Towards Practical Communication in Byzantine-Resistant DHTs

Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten

***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.

***Index Terms*—Computer network security; Distributed algorithms.**

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 [57], [63]. This is a practical concern since large-scale P2P systems currently exist such as the Azureus DHT [19] and the KAD DHT [58], each of which sees over one million users on a daily basis. In addition to file sharing, there are proposals for using P2P systems to protect archived data [23], combat computer worms [4] and re-implement the Domain Name System [62]; such applications would likely benefit from increased security as well.

Malicious activity is typically modeled by assuming an adversary that controls a significant fraction of the machines in the network. A machine that is controlled by the adversary is said to suffer a *Byzantine fault* and we often refer to a *Byzantine adversary* that uses these machines in concert in order to maximize disruption to the network. There are a number of results on P2P systems that can provably tolerate Byzantine faults [6]–[8], [21], [27], [32], [44], [53]. To date, the majority of results pertain to DHTs. A common technique in DHTs that tolerate adversarial faults is the use of *quorums* which are sets of peers where a bounded fraction (less than $1/2$) of the members are controlled by the adversary. A quorum replaces an individual peer as the atomic unit. Adversarial behavior can then 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 and this is our focus.

Before presenting our results, we lay the groundwork for the ideas in this paper. First, we elaborate on the relevance of Byzantine fault tolerance in P2P systems in practice. Second, there have been a number of advances towards achieving such fault tolerance using quorums, and we present an overview of the main ideas present in the literature.

A. The Byzantine Fault Model

The Byzantine fault model captures a wide variety of challenging attacks. Two common real-world examples are pollution attacks and index poisoning attacks which are both aimed at preventing peers from obtaining desired content. In a pollution attack, the attacker corrupts data and then facilitates the sharing of this data on a massive scale. Interested users spend resources obtaining this polluted data only to find it unusable. In a recent study, it was discovered that over 50% of files in Kazaa were polluted [39]. Similar in nature, an index poisoning attack is carried out by flooding the network with identifiers that do not correspond to any actual data [40]. Queries involving these identifiers will stall and eventually fail, consuming resources and frustrating the user. A recent study of the FastTrack and Overnet systems demonstrated instances where 50% to 90% of all advertisements for a particular file suffered from poisoning [40]. Such findings illustrate that, far from being isolated incidents, such attacks are prevalent.

Content sharing is clearly central to the P2P paradigm. However, providing content consumes the resources of the provider since upload bandwidth is often more restricted than download bandwidth; consequently, peers may be tempted to avoid advertising. This behaviour is known as *free riding* and it is common in many P2P systems. An analysis of the Gnutella network in 2000 revealed that nearly 70% of peers contributed zero content while approximately 50% of requested content was provided by only 1% of the peers [2]. A subsequent study in 2005 found that the situation had worsened with 85% of peers in the Gnutella network sharing no content [30].

M. Young is with the School of Computing at the National University of Singapore, COM 2 #02-19, Singapore, 117417. Email: dcsmmry@nus.edu.sg

A. Kate is with the Max Planck Institute for Software Systems, MPI-SWS Wartburg, Martin-Luther-Straße 12, D-66111 Saarbrücken, Germany. Email: aniket@mpi-sws.org

I. Goldberg and M. Karsten are with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada, N2L 3G1. Email: {iang,mkarsten}@cs.uwaterloo.ca

Certain malicious behaviour may also amount to denial-of-service (DoS) attacks. For instance, a Byzantine peer may route *any* query to itself and then offer only a corrupted version of data item [18]. Rapid joins and departures have also been considered as a DoS attack [63]. More straightforward attacks include Byzantine peers overwhelming targeted nodes by repeatedly issuing queries [57].

Finally, in discussing fault tolerance, it is important to address the Sybil attack [17]. Here the adversary controls a large number of identities; however, in contrast to the Byzantine fault model where the adversary typically controls machines, the identities generated by a Sybil adversary need not correspond to physical resources. Therefore, identities can be generated cheaply and, indeed, the adversary may even constitute a majority of network identities. In this case, algorithms that depend on majority action for fault tolerance are ineffective. In this work, we do not explicitly address such attacks. However, several schemes have been proposed for countering the Sybil attack and our results can likely be used in conjunction with these (see the survey of [61]).

B. Quorums and DHTs

A popular approach to dealing with the attacks discussed above is to use quorums [5]–[8], [21], [27], [44], [53] (discussed in Section II). A quorum is a set of peers where the majority of the members have not suffered Byzantine faults. While quorums have been applied to a variety of P2P structures, we restrict our scope to DHTs for ease of exposition and because a number of results pertaining to adaptive adversaries have been shown only for DHTs. Typically, a quorum consists of $\Theta(\log n)$ peers where n is the total number of peers in the system. If the Byzantine peers attempt to deviate from the protocol, this errant behavior can be overcome through majority action. For instance, content may be stored in a distributed and redundant fashion across members of a quorum such that the content cannot be polluted by a single host peer. Poisoning attacks can be mitigated by having peers belonging to the same quorum validate content before it is advertised. Furthermore, a useful property of quorums is that those peers who violate protocol can be ejected from their quorums which effectively removes them from the system.

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 $Q_1, Q_2, \ldots, Q_\ell$ 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 [60], $\ell = O(\log n)$ which gives a $O(\log^3 n)$ message complexity which is likely prohibitively expensive for practical values of n.

Note that this approach can fail if the adversary obtains a majority of corrupted peers in any quorum. This can happen if an adaptive adversary has its peers join and depart the network until a favorable placement is attained. For instance, the adversary may target a quorum in the following manner. The adversary adds a corrupted peer p into the system to see where it is placed. If p lands within the target quorum, the adversary keeps the peer active in the system; otherwise, p departs. Over a number of joins and departures the adversary may accumulate a large number of peers in the target quorum, eventually obtaining a majority. Remedies to this challenging *adaptive Byzantine adversary* have been proposed using quorums and we discuss this further in Section II.

To date, the previous best communication complexity for using quorums was given by Saia and Young [53] who give a randomized communication protocol for DHTs which achieves $O(\log^2 n)$ messages in expectation over a path of length $O(\log n)$. While communication between two quorums incurs an expected constant number of messages, the analysis in [53] yields a large constant. Furthermore, with probability $1-o(1)$, some peers will incur a non-constant ($\omega(1)$) message complexity (see [65]). 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 theoretical results exist on the feasibility of robust communication, they are likely not practical. This presents an impediment to the deployment of such systems and we seek to address this outstanding problem.

C. Our Contributions

We improve upon all previously known results involving communication between quorums [7], [21], [44], [53]. We summarize our main results below:

Theorem 1. *In the computational setting, for an adversary that controls 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 forwarding 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 forwarding 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 [21], 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 [53]. 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 indicates that our protocols perform well under significant levels of churn and faulty behaviour. In particular, for a 10^5-node system with $s = 30$ and $\ell = 20$, our results imply RCP-I and RCP-II both complete in under 5 seconds.

