# Exploring attacks on the Rapid membership system

Madhav Narayan
narayan@cs.utexas.edu

## Abstract

Maintaining membership information is an important task in large-scale distributed systems that change in size frequently, and often quickly. In order for membership information to be useful to the system, however, it must be *consistent* and *stable*; that is, each member in the system should see the same *view* of the system, and the membership view should not oscillate or change erratically. *Rapid* is a membership service that provides strong consistency and stability guarantees. In this project, we explore attacks on Rapid and its protocol. In particular, we study how Rapid's membership protocol performs in the presence of "adversarial" members trying to induce an incorrect membership view. We give bounds on the susceptibility of correct members to adversarial members in the system, and show that attacks are highly unlikely given a "real-world" system configuration.

## 1 Introduction

Many distributed systems rely on accurate information about the members in the system in order to run correctly and achieve high performance. To do so, distributed systems employ membership services that maintain information about the different members (say, processes) in the system. A *view* refers to a set of processes that are considered to be members of the system. For a membership service to be effective, it should provide membership views that are consistent and stable. A membership view is consistent if each member in the system sees the same view. The lack of a consistent membership view can affect a system's correctness: for example, *consistent hashing* (Karger et al., 1997) relies on having accurate information about the servers in the system to direct requests to the correct servers. Having an inconsistent membership view can also affect the performance of the system: for example, *Cassandra* (*Cassandra-3831: scaling to large clusters in GossipStage impossible due to calculatePendingRanges. Cassandra-6127: vnodes don't scale to hundreds of nodes.*)), an open-source database, may only allow one member to join the system at a time. A membership view is stable if the set of members does not change erratically, that is, members do not "oscillate" in-and-out of the system. Having an unstable membership view can increase the frequency of failure recovery operations, which tend to significantly affect the performance and availability of the system.

*Rapid* (Suresh et al., 2018) is a recently developed distributed membership service that provides strong consistency and stability guarantees. These guarantees improve on those provided by similar distributed membership systems. In addition to strengthened consistency and stability, Rapid is also able to initialize clusters of 2000 members faster than similar systems, and is able to remain robust in various kinds of failure scenarios.

Our goal for this project is to explore attacks on Rapid. In particular, we study how a group of "adversarial" members could target Rapid's protocol and diverge the membership view from its actual, correct state. We wish to show bounds on the extent to which individual members and consequently, the overall membership view, can be affected by adversarial members deviating from the protocol. This model of an adversary trying to attack the protocol serves as a natural and simple way of designing scenarios in which Byzantine failures can break the protocol, leading to incorrect membership views across the system.

## 2 Background

Membership services are widely used in distributed systems. Several membership systems and protocols have been developed and used in systems with various different requirements and sizes. Many of these systems offer varying degrees of consistency and stability. For example, membership systems that employ gossip (*Apache Cassandra*, *Akka*, *Serf*, *Redis*) to disseminate status notifications are resilient, but are unable to offer strong consistency and stability guarantees. Similarly, many membership services use *logically centralized* configurations (*Apache Zookeeper*, Burrows, 2006). In such configurations, an auxiliary collection of nodes is designated responsible for maintaining membership in the system. While these systems can be easier to reason about and provide strong consistency, they are unable to offer a resilient system. Rapid improves upon these services by providing strong stability and strict consistency guarantees.

Distributed systems have long been assessed for their resilience in the presence of adversarial members. For example, several systems have evaluated and explored adversarial attacks on their protocols. The *Fireflies* system (Johansen et al., 2015) which balances tolerance to Byzantine failures with scalability. It evaluates attacks on its protocol and shows the pseudorandom mesh structure it uses to organize processes makes the system resilient to adversarial attacks. In fact, Fireflies achieves this resilience despite providing full views of the system membership to all members, which in turn reduces the cost of messaging. Other systems, such as *Brahms* (Bortnikov et al., 2008) show worst-case divergence bounds of their protocols in response to adversarial attacks. Some systems also consider "Sybil attacks" (Douceur, 2002), in which adversarial members adopt different identities, thereby having more control over the system than they would with just a single identity.

In describing our attacks, we often borrow ideas and notation from these studies. In addition to considering some general attacks described in the studies above, we also consider attacks specific to Rapid's design and protocol.

