AnonyChat: An Anonymous Messaging Platform

Team 17

Mitchell Caisse

Mark Fitzgibbon

Benjamin Korza

# Chapter 1: Abstract

Many internet users desire a means of communication that guarantees privacy so that they can be confident their identities will remain unknown and their messages will be safe from prying eyes. This is especially the case nowadays as public concern increases about organizations spying on users in both foreign and domestic networks. Networks such as TOR already provide such a service; however, do not do so in a peer-to-peer manner. In this paper, we introduce Anonychat, an anonymous peer-to-peer chat protocol designed to keep the identities of its users a secret. Peers are linked by a central server, and utilize encryption in a method similar to onion routing to hide the contents of messages between users. Messages are sent out to nodes in a broadcast format to obfuscate the intended recipient.

In order to evaluate our success, we assessed two areas that would be crucial to the success of the protocol: security and practicality. To address security, we tested that our message source and destinations remain anonymous as well as message content remained secure. We found that message content remained secure, and our implementation would allow of source and destination to remain anonymous. Our scalability and practicality tests revealed that it would indeed scale well, and encryption and network did not add too much latency. Over all we received positive results from our testing.

# Chapter 2: Introduction

Concerns about privacy seem to be getting a lot more attention from the media in recent years. There have been many reported cases of companies spying on users in their networks, so this should come as no surprise. In addition, social media websites such as Facebook have been accused of buying and selling user information. As such, a public concern about privacy is justified. The issues presented here could also be important among people who work at corporations as well, considering packet sniffers or other external programs can be used to intercept messages and identify information about them. And finally, there is the concern of government monitoring. In the events that unfolded during Arab Spring, the need for an anonymous communication network became paramount as governments shut down cellular networks and hacked into Facebook pages. Our Anonychat will attempt to resolve this need by implementing a system in which packets can neither be read from external programs nor traced with certainty.

Several protocols already exist that provide anonymous chat. For instance, the Invisible Internet Project (I2P) is an ongoing effort to build a free, open source, and anonymous internet. However, I2P is designed at the network layer, and therefore restricts its users by forcing them to use the I2P network. Freenode is another protocol that uses encryption to hide the identity of a user who sends packets over the network. That being said, Freenode uses direct connections that allow its senders to be traced. Competition also extends to the peer-to-peer style communication network Skype, which uses a similar connection system we intend to implement (a central server to start, then peer-to-peer communication afterwards). However, Skype’s main focus is not anonymity, and its peer-to-peer connections use a direct method. Therefore, any anonymity the system does provide could cease to exist later. Other projects are in development squarely to accomplish anonymity; Quassel and Rust are two such examples. However, neither of these projects use the peer-to-peer model we will attempt.

To compete with these various systems, we have implemented a peer-to-peer chat system. Clients ping a central name server to determine and connect to their peers. Once connected, users can communicate with anyone else on the channel. Anonymity is the central focus of our protocol, providing us with an angle different than most of our competition. Second to anonymity is terms of importance is practicality; our protocol must provide a rapid means of communication without saturating the network to be truly successful. We maintain anonymity by broadcasting all sent messages and using two layers of encryption in a way similar to onion routing. When receiving messages, users attempt to decrypt it using public-private keys, and rebroadcast the message regardless of success. We maintain practicality by illustrating that a large network of clients can be sustained without suffering excessive slowdown or consuming too much of the network’s resources.

# Chapter 3: Methodology

In order to accomplish our goal we have designed a messaging protocol that will provide anonymity through the use of a distributed network of peers and multiple layers of encryption. In order to test our design we have implemented a test platform in C for Linux. This test application will allow us to perform testing on our design to ensure that it meets our goals and does not have a huge performance overhead. The details of our design and testing methods will be discussed in detail below.

## 3.1 Messaging Protocol

The Anonychat messaging protocol consists of a set of distributed clients that act as peers in transmitting messages, as well as a centralized name server to inform clients of available peers. These clients will send messages amongst each other using controlled flooding, encryption between the source and destination peers, and encryption between each intermediate peer. In order for this to work, we needed a way to ensure that each peer has a connection to all of the other peers. This is the role of our name server; it distributes peer lists to each peer, and make sure that they can all communicate.