II. Related Work

State machine replication (SMR) is a standard method for implementing highly fault-tolerant services [55]. Services are replicated over multiple servers to provide a high-integrity distributed system. While P2P systems do not align perfectly with the SMR paradigm [55], the literature on Byzantine fault-tolerant replication is relevant. Early work by Reiter [48] gives protocols for Byzantine agreement and atomic broadcast. Our first protocol shares some common features with the multicast protocol of [48], 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 [14] demonstrated efficient Byzantine fault-tolerant SMR; however, this seems unsuitable for a P2P setting due to scaling issues. Other Byzantine fault-tolerant systems exist such as SINTRA [12], FARSITE [3], the Query/Update protocol [1] and the HQ system [15]; however, these likely do not scale to the P2P domain.

Two *implemented* large-scale Byzantine fault-tolerant storage architectures are OceanStore [36] and Rosebud [51]. The latter scales up to tens of thousands of nodes and handles changing membership. However, 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 [50] and Rodrigues, Liskov and Shrira [52] 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 secure routing. Wang *et al.* [64] 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], [21], [27], [44]. These results make use of quorums of $\Theta(\log n)$ peers such that a majority of the peers in a quorum are correct. Awerbuch and Scheideler show how to tolerate a powerful adaptive adversary who attempts to gain a majority of Byzantine peers in a quorum [6]–[8]. Recent simulation work by Sen and Freedman [56] show how these results on maintaining a majority of correct peers in each quorum can be made practical in large-scale systems. Saia and Young [53] demonstrate more efficient robust communication but, as discussed earlier, several issues remain unresolved. Finally, a version of this current work appeared in [66].

Castro *et al.* [13], Halo [33], and Salsa [43] handle Byzantine faults by routing along multiple diverse routes. The proposal in [13] requires a CA whereas we do not rely on any trusted third party. In both [33] and [43], the guarantees are unclear against an adversary who owns a large IP-address space or targets identifiers over time as described in [7]. Such an adaptive adversary could potentially compromise the "knuckle" nodes in [33] or the global contacts used in [43]; in contrast, defenses against an adaptive adversary are known for quorum-based protocols [6]–[8].

There are several other works relating to issues of security in P2P networks. The ShadowWalker system [42] addresses the issue of anonymity and routes securely using the notion of multiple "shadows" which are similar to a quorum; however, our protocols differ significantly. The Brahms system [11] allows for uniform sampling of peers despite a Byzantine adversary. The Fireflies architecture [32] allows each peer to remain informed of live members despite Byzantine attacks; however, its applicability likely extends only to single hop overlays such as in work by Gupta *et al.* [26] and secure routing in multi-hop networks is not treated.

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 depart 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 [29]. Formally, let $delay(t)$ denote the time between the moment t when a message is initially sent and the moment when it is received at the destination; the sender retransmits the message until it is received correctly. Then the weak synchrony assumption is that the $delay(t)$ does not grow faster than t indefinitely. This assumption seems valid in practice assuming that network faults are eventually repaired, and it avoids the well-known impossibility result regarding liveness [22].

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

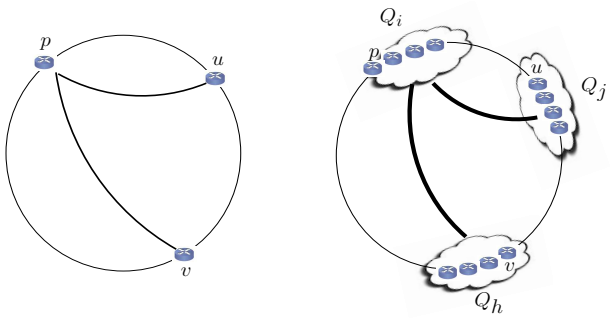

Fig. 1. (Left) Three peers on a DHT ring where p links to u and v. (Right) An example of a quorum topology in a DHT ring where $p \in Q_i$, $u \in Q_j$ and $v \in Q_h$. Thick lines signify all-to-all inter-quorum links.

A. The Quorum Topology

There are several approaches to how quorums are created and maintained [7], [44], [53]; we refer the reader to [21] for a detailed explanation of one approach. As a simple example, the links in a quorum topology consist of the union of Chord's links along with a set of redundant links. Specifically, for peer p's link to q in Chord, each peer in quorum Q_p has a link to every peer in Q_q (and links to every peer in its own quorum Q_p). A link to a Byzantine peer may be treated as faulty or unobtainable if the peer misbehaves, but only links between correct peers are required to be functional. Conceptually, despite a number of 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 call this 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 [21]). We assume the following four simple invariants are true:

1) *Goodness:* each quorum has size $\Theta(s)$ for $s = \Omega(\log n)$ 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 invariants are standard in the sense that previous work on quorums in DHTs ensure they hold with probability nearly equal to 1. For example, results for maintaining the goodness invariant are known [6]–[8]. For the membership invariant, there exist quorum topologies where a peer may belong to several quorums simultaneously [21], [44]. 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 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 on the network. 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 [28] in an appropriate group.

Our protocols guarantee successful transmission of m; however, feasibility is not enough. Our protocols must be efficient in terms of (1) the costs to correct peers for legitimate network operations *and* (2) the costs due to adversarial interference. 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 [20] or by a completely distributed approach using a DKG protocol [46]. 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 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.

We use the threshold version [9] of the Boneh-Lynn-Shacham (BLS) signature scheme [28] to authenticate communication between quorums. Its key generation does not mandate a trusted dealer and the signature generation protocol does not require any interaction among the signing parties or any zero-knowledge proofs.

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 across 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 by Kate and Goldberg [35] is the first DKG for an asynchronous setting; therefore, it is uniquely suitable for deployment in a P2P network. In addition to 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. Notably, the message complexity of a batch of peers P all joining and/or all leaving the quorum is the same as for a single peer joining/leaving the quorum, while the bit complexity increases only linearly with $|P|$ (see [35, Sec. 6]); for efficiency, we batch such operations during our analysis in Section V-B. The DKG protocol also considers *mobile adversary* [45] and provides proactive security using share renewal and share recovery protocols.

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 [57], [63]. 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 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.* [21]. 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, or tit-for-tat behavior for uploads/downloads. Such rules are known to everyone within a quorum and can be implemented at the software level or simply agreed upon by quorum members. As discussed in Section I-B, requests from a peer q that deviate from the rule set are ignored by the other members of its quorum, effectively removing q from the system.

E. Feasibility via Quorums, but Efficiency via Cryptography

We now make explicit a crucial point regarding the feasibility versus the efficiency of robust communication. As we have discussed, in the presence of Byzantine peers, no single peer can be trusted and quorums are employed to overcome this trust deficit through majority action. Using the simple protocol outlined in Section I, the transmission of a message is guaranteed. *Therefore, quorums allow for robust communication without the need for cryptographic techniques*. However, as we now discuss, cryptographic techniques are important to achieving efficient robust communication.

