Practical Robust Communication in DHTs Tolerating a Byzantine Adversary

Maxwell Young Aniket Kate Ian Goldberg Martin Karsten
David R. Cheriton School of Computer Science
University of Waterloo, ON, Canada
Email: {m22young,akate,iang,mkarsten}@cs.uwaterloo.ca

***Abstract*—There are several analytical results on distributed hash tables (DHTs) that can tolerate Byzantine faults. Unfortunately, in such systems, operations such as data retrieval and message sending incur significant communication costs. For example, a simple scheme used in many Byzantine fault-tolerant DHT constructions of n nodes requires $O(\log^3 n)$ messages; this is likely impractical for real-world applications. The previous best known message complexity is $O(\log^2 n)$ *in expectation*; however, the corresponding protocol suffers from prohibitive costs owing to hidden constants in the asymptotic notation and setup costs.**

In this paper, we focus on reducing the communication costs against a computationally bounded adversary. We employ threshold cryptography and distributed key generation to define two protocols both of which are more efficient than existing solutions. In comparison, our first protocol is *deterministic* with $O(\log^2 n)$ message complexity and our second protocol is randomized with expected $O(\log n)$ message complexity. Further, both the hidden constants and setup costs for our protocols are small and no trusted third party is required. Finally, we present results from microbenchmarks conducted over PlanetLab showing that our protocols are practical for deployment under significant levels of churn and adversarial behaviour.

I. Introduction

The peer-to-peer (P2P) paradigm is a popular approach to providing large-scale decentralized services. However, the lack of admission control in many such systems makes them vulnerable to malicious interference [1], [2]. This is a practical concern since large-scale P2P systems are in existence today such as the Azureus DHT [3] and the KAD DHT [4], each of which see more than one million users per day. In addition to file sharing, there are proposals for using P2P systems to protect archived data [5] and re-implement the Domain Name System [6]; such applications would likely benefit from increased security.

There are a number of results on P2P systems that can provably tolerate Byzantine faults [7]–[14]. This includes the Sybil attack [15] although for ease of exposition, we refer to a Byzantine adversary throughout this work; our results can also be used in conjunction with some proposals specific to the Sybil attack (see the survey of [16]). To date, the majority of results pertain to distributed hash tables (DHTs). A common technique in DHTs that tolerate adversarial faults is the use of *quorums* which are sets of peers such that a minority of the members suffer adversarial faults. A quorum replaces an individual peer as the atomic unit. Adversarial behavior can be overcome by majority action allowing for communication between correct peers; we call this *robust communication*. Since critical operations such as data queries are performed in concert by members of a quorum, robust communication must be efficient.

Several protocols using quorums have been proposed; however, there is a common theme in the way such quorums are utilized. A message m originating from a peer p traverses a sequence of quorums Q_1, Q_2, ..., Q_ℓ until a destination peer is reached. A typical example is a query for content where the destination is a peer q holding a data item. Initially p notifies its own quorum Q_1 that it wishes to transmit m. Each peer in Q_1 forwards m to all peers in Q_2. A peer in Q_2 determines the correct message by majority filtering on all incoming messages and, in turn, sends to all peers in the next quorum. This forwarding process continues until the quorum Q_ℓ holding p is reached. Assuming a majority of correct peers in each quorum, transmission of m is guaranteed. Unfortunately, this simple protocol is costly. If all quorums have size s and the path length is ℓ, then the message complexity is $\ell \cdot s^2$. Typically, $s = \Theta(\log n)$ and, as in Chord [17], $\ell = O(\log n)$ which gives a $O(\log^3 n)$ message complexity which is likely prohibitively expensive for practical values of n.

Saia and Young [10] give a randomized protocol which provably achieves $O(\log^2 n)$ messages in expectation. While communication between two quorums incurs an expected constant number of messages, the analysis in [10] yields a prohibitively large constant. Furthermore, with probability $1 - o(1)$ some peers will incur $\omega(1)$ message complexity (see our extended version [18]). The protocol also employs a link architecture between peers requiring the use of a Byzantine agreement protocol. Finally, maintenance and asynchronicity issues remain unresolved.

Therefore, while results exist on the feasibility of robust communication, work on the practicalities has lagged behind. This dearth presents an impediment to the deployment of such systems and we seek to address this outstanding problem.

A. Our Contributions

We improve over all previously known results involving communication between quorums [7], [10]–[12]. We summarize our main results below:

Theorem 1. *In the computational setting, for an adversary that controls up to an $\epsilon < 1/3$-fraction of any quorum* *of size at most s, there are two protocols for achieving robust communication of a message m to a set of peers $D \subseteq Q_i$ for some quorum Q_i over a path of length ℓ. Our Robust Communication Protocol I (RCP-I) has the following properties:*

- *The total* message complexity *(number of messages sent and received) and the message complexity of the sending peer is each at most $2 \cdot s + 4 \cdot s \cdot (\ell - 2) + |D|$.*
- *The message complexity of every non-sending peer along the lookup path is at most 4.*
- *The* latency *(number of roundtrip communication rounds) is at most $2 \cdot (\ell - 2) + 2$.*

For our Robust Communication Protocol II (RCP-II):

- *The expected total message complexity and the expected message complexity of the sending peer is each at most $2 \cdot s + \frac{(\ell-2)}{(1-\epsilon)\cdot c} + (\ell - 2) + |D|$.*
- *The expected message complexity of a non-sending peer on the lookup path is at most $\frac{2}{(1-\epsilon)\cdot c \cdot s}$.*
- *The expected latency is at most $\frac{(\ell-2)}{(1-\epsilon)\cdot c} + 2$.*

Here, the constant $c > 0$ is the probability that the response time of a correct peer is at most Δ.

Using the Chord-based construction of [11], the message complexity of RCP-I is $O(\log^2 n)$ and for RCP-II it is $O(\log n)$ in expectation. We tolerate a large fraction of adversarial peers; strictly less than a $1/3$-fraction compared to the roughly $1/4$-fraction in [10]. Our use of a distributed key generation (DKG) scheme allows for security *without* a trusted party or costly updating of public/private keys outside of each quorum. This obviates the need for a trusted third party. To the best of our knowledge, this is the first use of DKG in a Byzantine-tolerant P2P setting.

Finally, we provide microbenchmark results involving two quorums using PlanetLab. Our experimentation demonstrates that our protocols perform well under significant levels of churn and faulty behaviour. In particular, for a 10^5-node system with $\ell = 20$, our results imply RCP-I and RCP-II complete in under 4 seconds and 5 seconds, respectively.

II. Related Work

A large body of literature exists on implementing Byzantine protocols. While P2P systems do not align perfectly with the state machine replication (SMR) paradigm [19], the large body of literature on Byzantine fault-tolerant replication is relevant to our work. Early work by Reiter [20] yielded protocols for Byzantine agreement and atomic broadcast. Our first protocol shares some common features with the multicast protocol of [20], yet we differ significantly since in the P2P domain we must contend with issues of scalability, churn, and spurious requests aimed at consuming resources. More recently, Castro and Liskov [21] demonstrated efficient Byzantine fault-tolerant SMR; however, this seems unsuitable for a P2P setting due to scaling issues. Several other Byzantine fault-tolerant systems have been implemented such as SINTRA [22], FARSITE [23], the Query/Update protocol [24] and the HQ system [25]; however, scalability issues make the use of these protocols in a P2P setting unlikely.