### 3.1.1 Anonychat Name Server

The Anonychat Name Server (ANS) is responsible for creating a network of peers that provides each peer with a route to each other. Our protocol does not require a specific implementation for creating this network, but there are certain conditions that must remain true. The peer network must be changed periodically, allowing peers to send messages on different paths. The peer network must assign peers in a random fashion; it is recommended that peers with similar IP address prefixes not be assigned to each other. The peer network must not isolate any peers or group of peers; they must all form a small-world graph. The ANS must also maintain connection state on all peers, and be able to react accordingly when a peer drops from or joins into the network.

In our test implementation we maintained state by requiring each peer to uphold a TCP connection with the server. This allowed us both to determine when a peer has connected/disconnected and to send out peer updates to the peer when required. Our node network was implemented using a connection graph, which is detailed below:

1. An internal list of clients is re-arranged to occur in a random order. This should ensure that connections appear random as well for the following steps.
2. If there are more than five nodes connected, each of the nodes is arranged in a ring, and is connected to the two nodes ahead and behind it in the ring as well as the node directly across from it. Otherwise all of the nodes are simply connected to all of the other nodes. These connections are bidirectional, meaning that if node A is able to send to node B, node B is also able to send to node A. This situation is incidental, however and should not be assumed by the clients to always be true.
3. When the central server’s graph is completed, a message is sent to every node informing it of its assigned peers. No node is informed of who is able to send messages to it.
4. This cycle will repeat either (1) every time a client joins or leaves if the number of clients in the network is less than a number set by the ANS, or (2) every 30 seconds if there is a larger number of clients. The second case implies that, in larger networks, new clients may have to wait up to 30 seconds before it is able to broadcast messages, but this should also keep the name server from flooding the network. Messages in transit when connections are re-assigned behave as if nothing happened.

### 3.1.2 Anonychat Client

Our protocol implements two levels of encryption, message level and peer level. Both levels of encryption will use asymmetric encryption, implemented by the RSA algorithm. The first level of encryption, between the destination and source, guarantees that the destination knows the message is for itself. The source does not know any information about the destination other than its public key, and the fact that no one else can read the message. At this level of encryption keys are maintained by the user. The second level of encryption, between each intermediate peer, ensures that the source and destination of the messages remain anonymous. This level also ensures that the signature of the message changes as it is sent across the network. However, the content of the message remains the same. This approach is similar to onion routing; because the message signature is constantly changing, it disallows the message from being traced through the network. The keys used at this level are hidden from users.

Peers communicate with each other via controlled flooding. Specifically, they communicate by broadcasting the message to all of their peers, and their peers will then broadcast the message to their peers and so on. To determine if a received message is new, a hash of the source/destination encrypted message can be compared against a hash of previously received messages. When a peer receives a message that it has never seen before, it will always broadcast the message to its peers, regardless of whether it was the intended destination or not.

The message protocol that used by our messaging platform is relatively simple. It consists of a ten character or less command with the body following, delimited by a space. All messages have a fixed size to prevent an adversary from extrapolating data about users from message size. In our implementation each message is padded with a string of random characters, before the peer level encryption. This ensures that the padding is not obvious and randomness will not create a unique signature on messages.

## 3.2 Testing

To verify that our protocol is viable, has sufficient performance, and works as intended, we have run several tests on various attributes of the protocol. These tests include measuring message latency, latency introduced by encryption, time differences in successful and unsuccessful decryption, and the network utilization of our controlled flooding implementation. These experiments were run via a series of simulations on both a local machine and four computers spread out on different networks.

### 3.2.1 Latency

Given the indirect method our protocol uses to transmit messages, we expected to incur additional round trip time (RTT). However, as a metric for success we didn’t want this additional latency to be much bigger than the latency found in direct communications. To ensure our latency is acceptable, we set up test implementation on our four test machines. We then sent a series of ten messages between each machine using our messaging protocol, and recorded the RTT of each message. Next, we will sent another ten messages between the each of the four hosts themselves using ping, and again record the RTT of each message. After collecting the RTT of both messages, we compared them and verified that the RTT using our protocol is not greater than five times the direct RTT.