A Problem of Spamming: Note that spamming attacks can pose a critical problem in a system that employs quorums. 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. To investigate the implications of spamming, consider the case where peers act on any received request and call this the *passive scenario*. 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. Even if it is possible to detect spurious requests at a global scale, each correct peer would be required to maintain $O(n)$ records to exclude faulty peers from the system.

Therefore, the passive scenario is undesirable since spamming allows the adversary to consume the resources of correct peers at little cost to itself. 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.

Legitimacy and the Utility of the Quorum Topology: We now contrast the passive scenario against another general scenario that we call the *prove-and-verify scenario* which assumes that proofs and verifications are required to initiate operations. We argue that this is superior to the passive scenario and discuss how the quorum topology is used in previous works to provide the framework for proving and verifying operations.

P2P systems often lack admission control and, if forced to depart 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 are suggested in [13] and CAPTCHAs as suggested in [43]. 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 and so τ is large due to the abundance of potential targets.

In contrast, in the 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 [21], [44]; 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 greatly reduced. Furthermore, correct peers are not required to maintain records on misbehaving peers. However, while the prove-and-verify scenario is far more robust to spamming, there are shortcomings to this actual method of proof and verification and we discuss this

TABLE I
SUMMARY OF FREQUENTLY USED NOTATION.

Notation	Definition
m	Message being sent via our robust communication protocols
Q_p	Quorum of peer p
ϵ	Upper bound on the fraction of Byzantine peers in a quorum
s	Asymptotic size of quorums
ℓ	Number of hops traversed by communication of m
K_{Q_p}	Quorum public key of peer p's quorum
k_{Q_p}	Quorum private key of peer p's quorum
$\widehat{K}_{Q_p}$	Set of public key shares corresponding to K_{Q_p}
$(k_{Q_u})_p$	Peer p's individual share of k_{Q_u}
$\mathcal{RT}_{Q_p}$	Routing table information for all peers in Q_p
ts	Time stamp
S_i	Signature corresponding to the i^{th} hop in RCP-I
M_i	Chain of certificates corresponding to the i^{th} hop in RCP-II
D	Target set of peers who should receive m

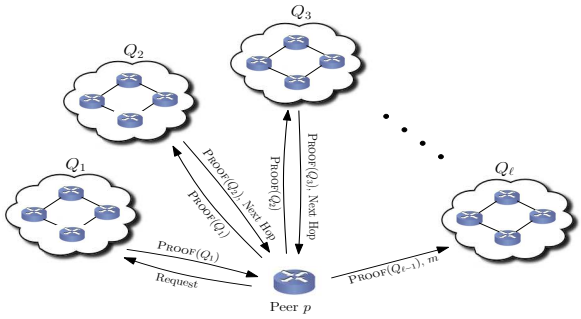

Fig. 2. Our general robust communication scheme. At step $i = 1, ..., \ell - 1$, peer p presents proof, $\text{PROOF}(Q_i)$, that quorum Q_i sanctions p's action, and receives new proof from Q_{i+1} in addition to routing information for the next hop. At the final step ℓ,peer p sends $\text{PROOF}(Q_{\ell-1})$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

next.

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 when membership changes. However, our experimental results in Section V suggest 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. 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; Table I summarizes our notation. Consider a sending peer p who wishes to send a message m to peer p'. We make the standard assumption that 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 $\text{PROOF}(Q_1)$. 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) $\text{PROOF}(Q_2)$, that will convince Q_3 that p's actions are legitimate. This continues iteratively until p contacts the quorum holding p' 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.

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 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.

System Churn and Protocol Design Considerations: The rate of churn in P2P networks has a significant impact on system performance and this is a concern reflected in the design of our two protocols. First, both of our protocols perform communication in an iterative fashion (in contrast to recursive routing) so that failures due to stale routing table information are detected immediately. While not specified in our pseudocode, upon experiencing a failure, the calling peer and its quorum may request that the responsible quorum update its routing table information, at which point the original request can proceed. Second, each of our protocols are designed to operate in different regimes of churn. As we will discuss later in Section IV-C, RCP-I is more tolerant of churn as it requires few updates when peers join or depart the system. On the other hand, RCP-II is more message-efficient and can be used in situations where the system is experiencing less churn.

Beyond these design choices, our protocols also inherit certain churn-resistant properties of the underlying quorum topology and routing mechanisms of DHTs. That is, the quorum topology requires only a polylogarithmic amount of routing table information that needs to be kept updated, while routing with our protocols still operates along $O(\log n)$ hops which yields a relatively small number of potential failure points. Additionally, the redundant link architecture used by quorums provides some robustness to rapid system

RCP-I: SENDING PEER p

Initial Step:

1: $p \in Q_1$ sends the following request to all peers in Q_1: $[p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts_1]$
2: p interpolates all received signature shares to form: $\mathcal{S}_{Q_1} \leftarrow [p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts_1]_{k_{Q_1}}$

Intermediate Steps:

3: **for** $i = 2$ to $\ell - 1$ **do**
4: p sends $\mathcal{S}_{i-1}$ and ts_i to every peer in Q_i and requests a signature $\mathcal{S}_i$, public key $K_{Q_{i+1}}$ and routing information for Q_{i+1}.
5: p interpolates received signature shares to form $\mathcal{S}_i \leftarrow [p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts_i]_{k_{Q_i}}$.
6: p verifies if $\mathcal{S}_i$ is valid using K_{Q_i}.
7: **if** ($\mathcal{S}_i$ is invalid) **then**
8: p sends signature shares to each peer in Q_i.

Final Step:

9: p sends $\mathcal{S}_{Q_{\ell-1}}$ to $D \subseteq Q_\ell$ along with m.

RCP-I: RECEIVING PEER $q \in Q_i$

Initial Step:

1: **if** ($q \in Q_1$ receives a request by p) **then**
2: 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:

3: **if** (q receives $\mathcal{S}_{i-1}$ and ts_i from p) **then**
4: 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.
5: **if** (q receives signature shares from p) **then**
6: q verifies all shares using public key shares and informs p of invalid shares.

Fig. 3. Pseudocode for RCP-I

membership changes along the lines investigated in [37].

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 $[p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts_1]$ 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 of the rule set to within some bound to compensate for clock drift. Peer p interpolates these signature shares to generate the signature: $\mathcal{S}_1 \leftarrow [p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts_1]_{k_{Q_1}}$.