## 3 The Rapid System

In this section, we briefly describe Rapid's design, protocol, and relevant parameters. Rapid's protocol involves three main components: monitoring, multi-process cut detection, and view-change consensus. Members monitor each other, broadcasting alerts about changes in a member's status to the entire system. Members aggregate alerts received about other members in the system, and propose configuration changes to the membership view based on the changes they are notified of in the system. Members then decide on the updated membership view by achieving consensus over the proposed view changes.

### 3.1 Monitoring

Rapid facilitates the notification of membership changes by having members monitor each other. Each member is monitored by $K$ *observers*, and in turn monitors $K$ *subjects*. This monitoring setup is realized by constructing $K$ pseudorandom rings.

Each ring is essentially a random permutation of all the members in the system. On each ring, a member is monitored by the member immediately preceding it, and itself monitors the member that immediately succeeds it. In this setup, it is possible for a member to monitor one of its own observers, and both these monitoring channels operate independently of each other. A key property of Rapid's monitoring topology is *expansion* — a characteristic signifying the connectivity of the monitoring graph — which ensures that correct members are notified of failures with high probability. In addition, since each member has exactly $2K$ monitoring relationships, the addition or removal of a member incurs exactly $2K$ modified monitoring edges. Similarly, the per-process monitoring overhead is $O(K)$ per round.

## 3.2 Multi-process cut detection

As observers see the status of their subjects change, they broadcast alerts to all the members in the system. An observer broadcasts a JOIN alert about a subject when it receives a request from the subject to join the system. Similarly, an observer broadcasts a REMOVE alert about a subject when it is unable to reach the subject. Given that the monitoring topology exhibits expansion, each correct member in the system receives alerts with high probability. Each member maintains a *tally* per member of the number of alerts it receives about each member in the system. A $tally(m)$ for a member $m$ represents the number of alerts received by a process from the observers of $m$. It is possible for two different correct members to have different tallies for another member $m$, however this occurs with low probability.

Since Rapid seeks to provide stable membership views, members must wait until the information they receive about other members is stable before proposing a view change. This is done by defining thresholds for the number of alerts received about a process below which information about a process is not considered stable. Rapid employs parameters $H$ and $L$ (where $1 \leq L \leq H \leq K$) as follows:

- A member $m$ is considered to be in *stable report mode* if $tally(m) \geq H$.
- A member $m$ is considered to be in *unstable report mode* if $L \leq tally(m) < H$.
- If $tally(m) < L$, the alerts about $m$ are considered to be noise.

Once *at least* one member is in stable report mode and *no* members are in unstable report mode at some member, the member may propose a change to the membership with the new information it has received about the other members in the system.

## 3.3 View-change

Once members propose changes to the membership change, they participate in a consensus process to reach agreement on a single *view-change*. A view-change is simply a change in the membership view of the system. The goal of this process is to agree on a single proposal. The consensus protocol is based on Fast Paxos (Lamport, 2006) and decides on a proposal once there is a quorum larger than three quarters of the membership. As a result, if more than three quarters of the membership have identical proposals, that proposal will be selected. Members gossip bitmaps of "votes" for each (unique) proposal, with each member voting for a proposal by setting a bit. A proposal is decided upon once a member receives a proposal for which at least three quarters of all the members have voted. If Fast Paxos is unable to decide on a proposal, consensus is then achieved using classical Paxos (Lamport, 2001).

# 4 Attack models

We consider the following attacks on the system:

- *Aggressive* attacks: An aggressive attack is one in which a corrupt observer of some correct process $p$ tries to have $p$ removed from the system. If at least $H$ of $p$'s $K$ observers are corrupt, they would be able to submit an *incorrect but stable* report about $p$. Other members receiving these reports would then propose for $p$ to be removed from the system.
- *Passive* attacks: A passive attack is one in which a corrupt observer of some faulty process $p$ tries to keep $p$ in the system. In such an attack, $p$'s corrupt observers would see that $p$ is faulty, but choose *not* to broadcast a REMOVE alert to the rest of the system.
- Attacks on Rapid's aggregation rules: Since Rapid prevents members from proposing configuration changes until all processes at a given member are in stable report mode, corrupt observers of a process $p$ could collude to issue *at least L* but *fewer* than $H$ alerts about $p$, thereby putting it in unstable report mode. In this way, corrupt observers could slow the system down by forcing members to delay proposing configuration changes.
- Attacks on consensus: As members propose changes to the membership view, they participate in voting process to reach consensus (agreement) on a single proposed membership view. This consensus protocol is built on *Fast Paxos*. If there is a quorum of at least three quarters of the membership on some proposal — that is, at least three quarters of the members in the system agree on the same proposal — that proposal is selected. In fact, because of the expansion of Rapid's monitoring topology, most processes have, with high probability, matching proposals. If Fast Paxos (Lamport, 2006) is unable to decide on a proposal, consensus is achieved using classical Paxos (Lamport, 2001).