Two *implemented* large-scale Byzantine fault tolerant storage architectures are OceanStore [26] and Rosebud [27]. The latter scales up to tens of thousands of nodes and handles changing membership. However, with only a single Byzantine node per replication group, Rosebud incurs significant overhead. In contrast, our protocols perform efficiently with 10% of the peers being Byzantine. Rosebud relies on a *configuration service* (CS) which tracks system membership, ejects faulty nodes, and handles new nodes. The CS, implemented over a set of nodes, introduces a potential bottleneck and a possible point of attack; similarly, a "primary tier" of replicas is used in OceanStore. In contrast, our protocol is completely decentralized and no special set of nodes is required.

Both Rodrigues, Kouznetsov and Bhattacharjee [28] and Rodrigues, Liskov and Shrira [29] give *proposals* for applying the SMR approach on a large scale; the latter describes a P2P system. However, both works rely on a CS and neither provides empirical results or discusses the details of secure data retrieval and message passing. Wang *et al.* [30] design and implement a routing scheme that tolerates Byzantine faults and yields good performance. However, they require both a certificate authority (CA) and a special set of nodes, called a neighborhood authority, similar to a CS.

There are several theoretical results on Byzantine fault-tolerant DHTs [7], [11]–[13]. These results make use of *quorums*, which are sets of $\Theta(\log n)$ peers such that a majority of the peers in a quorum are correct. Awerbuch and Scheideler show how to maintain quorums [7]–[9]. Saia and Young [10] demonstrate more efficient robust communication but, as discussed earlier, several issues remain unresolved.

Castro *et al.* [31], Halo [32], and Salsa [33] handle Byzantine faults by routing along multiple diverse routes. The proposal in [31] requires a CA whereas we do not rely on any trusted third party. In both [32] and [33], the guarantees are unclear against an adversary who owns a large IP-address space or targets identifiers over time as described in [7]. In contrast, defenses for quorum-based protocols are known [7]–[9]. Finally, the ShadowWalker system [34] routes securely using the notion of multiple "shadows" which are similar to a quorum; however, our protocols differ significantly.

III. System Model

Each peer p is assumed to have a unique identifier, $p_{\texttt{ID}}$, and a network address, p_{addr}. Byzantine peers are also referred to as *faulty* or *adversarial*; all other peers are called *correct*. A fraction of the correct peers may crash due to a system failure or leave gracefully. We model such peers as having *crashed*.

We adopt an asynchronous communication model with unbounded message delivery time. However, for liveness in DKG and in our second protocol, we use a *weak synchrony* assumption by Castro and Liskov [35].

Peers p and q are said to communicate directly if each has the other in its routing table. The target of m is a set of peers D within a single quorum; m may be a data item request and

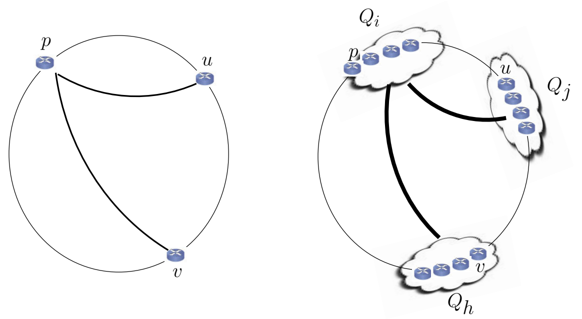

Fig. 1. (Left) Three peers on a DHT ring where p links to u and v. (Right) An example of a quorum topology in a DHT ring where $p \in Q_i$, $u \in Q_j$ and $v \in Q_h$. Thick lines signify inter-quorum links.

D may consist of a single peer or multiple peers depending on how data is stored.

A. The Quorum Topology

There are several different approaches to how quorums are created and maintained [7], [10], [12]; we refer the reader to [11] for a detailed explanation. Despite these different approaches, we may view the setup of quorums as a graph where nodes correspond to quorums and edges correspond to communication capability between quorums; we refer to this as the *quorum topology*. Figure 1 illustrates how quorums can be linked in a DHT such as Chord. Peers will likely have different views of the network and hence membership lists for Q_i may differ for two peers; however, such issues can be overcome (for example, see [11]). We assume the following four simple invariants are true:

1) *Goodness:* each quorum has size $s = \Omega(\log n)$ and possesses at most an ϵ-fraction of Byzantine peers for $\epsilon < 1/3$.
2) *Membership:* every peer belongs to at least one quorum.
3) *Intra-Quorum Communication:* every peer can communicate directly to all other members of its quorums.
4) *Inter-Quorum Communication:* if Q_i and Q_j share an edge in the quorum topology, then $p \in Q_i$ may communicate directly with any member of Q_j and vice-versa.

These four invariants are standard in the sense that previous works on quorums in DHTs ensure they hold with probability nearly equal to 1. For example, results for maintaining the goodness invariant in DHTs are known [7]–[9]. In terms of the membership invariant, there exist quorum topologies where a peer may belong to several different quorums simultaneously [11], [12]. Finally, to the best of our knowledge, no implementation of a quorum topology exists; this represents another gap between theory and practice. A number of challenges remain in bridging this gap and such an endeavor is outside the scope of this current work. However, the literature suggests that, with the proper deployment, maintaining these four invariants in real-world DHTs is plausible.

B. Assumptions

The adversary is assumed to have full knowledge of the network topology and control all faulty peers, which forms a constant fraction of all nodes in the system. In concert with the goodness invariant, strictly less than $1/3$ of the peers in any quorum can be faulty. These peers may collude and coordinate their attacks. Our adversary is computationally bounded with a security parameter κ and it has do 2^{κ} computation to break the security of the Gap Diffie-Hellman (GDH) problem [36] in an appropriate group.

Our protocols guarantee successful transmission of a message; however, feasibility is not enough. Our protocols must be efficient, both in terms of (1) the costs incurred by correct peers for legitimate network operations *and* (2) the costs incurred by adversarial behavior. The latter concern is crucial since it does no good to provide solutions that allow the adversary to easily launch costly attacks. We first discuss the cryptographic techniques for gaining efficiency and then elaborate on points (1) and (2).

C. Threshold Cryptography

We use threshold cryptography to authenticate messages. The idea behind an (η, t)-threshold scheme is to distribute a secret key among η parties in order to remove any single point of failure. Any subset of more than t parties can jointly reconstruct the secret key or perform the required computation securely in the presence of a Byzantine adversary which controls up to t parties. We use threshold signatures to authenticate the communication between quorums.

Threshold Signatures: In an (η, t)-threshold signature scheme, a signing (private) key k is distributed among η parties by a trusted dealer using a verifiable secret sharing protocol [37] or by a completely distributed approach using a DKG protocol [38]. Along with private key shares k_i for each party, the distribution algorithm also generates a verification (public) key K and the associated public key shares $\widehat{K}$. To sign a message m, any subset of $t+1$ or more parties use their shares to generate the signature shares σ_i. Any party can combine these signature shares to form a message-signature pair $S = (m, \sigma) = [m]_k$ that can be verified using the public key K; however, this does not reveal k. We refer to a message-signature pair S as a signature. It is also possible to verify σ_i using the public key shares $\widehat{K}$. We assume that no computationally bounded adversary that corrupts up to t parties can forge a signature $S' = (m', \sigma')$ for a message m'. Further, malicious behavior by up to t parties cannot prevent generation of a signature. Here, we use the threshold version [39] of the Boneh-Lynn-Shacham (BLS) signature scheme [36] secure under the GDH assumption (see [18] for more details).