In each intermediate step $i = 2, ..., \ell - 1$, p sends its most recent signature $\mathcal{S}_{i-1}$ 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 $\mathcal{S}_{i-1}$. If $\mathcal{S}_{i-1}$ 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 $\mathcal{S}_i$ 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 $\mathcal{S}_i$. We refer to this attack as the *share corruption attack*. Here, the individual public/private key shares play a crucial role. To obtain $\mathcal{S}_i$, 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 $\mathcal{S}_i$. 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 forwarding peer has message complexity at most 4 messages.*
- *The latency is at most $2 \cdot (\ell - 2) + 2$.*

Proof: First, we prove correctness. We show that if p is correct and has not violated the rule set, at each step i of the protocol p either (1) receives a valid signature and routing information for the next step or (2) terminates the protocol by delivering m to all members of D; correctness follows directly. Our proof is by induction on i:

Base Case: Consider the initial step $i = 1$ where p communicates with the peers in its quorum $Q_p = Q_1$ about sending the message m. If p is correct and has not violated the rule set, upon receiving $[p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts_1]$ all correct peers will send their shares to p. Therefore, p is guaranteed to form $\mathcal{S}$ by the goodness invariant. Peer p can then check whether $\mathcal{S}$ is valid and, if so, sets $\mathcal{S}$ to be $\mathcal{S}_1$. Otherwise, p must overcome the share corruption attack. Since p belongs to Q_1, peer p knows the individual public key shares of each peer in Q_1 and can therefore detect which shares are invalid and construct $\mathcal{S}_1$. Finally, p already has the routing information for Q_2; therefore, the base case holds.

Inductive Hypothesis: Assume that up to step $i < \ell$, p has obtained the correct signatures and routing information.

Inductive Step: At step $i+1$, peer p sends $\mathcal{S}_i$ to Q_{i+1}. By the inductive hypothesis, this signature is valid and p possesses the routing information for Q_{i+1}. If $i < \ell - 1$, and no corrupted share attack occurs, then p's request for $\mathcal{S}_{i+1}$ and the routing information for Q_{i+2} will be satisfied due to the goodness

invariant. Otherwise, p must overcome the corrupted share attack by sending all signed shares to all other peers in Q_{i+1}. Each correct peer in Q_{i+1} can detect and inform p which peers sent an invalid share. Due to the goodness invariant, peer p can majority filter on these responses to determine the invalid shares and then construct $\mathcal{S}_{i+1}$. If $i = \ell - 1$, p possesses the routing information for Q_ℓ to deliver m to all members of $D \subseteq Q_i$ and the protocol terminates successfully. In either case, the induction holds.

We now analyze the costs of our protocol. In the first step, even in the event that a share corruption attack occurs at most one round-trip round of communication occurs (between p and Q_1). For steps $i = 2, ..., \ell - 1$, if a share corruption attack occurs, at most two rounds of message exchange occur: (1) p sends to Q_i and Q_i sends back to p and (2) p transmits shares to Q_i who then send the correct shares back to p. Adding the last step, the latency is $2 \cdot (\ell - 2) + 2$. In terms of message complexity, in the first round, peer p must send a request to and receive a response from each peer in Q_1; this totals at most $2s$ messages. For steps $i = 2, ..., \ell - 1$ peer p must both send a request to and receive a response from each peer in a quorum; if a corruption attack occurs, p must send another message to each peer in a quorum (with all signed shares collected together) and receive back a response. Therefore, this incurs at most $4 \cdot s$ messages. In the last step p sends to all members of D. Hence, the message complexity is at most $4 \cdot s \cdot (\ell - 2) + |D| + 2s$. For every other involved peer $q \notin D$, $q's$ message complexity is at most 4; clearly, peers in D receive one message.

■

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 forwarding peers. In exchange, joins and departures 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 accommodated. For a peer $u \in Q_i$, each entry of its routing table has the form $[Q_j, p_{\texttt{ID}}, p'_{\texttt{ID}}, K_{Q_j}, ts]$. 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 $[p_{\texttt{ID}}|p_{\text{addr}}|\texttt{key}|ts]$ 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 $M_1 = [p_{\texttt{ID}}|p_{\text{addr}}|\ \texttt{key}|ts]_{k_{Q_1}}$.

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 [10].

Peer p may choose a Byzantine peer that does not respond. In that case, after an appropriate time interval, 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; *any* distribution suffices, including the Poisson, exponential, and gamma distributions previously used to characterize round

RCP-II: SENDING PEER p

Initial Step:

1: p sends the following to each peer $q \in Q_1$: $[p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts]$
2: p gathers all responses and constructs:

$$M_1 \leftarrow [p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts]_{k_{Q_1}}$$

Intermediate Steps:

3: **for** $i = 2$ to $\ell - 1$ **do**
4: **while** (p does not have M_i and has waited time Δ since previous selection) **do**
5: p sends M_{i-1} to $q \in Q_i$ selected u.a.r. without replacement.
6: **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**
7: p uses K_{Q_i} to verify $K_{Q_{i+1}}$, $\mathcal{RT}_{Q_{i+1}}$ and $K_{Q_{i-1}}$.
8: **if** ($K_{Q_{i+1}}$, $\mathcal{RT}_{Q_{i+1}}$ and $K_{Q_{i-1}}$ are all verified) **then**
9: $M_i \leftarrow [M_{i-1}|[K_{Q_{i-1}}]_{k_{Q_i}}]$

Final Step:

10: p sends $M_{\ell-1}$ to $D \subseteq Q_\ell$ along with m.

RCP-II: RECEIVING PEER q

Initial Step:

1: **if** ($q \in Q_1$ receives $[p_{\text{ID}}|p_{\text{addr}}|\mathsf{key}|ts]$ from $p \in Q_1$) **then**
2: q checks that p's request is legitimate and, if so, sends its signature share.

Intermediate Steps:

3: **if** ($q \in Q_i$ receives M_{i-1} from p) **then**
4: **for** $j = i - 1$ downto 1 **do**
5: q uses K_{Q_j} to verify $K_{Q_{j-1}}$.
6: Peer q uses K_{Q_1} to verify M_1.
7: **if** verification is successful **then**
8: 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

trip time (RTT) over the Internet. In practice, peer p would set its own Δ by sampling the network using methods for estimating RTT [31]. Since at most a constant fraction of peers are Byzantine, taking the median from a sufficiently large sample will determine Δ and p will receive a response from any of the previously selected peers - *this is in accordance with the weak synchrony assumption* stated 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 forwarding peer has expected message complexity at most $\frac{2}{(1-\epsilon)\cdot c \cdot s}$.*
- *The expected latency is at most $\frac{(\ell-2)}{(1-\epsilon)\cdot c} + 2$.*

Proof: First we prove the correctness of our protocol and, as before, we show that if p is correct and has not violated the rule set, at each step i of the protocol p either (1) establishes a valid M_i and receives the routing information for the next hop or (2) terminates the protocol by delivering m to all members of D. Our proof is by induction on i.

Base Case: Consider the initial step $i = 1$ where p communicates with the peers in its quorum $Q_p = Q_1$ about sending the message m. If p is correct and has not violated the rule set, upon receiving $[p_{\text{ID}}|p_{\text{addr}}|\ \mathsf{key}|ts]$ all correct peers will send their shares to p. Therefore, p is guaranteed to obtain M_1 by the majority invariant. Peer p already has the routing information for Q_2; therefore, the base case holds.

Inductive Hypothesis: Assume that at step $i < \ell$, p has obtained a correct M_i and routing information for Q_{i+1}.