The aggressive and passive attacks were introduced and explored in *Fireflies* (Johansen et al., 2015).

# 5 Analysis

Consider a system of $N$ processes (members). Each process is monitored by $K$ observers and in turn monitors $K$ subjects. The assignment of observers and subjects for each process is pseudo-random. Let each process be corrupt (that is, assume the role of an adversary) with uniform and independent probability $p_{\text{corrupt}}$[1]. We consider a process to be *safe* if none of its $K$ observers are corrupt.

**Lemma 5.1.** *The expected number of corrupt processes in the system is*

$$N \cdot p_{\text{corrupt}}$$

.

*Proof.* Each individual process is corrupt with uniform and independent probability $p_{\text{corrupt}}$. Using the linearity of expectation, then, adding the expectations of $N$ individual processes being corrupt, we expect the number of corrupt processes in the system to be $N \cdot p_{\text{corrupt}}$. □

**Lemma 5.2.** *The expected number of processes with no corrupt observers is*

$$N \cdot (1 - p_{\text{corrupt}})^K.$$

*Proof.* For any process, assume we have selected $K$ observers at random. Each of these observers is correct with probability $1 - p_{\text{corrupt}}$. The probability, then, that all $K$ observers are correct is $(1 - p_{\text{corrupt}})^K$. As a result, for $N$ processes, the expected number of processes with no corrupt observers is $N \cdot (1 - p_{\text{corrupt}})^K$. □

## 5.1 Aggressive attacks

A process is considered susceptible to an aggressive attack if at least $H$ of its $K$ observers are corrupt.

**Lemma 5.3.** *Let C be the number of corrupt observers for a process. The probability that at least*

*H of the process's K observers are corrupt is exponentially small in H, K, and $p_{\text{corrupt}}$. Specifically,*

$$\Pr(C \geq H) \leq \exp\left(-\frac{\left(H - Kp_{\text{corrupt}}\right)^2}{3Kp_{\text{corrupt}}}\right).$$

*Proof.* For each observer $i \in \{1, 2, \ldots, K\}$, we define the indicator random variable $C_i$ such that

$$C_i = \begin{cases} 1 & \text{if observer } i \text{ is corrupt,} \\ 0 & \text{otherwise.} \end{cases}$$

Here, $\mathbf{E}[C_i] = \Pr(C_i = 1) = p_{\text{corrupt}}$. The number of corrupt observers $C$ for a process is given by $C = \sum_{i=1}^{K} C_i$. We wish to bound the probability that $C \geq H$. We use the following version of the Chernoff bound (Equation 4.2, Mitzenmacher and Upfal, 2017):

$$\Pr(X \geq (1+\delta)\mu) \leq e^{-\mu\delta^2/3}$$

where $0 < \delta \leq 1$. We have

$$\Pr(C \geq H) = \Pr(C \geq (1+\delta)\mu) \leq e^{-\mu\delta^2/3}$$

Now, we have

$$\mu = \mathbf{E}[C] = \sum_{i=1}^{K} \mathbf{E}[C_i] = K \cdot p_{\text{corrupt}}$$

and $H = (1+\delta)\mu$. Solving for $\delta$, we get

$$\delta = \frac{H}{K \cdot p_{\text{corrupt}}} - 1$$

In order to employ this bound, we require $0 < \delta \leq 1$.

$$\begin{aligned} &\implies 0 < \frac{H}{K \cdot p_{\text{corrupt}}} - 1 \leq 1 \\ &\implies 1 < \frac{H}{K \cdot p_{\text{corrupt}}} \leq 2 \\ &\implies K < \frac{H}{p_{\text{corrupt}}} \leq 2K \end{aligned}$$

