Commensal Cuckoo: Secure Group Partitioning for Large-Scale Services

Siddhartha Sen
Princeton University
35 Olden Street, Princeton, NJ 08540
sssix@cs.princeton.edu

Michael J. Freedman
Princeton University
35 Olden Street, Princeton, NJ 08540
mfreed@cs.princeton.edu

ABSTRACT

We present *commensal cuckoo,*[*] a secure group partitioning scheme for large-scale systems that maintains the correctness of many small groups, despite a Byzantine adversary that controls a constant (global) fraction of all nodes. In particular, the adversary is allowed to repeatedly rejoin faulty nodes to the system in an arbitrary adaptive manner, *e.g.*, to collocate them in the same group. Commensal cuckoo addresses serious practical limitations of the state-of-the-art scheme, the cuckoo rule of Awerbuch and Scheideler, tolerating over 35x more faulty nodes with groups as small as 64 nodes (as compared to the hundreds required by the cuckoo rule). Secure group partitioning is a key component of highly-scalable, reliable systems such as Byzantine fault-tolerant distributed hash tables (DHTs).

1. INTRODUCTION

Many modern computing services are provided by building networked systems out of large sets of machines. System designers employ this cooperation in order to achieve the scalability and reliability requirements demanded by today's users. However, providing fault-tolerance is a challenge when working with such large sets of participating software and hardware systems, including those hosted by third-party cloud providers. In the case of open peer-to-peer systems like the Vuze DHT [42] (a million-node BitTorrent tracker), any machine connected to the Internet can participate; clearly, such machines cannot be trusted.

To cope with the possibility of arbitrary (or so-called Byzantine) failures—whether due to malice, bugs, or simply miscon-figurations—the academic community has proposed schemes such as Byzantine Fault Tolerant (BFT) replicated state machines [6, 38], in which system state and functionality are replicated and executed across multiple machines. These protocols ensure the correctness ("safety") and availability ("liveness") of the system when fewer than $1/3$ of the nodes are faulty. Unfortunately, despite considerable effort, state-of-the-art BFT protocols [6, 22] still scale poorly with system size, because they require each node to participate in every request, more than $2/3$ of the nodes to maintain the replicated state [45], and quadratic communication per request. Thus, the application of these protocols has been limited to "point solutions" on small sets of nodes within larger systems, *e.g.*, as a lock service [9].

To build BFT systems at a large scale, we must partition the system into smaller groups that operate on disjoint (or at least minimally-overlapping) partitions of the system state and functions. Client requests are routed to the appropriate group. The challenge is to ensure that every group maintains less than a fixed *local* fraction of faulty nodes (*e.g.*, $1/3$), given some fixed (smaller) *global* fraction of faulty nodes that are controlled by a Byzantine adversary. The system fails if even a single group becomes faulty (by exceeding the local fault threshold), since such a group may behave arbitrarily: for example, it may delete its portion of the system state and try to corrupt other groups. We assume that the adversary can coordinate the faulty nodes in an arbitrary, adaptive manner. In particular, it may initiate a *join-leave attack* [11, 12, 41], wherein it repeatedly rejoins certain faulty nodes to the system using fresh identities [12] in order to compromise one or more groups. Such attacks pose a significant threat [11, 12, 41] and have been launched successfully on real DHT systems [44]. We call this the *secure group partitioning* problem.

Prior work. Several systems [10, 21, 23, 31, 34, 43] and some proposals [33, 35] have used multiple BFT groups for scalability, but these solutions rely on a central configuration service to manage system-wide membership or they maintain this information at every node. Other systems offer better decentralization [1, 7, 17, 18, 26–28, 40], *e.g.*, using a group for each directory of a file system [1], but they assume that faulty nodes are distributed randomly or even perfectly across groups. Thus, to our knowledge, all systems are vulnerable to join-leave attacks. For example, Rodrigues and Liskov [32, 34] build a DHT by mapping nodes and data to a virtual $[0, 1)$ space using consistent hashing [19], and form BFT groups out of contiguous sets of four nodes, each group tolerating one fault. Even without a join-leave attack, such a perfect distribution of faults cannot be achieved even if faults occur uniformly randomly: some group will have $\omega(1)$ faulty nodes with high probability.