Inductive Step: First assume that $i = \ell - 1$. Then, by the induction hypothesis, peer p possesses $M_{\ell-1}$ and the necessary routing information to send this signature and message m to $D \subseteq Q_{i+1}$; thus the protocol terminates correctly. Otherwise, assume $i < \ell - 1$; we consider step $i + 1$. Peer p sends M_i to a peer $q \in Q_{i+1}$ selected uniformly at random without replacement. By the inductive hypothesis, the contents of M_i are valid and p possesses the necessary routing information. If q is a Byzantine peer, then p's request can fail and p can detect an invalid response using $K_{Q_{i+1}}$ obtained from the previous step. It is also possible that q is a correct but slow node and does not respond in a predefined time period. In this case, p re-issues its request to another randomly selected peer in Q_{i+1}; eventually, one of selected correct peers will respond with $[K_{Q_i}]_{k_{Q_{i+1}}}$, $[\mathcal{RT}_{Q_{i+2}}]_{k_{Q_{i+1}}}$ and $[K_{Q_{i+2}}]_{k_{Q_{i+1}}}$ to p. Peer p will verify this information and create a valid M_{i+1}. Therefore, at this point p possesses a correct M_{i+1} and routing information for Q_{i+2}; therefore, the induction holds.

Since RCP-II is a randomized algorithm, our costs are given in expectation. We assume the following: let X_i be a random variable denoting the time required for the i^{th} correct peer (note we condition on correctness) selected u.a.r without replacement by p to respond to p's request. We assume that $Pr[X_i \leq \Delta] = c$ where $c > 0$ is some constant probability.

We now calculate upper bounds of the expected resource costs. In the first step, in communicating with Q_1, peer p handles at most $2 \cdot s$ messages and the round-trip latency is 1. Then for each step $i = 2, ..., \ell-1$, let Y_i be the random variable with value 1 if the i^{th} peer is both correct and responds within time Δ; 0 otherwise. Then $Pr[Y_i = 1] \leq (1 - \epsilon) \cdot c$; for simplicity, set $\rho = (1 - \epsilon) \cdot c$ to be this probability of success. Let $Y = \sum_{i=1}^{s} Y_i$. The expected number of selections $E[Y]$ before p receives a response from a correct peer is at most:

$$\sum_{k=0}^{s} (1-\rho)^k \cdot \rho \cdot (k+1) = \rho \left(\sum_{k=0}^{s} (1-\rho)^k \cdot k + \sum_{k=0}^{s} (1-\rho)^k \right)$$

Therefore $E[Y] \leq \frac{1}{(1-\epsilon)\cdot c}$ and including the last step, the expected latency is at most $\frac{\ell-2}{(1-\epsilon)\cdot c} + 2$. The ℓ^{th} step requires D messages and one hop. In terms of expected message complexity, since each step requires at most 2 messages and the last step requires $|D|$ messages, we can give a crude upper

bound of $2s + \frac{2}{(1-\epsilon)\cdot c} \cdot (\ell - 2) + |D|$. However, note that once p hears back from a node, any message from any other previously selected nodes in the current step can be easily ignored/filtered. Therefore, per step, p handles $\frac{1}{(1-\epsilon)\cdot c} + 1$ messages. We can now give a more accurate upper bound of $2s + \frac{\ell-2}{(1-\epsilon)\cdot c} + (\ell - 2) + |D|$. Finally, while latency is measured in the number of communication rounds, the expected duration of time required for each intermediate round is $\frac{\Delta}{(1-\epsilon)\cdot c}$.

Regarding the expected message complexity of a forwarding peer $q \notin D$ along the lookup path, a correct peer chosen by p receives one message and sends one message. The probability that q is chosen is at most $1/((1-\epsilon)\cdot s)$; therefore the expected message complexity for q is at most $2/((1-\epsilon)\cdot s)$. ■

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 as shown in Lemma 2. 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. Rather than make it expensive for p to perform robust communication, RCP-II uses two properties to deter spamming: (1) it is inexpensive for a peer to detect spam and (2) the congestion suffered by a 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.

Faulty peers may misbehave in other ways with 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 limited since only p can use the certificate. An 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 the location indicated by the stale information was recently correct. Coupled with the standard assumption that ID collisions do not occur, this 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 Figure 4.

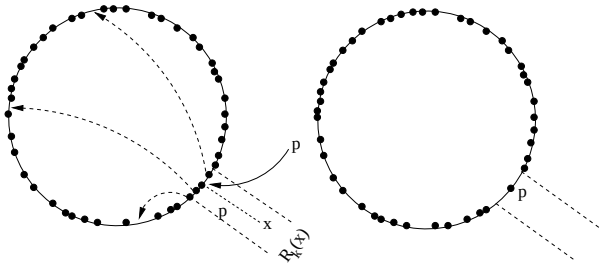

Fig. 5. An illustration of the cuckoo rule. (Left) Peer p is placed in its random location x. All peers in the k-region, $R_k(x)$ denoted by dashed lines, are assigned random locations in $[0, 1)$. (Right) After the cuckoo rule is executed, peer p is the only peer in $R_k(x)$.

Routing integrity is not compromised and, since routing tables can be signed periodically without significant CPU cost (see Section V), the impact of such an attack is negligible.

C. The Join Protocol and Membership Updates

For the sake of being self-contained, we describe how a peer would join our system. This involves a discussion of a result by Awerbuch and Scheideler [7] which guarantees the goodness invariant even if the number of joins and departures is polynomial in the size of the network n. More precisely, within a window of time, the adversary may opt to insert one of its spare faulty peers or remove a faulty peer already in the system. An adversary may attempt to gain a majority of Byzantine peers in a targeted quorum Q by having one of its peers q' join the system. If q''s location in the DHT does not allow it to become part of Q, then the adversary removes q' and has it rejoin for another attempt. By repeating this process, the adversary can eventually obtain a majority in Q, at which point the security of the system is compromised. The protocol for defending against such attacks is the *cuckoo rule* developed by Awerbuch and Scheideler [7]. Assume that the identifier space of the DHT is normalized to be [0,1). For any interval $I \subset [0, 1)$, the cuckoo rule maintains two invariants. The first is the *balancing invariant* which guarantees that I contains $\Theta(|I| \cdot n)$ peers. The second is the *majority invariant* which guarantees the majority of peers in I are correct. The authors show that for $|I| = \Theta(\log(n)/n)$ both invariants can be maintained with high probability over n^c joins and departures, where c is a constant that can be tuned. It follows that the peers in I can form a quorum that satisfies the goodness invariant.

To see how the cuckoo rule works, the ring $[0, 1)$ is assumed to be broken into disjoint segments of constant length k/n for some constant k. Each segment is called a *k-region* and $R_k(x)$ denotes the unique k-region containing x. When a peer p joins the network, it is assigned a random identifier $x \in [0, 1)$ and placed in this location. The use of random identifiers, rather than the original hashing scheme of earlier DHT constructions, is a minor modification that does not alter previous guarantees or analyses of such constructions (see [7] for more discussion). All nodes in $R_k(x)$ are evicted and placed into new locations chosen uniformly and independently at random from $[0, 1)$. Figure 5 illustrates these operations.