We know $H \leq K$, which gives $\frac{H}{K} \leq 1$. From $K < \frac{H}{p_{\text{corrupt}}}$, we have $p_{\text{corrupt}} < \frac{H}{K}$. Indeed,

$$p_{\text{corrupt}} \leq \frac{1}{2} \leq \frac{H}{K} \leq 1$$

Similarly, we have $\frac{H}{p_{\text{corrupt}}} \leq 2K$, which gives $\frac{H}{K} \leq 2p_{\text{corrupt}}$. Since $\frac{H}{K} \leq 1$, we have $1 \leq 2p_{\text{corrupt}}$. Now, applying the Chernoff bound, we have

$$\begin{aligned} \Pr(C \geq H) &\leq e^{-\mu\delta^2/3} \\ &= \exp\left(-\frac{\mu\delta^2}{3}\right) \\ &= \exp\left(-\frac{\left(Kp_{\text{corrupt}}\right)\left(\frac{H}{Kp_{\text{corrupt}}} - 1\right)^2}{3}\right) \\ &= \exp\left(-\frac{\left(Kp_{\text{corrupt}}\right)\frac{\left(H - Kp_{\text{corrupt}}\right)^2}{\left(Kp_{\text{corrupt}}\right)^2}}{3}\right) \\ &= \exp\left(-\frac{\frac{\left(H - Kp_{\text{corrupt}}\right)^2}{\left(Kp_{\text{corrupt}}\right)}}{3}\right) \\ &= \exp\left(-\frac{\left(H - Kp_{\text{corrupt}}\right)^2}{3Kp_{\text{corrupt}}}\right) \end{aligned}$$

□

## 5.2 Passive attacks

Rapid does not allow conflicting alerts about the same member; that is, there can not be JOIN and REMOVE alerts about the same member. As a result, we assume that if at least $H$ of a member's $K$ observers are correct, then status changes will be correctly reported. If fewer than $H$ observers are correct, however, then the remaining corrupt observers may choose *not* to notify the rest of the system if the member becomes faulty. We would like to show an upper bound on the probability that a member has fewer than $H$ correct observers.

**Lemma 5.4.** *Let X be the number of correct observers for a process. If H is at most the expectation of X, then the probability that fewer than H*

*of the process's K observers are correct is exponentially small in H, K, and $p_{\text{corrupt}}$. Specifically, if*

$$1 < H \leq \boldsymbol{E}[X],$$

*then*

$$\Pr(X < H) \leq \exp\left(-\frac{\left(K(1-p_{\text{corrupt}}) - H + 1\right)^2}{2K(1-p_{\text{corrupt}})}\right).$$

*Proof.* For each observer $i \in \{1, 2, \ldots, K\}$, we define the indicator random variable $X_i$ such that

$$X_i = \begin{cases} 1 & \text{if observer } i \text{ is correct,} \\ 0 & \text{otherwise.} \end{cases}$$

Here, $\mathbf{E}[X_i] = \Pr(X_i = 1) = 1 - p_{\text{corrupt}}$. The number of correct observers $X$ for a process is given by $X = \sum_{i=1}^{K} X_i$. We wish to bound the probability that $X < H$. We use the following version of the Chernoff bound (Equation 4.5, Mitzenmacher and Upfal, 2017):

$$\Pr(X \leq (1-\delta)\mu) \leq e^{-\mu\delta^2/2}$$

where $0 < \delta < 1$. We have

$$\Pr(X < H) = \Pr(X \leq (H-1)) = \Pr(X \leq (1-\delta)\mu) \leq e^{-\mu\delta^2/2}$$

Now, we have

$$\mu = \mathbf{E}[X] = \sum_{i=1}^{K} \mathbf{E}[X_i] = K \cdot (1 - p_{\text{corrupt}})$$

and $H - 1 = (1-\delta)\mu$. Solving for $\delta$, we get

$$\delta = 1 - \frac{(H-1)}{K(1-p_{\text{corrupt}})}$$

Now, applying the Chernoff bound, we have

$$\begin{aligned}
\Pr(X < H) &\leq e^{-\mu\delta^2/2} \\
&= \exp\left(-\mu\delta^2/2\right) \\
&= \exp\left(-\frac{K(1-p_{\text{corrupt}})\left(1 - \frac{(H-1)}{K(1-p_{\text{corrupt}})}\right)^2}{2}\right) \\
&= \exp\left(-\frac{K(1-p_{\text{corrupt}})\frac{\left(K(1-p_{\text{corrupt}}) - (H-1)\right)^2}{\left(K(1-p_{\text{corrupt}})\right)^2}}{2}\right) \\
&= \exp\left(-\frac{\left(K(1-p_{\text{corrupt}}) - H + 1\right)^2}{2K(1-p_{\text{corrupt}})}\right)
\end{aligned}$$

