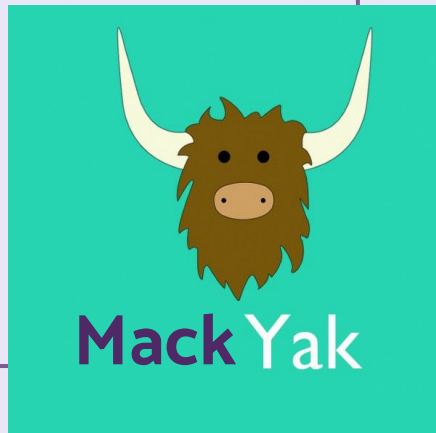
MackYack

By: Alexander Elguezabal, Derek Costello, Brandon Cash



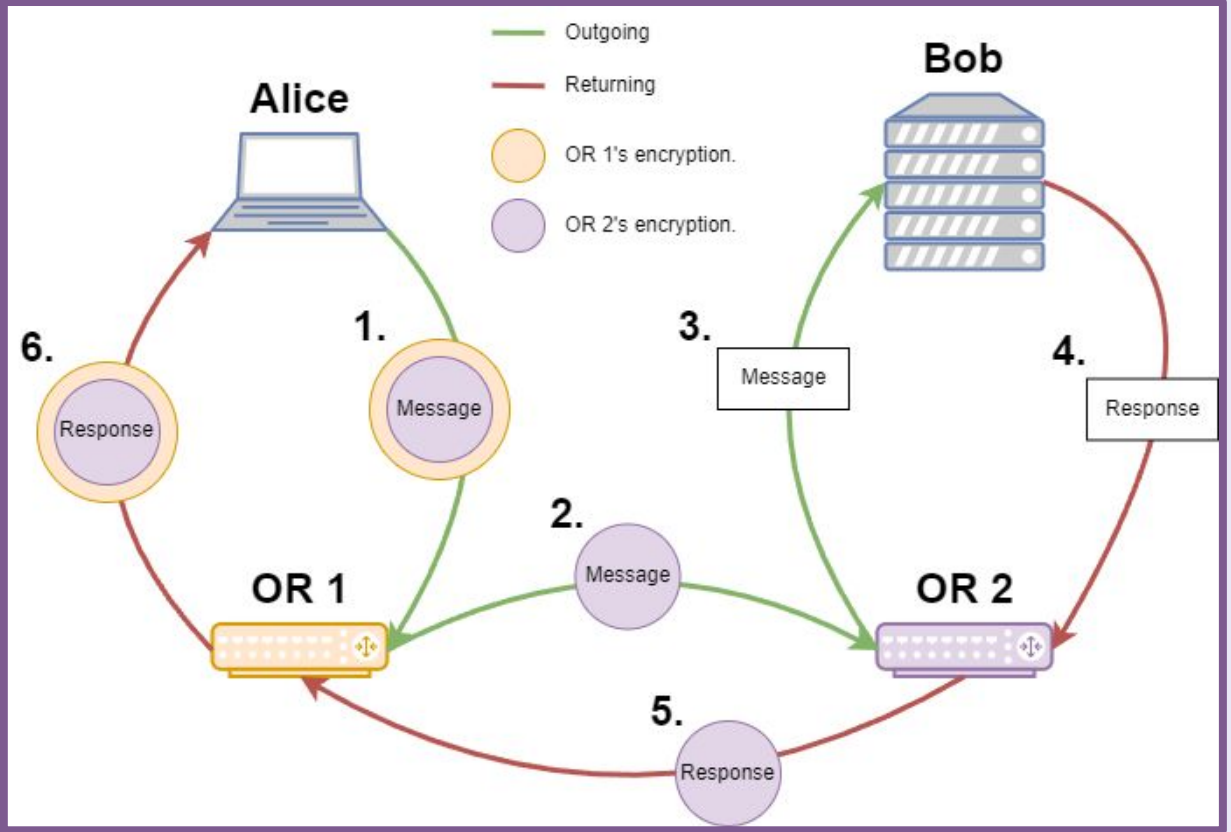
Our Project

- **MackYack** is an anonymous messaging service based off a real world application with a similar name.
- Clients are able to send messages to the server, where they are added to the MackYack board.
- Requests to the server are made periodically (every 3 seconds) to update the client's local view of the MackYack board.
- Utilizes TCP as the transport layer protocol to ensure reliable message delivery.

