declare a naming through an input text.
    - Visitors are granted view permission only.
    - Visitors are not allowed to edit/create content.

Authenticated-user mode: these users are required to login to authenticate and mapping the access control list.

- *The detailed specifications of this mode are:*
    - Authenticated-users is mandated to login
    - Authenticated-users are granted all edit/create content permission.
- *The user has different displayed status*
    - Online mode: its online status are shown visible to all other authenticated-users
    - Offline mode: it has no connection to the system.
    - Invisible mode: its status is displayed offline only, but it has connection to the system and works normally as same as online mode.

To shorten the text, we refer to the user as authenticated-user in the remainder of this document.

Channel

Channel is a single list of chatting messages. The users need to support the following channel operations:

- *List the channels that the users joined.*
- *Each channel has a scrolled window that displays the message list.*
- *Each channel supports a UI input component (input textbox, submission button) to allow users to create a new message and submit it.*
- *In the message list, we do not need to support edit/delete messages since it is not the objective of the network course. The course outcome has zero proportion of scoring on the convenience or beauty of the application.*
- *As the course outcome, it needs to send the data of the message to the destination. And it might create a notification when another message is created.*
- *Channel can support a customization policy on access list control but since we have much task to do and the main focus is the illustration of the course CO3093 outcome, in which it provide the data connection among computers, then access control is an option only (no scoring proportion)*

Personal own server or Channel hosting

NOTICE In this application style, the term "server" is NOT used to describe the physical machine where the services are deployed.

Server refers to the host that co-locates a set of channels (placed at the same location). As usual, that set belongs to one owner user. Other channels the owner users join are listed in the channel list not this channel hosting section . To clarify the functionality, we call it Channel-hosting. Students are free to interchange between the two terms, server as common sense application usage while channel hosting as theoretical paradigm.

- Channels in *channel* hosting vs joined-only channels: the former set of channels is owned by the current user and is stored on the peer that user is logging in. These channels might have a copy on the centralized server but these copies are mainly for backup purposes and are not primary storage repositories. The latter set of channels is simply cached content to reduce the access time to the remote peer. In accessing context, we use channels to represent both sets of channels, but in storage context, we need to indicate the exact type of the concerned channel set.
- *Channel-hosting* connection: when a user changes from offline to online, it needs to synchronize between the content in local peer and the content on the centralized server. During the online period, it keeps updating in both places synchronously.
  Remember that Discord was invented early for gaming streaming purposes where it has a large user actively for a while during the live session.

The direct connection between channel-hosting and viewer user is used in the peer-to-peer paradigm which supports a time sensitive data transfer and large scale. The maintenance of a copy on the centralized server costs only one connection and once transfer time. If all viewer users access the server it will cause the bottleneck as a traditional problem in client-server paradigm. The one copy on the centralized server is accessed when the live stream user is offline and out of the live session where the traffic amount is small therefore the centralized server is strong enough to handle it.

- The synchronization between channel-hosting and the centralized server is illustrated as in the following scenarios:

We denote A and B are the data content (text/chat/message)

- ***Channel hosting go offline***

Channel hosting	*Centralized server*	*Joined users X*	*Joined users Y*
N/A	**have a copy A**	fetch A from centralized server	fetch A from centralized server
N/A	sync to centralized server	**create new content B**	fetch B from centralized server
create new content A and cached it locally only	Do nothing (not available)	Do nothing (not available)	Do nothing (not available)

- ***Channel hosting go online***

Channel hosting	*Centralized server*	*Joined users X*	*Joined users Y*
having content A created during offline	sync between channel hosting and centralized server	fetch A from Channel hosting	fetch A from Channel hosting
create new content A	sync between channel hosting and centralized server	fetch A from Channel hosting	fetch A from Channel hosting
sync to channel hosting	sync to centralized server	**create new content B**	fetch B from Channel hosting

- ***Livestream period***

Channel hosting	*Centralize server*	*Joined users X*	*Joined users Y*

create new content A	copy A to the server	fetch A from Channel hosting	fetch A from Channel hosting
sync to channel hosting	sync to centralized server	**create new content B**	fetch B from Channel hosting

- ***Joined user go online***

Channel hosting	*Centralize server*	*Joined users X*	*Joined users Y*
has content A	has content A	fetch A from Channel hosting if it is online, otherwise fetch A from centralized server	fetch A from Channel hosting if it is online, otherwise fetch A from centralized server
sync B to channel hosting	sync B to centralized server	**has contend B during offline**	fetch B from Channel hosting if it is online, otherwise fetch B from centralized server
sync to channel hosting	sync to centralized server	**create new content B is as same as the case Channel hosting online/offline or livestream period**	fetch B from Channel hosting if it is online, otherwise fetch B from centralized server

- ***Joined user go offline***

Channel hosting	*Centralize server*	*Joined users X*	*Joined users Y*
Do nothing (not available)	Do nothing (not available)	**Create new content B and cached it locally only**	Do nothing (not available)

To keep it simple, we do not declare all small details. It is reasonable to assume that a joined user who creates new content is already an authenticated user. Therefore, it has the permission to edit/create new content. Visitors are excluded in that case since they have view permissions only and have no right to create a new content.