Now, we require $0 < \delta < 1$. We have

$$\begin{array}{rccccc}
 & 0 & < & \delta & < & 1 \\
\implies & 0 & < & 1 - \frac{(H-1)}{K(1-p_{\text{corrupt}})} & < & 1 \\
\implies & -1 & < & -\frac{(H-1)}{K(1-p_{\text{corrupt}})} & < & 0 \\
\implies & 1 & > & \frac{(H-1)}{K(1-p_{\text{corrupt}})} & > & 0 \\
= & 0 & < & \frac{(H-1)}{K(1-p_{\text{corrupt}})} & < & 1 \\
\implies & 0 & < & H-1 & < & K(1-p_{\text{corrupt}}) \\
\implies & 1 & < & H & < & K(1-p_{\text{corrupt}})+1 \\
= & 1 & < & H & \leq & K(1-p_{\text{corrupt}}) \\
= & 1 & < & H & \leq & \mathbf{E}[X]
\end{array}$$

which is assumed by the lemma. □

## 5.3 Attacks on Rapid's aggregation rules

A correct member $m$ can have fewer than $H$ corrupt observers, but still be removed from the system. If at least $L$ alerts are made about $m$, it is considered to be in unstable report mode. If no more corroborating reports are received, $m$ stays in unstable report mode, preventing other members from proposing configuration changes. However, Rapid provides mechanisms that ensure liveness in

order to prevent members from waiting for stability forever. In this case, after a specified timeout period elapses, Rapid will automatically broadcast alerts from $m$'s remaining $K-L$ observers. As a result, a member may in fact be affected by corrupt observers if at least $L$ of its $K$ observers are corrupt. This is weaker than requiring $H$ or more observers for susceptibility: indeed, $m$ would be able to withstand its corrupt observers for longer. However, $m$ would eventually be removed from the system.

**Lemma 5.5.** *Let $C$ be the number of corrupt observers for a process. If $L$ is greater than the expected number of corrupt observers and at most two times the expected number of corrupt observers, the probability that at least $L$ of the process's $K$ observers are corrupt is exponentially small in $L$, $K$, and $p_{\text{corrupt}}$. Specifically, if*

$$\boldsymbol{E}[C] < L \leq 2\boldsymbol{E}[C],$$

*then*

$$\Pr(C \geq L) \leq \exp\left(-\frac{\left(L - Kp_{\text{corrupt}}\right)^2}{3Kp_{\text{corrupt}}}\right).$$

*Proof.* The proof for this lemma very similar to that for Lemma 5.3, so we do not repeat it. However, it remains to show that the conditions for the Chernoff bound to apply still hold in this case.

We use the following version of the Chernoff bound (Corollary 4.2, Mitzenmacher and Upfal, 2017):

$$\Pr(X \geq (1+\delta)\mu) \leq e^{-\mu\delta^2/3}$$

where $0 < \delta \leq 1$. Here, we have $\mu = \mathbf{E}[C] = Kp_{\text{corrupt}}$. Solving for $\delta$, we get

$$\delta = \frac{L}{Kp_{\text{corrupt}}} - 1$$

We have

$$\begin{array}{rccccc} & 0 & < & \delta & \leq & 1 \\ \implies & 0 & < & \frac{L}{Kp_{\text{corrupt}}} - 1 & \leq & 1 \\ \implies & 1 & < & \frac{L}{Kp_{\text{corrupt}}} & \leq & 2 \\ \implies & Kp_{\text{corrupt}} & < & L & \leq & 2Kp_{\text{corrupt}} \\ \implies & \mathbf{E}[C] & < & L & \leq & 2\mathbf{E}[C] \end{array}$$

which is assumed by the lemma. □

## 5.4 Attacks on consensus

Rapid's consensus protocol *can* be attacked in the sense that correct members can agree upon an incorrect proposal that does not convey accurate membership information. However, we find that Rapid's consensus protocol *itself* is relatively immune to the presence of corrupt members.