Some theoretical constructions of fault-tolerant DHTs assume that nodes fail randomly (independent of their loca-

[*] **Com·men·sal·ism**. A symbiotic relationship in which one organism derives benefit while causing little or no harm to the other.

tion) [15,29], however these assumptions break down in the presence of adaptive join-leave attacks. Recent years have seen constructions that can provably withstand join-leave attacks [2–4,13,37]. Of these, the most promising scheme for DHTs that doesn't keep the system in a hyperactive state—*e.g.*, by forcing nodes to rejoin the system periodically [2]—is the scheme of Awerbuch and Scheideler [3]. They propose a simple, event-based scheme called the *cuckoo rule*: when a node wishes to join the system, place it at a random location $x \in [0, 1)$ and move, or *cuckoo*, all nodes in a constant-sized interval surrounding x to new random locations in $(0, 1]$. Using this rule, they prove that groups of size $O(\log n)$ remain correct for any polynomial number of rounds in n, where n is the number of correct nodes in the system. However, as we will show, the constants in their scheme are prohibitively large, so either groups must be very large (hundreds of nodes) or the global fraction of faults must be trivially low for the system to survive a reasonable number of rounds.

Our contributions. In this paper, we propose a scheme called *commensal cuckoo* that significantly improves the performance of the (parasitic) cuckoo rule. We demonstrate that the cuckoo rule fails largely due to "bad luck": bad events like repeated malicious joins to the same group that occur with non-negligible probability. Thus, our approach is to partially derandomize the cuckoo rule, which we do in two ways. First, we ensure that the number of nodes cuckood during a join deterministically matches the expected amount. Second, we allow groups to *vet* the join process, that is, reject join attempts if they have not received sufficient new nodes since the last join. Join vetting has surprisingly deeper benefits: it naturally addresses a known [3,5] vulnerability in the cuckoo rule and suggests the possibility of allowing faulty nodes to *choose* their join location. Using commensal cuckoo, we are able to maintain small groups of 64 nodes while tolerating a global *fraction* of faulty nodes that is over 35x larger than that of the cuckoo rule.

Paper organization. We define the secure group partitioning problem in §2 and examine the cuckoo rule in §3, using simulations to understand why it fails. We introduce our improved scheme, the commensal cuckoo rule, in §4. Commensal cuckoo is just one (critical) piece of a larger set of mechanisms needed to solve the secure group partitioning problem. We review these complementary problems in §5, as well as our proposed solutions. We conclude in §6.

2. PROBLEM ABSTRACTION

Let $N = n + \epsilon n$ be the size of the system, such that n nodes are correct and ϵn nodes are faulty. (The global fraction of faulty nodes is thus $\epsilon/(1 + \epsilon)$). Initially, the n correct nodes are mapped to random locations in $[0, 1]$. Next, the adversary joins the ϵn faulty nodes one by one. Finally, the adversary begins executing rejoin operations (a leave followed by a join) on whichever faulty nodes it wants, even basing its decision on the entire system state. Our goal is to devise an efficient join rule such that, with high probability (*i.e.*, at least $1 - 1/n$) and for any polynomial number of rounds (*i.e.*, rejoin operations), the system can be partitioned into intervals $I \in [0, 1)$ that satisfy the following conditions [3]:

- *Balance condition:* I contains $\Theta(|I| \cdot n)$ nodes.
- *Correctness condition:* I has less than $1/3$ faulty nodes.

The nodes in such an interval form a *group*. In line with our discussion above, we assume that groups are disjoint (to maximize parallelism) and run a BFT protocol that allows them, for example, to generate (pseudo)random numbers and agree on membership changes. In principle, the constant $1/3$ can be replaced with any constant less than $1/2$, a flexibility we exploit later.

Our discussion below abstracts away several lower-level mechanisms required for secure group partitioning, such as a mechanism for constructing and routing verifiable messages between groups. Since the algorithms we present can be understood without these mechanisms, we postpone their discussion to §5.

3. CUCKOO RULE

Awerbuch and Scheideler [3] propose the following simple join rule. For a fixed $k > 0$, define a *k-region* to be a region in $[0, 1)$ of size k/n that starts at an integer multiple of k/n. For technical (divisibility) reasons, k-regions are rounded from above to the closest value $1/2^r$ for some integer r.

