ASCII Chat

Lucas Chen (lmc336), Danny Qiu (dq29), Chesley Tan (ct353)

System Description

Vision

For our final project, we will be building a terminal-based peer-to-peer video chat application with end-to-end encryption. It will feature text-based video rendering, audio support, and messaging support. We envision that our application will have diverse applications, such as usage in video conferencing or collaboration in bandwidth-constrained or graphically-constrained environments.

Features

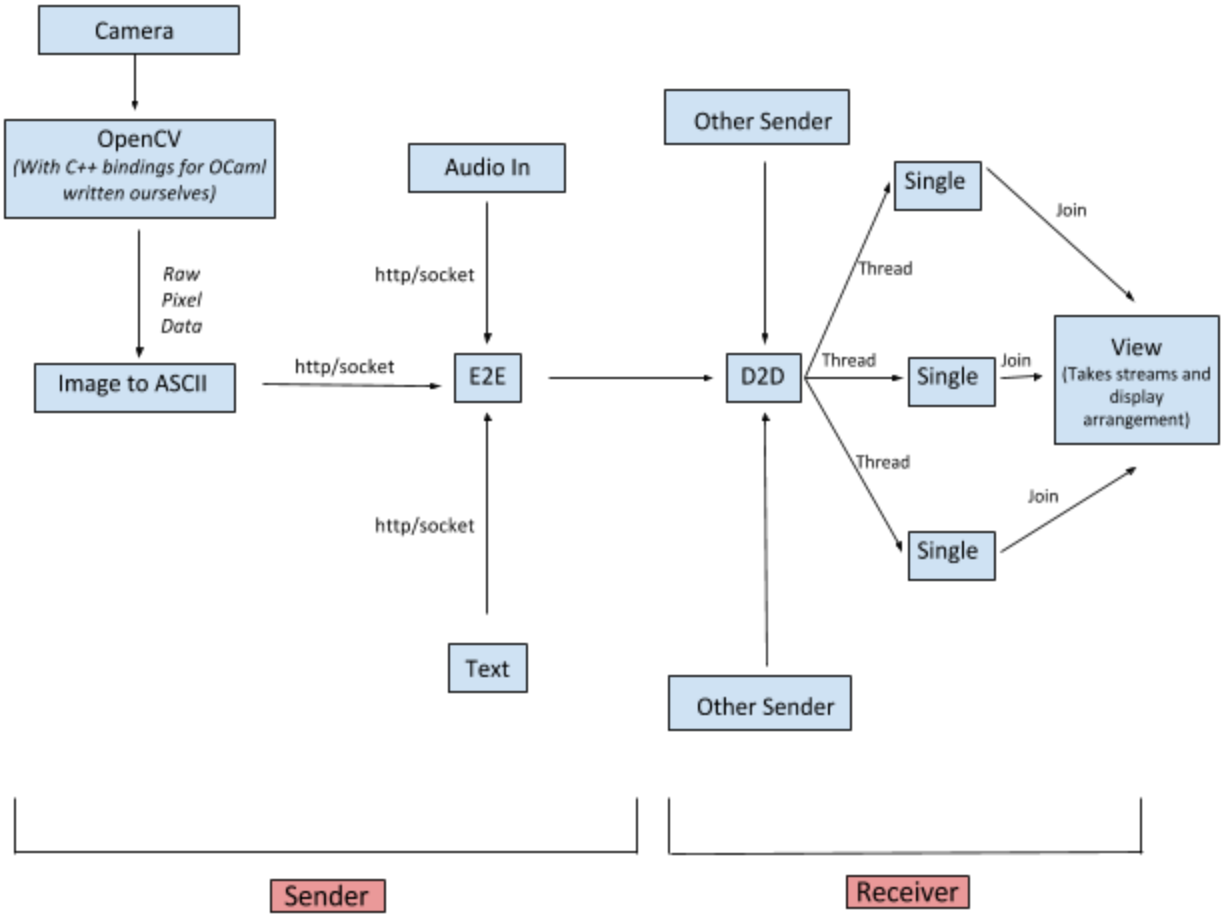
Our program will support/have:

- Colorized text-based video-chatting with up to 4 peers
- End-to-end encryption of video, audio, and messages
- Ephemeral messaging capability - messages are not permanently stored and are encrypted during transmission
- Audio/Voice chat
- Cross-platform compatibility

