 Brandon Li  
 CPSC 490: Special Projects  

 **Tolerating Server Failures in an Anytrust Anonymous Communication Protocol**

**Some Background**

Anonymous online communication is becoming increasingly desired and necessary by virtue of the now near ubiquity of Internet users and the cultural push towards unmonitored and uncensored speech. Several online anonymity techniques, most popularly Tor, have been developed in recent years, but with them a number of de-anonymizing methods have been developed as well. For example, Tor is susceptible to attacks based on network traffic analysis. Therefore, modern anonymity techniques are commonly forced to weigh a tradeoff between scalability and strength of anonymity. Dissent is a protocol that attempts to circumvent this tradeoff and achieve both scalability and strong anonymity [1].

**Current State of Dissent**

Dissent, a distributed protocol that relies on DC-nets and verifiable shuffles, was designed as a scalable strong-anonymity provider. However, there were several noted limitations that prevent the system from being fully practical. One of those limitations is that Dissent can handle client network churn (servers simply omit offline clients during each round), and it can even handle malicious behavior by either clients or servers (Dissent provides a blaming mechanism to enforce accountability, and it guarantees anonymity for clients if at least a single participating server is honest), but it cannot tolerate server churn. In fact, because all of the clients need to know in advance all of the servers that are participating in the round, if even one of the servers were to go offline, the protocol would stall indefinitely. Distributed systems are generally expected to ensure both “safety” (bad things do not happen) and “liveness” (good things happen eventually), and so this system is said to fail “liveness” in the case of a server failure [1].

**Proposed Contribution**

I aim to extend Dissent to tolerate server failures without significantly changing the existing Dissent protocol. Specifically, I plan to introduce a message exchange between a client and the group of servers in which the client will be notified of which servers to consider as part of the round (which will be a subset of the servers that are in the group). Since clients need to know all of the servers that will participate before the round even begins, this message exchange must happen before every round. The client will only use those servers to encrypt their message (using the shared secrets) as opposed to the entire set of servers that they could connect to, and only the servers that are selected for that round will actually participate in the
server-side of the protocol. The “liveness” condition will be reinforced as follow:

1. If any of the non-participating servers fail, the round will proceed as normal.
2. If any of the participating servers fail, the other participating must be able to recognize this (through perhaps a timeout mechanism). Then, a new “view” $V_{k+1}$ will be instated, in which there is a different set of participating servers. The round will be terminated and a new round will be initiated in which the set of active servers in the new view will participate in the protocol.

These views must be determinable by every one of the servers, either through a deterministic algorithm or through communication amongst the server group. Any view change will be reported to the clients, either when a client wishes to submit a message to the round or immediately after every time that the view changes. Ideally, the selection of active service groups, as well as the view changes, are all performed on the server-side. I would like for this protocol extension to create as little additional work for the client as possible.

The way that I currently envision selecting the server groups is through some well-known group membership algorithm (so that all servers can individually and deterministically know all of the servers associated with a particular view), although as mentioned, it would also be possible for
servers to learn of the active groups by communicating through messages. In principle, since all of the Dissent groups (the entire group, not the active servers in a particular view) are currently static, it is possible to have all of the views predetermined before any of the rounds and have the servers be able to cycle through them when a view change is issued. The exact details of this will be worked out throughout the implementation of this project.

In order to signify a view change, at least one of the servers must timeout and signal a requested view change. Perhaps all of the active servers will wait for the majority of the other servers to send it some “VIEW CHANGE REQUEST” message to determine when a view change actually occurs during a round, as modeled after the design that Castro and Liskov describe [2].

Implementation

Implementation will be done incrementally, and the initial mockups will be implemented outside of the main Dissent codebase and may abstract the notion of a server and client. For example, the first iteration of the project will simply demonstrate servers communicating a group to the clients that the clients then use to form their ciphertexts. The next iteration might add the view change feature. After the core functionality is implemented, I will integrate the work into the Dissent code.

The initial mockups may be implemented using a scripting language, but the integrated product will be written in Qt.

Deliverables

In addition to implementing the project, I will create and run tests to verify the behavior of the added functionality, as well as measure the impact of the extensions (i.e. look at how many server failures can actually be tolerated). I will also run performance diagnostics to determine any added overhead. Finally, I will document my work in a research paper.

Non-goals

This project will not attempt to implement a full state machine replication service for Byzantine Fault Tolerance. Furthermore, this project as it is currently planned is not intended to resolve malicious server behavior other than server crashes (malicious server and client behavior are already handled by the existing protocols).