Cuckoo rule. *When a new node wants to join the system, place it at a random $x \in [0, 1)$ and move (cuckoo) all nodes in the unique k-region containing x to random locations in* $[0, 1)$.

We call the new node's join a *primary join* and the subsequent joins of the cuckood nodes *secondary joins*. Awerbuch and Scheideler prove that in steady state, provided $\epsilon < 1/2 - 1/k$, all regions of size $O(\log n)/n$ have $O(\log n)$ nodes (*i.e.*, they are balanced) of which less than $1/3$ are faulty (*i.e.*, they are correct), with high probability, for any polynomial number of rounds. Thus these intervals are the groups. Their analysis further implies that the best adversarial strategy is to target a single group and repeatedly rejoin faulty nodes that lie outside the group.

In practice, the random location of the primary join is generated by the group initially contacted by the new node, and the random locations of the secondary joins are generated by the group that owns the primary join location (where the new node ultimately joins).

Cuckoo rule analysis. We first observe that the optimal strategy of the adversary is actually different from that claimed in [3]—namely, target a single group, and have nodes not in that group rejoin—once constant factors are taken into account. At the beginning of each round, the adversary should sort all groups by increasing fraction of faulty nodes, and should have a faulty node belonging to the group with the lowest fraction attempt to rejoin the system. This Markovian strategy always maintains the largest and most promising number of targeted groups.

Using this modified adversarial strategy, we simulated the cuckoo rule to investigate the different constant factors involved. These factors arise from the use of Chernoff bounds in the analysis, as well as union bounds over all groups and all rounds for which the balance and correctness conditions must hold. Figure 1 shows the minimum (average) group size required to last 100,000 rounds, for different values of N and ϵ, while optimizing over k. We increased the group size in powers of 2 to avoid divisibility problems. As the figure shows, this size is in the hundreds of nodes for any reasonable value of ϵ. Even when $\epsilon = 0.01$, the minimum group size is 256 for $N \geq 2048$. This situation is degenerate

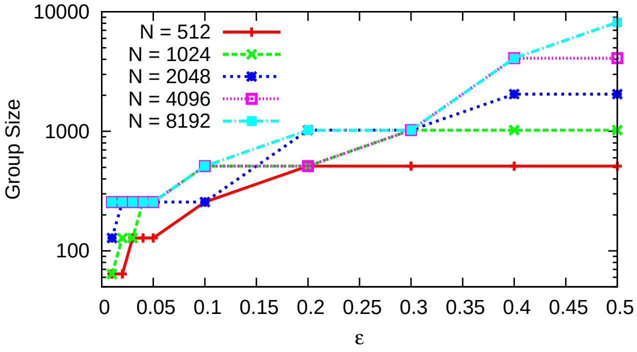

Figure 1: (Cuckoo rule) Minimum group size (in powers of 2) needed to tolerate different ϵ for 100,000 rounds, where $\epsilon/(1+\epsilon)$ is the global fraction of faulty nodes. Groups must be large (*i.e.*, 100s to 1000s of nodes) to guarantee correctness.

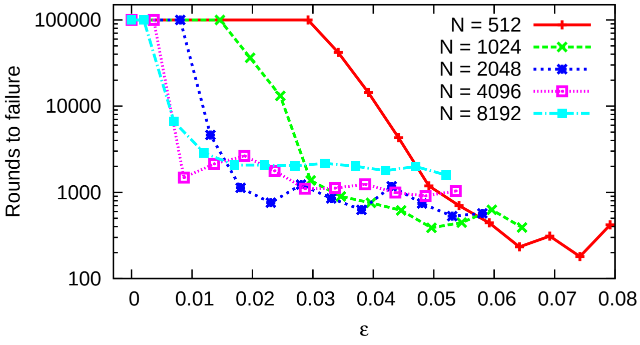

Figure 2: (Cuckoo rule) Number of rounds the system maintained correctness with an average group size of 64 nodes, for varied ϵ. Simulation was halted after 100,000 rounds. Failure rates drop dramatically past a certain threshold for different N.

because the *total* number of faulty nodes in the system is (much) less than 1/3 of the average group size, which suggests that the cuckoo rule causes groups to become highly imbalanced. In fact, the optimal value of k in these cases was always less than 2, indicating that the system preferred to cuckoo very few nodes (or none at all).