The Protocol

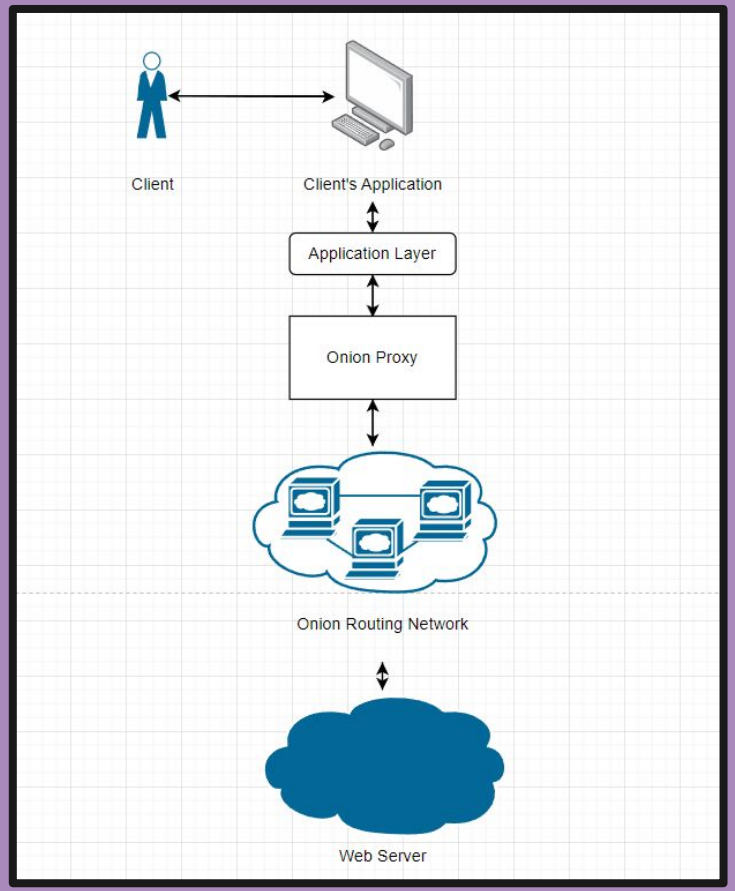
- MackYack uses Onion Routing to maintain the **anonymity** of each client.
  - Onion Routing achieves anonymous communication by routing data through layers of intermediate nodes called Onion Routers. At each layer a router peels off a layer of encryption, revealing the next destination for the message. This method prevents any single node from knowing both the sender and recipient.
- Our Onion Routing overlay network is built in reference of *Tor: The Second-Generation Onion Router*.
  - It is worth mentioning that this is **not** a faithful implementation of Tor. There are a number of relaxations to the original specification.

The Protocol (cont.)



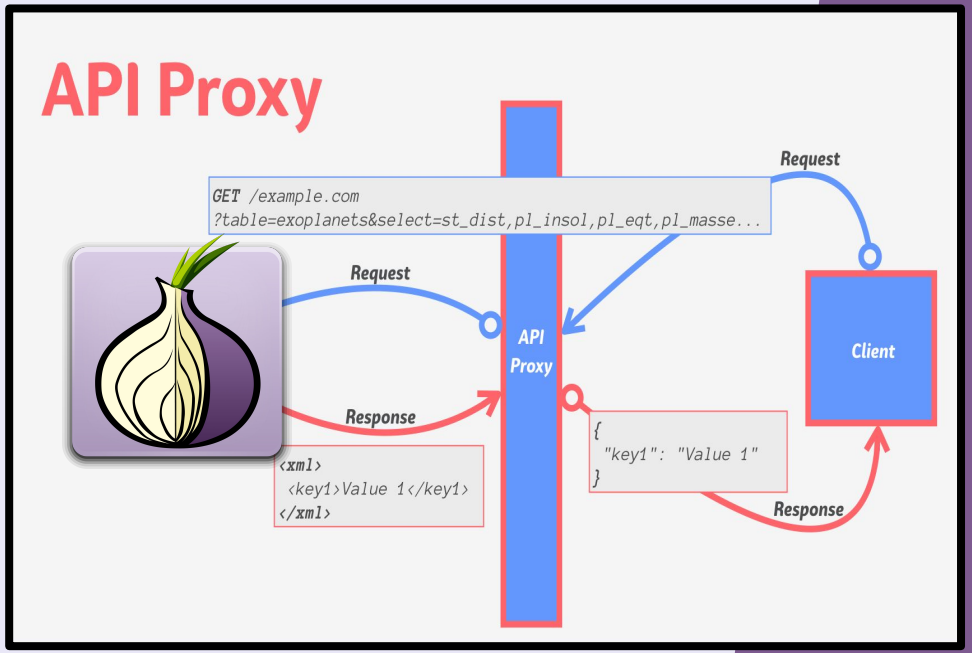
Application Layer

- MackYack uses a Client Server model for handling operations on the board.
  - Clients can receive and update new messages on the board
  - The Server is allowed to respond with the current state of the board and update the board when a new message is received.