Distributed Key Generation (DKG): In absence of a trusted party in the P2P paradigm, we use a DKG scheme to generate the (distributed) private key. An (η, t)-DKG protocol allows a set of η nodes to construct a shared secret key k such that its shares k_i are distributed over the nodes and no coalition of fewer than t nodes may reconstruct the secret; no trusted dealer is required. There is also an associated public key K and a set of public key shares $\widehat{K}$ for verification.

The protocol in [40] is the first DKG for an asynchronous setting; therefore, it is uniquely suitable for deployment in a

P2P network. Along with a Byzantine adversary, this protocol also tolerates crash failures. For a quorum of size $s = \eta$, with t Byzantine nodes and f correct nodes that can crash, the DKG protocol requires that $s \geq 3t + 2f + 1$. In our case, this *security threshold* holds due to the goodness invariant in Section III-A. The DKG protocol allows for system dynamics without changing the system public key K and this can be done efficiently by batching; details are given in Section V-B.

D. Spamming Attacks

A critical concern is that the adversary may launch spurious communications aimed at consuming resources; we refer to such behavior as *spamming*. For example, a malicious peer may initiate a number of data retrieval requests [1], [2]. Here the situation is more dire since the impact of such attacks is multiplied by the group action in a quorum-based system.

Ultimately, there is no perfect defence against an adversary with the resources to initiate massive spamming attacks (or denial-of-service (DoS) attacks; see the extended version [18] for more discussion) and this is not our focus. Rather we show that our protocols do not afford the adversary an advantage in launching such attacks. Our goal is to prevent the adversary from forcing a peer to perform expensive operations with impunity. For any operation initiated by a spammer p, this can be accomplished by either (A) placing the bulk of the cost of executing said operation on p or (B) making the detection of spamming inexpensive. As we will show in Section IV, our protocol RCP-1 in Section IV-A employs principle (A) while our protocol RCP-II in Section IV-B employs principle (B).

In addition to cryptographic techniques, we assume a *rule set* to reduce the impact of spamming attacks as introduced by Fiat *et al.* [11]. A rule set defines acceptable behavior in a quorum; for example, the number of data lookup operations a peer may execute per duration of time. Such rules are known to everyone within a quorum and can be implemented at the software level or agreed upon by quorum members. Requests from a peer that deviates from the rule set are ignored by the other members of its quorum.

E. Efficiency (Not Feasibility) Through Cryptography: The Prove-and-Verify Scenario

We now discuss the merits of employing cryptographic techniques. In the presence of Byzantine peers, no single peer can be trusted. Quorums are employed to overcome this trust deficit through majority action. Using the simple protocol outlined in Section I, transmission of a message is guaranteed. *Therefore, quorums allow for robust communication without the need for cryptographic techniques.*

However, spamming attacks still pose a critical problem. For example, a group of Byzantine peers may pretend to be a quorum and initiate requests. Therefore, simply obeying a request because it *appears* to come from a quorum does not prevent spamming. A standard fix is that a quorum responds only to requests that are "proven" to be legitimate. Yet, there is a cost to proving legitimacy; we explore this to motivate our protocols. First, we expand on the utility of a quorum topology in proving legitimacy. We then show how cryptographic techniques improve the efficiency of this task.

Utility of the Quorum Topology: We compare two general scenarios in order to demonstrate the utility of a quorum topology in proving requests legitimate. The first assumes that proofs and verifications are required to initiate operations; call this the *prove-and-verify scenario*. The second assumes no proof is required before acting (although, each peer may keep a record of misbehaving peers); call this the *passive scenario*. P2P systems often lack admission control and, if forced to leave the system, a Byzantine peer may simply rejoin the network with a new identity. In the worst case, perpetual and rapid rejoin operations result in a DoS attack. Therefore, we make the standard assumption that there is a cost for joining the network (for example, monetary costs as in [31] or CAPTCHAs as suggested in [33]). The best method of enforcing this cost is beyond the scope of this work.

Let τ denote the rate at which p can issue spurious requests before being forced to rejoin the system. In the passive scenario, a Byzantine peer p can contact *any* quorum Q_i by colluding with other faulty peers to obtain necessary routing information. Members of Q_i act on any request coming from p. Therefore, a correct peer may be required to maintain $O(n)$ records so that spam requests are ignored. Moreover, here τ is large due to the abundance of potential targets. In contrast, in a prove-and-verify system the members of Q_i must verify p's proof before acting. Proof and verification may take different forms. For instance, constructions exist where two peers communicate only if their respective quorums are linked [11], [12]; that is, the quorum topology itself acts as proof. Verification occurs by having a quorum Q_i act on p's request only if each peer in Q_i receives messages from a majority in Q_p. Here τ is smaller; however, there are still shortcomings to this method of proof and verification.

Efficiency in the Prove-and-Verify Scenario: We argue two things: (1) the form of proof discussed above is restrictive and (2) verification is expensive. First, the proof is restrictive since for Q_i and Q_j to communicate without sending through intermediary quorums, they must maintain links to one another; such maintenance is costly. Second, the verification process is expensive because when communication occurs from Q_i to Q_j, a correct peer $q \in Q_j$ must know to which peers in Q_i it must listen; this incurs more maintenance costs. These are two significant problems with existing schemes.

Cryptography allows us to improve asymptotically on the message complexity of verification. Under our protocols, each quorum has a public and private key established using DKG. Communication can occur between any two quorums that know and can verify each other's public key. Therefore, the form of proof is not as restricted by the quorum topology and we exploit this in RCP-II. Furthermore, verification is cheaper; using $O(s)$ messages in RCP-I or $O(1)$ expected messages in RCP-II. Of course, overhead is incurred by using cryptography. Message sizes increase by an additional $O(\kappa)$ bits and keys shares, but *not* the key itself, must be updated

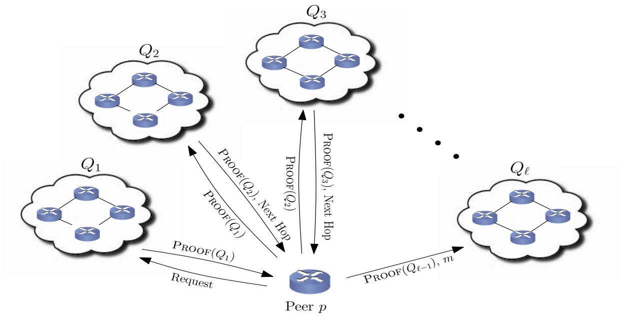

Fig. 2. Our general robust communication scheme. At step $i = 1, ..., \ell - 1$, peer p presents proof, PROOF(Q_i), that quorum Q_i sanctions p's action, and receives new proof from Q_{i+1} in addition to routing information for the next hop. At the final step ℓ,peer p sends PROOF($Q_{\ell-1}$) and m.

when membership changes. However, our experimental results in Section V show that this overhead is tolerable since the computation costs are significantly smaller than the network latency. Hence, cryptography provides a more efficient and flexible implementation of the prove-and-verify scenario.