To determine what it would take to support groups of size 64—perhaps an upper-bound for any type of performant system—we ran the simulation with this constraint and optimized over ϵ and k. Figure 2 shows the results, where each simulation ran for a maximum of 100,000 rounds, at which point it was deemed non-faulty. For $N \geq 1024$, $\epsilon < 0.015$ to achieve a non-faulty result, dropping to 0.002 when $N = 8192$. (Note that we expect this threshold to lower with increasing N, as the figure shows, because the fixed average group size becomes smaller relative to the global number of faulty nodes ϵn.) Again, these thresholds fall in the degenerate realm described above. The thresholds are also sharp, as the number of rounds to failure drops dramatically when ϵ is increased (note the log scale). Our goal is to increase this threshold.

In order to achieve this increase, we gain a deeper understanding of what goes wrong by examining the evolution of a group that eventually fails. Figure 3 plots the fraction of faulty nodes in such a group over time, for a system with $N = 8192, \epsilon = 0.05, k = 4$ and an average group size of 64 as before. We immediately see two problems. First, each primary join (indicated by a black cross) causes the faulty fraction of the group to jump. Although the threshold recovers slightly due to churn in the system due to joins in other groups, it does not fully recover its original threshold. Second, some primary joins occur very close together—the probability of such bad events is non-negligible, and can be easily calculated; these joins have the worst impact. In contrast, during the period from round 507 to 858, there are relatively few primary joins, giving the group enough time to improve its faulty fraction.

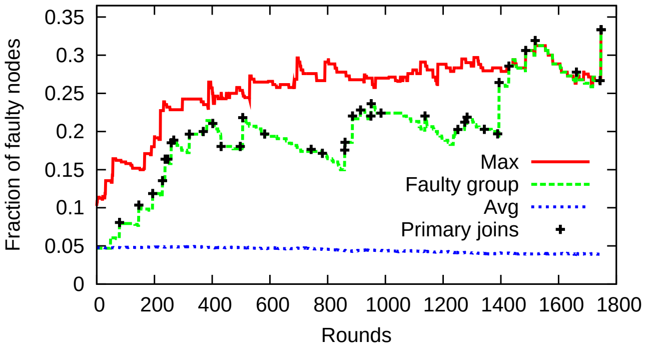

Figure 3: (Cuckoo rule) Evolution of a faulty group that fails at round 1749. Primary joins are often to blame for the group's ultimate failure, the maximum fraction of faulty nodes per group is much higher than the average, and the initial maximum fraction group was not the final one that fails.

To investigate the first problem, Figure 4 (left) shows the number of nodes cuckood in each round. Although this number starts out concentrated around its expected value of $k = 4$, it spreads out over time, ranging from 0 to as high as 18. This indicates that k-regions are getting increasingly "clumpy", likely due to the fact that primary joins empty out an entire k-region. To investigate the second problem, Figure 5 plots a CDF of the number of secondary joins in between successive primary joins to the failed group. This number is also expected to be $k = 4$, since an interval of size i is joined every $1/i$ rounds in expectation, during which time k/i other nodes are expected to be cuckood, of which $(i/1)(k/i) = k$ are expected to land in the interval. However, the number shows enough variance to be problematic, with a considerable fraction at 0, and values as high as 16 for the faulty group (and 37 over all groups).

4. COMMENSAL CUCKOO RULE

The two problems discovered in Section 3 can be remedied in two natural ways. First, to make cuckoos more consistently sized and evade the "clumpiness" problem, we cuckoo k nodes chosen randomly from the group being joined, instead of all nodes in the k-region surrounding the join location. However, using a fixed number for k actually makes things worse, because a group that is light (*i.e.*, has less than the average group size g), will continue to become lighter, eventually allowing the adversary to compromise it. Thus, to ensure that lighter groups cuckoo less and heavier groups cuckoo more, we scale k by the group's size relative to g.