### 3.2.2 Encryption Overhead

Encryption plays a big part in our messaging protocol, but it can also be a very expensive computation. For the sake of rapid communication, we didn’t want encryption to add a huge overhead to our protocol. To ensure this is not the case, we tested the time it takes for messages of various different sizes to be encrypted and then decrypted. We sent messages with lengths in a range starting from 20 to 1000, increasing by increments of 20 each iteration. Using these tests, we calculated the time each of these messages took to encrypt and decrypt. This will add additional latency to message sending, but we hoped it would be less than half of the network latency.

In addition to testing the overhead added by encryption, we also needed to test that the time difference between a successful and unsuccessful decryption is not noticeable. If this difference is noticeable, it could allow an adversary to determine if a particular node was the intended recipient of a message. To do this we performed a similar test to our total encryption/decryption overhead test mentioned above. However, this time we decrypted the message twice: once with the correct private key, and the second time with an invalid private key. Comparing these values allow us to determine if a successful decryption is noticeable.

### 3.2.3 Network Utilization

The network utilization of our protocol also needed to be tested. We don’t want our protocol to use a vast majority of the available bandwidth by flooding messages across the network. In order to test our utilization we ran a simulation with around six clients running locally. The simulation lasted for five minutes with each client periodically sending messages to other clients. To do this, each client picked a random time from five to 35 seconds. The client then sent a message after this time expired and repeated the process. As this occurred we analyzed the network using Wireshark and took notes of how many packets were in the network for five second intervals. We also ran a five minute simulation of normal computer use, such as web browsing, and heavy network use such as a file transfer. This allowed us to compare our protocol’s network usage against these two, giving us a sense of how our network utilization compares to normal and heavy network use.

### 3.2.4 Scalability

As a chat program could possibly be used by different amounts of people at a time, it seemed paramount we test how well our messaging protocol will scale. To do this we performed a test similar to the network utilization test; however, instead of analyzing the network, we analyzed the resource usage of each node and the ANS. During this simulation we recorded the number of messages each node processed, and the processing time of each of those messages. We then recorded the RTT of each message to determine the effect more clients have on it. We also analyzed the ANS, measuring how many times it reshuffles the peer list and how long each of those peer reshuffles took. Unlike the network utilization test, we ran this test on a range of clients from six to 18, increasing by six clients each time. This gave us a clear measure of how well our protocol scales with varying numbers of clients.

### 3.2.5 Anonymity

The final test that we performed is the anonymity test of the source of our messages. To do this, we ran a simulation with around ten clients sending messages back and forth to each other at varying intervals. We used Wireshark to examine the sent packets, and took note of whether the packets were encrypted as well as confirmed there were no identical packets sent between peers. This helped us tell if it is possible to determine the source of a particular message by analyzing the packets sent between nodes.

# Chapter 4: 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.

## 4.1 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

## 4.2 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 4 Compared Round Trip Times

## 4.3 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

## 4.4 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.

We had initially also wanted to record the RTT of each of the messages received during this test, however due to unsynchronized clocks, we were unable to obtain accurate results for these values, as even a time difference of two minutes had completely skewed our data. If each of the machines used during the test had synchronized clocks we would have been able to get better data, and are more complete analysis about scalability by comparing the RTT. However if the RTT followed the same trend as the reshuffle time and message processing time, it would not have a negative effect on the scalability of our messaging protocol.

Figure 6 Average ANS Reshuffle Time vs Number Clients

Figure 7 Average Message Processing Time vs Number Clients

## 4.5 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.

# Chapter 5: Conclusion

We have implemented a tested a message protocol that guarantees the anonymity of the source and destination of it’s messaging by the use of controlled flooding, multiple layers of encryption, and fixed sized packets. We have also guaranteed message confidentially by the use asymmetric encryption of messages sent. We have also tested our implementation to test its scalability and practicality of the protocol. We found that while we were unable to get accurate latency tests that the protocol performed quite favorably and used less bandwidth that your average web browsing session. This compared with the ability of the protocol to scale, proves that we have accomplished our goal in creating a message protocol that is anonymous and confidential.

Our implementation is far from perfect and has room for future improvement. It can be improved upon by adding more advanced messaging features and nick names for clients. Right now it just supports basic message sending as more of a proof of concept. A proper implementation of peer level encryption could be added as well, currently our protocol defines that it is necessary however we were unable to implement it successfully. Taking these changes into consideration, our design still accomplishes its goal.