IV. Our Robust Communication Protocols

We propose two robust communication protocols: RCP-I and RCP-II. Here we outline a general scheme in Figure 2 that is later refined to give our two protocols. Consider a sending peer p who wishes to send a message m to peer q. We assume m is associated with a key value which yields information necessary for distributed routing; that is, the next peer to which m should be forwarded is always known. Peer p notifies its quorum Q_1 that it is performing robust communication and receives PROOF(Q_1). Peer p sends this to Q_2 as proof that p's actions are legitimate; the form of this proof is discussed later. Depending on the scheme, one or more members of Q_2 examines the proof and, upon verifying it, sends to p: (1) routing information for Q_3 and (2) PROOF(Q_2), that will convince Q_3 that p's actions are legitimate. This continues iteratively until p contacts the quorum holding q and m is delivered. We employ the following concepts:

Quorum Public/Private Keys: Each quorum Q_i is associated with a (distributed) public/private key pair (K_{Q_i}, k_{Q_i}); however, there are two crucial differences between how such a key pair is utilized here in comparison to traditional implementations. First, only those quorums linked to Q_i in the quorum topology, and not everyone in the network, need to know K_{Q_i}. Second, (K_{Q_i}, k_{Q_i}) is created using the DKG protocol and $\widehat{K}_{Q_i}$ is the associated set of public key shares.

Individual Public/Private Key Shares: Each peer $p \in Q_i$ possesses a private key share $(k_{Q_i})_p$ of k_{Q_i} produced using DKG. Unlike the quorum public/private key pair of Q_i which must be known to all quorums to which Q_i is linked in the quorum topology, only the members of Q_i need to know the corresponding public key shares $\widehat{K}_{Q_i}$, which plays an important role in allowing members of Q_i to verify that the signature share sent to peer p is valid.

```
RCP-I: SENDING PEER $p$
Initial Step:
  $p \in Q_1$ sends the following request to all peers in $Q_1$: $[p_{\mathrm{ID}}|p_{\mathrm{addr}}|\mathtt{key}|ts_1]$
  $p$ interpolates all received signature shares to form: $\mathcal{S}_{Q_1} \leftarrow [p_{\mathrm{ID}}|p_{\mathrm{addr}}|\mathtt{key}|ts_1]_{k_{Q_1}}$
Intermediate Steps:
  for $i = 2$ to $\ell - 1$ do
    $p$ sends $\mathcal{S}_{Q_{i-1}}$ and $ts_i$ to every peer in $Q_i$ and requests a signature $\mathcal{S}_{Q_i}$, public key $K_{Q_{i+1}}$ and routing information for $Q_{i+1}$.
    $p$ interpolates received signature shares to form $\mathcal{S}_{Q_i} \leftarrow [p_{\mathrm{ID}}|p_{\mathrm{addr}}|\mathtt{key}|ts_i]_{k_{Q_i}}$.
    $p$ verifies if $\mathcal{S}_{Q_i}$ is valid using $K_{Q_i}$.
    if ($\mathcal{S}_{Q_i}$ is invalid) then
      $p$ sends signature shares to each peer in $Q_i$.
Final Step:
  $p$ sends $\mathcal{S}_{\ell-1}$ to $D \subseteq Q_\ell$ along with $m$.
```

```
RCP-I: RECEIVING PEER $q \in Q_i$
Initial Step:
  if ($q \in Q_1$ receives a request by $p$) then
    $q$ checks that a request by $p$ does not violate the rule set. If the request is legitimate, $q$ sends its signature share to $p$.
Intermediate Steps:
  if ($q$ receives $\mathcal{S}_{Q_{i-1}}$ and $ts_i$ from $p$) then
    $q$ verifies a $\mathcal{S}_{Q_{i-1}}$ using $K_{Q_{i-1}}$ and validates $ts_i$; if successful, $q$ sends its signature share, $K_{Q_{i+1}}$ and routing information for $Q_{i+1}$ to $p$.
  if ($q$ receives signature shares from $p$) then
    $q$ verifies all shares using public key shares and informs $p$ of invalid shares.
```

Fig. 3. Pseudocode for RCP-I

A. Robust Communication Protocol I

We now illustrate RCP-I for a peer p who wishes to send a message m. The path m takes through quorums is denoted by $Q_1, ..., Q_\ell$. We assume that $p \in Q_1$ and the target of the message is a set of peers $D \subseteq Q_\ell$.

Overview: We outline RCP-I; the pseudocode is given in Figure 3. Initially, the correct peers of Q_1 must acquiesce to p's request. Peer p begins by sending $[p_{\mathrm{ID}}|p_{\mathrm{addr}}|\mathtt{key}|ts_1]$ to all peers in its quorum Q_1. The value key corresponds to the intended destination of m and ts_1 is a time stamp. The message m can also be sent, and its hash can be added inside the signature below; however, for simplicity, we assume m is sent only in the last step. Each correct peer $q \in Q_1$ then consults the rule set and sends its signature share to p if p is not in violation. Peer p interpolates these signature shares to generate the signature: $S_1 \leftarrow [p_{\mathrm{ID}}|p_{\mathrm{addr}}|\mathtt{key}|ts_1]_{k_{Q_1}}$.

In each intermediate step $i = 2, ..., \ell - 1$, p sends its most recent signature S_{i-1} and a new time stamp ts_i to each peer $q \in Q_i$ along the lookup path. Since Q_i is linked to Q_{i-1} in the quorum topology, each q knows the public key $K_{Q_{i-1}}$ to verify S_{i-1}. If S_{i-1} is verified and ts_i is valid, q sends back its signature share, $K_{Q_{i+1}}$ and the routing information. Peer p collects the shares to form S_i and majority filters on the routing information for Q_{i+1}. In terms of majority filtering, both group membership and the corresponding routing information are agreed upon using DKG. Finally, for Q_ℓ, p sends m along with $\mathcal{S}_{\ell-1}$ to peers in the set D.

Share Corruption Attack: Note the following attack: a set of Byzantine peers $B \subsetneq Q_i$ send invalid shares to p and, therefore, p will fail to construct S_i. We refer to this attack as the *share corruption attack*. Here, the individual public/private key shares play a crucial role. To obtain S_i, p sends the received shares to each peer in Q_i using one message per peer. For a share sent to p by a peer in Q_i, each correct peer in Q_i verifies the share using $\widehat{K}_{Q_i}$. All valid shares are then sent back to p who creates S_i. While members of Q_i may identify those peers which p alleges sent an incorrect share, punitive action is limited since p could be Byzantine. Note that the shares are not recomputed; hence, the adversary can only perform this attack once per step.

Lemma 1. *RCP I guarantees that m is transmitted to a target set of peers $D \subseteq Q_i$ for some quorum Q_i over a path of length ℓ with the following properties:*

- *Both the total message complexity and the message complexity of the sending peer is each at most $2 \cdot s + 4 \cdot s \cdot (\ell - 2) + |D|$.*
- *Each non-sending peer has message complexity at most 4 messages.*
- *The latency is at most $2 \cdot (\ell - 2) + 2$.*

For our proof, refer to our extended version [18].

Spamming Attacks: The sending peer p experiences more cost than other participating peers. In part, this is due to the iterative nature of the protocol; however, largely this is because p must send and receive $O(s)$ messages per step. In contrast, other participating peers need only send and receive a constant number of messages over the execution of the protocol.

Peer p may misbehave in other ways. For instance, p may repeatedly contact its quorum to initiate robust communication; however, eventually all correct peers will ignore p. Similarly, using a correct signature, p may repeatedly ask q in another quorum for proof and/or routing information; however, time stamps limit such replay attacks. In conclusion, such actions cannot cause correct peers to perform expensive operations.

B. Robust Communication Protocol II

RCP-II is randomized yielding a small expected message complexity for both the sending peer and non-sending peers. In exchange, join and leave operations incur additional cost in comparison to RCP-I; we discuss this in Section IV-C.