The node placements required by the cuckoo rule can be executed by having quorums use robust communication in order to inform each other about the arrival of the evicted

nodes at their new locations. Once a quorum Q_i knows about the presence of a recently evicted node q, all correct members of Q_i update their membership lists, share IP addresses, and aid q in setting up any required links (i.e. such as finger links in Chord). A detailed discussion of how this could be done is presented in [21] and random numbers can be generated using the protocol of [6]. We finish our discussion of a join protocol by discussing the steps necessary for maintaining DKG and the consequent cost of membership changes.

RCP-I: Consider a quorum Q_i to which a new peer is added. The membership update protocol of DKG [35] 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 departs 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 (share renewal) protocol need not be executed. We use this to associate the system churn rate to DKG session time. Note that the adversary may crash some of its t nodes, and in principle, the system can handle $t + f$ node departures. However, we cannot associate these additional t crashes with the system churn due to the inherent arbitrary nature of Byzantine peers.

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 departs Q_i, DKG may be required 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 a join or departure is higher in comparison to RCP-I.

V. Experimental Results

We examine the performance of DKG and our two protocols on the PlanetLab platform [47]. 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 [34] to measure the performance of DKG. We incorporate threshold BLS signatures and realize our two protocols using this setup on PlanetLab.

Distributed Key Generation: We test the DKG implementation for quorum sizes $s = 10, 20, 30, 40, 50$ and present median completion times and median CPU usage in Table II along with 95% one-sided confidence intervals. We describe our experimental setup in the context of [24], [38]. Our experiments are terminating and conducted via the method of independent replications. A single replication consists of s individual observations each corresponding to the time required for a participating peer in the quorum to complete the DKG protocol; there are 10 replications for each s value. For each s value, the PlanetLab machines used are chosen from around the world with roughly 44% located in North America, 30% located in East Asia and the remaining 26% located in Europe. Using independent replications, an unbiased sample point estimator for variance is calculated and used to obtain our one-sided confidence intervals using the t-distribution.

The median completion periods vary from roughly 4 seconds for $s = 10$ to roughly 117 seconds for $s = 50$. Notably, the bulk of this latency is due to network delay whereas 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 are set up so that correctness is guaranteed so long as at most 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. Pseudorandom values are generated using the well-known Number Theory Library (NTL) and Pairing-Based Cryptography (PBC) library.

RCP-I and RCP-II: For our RCP-I and RCP-II experiments, we set $s = 30$, $t = 3$, and $f = 10$. We conduct terminating experiments again via the method of independent replications where each of the 5 replications consists of 30 observations. In RCP-I, a node requires an average of 0.14 ± 0.0075 seconds (95% one-sided confidence interval) 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 ± 0.015 seconds (95% one-sided confidence interval) in the case of a share corruption attack. Extrapolating to a path length ℓ, an operation should take between $0.14 \cdot \ell \pm 0.0075 \cdot \ell$ or $0.23 \cdot \ell \pm 0.015 \cdot \ell$ seconds on average. For a DHT with 10^5 nodes, the average total time for RCP-I is then 2.8 ± 0.15 to 4.6 ± 0.3 seconds with $\ell = 20$.

In RCP-II, a node takes 0.042 ± 0.014 seconds (95% one-sided confidence interval) on average to obtain the required signed public keys and the signed routing information from a correct peer. A single signature verification takes negligible time; however, for completeness we report the average value of 0.0045 ± 0.0028 seconds (95% one-sided confidence interval). The median latency value over *all* pairs of PlanetLab nodes is roughly 0.08 seconds [16]; that is, $\Delta = 0.08$ seconds for $c = 0.5$. With a chain of signed public keys of length ℓ, the total communication time is $0.14 \pm 0.0075 + (0.042 \pm 0.014) \cdot (\ell - 1) + \frac{\Delta \cdot (\ell - 2)}{c \cdot (1 - \epsilon)} + (0.0045 \pm 0.0028) \cdot \frac{\ell(\ell-1)}{2}$ which for 10% Byzantine peers, is 4.94 ± 1.60 seconds in expectation for $\ell = 20$. 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 cal-

TABLE II
MEDIAN VALUES OF DKG COMPLETION TIME AND CPU TIME PER NODE FOR VARIOUS SYSTEM SIZES.

s	t	f	**Time** (seconds)	**CPU Seconds/Node**
10	1	3	4.32 $\pm$ 1.54	1.20 $\pm$ 0.06
20	2	6	7.50 $\pm$ 0.67	2.53 $\pm$ 0.47
30	3	10	15.17 $\pm$ 0.68	7.98 $\pm$ 3.39
40	4	13	36.80 $\pm$ 6.12	20.21 $\pm$ 9.85
50	5	17	116.63 $\pm$ 6.47	55.31 $\pm$ 30.16

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

s	10			20			30		
n_Q	1	2	3	1	2	3	1	2	3
r_Q	526	351	175	263	105	53	175	87	58

s	40			50		
n_Q	1	2	3	1	2	3
r_Q	132	66	44	105	53	35

culations to obtain the expected order of magnitude for our performance figures. For RCP-I, from the above discussion, each signature verification takes 0.0045 ± 0.0028 seconds; thus, the total CPU time required per execution is $(0.0045 \pm 0.0028) \cdot \ell \cdot (1 + s + s^2)$; this includes the costs due to share corruption attacks. For $\ell = 20$ and $s = 30$, this value is 74.48 ± 52.14 CPU seconds, spread out over 600 nodes. Therefore, the number of executions of RCP-I that can be started per second on average when $n = 10^5$ is roughly 10^3; note this rate value is for *the entire system*. Now, if no share corruption attacks occur, the total CPU time required per execution becomes $(0.0045 \pm 0.0028) \cdot \ell \cdot (1 + s)$ which, for the same parameter values, is 2.5 ± 1.76 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 $(0.0045 \pm 0.0028) \cdot \left(\ell + \frac{(\ell-1)\cdot\ell}{2\cdot(1-\epsilon)}\right)$ which, for the same parameters and $\epsilon = 1/10$ is 1.040 ± 0.65 CPU seconds 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 since asymptotic analysis is primarily 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: We now return to the issue of system churn which was discussed earlier in Section IV. A common metric for measuring the degree of churn is *session time*: the time between when a node joins the network and when it departs [49]. 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 [25]. In the Gnutella and Napster networks, the median session time was measured to be approximately 60 minutes [54]. In the KAD DHT, 155 minutes was the measured median [59]. Here, we temporarily assume a median session time of 60 minutes and a standard Poisson model of peer arrivals/departures as in [41], [49]. 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 [49]: $r = (n \cdot \ln 2)/t_{med}$. For $n = 10^5$ and $t_{med} = 3600$ seconds, $r \approx 19$. Assuming that join and departure events occur independently of each other, Table III 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 assumption that overlap occurs only with neighboring quorums in the ID space.