**Corollary 5.5.1.** *We require $p_{\text{corrupt}}$ to be very high for corrupt members participating in Rapid's consensus process to decide on an incorrect proposal.*

*Proof.* Rapid uses Fast Paxos (Lamport, 2006) to achieve consensus on a single proposed membership view. Fast Paxos decides on a proposal if there is a quorum of greater than three quarters of the system membership; that is, if at least three quarters of the members have an identical proposal. From Lemma 1, we expect the number of corrupt processes in the system to be $n \cdot p_{\text{corrupt}}$. At least three quarters of the members must be corrupt:

$$\begin{aligned} n \cdot p_{\text{corrupt}} &> \frac{3}{4}n \\ p_{\text{corrupt}} &> \frac{3}{4} = 0.75 \end{aligned}$$

Similarly, even when Fast Paxos is unable to decide on a proposal and must resort to classical Paxos, we still require a majority of acceptors to be correct, which gives

$$p_{\text{corrupt}} > \frac{1}{2}.$$

In our preceding analysis, however, we require and assume that the (configured) probability $p_{\text{corrupt}}$ is strictly less than $1/2$. □

## 6 Results and Discussion

In summary, we show the following:

- The expected number of corrupt processes in the system is $N \cdot p_{\text{corrupt}}$.
- The expected number of processes with no corrupt observers is $N \cdot (1 - p_{\text{corrupt}})^K$.
- The probability that at least $H$ of a process's $K$ observers are corrupt is given by

$$\Pr(C \geq H) \leq \exp\left(-\frac{\left(H - Kp_{\text{corrupt}}\right)^2}{3Kp_{\text{corrupt}}}\right).$$

- The probability that fewer than $H$ of a process's $K$ observers are correct is given by

$$\Pr(X < H) \leq \exp\left(-\frac{\left(K(1-p_{\text{corrupt}}) - H + 1\right)^2}{2K(1-p_{\text{corrupt}})}\right)$$

- The probability that at least $L$ of a process's $K$ observers are corrupt is given by

$$\Pr(C \geq L) \leq \exp\left(-\frac{\left(L - Kp_{\text{corrupt}}\right)^2}{3Kp_{\text{corrupt}}}\right).$$

- Rapid's consensus protocol is itself immune to the presence of adversarial members. The required probability for a given member to be corrupt is very high for adversarial members to directly impact consensus.

In evaluating Rapid, the designers of the system used a system with between 1000 and 2000 members, and set parameters $K = 10$, $H = 9$, and $L = 3$.

| $p_{\text{corrupt}}$ | **Expected number of corrupt processes** |
|---|---|
| 0.1 | 100 |
| 0.25 | 250 |
| 0.5 | 500 |
| 0.75 | 750 |
| 1 | 1000 |

Table 1: Expected number of corrupt processes in a system with $N = 1000$ members

| $p_{\text{corrupt}}$ | **Expected number of unaffected processes** |
|---|---|
| 0.1 | 349 |
| 0.25 | 56 |
| 0.5 | 1 |
| 0.75 | 0 |
| 1 | 0 |

Table 2: Expected number of unaffected processes in a system with $N = 1000$ members

Table 1 shows that, as expected, the number of corrupt processes grows linearly as the independent probability of each individual process being corrupt is increased. We also saw that the expected number of process with no corrupt observers was given by $N \cdot (1 - p_{\text{corrupt}})^K$. We refer to such processes as *unaffected* processes.

In Table 2, the number of processes unaffected by corrupt members in the system falls rapidly as the independent probability of each individual process being corrupt is increased.

We saw that if a process has at least $H$ corrupt members, it is susceptible to an aggressive attack. We refer to such processes as *fully affected* processes.

Table 3 shows the probabilities that a given process is fully affected as $p_{\text{corrupt}}$ increases.

| $p_{\text{corrupt}}$ | **Probability a process is fully affected** |
|---|---|
| 0.1 | $5.43 \times 10^{-10}$ |
| 0.25 | 0.0035 |
| 0.5 | 0.3441 |
| 0.75 | 0.9048 |
| 1 | 0.9672 |

Table 3: The probability $\Pr(C \geq H)$ ($0 \leq \Pr(C \geq H) \leq 1$) that a process is fully affected. Here, $K = 10$ and $H = 9$.