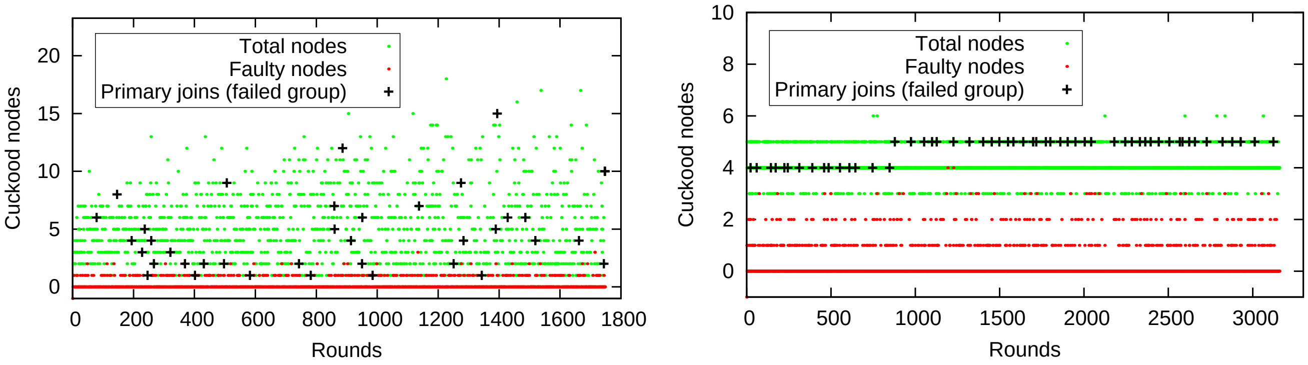

Figure 4: Number of cuckood nodes during joins for the cuckoo rule (left) vs. commensal cuckoo rule (right). Black crosses correspond to primary joins in the group that ultimately fails. The cuckoo rule yields higher variance in the group size being evicted, likely due to increasing non-uniformity of nodes across the keyspace, as k-regions are evicted *en masse*.

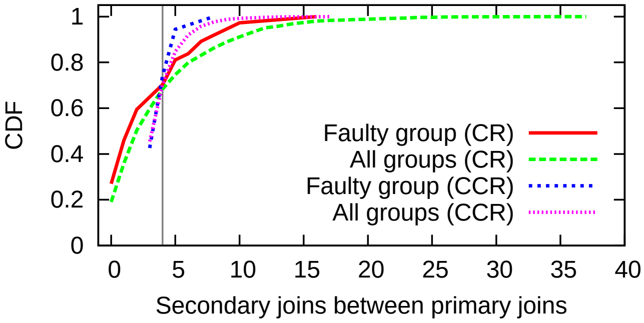

Figure 5: Number of secondary joins between successive joins for the cuckoo rule (CR) vs. commensal cuckoo rule (CCR). Commensal cuckoo leads to results more tightly concentrated around the expected value of 4, shown as a vertical line.

Most prior work (*e.g.*, [2,13]) calculates g based on an estimate of n and a target group size $c \log n$ for some *a priori* fixed $c > 0$. However, g can also be chosen at the onset of the system (as in most of our experiments), not as a function of n, to reflect the scalability of the higher-level application running in each group. Such an approach is practical if g is large enough for the maximum n ever expected to be seen.

Second, to address the problem of inconsistently spaced primary joins, we allow a group to *vet* join attempts by refusing them if it has not received a sufficient number of secondary joins since the last primary join. Together, these techniques derandomize crucial aspects of the cuckoo rule, yielding our new rule:

Commensal cuckoo rule. *When a new node wants to join the system, pick a random $x \in [0, 1)$. If the group containing x has not received at least $k - 1$ secondary joins since its last primary join, start over with a new random $x \in [0, 1)$. Otherwise, place the node at x and move (cuckoo) kg'/g random nodes in the group to random locations in $[0, 1)$, where g' is the group's current size and g is the average group size.*

Interestingly, these two modifications are synergistic. By ensuring the expected (weighted) number of nodes are cuckood during each join, commensal cuckooing ensures a sufficient number of secondary joins, which allows a group to wait for enough of them before accepting another join. At the same time, by vetting repeated joins attempts, the adversary is forced to join distinct groups, which roughly speaking ensures that all groups are joined, resulting in $\sum kg'/g = (k/g) \sum g' = Nk/g$ total secondary joins. Thus, if groups wait for slightly less than k secondary joins between primary joins, a joining node need only try O(1) times before finding a group that accepts it. If too few secondary joins occurred, however—as in the case of the cuckoo rule—the system would risk deadlocking because no groups would accept a join.