RCP-II utilizes signed routing table information. As a concrete example, we assume a Chord-like DHT although other DHT designs can be accomodated. For a peer $u \in Q_i$, each entry of its routing table has the form $[Q_j, p_{\mathtt{ID}}, p'_{\mathtt{ID}}, K_{Q_j}, ts]$. Here $p \in Q_j$ and $p' \in Q_{j-1}$ where (1) Q_i links to Q_j and Q_{j-1} in the quorum topology, (2) Q_{j-1} immediately precedes Q_j clockwise in the identifier space and (3) p and p' are respectively located clockwise of all other peers in Q_j and Q_{j-1}. K_{Q_j} is the quorum public key of Q_j, and ts is a time stamp for when this entry was created. Note that any point in the identifier space falls between unique points p_{ID} and p'_{ID}. Given this property, and that entries are signed by a quorum, any attempt by a malicious peer along the lookup path to return incorrect routing information can be detected. $\mathcal{RT}_{Q_j}$ denotes the routing table information for all peers in Q_j. $[K_{Q_j}]_{k_{Q_i}}$ is the quorum public key of Q_j signed using the private quorum key of Q_i; recall, neighbors in the quorum topology know each others' public key. $[\mathcal{RT}_{Q_j}]_{k_{Q_i}}$ is the routing information signed with the private key of Q_i; entries of the routing table are signed separately. Routing table information is time stamped and re-signed periodically when DKG is executed.

Overview: We sketch RCP-II here. For simplicity, we temporarily assume that peers act correctly; our pseudocode in Figure 4 is complete for when peers fail to respond to requests by p. Initially, each correct peer in Q_1 receives $[p_{\mathtt{ID}}|p_{\text{addr}}|\mathtt{key}|ts]$ from p. The time stamp ts is chosen by p and peers in Q_1 will acquiesce to the value if it agrees with the rule set to within some bound to compensate for clock drift. If the request does not violate the rule set, then the information is signed allowing p to form $M_1 = [p_{\mathtt{ID}}|p_{\text{addr}}|\ \mathtt{key}|ts]_{k_{Q_1}}$.

In the second step of the protocol, p knows the membership of Q_2 and selects a peer $q_2 \in Q_2$ uniformly at random (u.a.r.) without replacement. Peer p then sends M_1 to q_2. Assuming q_2 is correct, it verifies M_1 using K_{Q_1} and checks that the ts is valid; the duration for which a time stamp is valid would be specified by the rule set. Once verified q_2 sends p the information $[K_{Q_1}]_{k_{Q_2}}$, $[\mathcal{RT}_{Q_3}]_{k_{Q_2}}$ and $[K_{Q_3}]_{k_{Q_2}}$. Peer p knows K_{Q_2} since Q_1 links to Q_2 and verifies $[K_{Q_1}]_{k_{Q_2}}$, $[\mathcal{RT}_{Q_3}]_{k_{Q_2}}$ and $[K_{Q_3}]_{k_{Q_2}}$, and checks that the time stamp on the routing information is valid. If so, p constructs $M_2 = [M_1|[K_{Q_1}]_{k_{Q_2}}]$. Here $[K_{Q_1}]_{k_{Q_2}}$ will allow some peer in Q_3 to verify K_{Q_1} and M_1, while the signed verified K_{Q_3} will allow p to check the response from that peer in Q_3.

This process repeats with minor changes for the remaining steps. Using $\mathcal{RT}_{Q_3}$ from the previous step, p selects a peer q_3 randomly from Q_3 and sends M_2. Since Q_3 is linked with Q_2 in the quorum topology, q_3 knows K_{Q_2}, which it uses to verify $[K_{Q_1}]_{k_{Q_2}}$; this allows q_3 to verify M_1 signed with k_{Q_1}. Peer q_3 then confirms that ts is valid and sends $[K_{Q_2}]_{k_{Q_3}}$, $[\mathcal{RT}_{Q_4}]_{k_{Q_3}}$ and $[K_{Q_4}]_{k_{Q_3}}$ to p. Peer p has a verified public key K_{Q_3} from the previous step and uses it to verify $[K_{Q_2}]_{k_{Q_3}}$, $[\mathcal{RT}_{Q_4}]_{k_{Q_3}}$, and $[K_{Q_4}]_{k_{Q_3}}$. Then p constructs $M_3 = [M_2|[K_{Q_2}]_{k_{Q_3}}] = [M_1|[K_{Q_1}]_{k_{Q_2}}|[K_{Q_2}]_{k_{Q_3}}]$. This process continues until m is delivered. Figure 4 gives the pseudocode for RCP-II. Every peer contacted by p verifies a chain of certificates, which can be converted into a single signature using the concept of aggregate signatures [41].

It is possible that p chooses a Byzantine peer that may not respond. In that case, after some appropriate time interval,

```
RCP-II: SENDING PEER $p$
Initial Step:
$p$ sends the following to each peer $q \in Q_1$: $[p_{\text{ID}}|p_{\text{addr}}|\texttt{key}|ts]$
$p$ gathers all responses and constructs:
    $M_1 \leftarrow [p_{\text{ID}}|p_{\text{addr}}|\texttt{key}|ts]_{k_{Q_1}}$
Intermediate Steps:
for $i = 2$ to $\ell - 1$ do
    while ($p$ does not have $M_i$ and has waited time $\Delta$ since previous selection) do
        $p$ sends $M_{i-1}$ to $q \in Q_i$ selected u.a.r. without replacement.
        if ($[K_{Q_{i-1}}]_{k_{Q_i}}$, $[\mathcal{RT}_{Q_{i+1}}]_{k_{Q_i}}$ and $[K_{Q_{i+1}}]_{k_{Q_i}}$ are received from any peer in $Q_i$ previously selected) then
            $p$ uses $K_{Q_i}$ to verify $K_{Q_{i+1}}$, $\mathcal{RT}_{Q_{i+1}}$ and $K_{Q_{i-1}}$.
        if ($K_{Q_{i+1}}$, $\mathcal{RT}_{Q_{i+1}}$ and $K_{Q_{i-1}}$ are all verified) then
            $M_i \leftarrow [M_{i-1}|[K_{Q_{i-1}}]_{k_{Q_i}}]$
Final Step:
$p$ sends $M_{\ell-1}$ to $D \subseteq Q_\ell$ along with $m$.
```

```
RCP-II: RECEIVING PEER $q$
Initial Step:
if ($q \in Q_1$ recives $[p_{\text{ID}}|p_{\text{addr}}|\texttt{key}|ts]$ from $p \in Q_1$) then
    $q$ checks that $p$'s request is legitimate and, if so, sends its signature share.
Intermediate Steps:
if ($q \in Q_i$ receives $M_{i-1}$ from $p$) then
    for $j = i - 1$ downto 1 do
        $q$ uses $K_{Q_j}$ to verify $K_{Q_{j-1}}$.
    Peer $q$ uses $K_{Q_1}$ to verify $M_1$.
    if verification is successful then
        $q$ sends $[K_{Q_{i-1}}]_{k_{Q_i}}$, $[\mathcal{RT}_{Q_{i+1}}]_{k_{Q_i}}$ and $[K_{Q_{i+1}}]_{k_{Q_i}}$ to $p$.
```

Fig. 4. Pseudocode for RCP-II

p will select an additional peer in the quorum. Let X be a random variable denoting the time required for a correct peer to respond. We make a weak assumption that $Pr[X \leq \Delta] \geq c$ where Δ is any duration of time and $c > 0$ is any constant probability. This does not circumscribe a particular distribution for response times; in fact, *any* distribution suffices, including the Poisson, exponential, and gamma distributions previously used to characterize round trip time (RTT) over the Internet. In practice, a peer p would set its own Δ value by sampling the network using methods for estimating RTT [42]. Since there are only a constant fraction of Byzantine peers, taking the median from a sufficiently large sample will allow p to determine Δ. As p receives a response from any of the previously selected peers in Q_i, *this is in accordance with the weak synchrony assumption* in Section III.

