# Results

The main goal of our project was to create an Anonymous chat program to allow users to communicate with each other anonymously. We have ran several tests to determine how practical our implementation is. These tests include anonymity tests, network utilization tests, and scalability tests. The result and analysis of the tests that we ran are discussed in detail below.

## Encryption and Decryption

To ensure the use of encryption wouldn’t add significant overhead to our protocol, we tested the time it takes for messages of various sizes to be encrypted and then decrypted. In order to do this we encrypted messages with a length with a range of 20 to 1000 increasing by 20 each time. As Figure 1 shows, the time required to encrypt a message is relatively minimum with the average time being around 0.15 milliseconds. The time required to decrypt a message is longer than encrypting, 8.13 milliseconds on average, but is still not that much of an overhead. These times are less than our metric of 250ms for encryption and decryption.



Figure 1 Encryption and Decryption Times

Encryption time does vary greatly depending on the length of the message, as can be seen below in Figure 2. Messages with a longer length take considerably less time to encrypt than messages with a smaller length. However after the message length gets to around 150 characters encryption time levels out at around 0.15 milliseconds. There is a spike around 640 characters, which is most likely due to the encryption algorithm used. These results show that the encryption over head is dependent upon the message length, and there for the use of our chat program. If a user sends a lot of longer messages, encryption overhead will be lower than if they send longer messages versus smaller messages. As can be seen in Figure 3, the decryption time follows a similar trend to encryption.

Figure 2 Encryption Time Trend

In addition to ensuring that encryption and decryption does not add too much over head to our protocol, we also wanted to ensure that a successful decryption will not expose a peer as a recipient. To do this we ran a similar test as above, but this time decrypted each message with the correct private key and an incorrect private key. In Figure 3, it can be seen that the time for an unsuccessful encryption does not differ by much of that of a successful decryption. An unsuccessful encryption is not always faster than a successful encryption. The average time difference between a successful and unsuccessful encryption is 0.0527 milliseconds. This difference is miniscule and will make it hard to determine the recipient of a message by analyzing decryption times.

Figure 3: Unsuccessful vs Successful Decryption Times

## Latency

An important issue with messaging protocols is network latency, we wanted to ensure that our distributed protocol does not increase the round trip time of messages by more than five times that of direct communication. Our results show the RTT of messages sent, including encryption time averaged around 12 milliseconds, the direct RTT averaged at about 1.3 milliseconds, and the RTT not including encryption (adjusted RTT) averaged 2.95 milliseconds. The adjusted RTT is only around twice that of the direct RTT. Figure 4 shows the trends of the RTT for each of the twelve messages sent during our test.

While these results meet our metric for success, they do not accurately represent the RTT with a large number of clients participating. This test was ran with only three clients, all connected on the same Local Area Network (LAN) as we were unable to run a more comprehensive test due to resource and time limitations. As a result of this each of the nodes had each other as a peer, resulting in their RTT being very similar to their Direct RTT’s which is evident in Figure 4. We believe that had we been able to test this with around ten clients, on a more diverse network, not just a LAN, we would have seen very different results.

Figure Compared Round Trip Times

## Network Utilization

The network utilization of our messaging protocol is very important, as it uses controlled flooding to send messages, we didn’t want our protocol to use too many resources or saturate the network. To test our utilization we ran a test with six clients running locally all sending messages to each other periodically as described in Chapter 3. We then compared these results to a session of web browsing. As you can see in Figure 5, the web browsing session utilized the network much more than Anonychat. Web browsing had almost ten times the total number of packets, eight times more packets per second, double the average packet size, and almost 20 times the bytes per second. Our network utilization was relatively small compared to that of web browsing, and this was analyze six clients traffic. Even with using controlled flooding, our network utilization remained much lower than a simple web browsing session.



Figure 5 AnonyChat and Web Browsing Utilization

## Scalability

The scalability of our protocol is something we need to consider. We ran a test involving 6, 12 and 18 clients, to determine the effect on the name server peer shuffling, and the average processing time of messages. In Figure 6, it shows that as the number of clients increases, so does the average reshuffling time for the peer list, however this rate is less than a linear rate, which meets our goal of no more than linear growth in time. We also compared the average processing time for each message, as Figure 7 shows, as more clients were added the processing time decreased linearly. Using this we inferred that our name server implementation will scale relatively easy for a large number of clients.

Figure Average ANS Reshuffle Time vs Number Clients

Figure Average Message Processing Time vs Number Clients

## Anonymity

As we have developed an anonymous messaging protocol, the anonymity of the source and destination of messages is very important. As discussed in Chapter 3 we ran an anonymity test to determine if the contents of the packets remained confidential and the message was not easily traceable. After analyzing the Wireshark capture from our test, we have determined that the contents of the message remain encrypted, and all messages are sent with a fixed size. However as we were unable to implement peer-level encryption, we do not have results to verify that it does indeed prevent message tracing. The source of the message does remain anonymous, as all packets containing messages have the same size of 1416, and as shown earlier there is no noticeable different between an unsuccessful and successful decryption.