User access

- *user* fetch: user fetch content when it changes from offline to online.
- *user* notification: when a new content is created, it is notified to the user by notification mechanism and fetches the new content to the user application instance. Students are free to design such a beautiful notification mechanism,

but in the case we lack the idea, a simple mechanism of getting a user list from the centralized server and polling to send a notification to each user in the list is a nice start.

- A note of user list: on each peer, there may be more than one instance of this application, each instance has a unique peer port as usual. A user can log in multiple times on different application instances via multiple sessions, then items in the user list are unique by the tuple of (peer IP and port, username, session ID). The user list is a list of these tuples.

System Log

To track the initialization of the connection in multi-direction communication. You might write a log entry for each connection and data transfer. The log file, as usual, is an ASCII text file to keep it simple format.

Extra credit:

Seeding strategies: a simple download strategy is designed for a single livestreamer. But if you choose to apply something fancier (like multi peer seeding), you need to document it and provide supplemental materials.

Large scalability examination: To support billions of messages, Discord has passed through various technology improvements (*). If you have some estimation and collect some measurements to reproduce the numerical that shows different scalability on these technologies, it is a plus.

- In 2017, Discord use MongoDB to support 100 million stored message
- By 2022, their Cassandra cluster had 177 nodes with trillions of messages, but faced serious performance issues.
- They decided to migrate to ScyllaDB, a Cassandra-compatible database written in C++ which promised better performance, faster repairs, and stronger workload isolation.

*https://discord.com/blog/how-discord-stores-trillions-of-messages

System API You can support API ase ((BASE_URL))/users/{user_id}/channels/ or some basics to demo the capabilities of autorunning bot through programmable API. The definition of the API is freely available for your brainstorming. You need to document it and provide supplemental materials.

Bot or automation support: demo that you can create an autoresponder bot in the application.

Q&A

https://docs.google.com/spreadsheets/d/1rUlqmEX1mX3PwfKobLRqWlWNCm08wqcHmG6Qc0PcVXw/edit?usp=sharing

Grading

Tracker Protocol - 20% (the initialization phase) successfully parses the metainfo, sends a request to the tracker, obtains its response with the list of corresponding peers, and parses the list of peers into useful 'ip' and 'port' pairs.
Client server paradigm - 20% data transfer when a peer need to connect to the tracker on the centralized server

Peer to peer paradigm - 20% (the live stream phase) data transfer when a peer becomes a living streamer and supports peer connections and data transfer among peers.

Readme - 10% document your design, as well as any errors or features.

Synchronization between Channel-hosting and centralized server feature - 10%

Connection logging - it needs to log the connected host (either centralized host or channel hosting) for each message or notification. Simple log file is fine, we can clear and log a new file when the log is longer than 10.000 records. But this log is used to evaluate where your data are synchronized from. **10%**

Advanced feature - 10%

WORKING PLAN

- Work in a team.
- Each group has 2-3 or up to 4 members.

Phase 1 (First 2 weeks):
- Define specific functions of the file-sharing application
- Define the communication protocols used for each function

Phase 2 (Next 2 weeks)
- Implement and refine the application according to the functions and protocols defined in Phase 1

RESULTS

Phase 1 (softcopy):
- Submit the report file of the first phase to BKeL in the predefined section as announced on BKeL.
- File format (pdf): **ASS1_P1_<<Group_Name>>.pdf**

This report includes:
- Define and describe the functions of the application

- Define the protocols used for each function

Phase 2 (hard and softcopy):

- Submit the report file of the assignment to BKeL
- File format (.rar): ASS1_**<<Group_name>>**.rar

This report includes:

- Phase 1 content
- Describe each specific function of the application
- Detailed application design (architecture, class diagrams, main classes ...)
- Validation (sanity test) and evaluation of actual result (performance)
- Extension functions of the system in addition to the requirements specified in section 2
- Participants' roles and responsibilities
- Manual document
- Source code (softcopy)
- Application file (softcopy) compiled from source code

EVALUATION

Assignment 1 is worth 15% of the course grade.

DEADLINE FOR SUBMISSION

3-4 weeks from the assignment announcement depends on the Lab teacher.