4.1 Comparative evaluation

Figures 4 (right) and 5 show the number of cuckood nodes and the number of secondary joins between successive primary joins, respectively, using the commensal cuckoo rule. Compared to the results of the cuckoo rule, these numbers are tightly concentrated around their expected values.

Table 1 shows the largest value of ϵ achieved by each scheme, optimizing over k, that can allow a system with an average group size $g = 64$ to remain correct for at least 100,000 rounds. We find that the performance of commensal cuckoo improves with increasing N, achieving an ϵ value up to 35x larger than that of the cuckoo rule while tolerating $< 1/3$ faulty nodes. Since commensal cuckoo also improves with increasing k (not shown), we enforced an upper bound of $k = 8$. The cuckoo rule does not improve by increasing k, however, due to the very problems it suffers when the average group size is small (as discussed in §3). In general, commensal cuckoo evicts more nodes per join than the cuckoo rule, but this eviction is needed to redistribute faulty nodes in the system.

Interestingly, commensal cuckoo's technique of join vetting has deeper benefits than those outlined above. As presented, the cuckoo rule in §3 has a known [3,5] vulnerability: when joining a faulty node, the adversary may repeatedly cause the random number generation for the primary join location to fail—causing no cuckoos to occur—until it receives a join location of its liking. Awerbuch and Scheideler address this problem [5] by effectively spawning artificial cuckoos whenever a join attempt fails. Join vetting, on the other hand, naturally evades this vulnerability, because regardless of the adversary's behavior, a group will not ac-

N	<1/3 faulty			<1/2 faulty		
	CR	CCR	Gain	CR	CCR	Gain
512	0.0292	0.0486	1.66x	0.0564	0.0856	1.52x
1024	0.0146	0.0809	5.54x	0.0302	0.1940	6.42x
2048	0.0080	0.0629	7.82x	0.0146	0.1917	13.1x
4096	0.0037	0.0771	20.8x	0.0081	0.2169	26.8x
8192	0.0020	0.0702	35.1x	0.0040	0.1997	49.6x

Table 1: Maximum ϵ of the cuckoo rule (CR) vs. commensal cuckoo rule (CCR), in order for groups of average size 64 to remain correct for at least 100,000 rounds.

cept a primary join unless sufficient secondary joins have occurred in the system. Moreover, this suggests the following modification to the commensal cuckoo rule: *allow faulty nodes to (attempt to) join at* any *primary join location*, not just a random one. We conjecture that commensal cuckoo with this modification remains secure.

4.2 Higher fault thresholds

Table 1 also lists the largest ϵ achieved when only a majority of the nodes in a group are required to be correct. These results are even better: commensal cuckoo tolerates an ϵ almost 50x better than the cuckoo rule, which corresponds to a global fault fraction of almost 18%. While several techniques exist for improving the resiliency of BFT protocols to 1/2, they either require broadcast channels (*e.g.*, [16,30]) or trusted primitives (*e.g.*, [8,25]), neither of which is practical for groups of nodes in an open peer-to-peer setting. However, one technique—separating the thresholds of consistency and availability [24]—allows a group to remain correct as long as fewer than 1/2 of its nodes are faulty, even though it may become unavailable before that. The drawback of this approach is that a correct group may lose some of its most recent updates if it becomes unresponsive.

5. TOWARDS A COMPLETE SOLUTION

Commensal cuckoo relies on several lower-level mechanisms that we have assumed in previous sections. We briefly discuss some of those mechanisms here.

Secure routing. In order to send messages between the groups involved in a join operation, as well as to process client requests that may arrive at any group, we need a mechanism for routing messages. The cuckoo rule [3] uses a routing scheme based on de Bruijn graphs that requires $O(\log n)$ hops and $O(\log^3 n)$ total messages. Recent work [36, 46] reduces the latter overhead to $O(\log n)$ messages in expectation. We propose combining these ideas with O(1)-hop DHT constructions [14] to reduce both the number of hops and messages, at the cost of increased per-group state.