Lemma 2. *RCP- II guarantees that m is transmitted to a target set of peers $D \subseteq Q_i$ for some quorum Q_i over a path of length ℓ with the following properties:*

- *Both the total message complexity and the message complexity of the sending peer is each at most $2 \cdot s + \frac{(\ell-2)}{(1-\epsilon)\cdot c} + (\ell - 2) + |D|$.*
- *Each non-sending peer has expected message complexity at most $\frac{2}{(1-\epsilon)\cdot c \cdot s}$.*
- *The expected latency is at most $\frac{(\ell-2)}{(1-\epsilon)\cdot c} + 2$.*

For our proof, refer to our extended version [18].

While latency is measured in communication rounds, the time for executing RCP-II depends on Δ and we discuss this briefly. Accounting for the response time incurred in the intermediate steps, p waits for at most time $\frac{\Delta}{(1-\epsilon)\cdot c}$ per step in expectation (again, see [18]). Since peer p will have knowledge of the response time distribution, p may optimize performance by selecting Δ so that $\frac{\Delta}{c}$ is minimized.

Spamming Attacks: Due to the iterative nature of RCP-II, p sends more messages than other participating peers, but not to the degree seen in RCP-I. Instead of making it expensive for p to perform robust communication, RCP-II uses the following two properties to deter spamming: (1) it is inexpensive for a correct peer to detect spam and (2) the congestion suffered by a correct peer is low since the number of messages is not magnified by the use of quorums.

To address our first point, p may launch as many robust communication operations as the rule set allows; p may even try to circumvent the rule set by directly sending to a correct peer q; however, it is inexpensive for q to verify that the proof being sent is invalid. The operation terminates at that point since q will not reply. In contrast to the passive scenario of Section III-E, q need not keep a history to judge the legitimacy of a request; it simply verifies the accompanying certificate.

Our second point, and a key difference between RCP-I and RCP-II, is that with RCP-II an operation incurs only expected $O(\ell)$ messages which compares favourably to a system *without a quorum topology*. Therefore, the congestion caused by such requests is not significantly magnified by the use of quorums which was a key concern regarding spamming.

Adversarial peers may misbehave in other ways with many of the same consequences and remedies as discussed in RCP-I. Even with a generous upper bound on the expiration of ts, the congestion p can cause with a replay attack is again limited since only p can use the certificate. A notable attack, unique to RCP-II, occurs when a faulty peer gives p stale routing table information. Since entries are signed and time stamped, we are guaranteed that in the fairly recent past, the location indicated by the stale information was indeed correct. This fact, coupled with the standard assumption that ID collisions do not occur, guarantees that the adversary *cannot* engineer a situation where requests are forwarded to a faulty peer. Consequently, the impact of this attack is limited. The search path may be slightly lengthened by forwarding to an older location. Alternatively, stale information may point to a peer that no longer exists or is not the correct recipient, which forces p to backtrack one hop. These cases are handled easily, but for ease of exposition, they are not treated in our pseudocode in Figure 4. In short, routing integrity is not compromised. The fact that routing tables can be signed periodically every several minutes without significant CPU cost (see Section V) implies that the impact of such an attack is negligible.

C. Membership Updates

We discuss the implications of membership changes:

RCP-I: Consider a quorum Q_i to which a new peer is added. The membership update protocol of DKG [40] is executed to redistribute the shares of the public/private quorum key pair over all members of Q_i. In the process, the individual public/private key shares are also updated. Notably, *no other quorums are affected by this process* as the quorum key pair remains the same and the individual key shares need only be known to members of Q_i. When a peer leaves Q_i, the departure can be treated as a crash and, so long as the number of crashes does not exceed the crash-limit f, the DKG protocol need not be executed. We use this to associate the system churn rate to the session time of the DKG system. A membership update may also lead to modification in the crash limit f or the security threshold t. This can be handled using the group modification agreement of the DKG protocol [40]. Note that the adversary may crash some of its t nodes, and in principle, the system can handle $t + f$ node leaves. However, we cannot associate these additional t crashes with the system churn due to the inherent arbitrary nature of Byzantine peers. In terms of signature generation, a quorun can sign messages as long as any $t + 1$ of $n - t$ honest nodes are up.

RCP-II: When a peer q joins Q_i, the DKG protocol needs to be executed as in the case of RCP-I; however, there are additional costs due to the need to update and re-sign the routing table information. In particular, not only do the peers in Q_i need to update and have signed their routing table information (to reflect the addition of q), all quorums to which Q_i is linked under the quorum topology also need to update and re-sign their routing table information; note that this *does not require any revocation* since the public key does not change.Therefore, a join event under this scheme *does affect other quorums*. When a peer leaves Q_i, the DKG protocol may need to be executed as in the case of RCP-I. However, routing table information for Q_i and the quorums to which it links must again update and re-sign their routing table information. Therefore, while RCP-II reduces message complexity, the cost of join/leave operations is higher in comparison to RCP-I.

V. Experimental Results

We examine the performance of DKG and our two protocols on the PlanetLab platform [43]. Based on our experimental results and known churn rates, we propose parameters for DHTs using our protocols.

A. Implementation and Microbenchmarks

The DKG protocol is a crucial component of our protocols. It is required to initiate a threshold signature system in a quorum and to securely manage membership changes. We use a C++ implementation [44]. We incorporate threshold BLS signatures into this implementation and realize our two protocols using this setup on PlanetLab.

TABLE I
Median values of DKG completion time and CPU time per node for various s values.

s	t	f	**Completion Time** (sec)	**CPU Seconds/Node**
10	1	3	5.73	0.76
15	2	4	18.0	1.94
20	2	6	68.0	2.55
25	3	7	290.9	6.13
30	3	10	336.7	7.27

Distributed Key Generation: We test the DKG implementation for quorum sizes $s = 10, 15, 20, 25, 30$ and present median completion times and median CPU usage in Table I. The median completion periods vary from 6 seconds for $s = 10$ to more than 5 minutes for $s = 30$. The bulk of this latency is due to network delays; in contrast, the required CPU time is far smaller than the completion periods.

In the next subsection, we examine the feasibility of these completion periods. Our DKG experiments assume that 30% of the peers may crash and 10% of the peers may be Byzantine. While we can tolerate any fraction of Byzantine peers less than $1/3$, we use these numbers since in many practical scenarios we expect the fraction of Byzantine faults to be less than 10% and modest compared to the fraction of crash failures.

RCP-I and RCP-II: For our RCP-I and RCP-II experiments, we set $s = 30$, $t = 3$, and $f = 10$. In RCP-I, a node requires 0.14 seconds on average to obtain a threshold signature from a quorum, if all of the obtained signature shares are correct. The average execution time increases to 0.23 seconds in case of a share corruption attack. Extrapolating to a path length ℓ, an operation should take $0.14 \cdot \ell$ to $0.23 \cdot \ell$ seconds on average. For a DHT with 10^5 nodes, the average total time for RCP-I is then 3 to 4 seconds with $\ell = 20$.