Description

To video-chat with peers, our program will accept an IP address or URL of the hosting user, who acts as the gateway for other users to join a video conference. Each user within the video conference will act as both a server and a client in that each user's server sends its video to other clients, as well as receives video from other clients. The video will be displayed in ASCII, where the variation of characters will be used to represent differing local features within the image. In addition, the color capabilities of the user's terminal will be utilized. For minimizing bandwidth, colors in the video will be constrained to a set of 256 predefined colors that span a wide range of the color spectrum. Each user's server will locally process the video input from the user's camera to generate an ASCII representation of the image, as well as compress the colors in the image and resize the image appropriately. To support video conferencing with up to 4 peers, the user interface will automatically re-orient itself to maximize screen usage and equalize screen space for each video stream. Users will also be able to send text messages to each other, as well as transfer audio from the user's microphone to other users. Finally, our program respects user privacy as all data that is transmitted will be end-to-end encrypted, and no data will be persistently stored.

Architecture

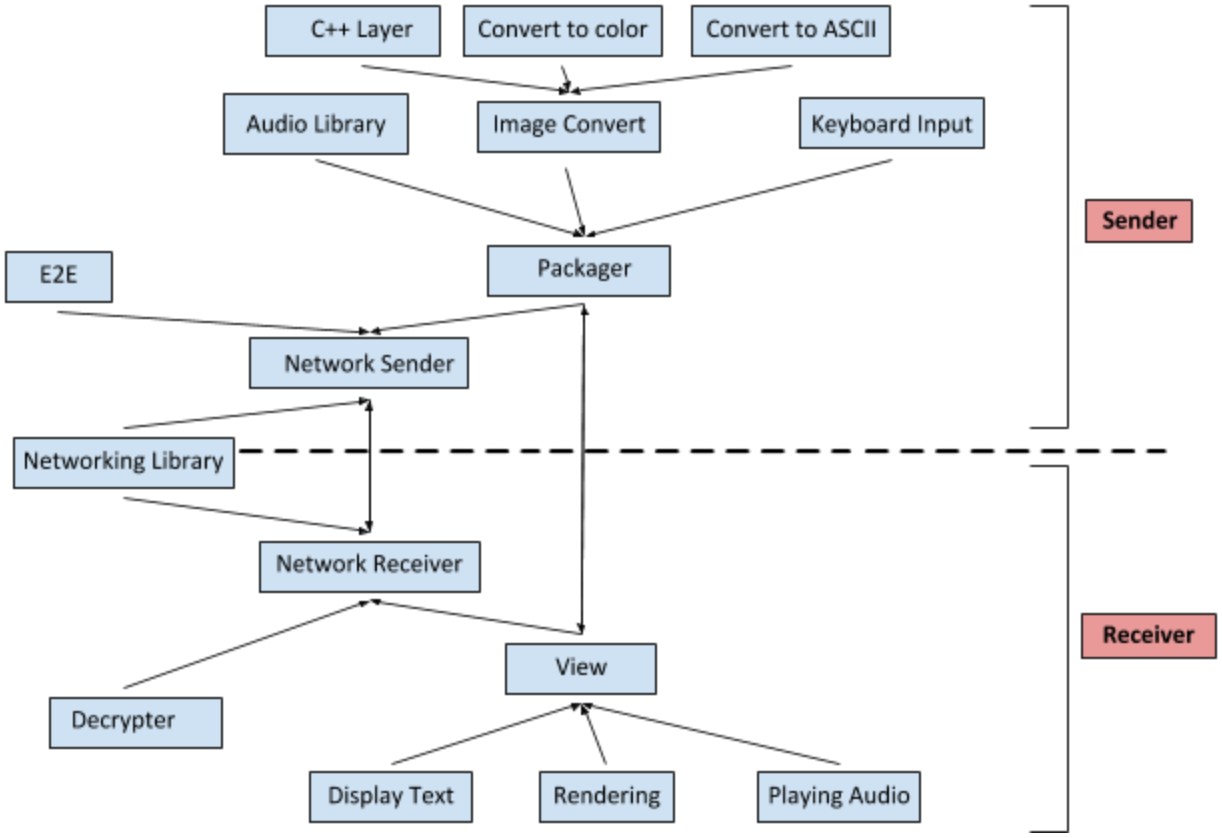


Our system uses strong MVC design, in which we make sure to decouple different components and put similar functionality together. The model is represented by the video, text, and audio data. The controller is represented by the E2E (end-to-end encryption) which takes the model data and encrypts it for sending, and the D2D (end-to-end decryption) which decrypts the encrypted data and reconstructs the model. The view is represented by the component which combines the decrypted data and displays it on the terminal.
It also allows the system to be scalable since each client will act as both a sender and a receiver, leading to the desired decentralized ascii video chat. This peer to peer architecture will emphasize strong networking principles to lower the bandwidth used and decrease the latency for smooth video chatting. It will also use the pipe and filter model to ensure that the client has no performance delay.

System Design

- **Convert to ASCII:** Renders the the raw pixel data as ASCII characters

- **Convert to Color:** Assigns color to each of the ASCII characters
- **Audio LIbrary:** Takes in constant audio feed from system microphone
- **Image Convert:** Combines all the layers and feeds the stream of ASCII characters
- **Keyboard Input:** Takes text input from user keyboard.
- **Packager:** Combines all audio, visual, and text information.
- **E2E:** Encrypts all the data that is sent
- **Network Sender:** Sends data to host
- **Network Receiver:** Receives data from the network socket
- **Decrypter:** Decrypts data that is received
- **Viewer:** Processes received information and passes it on as text, visual data, or audio
- **Display Text:** Displays the next available text from all clients in chatbox.
- **Rendering:** Displays the next available image from all clients
- **Playing Audio:** Passes audio onto system speakers.



Module Design

What is the interface to each module? Write a .mli file making each interface precise with names, types, and brief specifications (which you will plan to elaborate later); submit a zip file named interfaces.zip containing those .mli files along with your design document.