Group authentication. A group must be able to verify a message it receives even if it has no knowledge of the sending group's membership. The cuckoo rule relies on all-to-all connectivity between adjacent groups along a routing path to authenticate messages. However, cryptographic techniques can significantly improve the performance and flexibility of authentication in the DHT [46]. In particular, threshold signatures [39] and distributed key generation [20] can be used to assign a public/private key pair to each group that remains constant despite changes in the group's membership. Specifically, the g group members use a (t, g)-threshold signature scheme to cooperatively sign a message with their private key shares, despite up to t faulty nodes. When the group's membership changes, a new set of key shares corresponding to the same public/private key pair is generated and distributed. The new shares reveal no additional information to faulty nodes than their old shares, even in combination.

Bootstrapping and heavy churn. Our analysis of the cuckoo and commensal cuckoo rules assumes the system is in steady state. However, protocols to bootstrap the system and handle heavy churn are also needed. Prior fault-tolerant DHTs describe such protocols [2, 13], but they use groups that overlap and are based on the current (estimated) value of n. Thus, all groups necessarily change if n changes by a large enough amount. Our scheme supports an alternative approach that chooses a target group size g at the onset of the system, based on the scalability of the intra-group protocol, as discussed in §4. This enables a bootstrapping protocol that creates an initial group spanning the entire $[0, 1)$ interval and uses split and merge operations to divide a region or merge two regions, respectively, if they get too heavy or too light. Such a protocol is simpler when n is small, and it localizes the effect of increasing n. For the bootstrapping protocols to work, we must assume that the number of faulty nodes is at most ϵn for all values of n.

Other attacks. Our fault model currently allows the adversary to control the behavior of only faulty nodes. A different type of attack is one of denial-of-service (DoS), in which the adversary forces a correct node to leave the system, *e.g.*, by overwhelming it with spurious traffic. Awerbuch and Scheideler propose an extension [4] to the cuckoo rule that withstands such an attack; commensal cuckoo is compatible with this extension, and we believe some of the ideas presented in this paper can be applied to the extended rule as well.

The adversary may also launch a DoS attack on the data layer of the DHT, for example by crafting a series of insert or lookup requests that target a particular region or node. Awerbuch and Scheideler handle such attacks by *proactively* replicating data items across groups [3]. A *reactive* approach may also be practical and more efficient. This is because, unlike with join-leave attacks, it is possible to *detect* when a data-layer attack occurs by measuring the current request load. The system could use this information to adaptively replicate data items on-the-fly.

6. CONCLUSIONS

Commensal cuckoo is a practical scheme for partitioning a large-scale system into many small groups that remain correct despite adversarial join-leave attacks. By carefully managing when it is acceptable for new nodes to join groups, as well as balancing which existing nodes are evicted by such joins, commensal cuckoo can support significantly smaller group sizes and a higher fraction of faulty nodes than the state-of-the-art, the cuckoo rule. Commensal cuckoo relies on several important mechanisms to solve the secure group partitioning problem. In our future work, we plan to design protocols for these mechanisms along the lines of §5.

7. REFERENCES

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *OSDI*, pages 1–14, 2002.

[2] B. Awerbuch and C. Scheideler. Group spreading: A protocol for provably secure distributed name service. In *ICALP*, pages 183–195, 2004.

[3] B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. In *SPAA*, pages 318–327, 2006.

[4] B. Awerbuch and C. Scheideler. Towards scalable and robust overlay networks. In *IPTPS*, 2007.

[5] B. Awerbuch and C. Scheideler. Robust random number generation for peer-to-peer systems. *Theor. Comput. Sci.*, 410:453–466, 2009.

[6] M. Castro. *Practical Byzantine Fault-Tolerance*. PhD thesis, M.I.T., 2000.

[7] M. Castro, P. Druschel, A. J. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, pages 299–314, 2002.

[8] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *SOSP*, pages 189–204, 2007.

[9] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *SOSP*, pages 277–290, 2009.