For the larger quorums, several of the r_Q values are less than the corresponding median DKG completion times in Table II. Therefore, a quorum may not be able to run 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 [35, 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 depart 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 II for $s = 20$ and $n_Q = 1$, DKG completes within 7.5 seconds. The number of departures a quorum can suffer while not exceeding the crash limit is $f = 6$. If Byzantine peers depart, more crashes are tolerable; however, identifying such events is impossible, so we assume the worst case of $f = 6$. Assuming DKG executes every $r_{DKG} = 600$ seconds (10 minutes), we seek the median session time such that at most 6 peers depart the system within 607.5 seconds. With $n/s = 5000$ quorums in the system, each experiencing 6 departures within 607.5 seconds, the system churn rate is roughly $r = 49$. This gives $t_{med} = 1415$ or, equivalently, 24 minutes. Therefore, with this t_{med} or larger, we expect the system to remain secure. Moreover, a quorum only spends $7.5/607.5 = 1.23\%$ of the time executing DKG.

We can decrease the required median session times by decreasing r_{DKG}; however, the percentage of time spent on DKG increases. Such tuning would depend on the desired system performance, the application, s, and n_Q. Table IV gives session time calculations for other values of s, r_{DKG} and n_Q.

Required session times increase with s. Notably, for $s = 50$ and $n_Q = 1$, t_{med} does not exceed the (roughly) 60 minutes in the Gnutella and Napster networks [54]. As n_Q increases, the

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

s	10			20			30		
r_{DKG}	2			5			9		
n_Q	1	2	3	1	2	3	1	2	3
t_{med}	4.79	9.57	14.36	11.84	23.68	35.52	19.24	38.48	57.72

	40			50		
	14			20		
	1	2	3	1	2	3
t_{med}	31.17	62.33	93.50	44.74	88.47	134.20

required session times grow linearly. However, our maximum of 134 minutes is still less than $t_{med} = 155$ minutes measured in the KAD DHT [59]. These results are encouraging as the recent results in [56] suggest that quorums as small as 64 peers can facilitate the cuckoo rule. We tentatively conclude that our protocols can be deployed in applications where session times range from a few minutes to just over two hours and that such applications currently exist.

VI. CONCLUSIONS AND FUTURE WORK

We have provided two new robust communication protocols that leverage cryptographic techniques to improve asymptotically on the message complexity of previous results. Our experimental work suggests that our protocols are practical for a number of application scenarios. In terms of future work, the performance of a complete system is an important open question - the quorum topology chosen is crucial and optimizing this in practice requires further study. 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: We gratefully acknowledge Yizhou Huang for his performance improvements to the DKG implementation. This work is supported by NSERC, MITACS, and David R. Cheriton Graduate Scholarships.

REFERENCES

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-Scalable Byzantine Fault-Tolerant Services. In *Proc. Symp. on Operating Systems Principles (SOSP)*, pages 59–74, 2005.

[2] E. Adar and B. A. Huberman. Free Riding on Gnutella. http://www.hpl.hp.com/research/idl/papers/gnutella/gnutella.pdf, 2000.

[3] 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 *Proc. Operating Systems Design and Implementation (OSDI)*, pages 1–14, 2002.

[4] J. Aspnes, N. Rustagi, and J. Saia. Worm versus alert: Who Wins in a Battle for Control of a Large-Scale Network? In *Proc. Intl. Conference on Principles of Distributed Systems*, pages 443–456, 2007.

[5] B. Awerbuch and C. Scheideler. Group Spreading: A Protocol for Provably Secure Distributed Name Service. In *Proc. Intl. Colloquium on Automata, Languages and Programming*, pages 183–195, 2004.

[6] B. Awerbuch and C. Scheideler. Robust Random Number Generation for Peer-to-Peer Systems. In *Proc. International Conference on Principles of Distributed Systems (OPODIS)*, pages 275–289, 2006.

[7] B. Awerbuch and C. Scheideler. Towards a Scalable and Robust DHT. In *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 318–327, 2006.

[8] B. Awerbuch and C. Scheideler. Towards Scalable and Robust Overlay Networks. In *Proc. Intl. Workshop on Peer-to-Peer Systems*, 2007.

[9] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Proc. International Workshop on Theory and Practic in Public Key Crptography (PKC)*, pages 31–46, 2003.

[10] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Proc. International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 416–432, 2003.

[11] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer. Brahms: Byzantine Resilient Random Membership Sampling. In *Proc. Symposium on Principles Distributed Computing*, pages 145–154, 2008.

[12] C. Cachin and J. Poritz. Secure Intrusion-tolerant Replication on the Internet. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, pages 167–176, 2002.

[13] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proc. Operating Systems Design and Implementation*, pages 299–314, 2002.

[14] M. Castro and B. Liskov. Byzantine Fault Tolerance Can Be Fast. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, pages 513–518, 2001.

[15] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance. In *Proc. Operating Systems Design and Implem.*, pages 177–190, 1999.

[16] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for Low Latency and High Throughput. In *Proc. Symp. on Networked Sys. Design and Implementation*, pages 85–98, 2004.

[17] J. Douceur. The Sybil Attack. In *Proc. Intl. Workshop on Peer-to-Peer Systems*, 2002.

[18] D. Dumitriu, E. W. Knightly, A. Kuzmanovic, I. Stoica, and Willy Zwaenepoel. Denial-of-Service Resilience in Peer-to-Peer File Sharing Systems. In *Proc. International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, pages 38–49, 2005.

[19] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a Million User DHT. In *Proc. Internet Measurement Conference*, pages 129 – 134, 2007.

[20] P. Feldman. A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In *Proc. Symposium on the Foundations on Computer Science (FOCS)*, pages 427–437, 1987.

[21] A. Fiat, J. Saia, and M. Young. Making Chord Robust to Byzantine Attacks. In *Proc. European Symp. on Algorithms*, pages 803–814, 2005.

[22] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus With One Faulty Process. *Journal of the ACM*, 32(2):374–382, 1985.

[23] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *Proc. USENIX Security Symposium*, pages 299–315, 2009.

[24] D. Goldsman and G. Tokol. Output Analysis: Output Analysis Procedures for Computer Simulations. In *Proceedings of the 32^{nd} Conference on Winter Simulation*, pages 39–45, 2000.

[25] 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 *Proc. Symposium on Operating Systems Principles (SOSP)*, pages 314–329, 2003.

[26] A. Gupta, B. Liskov, and R. Rodrigues. One Hop Lookups for Peer-to-Peer Overlays. In *Proc. Hot Topics in Operating Sys.*, page 2, 2003.

[27] K. Hildrum and J. Kubiatowicz. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-peer Networks. In *Proc. Symposium on Distributed Computing (DISC)*, pages 321–336, 2004.

[28] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *Proc. Intl. Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, 2001.

[29] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Computer Systems*, 20(4):398–461, 2002.

[30] D. Hughes, G. Coulson, and J. Walkerdine. Free Riding on Gnutella Revisited: The Bell Tolls? *IEEE Distributed Systems Online*, 6(6), 2005.