Data

In accordance with the privacy-oriented design of the application, no data will be persistently stored by the application to storage outside of memory. Because the video chat implementation is text-based, videos will be represented as a sequence of frames, which are represented as encrypted strings (using end-to-end encryption scheme) when being transferred in a peer-to-peer fashion between clients. Textual data will be transferred along with the ascii video and encrypted together. Audio will be transferred between clients in a binary format.

We expect to use hash tables as a method of storing the most recent video frame associated with a given peer, so that we can perform a constant-time indexing operation given the peer whose frame we want to look up. We expect to use arrays to store image data so that we can process (convert to text) them with reasonable efficiency, since the conversion algorithm will rely on a pixel's neighboring pixels as well. We also expect to use the hash tables to provide fast lookup of encryption keys for the given peer.

For a more efficient implementation of multi-peer chats, it may be helpful to use a decentralized network where peers retransmit data to get data from one end of the network to the other. This allows for less network overhead and more efficient routing. Much of this will be implemented using distance-vector routing, which would require dijkstra's algorithm on a graph, which contains all the connection nodes. A routing table, which would be a tree-based map would most likely be needed in this case.

External Dependencies

We plan on leveraging the OpenCV library in order to read camera images from the user's webcam. We chose to use OpenCV, because of its cross-compatibility, portability, and performance. Because there does not exist any OpenCV bindings for OCaml, we may need to write a small API in C++ to read frames from the user's camera and return the image data as a primitive data type, and then use the ocaml-ctypes library to call the C++ function from within OCaml. We purposefully avoid doing a large amount of work within the C++ component, where we have the benefit of OpenCV's extensive API, in order to focus on using OCaml.

We will also be using ANSITerminal to control the user's terminal to display the ascii video, resize the terminal window, and allow for colorized output. Though, we will be writing many helper functions on

top of ANSITerminal to create the desired layout for video chatting, ANSITerminal will serve as a good base for what we need.

Testing Plan

We plan on testing our system throughout development by writing unit tests for individual functions as we implement the system in a bottom-up manner. By employing a bottom-up implementation with fastidious unit testing, we can ensure we catch bugs in critical functions early on so that they do not cause bugs in other functions which utilize them. We will each commit to peer-reviewing each other's code/commits to ensure that good design is being followed and that the code is elegant, readable, and bug-free.

Unit tests will be written for video to ascii conversion to maintain correctness over time. Tests will also be written for the end-to-end encryption to ensure correctness in both the encryption and decryption phases. Lastly, as we encounter bugs and fix them, we will add integration/unit tests so that bugs do not reappear in the future.