[10] J. Cowling, D. R. K. Ports, B. Liskov, R. A. Popa, and A. Gaikwad. Census: Location-aware membership management for large-scale distributed systems. In *USENIX ATC*, pages 159–172, 2009.

[11] S. A. Crosby and D. S. Wallach. Denial of service via algorithmic complexity attacks. In *USENIX Security*, pages 29–44, 2003.

[12] J. R. Douceur. The Sybil attack. In *IPTPS*, pages 251–260, 2002.

[13] A. Fiat, J. Saia, and M. Young. Making Chord robust to Byzantine attacks. In *ESA*, pages 803–814, 2005.

[14] I. Gupta, K. P. Birman, P. Linga, A. J. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *IPTPS*, pages 160–169, 2003.

[15] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *DISC*, pages 321–336, 2003.

[16] A. Jaffe, T. Moscibroda, and S. Sen. The price of equivocation: Characterizing Byzantine agreement via hypergraph coloring. Manuscript, 2011.

[17] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable support for intrusion-tolerant network overlays. In *EuroSys*, pages 3–13, 2006.

[18] A. Kapadia and N. Triandopoulos. Halo: High-assurance locate for distributed hash tables. In *NDSS*, pages 61–79, 2008.

[19] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC*, pages 654–663, 1997.

[20] A. Kate and I. Goldberg. Distributed key generation for the internet. In *ICDCS*, pages 119–128, 2009.

[21] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The SecureRing protocols for securing group communication. In *HICSS*, pages 317–326, 1998.

[22] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *SOSP*, pages 45–58, 2007.

[23] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *ASPLOS*, pages 190–201, 2000.

[24] L. Lamport. Lower bounds for asynchronous consensus. In *FuDiCo*, pages 22–23, 2003.

[25] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. TrInc: Small trusted hardware for large distributed systems. In *NSDI*, pages 1–14, 2009.

[26] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin. Flightpath: Obedience vs. choice in cooperative services. In *OSDI*, pages 355–368, 2008.

[27] P. Mittal and N. Borisov. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. In *CSS*, pages 161–172, 2009.

[28] A. Nambiar and M. Wright. Salsa: A structured approach to large-scale anonymity. In *CSS*, pages 17–26, 2006.

[29] M. Naor and U. Wieder. Novel architectures for P2P applications: The continuous-discrete approach. *ACM Trans. Algorithms*, 3(3), 2007.

[30] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC*, pages 73–85, 1989.

[31] M. K. Reiter. The Rampart toolkit for building high-integrity services. In *Workshop on Theory and Practice in Distributed Systems*, pages 99–110. 1995.

[32] R. Rodrigues. *Robust Services in Dynamic Systems.* PhD thesis, M.I.T., 2005.

[33] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee. Large-scale Byzantine fault tolerance: Safe but not always live. In *HotDep*, 2007.

[34] R. Rodrigues and B. Liskov. Rosebud: A scalable Byzantine-fault-tolerant storage architecture. Technical Report TR-2003-035, M.I.T., CSAIL, 2003.

[35] R. Rodrigues, B. Liskov, and L. Shrira. The design of a robust peer-to-peer system. In *SIGOPS European workshop*, pages 117–124, 2002.

[36] J. Saia and M. Young. Reducing communication costs in robust peer-to-peer networks. *Inf. Process. Lett.*, 106:152–158, 2008.

[37] C. Scheideler. How to spread adversarial nodes?: rotate! In *STOC*, pages 704–713, 2005.

[38] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4), 1990.

[39] V. Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220, 2000.

[40] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *INFOCOM*, pages 1–12, 2006.

[41] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *ACSAC*, pages 252–261, 2004.

[42] Vuze. `http://www.vuze.com/`.

[43] P. Wang, I. Osipkov, N. Hopper, and Y. Kim. Myrmic: Provably secure and efficient DHT routing. Technical Report 2006/20, Univ. Minnesota, DTC, 2006.

[44] S. Wolchok, O. S. Hofmann, N. Heninger, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel. Defeating Vanish with low-cost Sybil attacks against large DHTs. In *NDSS*, 2010.

[45] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In *SOSP*, pages 253–267, 2003.

[46] M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical robust communication in dhts tolerating a Byzantine adversary. In *ICDCS*, pages 263–272, 2010.