[31] H. Jiang and C. Dovrolis. Passive Estimation of TCP Round-Trip Times. *ACM SIGCOMM Computer Communication Review*, 32:75–88, 2002.

[32] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. In *Proc. EuroSys Conference*, pages 3–13, 2006.

[33] A. Kapadia and N. Triandopoulos. Halo: High-Assurance Locate for Distributed Hash Tables. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2008.

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

[35] A. Kate and I. Goldberg. Distributed Key Generation for the Internet. In

Proc. Intl. Conf. on Distributed Computing Systems, pages 119–128,2009.
[36] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, W. Weimer H. Weatherspoon, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 190–201, 2000.
[37] F. Kuhn, S. Schmid, and R.Wattenhofer. Towards Worst-Case Churn Resistant Peer-to-Peer Systems. *Distributed Computing*, 22(4):249–267, 2010.
[38] S. Kurkowski, T. Camp, and M. Colagrosso. Manet Simulation Studies: The Incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9:50–61, 2005.
[39] J. Liang and R. Kumar. Pollution in P2P file sharing systems. In *Proc. IEEE Intl. Conf. on Computer Communications (INFOCOM)*, 2005.
[40] J. Liang, N. Naoumov, and K. W. Ross. The Index Poisoning Attack in P2P File Sharing Systems. In *Proc. IEEE Intl. Conf. on Computer Communications (INFOCOM)*, pages 1–12, 2006.
[41] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *Proc. Symposium on Principles Distributed Computing (PODC)*, pages 233–242, 2002.
[42] P. Mittal and N. Borisov. ShadowWalker: Peer-to-peer Anonymous Communication using Redundant Structured Topologies. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2009.
[43] A. Nambiar and M. Wright. Salsa: A Structured Approach to Large-Scale Anonymity. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 17–26, 2006.
[44] M. Naor and U. Wieder. A Simple Fault Tolerant Distributed Hash Table. In *Proc. Intl. Workshop on Peer-to-Peer Sys.*, pages 88–97, 2003.
[45] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks (Ext. Abstract). In *Proc. Symposium on Principles Distributed Computing (PODC)*, pages 51–59, 1991.
[46] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proc. International Cryptology Conference (CRYPTO)*, pages 129–140, 1991.
[47] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *ACM SIGCOMM Computer Communication Review (CCR)*, 33(1):59–64, 2003.
[48] M. K. Reiter. The Rampart Toolkit for Building High-Integrity Services. In *Proc. International Workshop on Theory and Practice in Distributed Systems*, pages 99–110, 1995.
[49] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proc.* USENIX *Technical Conference*, pages 127–140, 2004.
[50] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee. Large-Scale Byzantine Fault Tolerance: Safe but Not Always Live. In *Proc. Hot Topics in Dependable Systems (HotDep)*, 2007.
[51] R. Rodrigues and B. Liskov. Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture. Technical Report TR/932, MIT LCS, December 2003.
[52] R. Rodrigues, B. Liskov, and L. Shrira. The Design of a Robust Peer-to-Peer System. In *Proc. ACM SIGOPS European Workshop*, pages 117–124, 2002.
[53] J. Saia and M. Young. Reducing Communication Costs in Robust Peer-to-Peer Networks. *Inform. Process. Lett.*, 106(4):152–158, 2008.
[54] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. Multimedia Computing and Networking (MMCN)*, pages 314–329, 2002.
[55] F. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comp. Surv.*, 22(4):299–319, 1990.
[56] Siddhartha Sen and Michael J. Freedman. Commensal Cuckoo: Secure Group Partitioning for Large-Scale Services. *ACM SIGOPS Operating Systems Review*, 46(1):33–39, February 2012.
[57] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *Proc. Intl.l Workshop on Peer-to-Peer Systems*, pages 261–269, 2002.
[58] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *Proc. Internet Measurement Conference*, pages 117 – 122, 2007.
[59] M. Steiner, T. En-Najjary, and E.W. Biersack. Long Term Study of Peer Behavior in the KAD DHT. *IEEE/ACM Transactions on Networking*, 17(5):1371–1384, 2009.
[60] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM Conference*, pages 149–160, 2001.
[61] G. Urdaneta, G. Pierre, and M. van Steen. A Survey of DHT Security Techniques. *ACM Computing Surveys*, 43(2):1–53, 2011.
[62] V. Pappas and D. Massey and A. Terzis and L. Zhang. A Comparative Study of the DNS Design with DHT-Based Alternatives. In *Proc. IEEE Intl. Conference on Computer Communications*, pages 1–13, 2006.
[63] D. S. Wallach. A Survey of Peer-to-Peer Security Issues. In *Proc. Intl. Conf. on Software Security: Theories and Sys.*, pages 253–258, 2002.
[64] P. Weng, N. Hopper, I. Osipkov, and Y. Kim. Myrmic: Secure and Robust DHT Routing. Technical Report 2006/20, Univ. of Minnesota, 2006.
[65] M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical Robust Communication in DHTs Tolerating a Byzantine Adversary. Technical Report CACR 2009-31, 2009. http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-31.pdf.
[66] M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical Robust Communication in DHTs Tolerating a Byzantine Adversary. In *Proc. Intl. Conf. on Distributed Computing Systems*, pages 263–272, 2010.

Maxwell Young is a Research Fellow at the National University of Singapore. He received his M.S. in computer science from the University of New Mexico, USA in 2006 and his Ph.D. from the University of Waterloo in Ontario, Canada in 2011. His research interests include security and distributed computing with a focus on adversarial fault tolerance in large decentralized networks.

Aniket Kate is a postdoctoral researcher at the Max Planck Institute for Software Systems (MPI-SWS), Germany. Prior to that, he completed his PhD at the University of Waterloo, Canada in 2010 and his Masters at the Indian Institute of Technology (IIT) - Bombay, India in 2006. His research aims at bridging the gap between cryptography, and system security and privacy research.

Ian Goldberg is an Associate Professor of Computer Science at the University of Waterloo, where he is a founding member of the Cryptography, Security, and Privacy (CrySP) research group. He holds a Ph.D. from the University of California, Berkeley, where he discovered serious weaknesses in a number of widely deployed security systems, including those used by cellular phones and wireless networks. He also studied systems for protecting the personal privacy of Internet users, which led to his role as Chief Scientist at Zero-Knowledge Systems (now Radialpoint). His research currently focuses on developing usable and useful technologies to help Internet users maintain their security and privacy. He is a Senior Member of the ACM and a winner of Early Researcher Award, the Outstanding Young Computer Science Researcher Award, and the Electronic Frontier Foundation's Pioneer Award.

Martin Karsten (M'98 / ACM'98) obtained his first university degree in joint business administration and computer science from the University of Mannheim (Germany). He received his doctoral degree in computer science from TU Darmstadt (Germany) in 2000. He later joined the University of Waterloo (Canada) in 2002 where he is currently an Associate Professor in the David R. Cheriton School of Computer Science. His main research interest is the systems-level software infrastructure of networks and distributed systems.