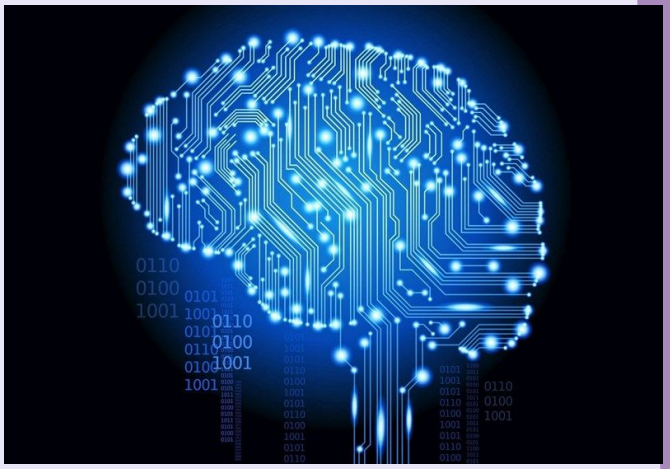
Onion Proxy

- Serves as a proxy within the Onion Routing Network by facilitating communication between a client and an Entrance Onion Router
- Manages the establishment of secure channels, message relay, and reception
- Abstracted from client so that it can be ran as a layer between Application layer and Onion Network
- Is utilized by MackYack clients for accessing the Onion Routing overlay network, running on the client's local computer upon the MackYack application's launch.



Onion Routers' Functionality

- Provides the essential functionalities for an onion router by handling messages for onion routers, proxies, or web servers by managing tables and cryptographic operations.
- Operates using a threaded service model, executing actions based on received cell (message) types.
- Employs cryptographic functions for encryption and decryption, including AES encryption for message security.
- Facilitates communication with other nodes and with destinations (AKA the **MackYack server**) by sending messages to specified destinations
- Handles destroying a circuit from the Onion Router when a client leaves the network.



Cells

- **Cells are the messages sent between the client, server, and onion routers.**
- **Create:**
  - Sent from Client to the first onion router to create a circuit, or...
  - Sent from OR to the new OR to be added to extend a circuit.
- **Created:** Sent from the newly-added onion router to the client confirming the creation/extension of a circuit.
- **Destroy:** Sent from Client to the first onion router to break down the established circuit (recursively).
- **Data:** Sent from the client to the last OR in the circuit (which is then passed to the server). Contains the data that will be sent out to the server.
- **Relay:** Sent by the client or an OR to the next OR in the circuit. Its function is for the receiving OR to relay the data to the next OR in the chain without interpreting the data.
  - **RelaySecret:** Contains the secret within a **Relay** cell to be decrypted by the OR.

Limitations of MackYack and the Onion Routing Implementation

- The last message sent from the exit node is not encrypted.
- Requires some tweaking before onion routing can be deployed w/ other applications.
  - Application Layer must interact with Onion Proxy to send messages. All server must use Java websockets.
- Not incredibly fault tolerant
  - Original specification protects against *break a node and see which circuits go down* attacks (or simply: nodes going offline unexpectedly). We did not implement this.
- Onion Routing in general is ***SLOW***.
  - Hopping between ORs adds distance + overhead
  - Variable performance between ORs
  - Congestion can exist on the Onion Network

New Stuff We Learned

- Greater understanding of crypto schemes & implementation of crypto ideas.
- Difficulties of building overlay networks.
- Reading & digesting academic papers.
  - Especially when it's not abundantly clear how to implement the details...
- Why onion routing provides anonymity.
  - The message is sent from a random other host (exit OR), so it isn't traced back to you.

Challenges

1. Needed to include temporary symmetric keys for Diffie-Hellman KEX, as the public key information (g^x) was too large to encrypt asymmetrically.
2. Trouble with adding BouncyCastleProvider() b/c of build.xml jar dependencies.
3. Had to change circuit IDs from integers to a String-representation of UUIDs because upon decryption the integers would randomly be in scientific notation?
4. **It took ages to figure out how to send data through the Onion Network in the reverse direction.**
    a. Had to add srcAddr/srcPort fields to Create cells and add it to a reverse lookup table from the given circuit ID. It was a mess...
5. Understanding what tables were needed in the ORs to forward data (and send it in the reverse direction).
6. Minor trouble encrypting the layers w/ Relay cells (wound up being an ordering issue in the Onion Proxy (client-side)).

Lets see it in action!

Try It Out!

~Fin