# 7 Future Work

There are several ways in this work can be extended and further explored:

- **Obtaining "global" bounds on worst-case membership divergence:** It would be useful to have bounds that can be evaluated at the system level; that is, bounds that could be used to quantify how the system as a whole deviates from the actual, correct membership state in the worst case. We obtained more myopic bounds on individual processes and specific components in the system. A global bound, derived as a function of these smaller bounds, would give a higher-level, broader sense of the resilience of Rapid's membership protocol in response to the presence of adversarial members.

- **Exploring *eclipse* attacks:** The Fireflies system considers *eclipse* attacks (Singh et al., 2004) on its protocol. An eclipse attack is one in which the overlay topology of the system is "biased" such that correct members (unknowingly) favor communicating with corrupt members. Rapid's monitoring topology could also be susceptible to such an attack, and it is worth exploring how a process' observers and subjects might collude to propagate incorrect information about the process.

- **Exploring attacks on logically centralized configurations of Rapid:** Rapid supports both logically centralized configurations, in which a designated, auxiliary set of members maintain membership information, as well as decentralized configurations, in which all the members in the system communicate with each other to maintain membership information. In this project, we only explored attacks on the decentralized version of the membership protocol. A possible extension could be to explore and obtain bounds on the resilience of logically centralized configurations of Rapid in the presence of adversarial members. Our hypothesis is that logically centralized configurations might be more resilient to attacks since there is greater control over and visibility of the flow of updates in the system.

- **Exploring how gossip-based broadcast could be exploited:** Rapid currently employs unicast-to-all, but a robust gossip-based broadcast implementation is also being developed. We would like to explore if adversarial members could take advantage of this communication strategy by slowing messages down. By succeeding to slow "favorable" messages down, an adversary could ensure that consensus is *not* achieved in the primary, almost-everywhere agreement phase, and always require consensus to go through the slower consensus protocol built on classical Paxos.

# 8 Acknowledgments

We would like to thank Dr. Lalith Suresh, Dr. Dahlia Malkhi, and VMware Research Group for sharing Rapid with us, and for the opportunity to work on an extension to the system. We would also like to thank Darshan Thaker, Guneet Dhillon, and Max Gray for helpful discussions and suggestions about the proofs in this project.

## Notes

[1]This notation has been adapted from *Fireflies* (Johansen et al., 2015).

## References


Bortnikov, Edward et al. (2008). "Brahms: Byzantine resilient random membership sampling". In: *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing*.

Burrows, Mike (2006). "The chubby lock service for loosely-coupled distributed systems". In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*.

Cassandra, Apache. *Apache Cassandra*. `https://cassandra.apache.org`. 2007.

– *Cassandra-3831: scaling to large clusters in GossipStage impossible due to calculatePendingRanges*. `https://issues.apache.org/jira/browse/CASSANDRA-3831`. 2015.

– *Cassandra-6127: vnodes don't scale to hundreds of nodes*. `https://issues.apache.org/jira/browse/CASSANDRA-6127`. 2013.

Douceur, John R. (2002). "The sybil attack". In: *Lecture Notes on Computer Science, Vol. 2429. Springer*.

Hashicorp. *Serf*. `https://serf.io`. 2013.

Johansen, Havard D. et al. (2015). "Fireflies: A secure and scalable membership and gossip service". In: *ACM Transactions on Computer Systems, Vol. 33, No. 2*.

Karger, David et al. (1997). "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web". In: *Proceedings of the twenty-ninth annual ACM Symposium on Theory of Computing*.

Lamport, Leslie (2001). "Paxos made simple". In: *ACM Sigact News 32.4*.

– (2006). "Fast Paxos". In: *Distributed Computing*.

Mitzenmacher, Michael and Eli Upfal (2017). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

Redis. *Redis*. `https://redis.io`. 2009.

Singh, Atul et al. (2004). "Defending against Eclipse attacks on overlay networks". In: *Proceedings of the 11th ACM SIGOPS European Workshop*.

Suresh, Lalith et al. (2018). "Stable and Consistent Membership at Scale with Rapid". In: *USENIX ATC 2018*.

Typesafe. *Akka*. `https://akka.io`. 2009.

Zookeper, Apache. *Apache Zookeeper*. `https://zookeper.apache.org/`. 2010.