# Approach review

This project follows the system illustrated in figure X. There are two integral parts involved: the tunnel server that communicates with clients and the Internet, and the tunnel broker which authenticates clients and allows automatic tunnel setup.

There are multiple existing protocols to choose from for the tunnel broker and the tunnel server. And there exists options available to fulfil the goal of this project, therefore this project will not propose a protocol from scratch. As stated earlier, this project focuses a solution to address NAT issues with the proto-41 protocol. AYIYA and Teredo are solutions available. AYIYA is significantly simpler than Teredo in terms of how the system operates and procedures to handle a client connection. Hence, AYIYA is the preferred option for this project. Another advantage of choosing AYIYA is the ability to reutilise the AICCU client. In such a manner, the tunnel broker will implement the Tunnel Information and Control (TIC) protocol.

There are two paths for building the tunnel server. One is to develop a tunnel server from scratch, and the other is to work on the open-sourced sixxsd, with the latter developed and used by SixXS in their operation. The workload of creating our own tunnel server will be considerably heavy. Therefore, python is a desirable choice for the programming language in this case. It has rich libraries and good compatibility with network programming to allow faster development. The tunnel server may also be developed in C, which oversees a performance advantage than python. But the project emphasis on the implementation of the goals rather than how the program performs. So, python is preferred. In the case of utilising existing code, the first step is to verify whether the published code works and if any changes is needed. Then the project can focus on improving the code and AYIYA protocol. Options available includes replace the obsolete hashing algorithms, adding the ability to allocate arbitrary sized IPv6 addresses to clients, and adds more supported tunnel protocols (L2TP for example) to the tunnel server etc.

The tunnel broker is not a performance sensitive instance. The performance will be considered good enough if the client’s requests are handled within a sensible time. The use case of the tunnel server oversees each time a client starts the AICCU client. Once it obtains the tunnel information, they will disconnect from the tunnel broker. That should see a small number of active connections on the tunnel broker concurrently. Hence the tunnel broker does not require a high- throughput. TIC protocol requires the tunnel server listening on port 3874 and the tunnel broker will be a socket server. The above-mentioned reasons justify the use of Python for this part. Python has a good socket programming library; it allows multi-threading and is easy to code and debug. It also has other libraries such as MySQL connector to allow store and retrieving data in MySQL database. When doing background research, no source code was found for Sixxs’s TIC implementation. Hence, this project has to build the own implementation.

# Final approach

Sixxsd was tested and proved it still functions as intended. So, this project will utilise the previous work from Sixxs and address security and useability improvements. As MD5 and SHA1 no longer considered being secure, this project aims to implement SHA256 as a replacement to the obsoleted algorithms.

Sixxsd also enforce a block size of /44 or /56 as the address pool where the tunnelled IPv6s comes from, and tunnel endpoint shall only receive a block of /64. This kind of address planning is not very suitable for individual users given the difficulty of acquiring such an amount of IP addresses. Therefore, the project aims to tune the software such that it can accept any arbitrary sized address space, and ideally arbitrary sized IPv6 blocks for the tunnel’s communication. However, the address management structure employed by the legacy sixxsd may requires a major rework to support arbitrary tunnel size. So, this project aims to adjust the tunnel size to /127 for now, as to cope with the change to arbitrarily size address pool. Another advantage of setting the tunnel size to /127 is that the communication is limited between the host and the tunnel server.

The process of setting up an AYIYA tunnel relies on the AICCU client. Which, in turn needs a tunnel information server to retrieve information. Given Sixxs did not publish their tunnel information server, the other part of this project is to develop a tunnel broker that implements the TIC protocol. It is worth noting that the original TIC protocol states the challenge algorithm can be clear, md5 or cookie. It is unclear how cookie will serve as a challenge method in this scenario with resources available. SHA256 will be used by this project as the challenge method during authentication to improve security. The project kinds in mind that the tunnel broker may serves multiple clients, and each client may have more than one tunnels. Thus, it makes sense to store the accounts and configured tunnels in a database to facilitate management process.

Above changes to the TIC and AYIYA see a minor rework on the AICCU client to make it compatible with the newly added feature.

# The implementation

This project has built a tunnel server that supports 6in4 protocol, and partially completed SixXS’s heartbeat protocol in python. This python implementation helps verify the background research conducted through the early stage of the project and have a better understanding of the mechanisms. Figure X outlines the classes in the tunnel server. The SocketInterface handles receiving and sending packets. holds the protocol number of proto41, which is 41. Each instance of the SocketInterface can bind one handler to process a protocol. Each handle also has its own PacketFilter. The primary job of the PacketFilter is checking captured packets originated from a valid source. This include checking the source IP address of the encapsulated IPv6 address is not listed in the restricted address range, the packet is from a registered tunnel. The allocation of tunnelled IPv6 and its endpoint is stored in a text file.

A SocketInterface instance will have 2 pairs of sockets, one for handling IPv4 packets and one for IPv6 packets. Within each pair, one raw socket is will listen on a specific interface for the ethernet frames received. The other raw socket is charged to send out IP packets to the ethernet. On receiving a packet from the tunnel, it is validated first then pass to the handler’s decapsulate method to obtain the encapsulated IPv6 packet. Once the IPv6 packet is ready, it will be sent to the Internet with the inject6 method. The process of encapsulating an IPv6 packet from the Internet to the tunnel endpoint follows the same order but different functions will be called.

When the proto41 handler decapsulating a packet, it will recalculate the size of the IPv6 packet because IPv4 header might be padded [6in4 specs]. This implementation does not handle the next header in the IPv6 packets. And when encapsulating a packet, the important thing is to craft an IPv4 header for the packet. This implementation does not handle the fragmentation of carrying packets. The crafted header sees the type of service being set to 0, identification to 0, flags to 0, and fragment offsets to 0. Other fields in the header are filled in respect to the tunnel. The heartbeat handler is partially completed, only outlines the skeleton of the protocol. The handler will have an additional UDP socket to listen on port 3740 for heartbeat packets. The handler then must validate the packet, looking for a fixed starting text, registered endpoint, and genuine md5 signature. It features a tick function to mark tunnels without a heartbeat message for more than 120 seconds to offline. The design should see a modification on the PacketFilter to allow the binding of the heartbeat handler. It should not be used with SocketInterface because the heartbeat protocol is an extension to 6in4. The implementation of heartbeat handler is incomplete due to project management changes.

# Project management

At the start of the project, it is believed that Sixxs’s tunnel server is not published. When answering the question of will the SixXS code to be open-sourced, they explicated states that “We do not currently have plans to open source any code that is not already publicly available.” And as the tunnel server sixxsd, “offering the code base will not be necessarily useful for others.” reference to sunset page. Later when the bibliography was gathered, source code of sixxsd was found in a GitHub repository. The Sixxs website does not contain a reference to the repository. This discovery enables the tunnel server to be reused with the AICCU client, and this project shall focus on improving AYIYA.

# Further work

Handle next header. Complete Heartbeat.