In RCP-II, a node takes 0.04 seconds on average to obtain the required signed public keys and the signed routing information from a correct peer. A single signature verification takes 0.004 seconds on average. The median latency value over Planetlab is roughly 0.08 seconds [45]; $\Delta = 0.08$ seconds for $c = 0.5$. With a chain of signed public keys of length ℓ, the total communication time is $0.14 + 0.04 \cdot (\ell - 1) + \cdot \frac{\Delta \cdot (\ell - 2)}{c \cdot (1 - \epsilon)} + 0.004 \cdot \frac{\ell(\ell-1)}{2}$ which for 10% Byzantine peers, is 4.68 seconds in expectation. To a first-approximation, the execution times of our protocols seem quite reasonable.

System Load: We address the issue of system load under the assumption that signature verification is the most significant computational operation. We make back-of-the-envelope calculations to obtain the expected order of magnitude for our performance figures. For RCP-I, from the above discussion, each signature verification takes 0.004 seconds; thus, the total CPU time required per execution is $0.004 \cdot \ell \cdot (1 + s + s^2)$; this includes the costs due to share corruption attacks. For $\ell = 20$ and $s = 30$, this value is 75 CPU seconds, spread out over 600 nodes. Therefore, the number of executions of RCP-I that can be started per second on average is $n/75 \approx 10^3$ when $n = 10^5$. Note this rate value is for *the entire system*. Now, if no share corruption attacks occur, the total CPU time

TABLE II
THE EXPECTED NUMBER OF SECONDS BEFORE A QUORUM EXPERIENCES A MEMBERSHIP CHANGE (r_Q).

s	10			15			20			25			30		
n_Q	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
r_Q	526	351	175	350	234	117	263	132	88	210	140	70	175	87	58

required per execution becomes $0.004 \cdot \ell \cdot (1 + s)$ which, for the same parameter values, is 2.5 CPU seconds. This implies that $4 \cdot 10^4$ executions can be started per second on average in the entire system. For RCP-II, the total CPU time required for execution is given by $0.004 \cdot \left(\ell + \frac{(\ell-1)\cdot\ell}{2\cdot(1-\epsilon)} \right)$ which, for the same parameters and $\epsilon = 1/10$ is roughly 1 CPU second on average. Therefore, approximately 10^5 executions can be started per second on average in the entire system.

B. Analysis and Discussion

As mentioned in Section III-A, important questions remain with regards to translating theoretical results to a practical setting. In particular, two quantities of interest are the size of quorums, s, and the number of quorums to which each peer belongs, n_Q. Unfortunately, pinning down these quantities is non-trivial. Only asymptotic analysis is present in the literature. Furthermore, it is not a simple case of substituting hard numbers because s depends on a number of parameters: (1) the exact guarantees being made, (2) algorithms for quorum maintenance, (3) the tools of analysis (i.e. form of Chernoff bounds used) and many more. Evaluating these parameters is outside the scope of this work. Instead, we assume a range of values for s and n_Q. As our protocols appear to be the most efficient to date, the following results illuminate what currently seems possible in practice.

System Churn and DKG: The performance of our two protocols will likely depend on system churn. A common metric for measuring the degree of churn is *session time*: the time between when a node joins the network and when it leaves [46]. As discussed in Section III-E, we make the standard assumption that the cost of joining the network is large enough so as to prevent the adversary from substantially increasing the rate of churn through rapid rejoin operations.

Part I - An Argument for Batching: Investigations have yielded differing measurements for median session times. The Kazaa system was found to have a median session time of 144 seconds [47]. In the Gnutella and Napster networks, the median session time was measured to be approximately 60 minutes [48]. Measurements of the Skype P2P network yielded a median session time of 5.5 hours for super-peers [49]. Here, we temporarily assume a median session time of 60 minutes and a standard Poisson model of peer arrivals/departures as in [46], [50]. To calculate churn rate, r (number of arrivals/departures per second), based on the median session time t_{med} (in seconds), we use the formula of [46]: $r = (n \cdot \ln 2)/t_{med}$. For $n = 10^5$ and $t_{med} = 3600$ seconds, $r \approx 19$. Assuming that join and leave events occur independently of each other, Table II gives the expected number of seconds, r_Q, at which point a quorum will undergo a membership change when each peer belongs to n_Q quorums. Our choice of $n_Q \leq 3$ is based upon the reasonable assumption that overlap occurs only with immediate neighboring quorums in the ID space.

TABLE III
MEDIAN SESSION TIMES (IN HOURS) DERIVED FROM VALUES FOR s, n_Q AND r_{DKG} (IN HOURS).

s	10			15			20			25			30		
r_{DKG}	0.167			0.25			0.33			0.42			0.5		
n_Q	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
t_{med}	0.19	0.39	0.58	0.66	1.32	2.00	0.81	1.62	2.44	1.23	2.46	3.70	1.23	2.47	3.70

In several cases, the r_Q values are less than the corresponding median DKG completion times in Table I. Therefore, a quorum may not be able to execute DKG often enough to accommodate each membership change. However, join operations can be queued and performed in batches. Executing DKG for a batch of joins does not increase the message complexity and message size increases only linearly in the batch size (see [40, Sec. 6]). Therefore, batching can mitigate the effects of churn and it seems plausible that peers would tolerate some delay in joining in exchange for security.

Part II - Batching and the Security Threshold: Batching join events improves performance; however, many peers might leave a quorum before a new batch is added, thus violating the security threshold. Hence, we are interested in the session time value required such that this is not likely to occur. Based on Table I for $s = 20$ and $n_Q = 1$, DKG completes within 68 seconds. The number of leave events a quorum can suffer while not exceeding the crash limit is $f = 6$. If Byzantine peers leave, more crashes are tolerable; however, identifying such events is impossible, so we assume the worst case of $f = 6$. Assuming DKG executes every $r_{DKG} = 1200$ seconds, we seek the median session time such that at most 6 peers leave the system within 1268 seconds. With $n/s = 5000$ quorums in the system, each experiencing 6 leave events within 1268 seconds, the system churn rate is $r = 23.7$. This gives $t_{med} = 2930$ or, equivalently, 49 minutes. Therefore, with this t_{med}, we expect the system to remain secure. Moreover, a quorum only spends $68/1268 = 5.4\%$ of the time executing DKG.

Certain parameters can be tuned to offer performance tradeoffs. Decreasing r_{DKG} yields smaller required median session times; however, the percentage of time spent on DKG increases. Such tuning would depend on the desired system performance, the application, and s and n_Q. Table III gives session time calculations for other values of s, r_{DKG} and n_Q.

Required session times increase with s. Notably, for $s = 30$ and $n_Q = 1$, t_{med} does not far exceed the 60 minutes in [48]. As n_Q increases, the required session times grow linearly. However, our maximum of 3.7 hours is still less than t_{med} measured for super-peers in the Skype network [49]. We tentatively conclude that our protocols can be deployed in applications where session times range from 10 minutes to a few hours and that such applications currently exist.

VI. Future Work

The performance of a complete system is an important open question. The quorum topology chosen is crucial and optimizing this in practice is a topic of future work. While we focus on DHTs, our results may apply to other P2P designs and more general settings where groups of machines, some with untrustworthy members, must communicate; it would be of interest to identify such applications.

Acknowledgements: This work is supported by NSERC, MITACS, and David R. Cheriton Graduate Scholarships. We thank both Douglas Reeves and the anonymous reviewers for their constructive feedback.

References

[1] E. Sit and R. Morris, “Security Considerations for Peer-to-Peer Distributed Hash Tables,” in 1^{st} *Intl. Workshop on Peer-to-Peer Systems*, 2002, pp. 261–269.

[2] D. S. Wallach, “A Survey of Peer-to-Peer Security Issues,” in *ISSS*, 2002, pp. 253–258.

[3] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson, “Profiling a Million User DHT,” in *IMC*, 2007, pp. 129 – 134.

[4] M. Steiner, T. En-Najjary, and E. W. Biersack, “A Global View of KAD,” in *IMC*, 2007, pp. 117 – 122.

[5] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, “Vanish: Increasing Data Privacy with Self-Destructing Data,” in *USENIX Security Symp.*, 2009, pp. 299–315.

[6] V. Pappas, D. Massey, A. Terzis, and L. Zhang, “A Comparative Study of the DNS Design with DHT-Based Alternatives,” in *INFOCOM*, 2006, pp. 1–13.

[7] B. Awerbuch and C. Scheideler, “Towards a Scalable and Robust DHT,” in *SPAA*, 2006, pp. 318–327.

[8] ——, “Towards Scalable and Robust Overlay Networks,” in *IPTPS*, 2007.

[9] ——, “Robust Random Number Generation for Peer-to-Peer Systems,” in *OPODIS*, 2006, pp. 275–289.

[10] J. Saia and M. Young, “Reducing Communication Costs in Robust Peer-to-Peer Networks,” *Information Processing Letters*, vol. 106(4), pp. 152–158, 2008.

[11] A. Fiat, J. Saia, and M. Young, “Making Chord Robust to Byzantine Attacks,” in *ESA*, 2005, pp. 803–814.

[12] M. Naor and U. Wieder, “A Simple Fault Tolerant Distributed Hash Table,” in *IPTPS*, 2003, pp. 88–97.

[13] K. Hildrum and J. Kubiatowicz, “Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-peer Networks,” in *DISC*, 2004, pp. 321–336.

[14] H. Johansen, A. Allavena, and R. van Renesse, “Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays,” in *OSR*, 2006, pp. 3–13.

[15] J. Douceur, “The Sybil Attack,” in *IPTPS*, 2002.

[16] G. Urdaneta, G. Pierre, and M. van Steen, “A Survey of DHT Security Techniques,” *ACM Computing Surveys*, 2009, to appear.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications,” in *SIGCOMM*, 2001, pp. 149–160.

[18] M. Young, A. Kate, I. Goldberg, and M. Karsten, “Practical Robust Communication in DHTs Tolerating a Byzantine Adversary,” Tech. Rep. CACR 2009-31, 2009, http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-31.pdf.

[19] F. B. Schneider, “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial,” *ACM Computing Surveys*, vol. 22(4), pp. 299–319, 1990.

[20] M. K. Reiter, “The Rampart Toolkit for Building High-Integrity Services,” in *Intl. Workshop on Theory and Practice in Distributed Systems*, 1995, pp. 99–110.

[21] M. Castro and B. Liskov, “Byzantine Fault Tolerance Can Be Fast,” in *DSN*, 2001, pp. 513–518.

[22] C. Cachin and J. Poritz, ‘Secure Intrusion-tolerant Replication on the Internet,” in *DSN*, 2002, pp. 167–176.

[23] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, ‘FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted environment,” in *OSDI*, 2002, pp. 1–14.

[24] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, “Fault-Scalable Byzantine Fault-Tolerant Services,” in *SOSP*, 2005, pp. 59–74.

[25] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, “HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance,” in *OSDI*, 1999, pp. 177–190.

[26] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, W. W. H. Weatherspoon, C. Wells, and B. Zhao, “OceanStore: An Architecture for Global-Scale Persistent Storage,” in *ASPLOS*, 2000, pp. 190–201.

[27] R. Rodrigues and B. Liskov, “Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture,” MIT LCS, Tech. Rep. TR/932, December 2003.

[28] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee, “Large-Scale Byzantine Fault Tolerance: Safe but Not Always Live,” in *HotDep*, 2007.

[29] R. Rodrigues, B. Liskov, and L. Shrira, “The Design of a Robust Peer-to-Peer System,” in 10^{th} *ACM SIGOPS European Workshop*, 117-124, p. 2002.

[30] P. Weng, N. Hopper, I. Osipkov, and Y. Kim, “Myrmic: Secure and Robust DHT Routing,” University of Minnesota, Tech. Rep. 2006/20, 2006.

[31] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach, ‘Secure Routing for Structured Peer-to-Peer Overlay Networks,” in *OSDI*, 2002, pp. 299–314.

[32] A. Kapadia and N. Triandopoulos, “Halo: High-Assurance Locate for Distributed Hash Tables,” in *NDSS*, 2008.

[33] A. Nambiar and M. Wright, “Salsa: A Structured Approach to Large-Scale Anonymity,” in *CCS*, 2006, pp. 17–26.

[34] P. Mittal and N. Borisov, “ShadowWalker: Peer-to-peer Anonymous Communication using Redundant Structured Topologies,” in *CCS*, 2009.

[35] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance and Proactive Recovery,” *ACM Transactions on Computer Systems*, vol. 20(4), pp. 398–461, 2002.

[36] D. Boneh, B. Lynn, and H. Shacham, “Short Signatures from the Weil Pairing,” in *ASIACRYPT*, 2001, pp. 514–532.

[37] P. Feldman, ‘A Practical Scheme for Non-Interactive Verifiable Secret Sharing,” in *FOCS*, 1987, pp. 427–437.

[38] T. P. Pedersen, “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing,” in *CRYPTO*, 1991, pp. 129–140.

[39] A. Boldyreva, “Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme,” in *PKC*, 2003, pp. 31–46.

[40] A. Kate and I. Goldberg, “Distributed Key Generation for the Internet,” in *ICDCS*, 2009, pp. 119–128.

[41] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, “Aggregate and Verifiably Encrypted Signatures from Bilinear Maps,” in *EUROCRYPT*, 2003, pp. 416–432.

[42] H. Jiang and C. Dovrolis, “Passive Estimation of TCP Round-Trip Times,” *CCR*, vol. 32, pp. 75–88, 2002.

[43] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, “A Blueprint for Introducing Disruptive Technology into the Internet,” *CCR*, vol. 33, no. 1, pp. 59–64, 2003.

[44] A. Kate and I. Goldberg, “Asynchronous Distributed Private-Key Generators for Identity-Based Cryptography,” Cryptology ePrint Archive, Report 355, 2009.

[45] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, “Designing a DHT for Low Latency and High Throughput,” in *NSDI*, 2004, pp. 85–98.

[46] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, “Handling Churn in a DHT,” in USENIX *Annual Technical Conf.*, 2004, pp. 127–140.

[47] P. K. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, “Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload,” in *SOSP*, 2003, pp. 314–329.

[48] S. Saroiu, P. K. Gummadi, and S. D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems,” in *MMCN*, 2002, pp. 314–329.

[49] S. Guha, N. Daswani, and R. Jain, ‘An Experimental Study of the Skype Peer-to-Peer VoIP System,” in *IPTPS*, 2006.

[50] D. Liben-Nowell, H. Balakrishnan, and D. Karger, “Analysis of the Evolution of Peer-to-Peer Systems,” in *PODC*, 2002, pp. 233–242.