Towards Secure Distributed Hash Table

Zhe Wang(✉) and Naftaly H. Minsky

Rutgers University, Piscataway, NJ 08854, USA
{zhewang,minsky}@cs.rutgers.edu

Abstract. A distributed hash table (DHT) provides decentralized lookup service for distributed applications. All current implementations of DHT are achieved by the individual components being run by the participants of the application in question. Namely, the correctness of the DHT relies on that all the participants follow the same protocol. Unfortunately, this aspect of the current approach makes DHT seriously vulnerable to attacks. Such security and fault tolerance concerns about DHT prompted several attempts to improve the vulnerability of DHT. However, all the proposed solutions also rely on the code to be executed correctly. We present in this paper a novel way for implementing DHT, giving rise to an architecture we call GDHT, for Governed Distributed Hash Table. GDHT implements the required protocol with a powerful means for establishing policies governing the behaviors of the participants of DHT. By carrying out the protocol by an equally distributed middleware, the correctness of the execution of routing algorithm is guaranteed. Moreover, the execution of the security module and improvements on routing algorithm can also be ensured.

Keywords: Distributed hash table · Fault tolerance · Collaboration · Security · Governed · Chord · Sybil attack · Routing attack

1 Introduction

A *distributed hash table* (DHT) is a distributed group of components that collaborate in forming a decentralized lookup service for distributed, mostly P2P type, applications. It has been used for a variety of applications, such P2P file sharing systems [4], distributed file systems [5], domain name services [10], instant messaging [12], and recently, distributed online social networks [17,18]. The concept of DHT has many different implementations, such as Chord [14], Pastry [13], CAN [11] and Kademlia [8], which are based on different coordination protocols that all the components of a given DHT must observe.

All current implementations of DHT have this in common: the individual components of a DHT must be run by the participants of the application in question—henceforth to be referred to simply as the participants—who use their component as a gateway to the DHT at large. Unfortunately, this aspect of the current approach to the implementation of DHTs makes it seriously vulnerable to attacks. Although all the participants get the software of the DHT components, for them to run it locally, there is no guarantee that everyone is going to

© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2016
S. Guo et al. (Eds.): CollaborateCom 2015, LNICST 163, pp. 257–266, 2016.
DOI: 10.1007/978-3-319-28910-6_23

follow the required protocol—particularly because in many applications of DHT the participants are unknown and cannot be trusted. For example, a malicious participant may change the code to forward a lookup request to a wrong node, or to claim it is responsible for the requested key and then respond with bogus result, or simply deny the existence of certain key.

This is a well known problem, which has been studied by several researchers. This research has been reviewed in [15], which discusses several types of attacks on a DHT, and describes protective measures against them, by changing the DHT protocol in various ways. But since such a changed protocol is to be executed locally, the protective measures built into it are themselves vulnerable to malicious attacks by some participants.

The Contribution of this Paper: We address this issue by taking the responsibility of carrying out the DHT protocol in question away from the untrusted participants, entrusting it to a trustworthy middleware used to govern the interaction of the participants with the DHT. The resulting architecture is called GDHT, for Governed DHT. And the middleware on which it is based is called law-governed interaction, or LGI. This middleware is decentralized, and thus scalable, and stateful—it needs to be stateful to be able to handle the highly stateful nature of the various DHT protocols. As a proof of concept we describe here the implementation of GDHT for the Chord version of DHT.

The rest of this paper is organized as follows: Sect. 2 discusses the attack models that the DHTs are mostly vulnerable to, and the proposed approaches for resolving them. Section 3 introduces the model of GDHT. Section 4 describes our implemented Chord version of GDHT that demonstrates how this abstract model can be used for a concrete application. And we conclude in Sect. 5.

2 Attack Models and the Limitations of Current Solutions

While the users of DHT benefit from the availability and scalability the DHT provides, they also face certain security and fault tolerance issues, especially when the information and resources, which are stored and transmitted in participants of DHT, are critical and sensitive. Because the implementations of DHT are usually distributed through releasing a suite of software for downloading. The participants get the software and run it locally. They rely on that all the participants follow the same protocol, by running the same code of software. However, since the software is run at each participant's computer, there is no guarantee that everyone is going to follow the same protocol. For example, a malicious participant may change the code to forward a lookup request to a wrong node, or to claim it is responsible for the requested key and then respond with bogus result, or simply deny the existence of certain key.

There are a wide range of attacks that malicious participants could exploit and launch on the software to gain illicit benefit. Several attack models focus on the nature of DHT participants or the routing between them. Chief among

them are *routing attack* and *the Sybil attack*. Basically, these attacks try to create malicious nodes (numerously) and then deceive the benign nodes collaboratively.

In the rest of this section, we are going to first describe the two attack models, analyze the solutions that researchers proposed, and then show their limitations.

2.1 Routing Attack

A routing attack is generally an attempt to prevent the routing of a lookup request from being successful [15]. For example, a malicious participant can refuse to forward lookup requests. Or it can forward the request to a non-existing or another compromised participant. Moreover, it could pretend to be responsible for certain key. It's also possible that the malicious participant routes requests normally, but denies the existence of a valid key or to respond with bogus result.

The approaches of defending routing attacks can be generalized to two main categories—redundant routing [3,7] and redundant storage [12]. Redundant routing employs either mechanisms like wide path or multiple routing table, while redundant storage replicates the data (or metadata of the location). Both methods are very costly as they increase the overhead of each routing hop or the numbers of transmission operations and the actual storage space—not to mention the synchronization issue between replicas. Even so, the cost is not the major problem. What is worse is that the redundancy cannot guarantee the success of the routing. Those solutions are feasible only when a reasonably low fraction f of participants are malicious [15]. We are going to show, in next section that a Sybil attack easily breaks these defenses by effectively increasing f.

We also demonstrate, in our implementation in Sect. 4, that how our model can enforce everyone to follow the same protocol, while not assuming there is only very small percentage of malicious participants.

2.2 Sybil Attack

A Sybil attack exploits the fact that in a distributed system, if the system fails to guarantee that each logical identity refers to a single remote physical entity, an attacker could create a large number of identities and dominate the overlay network by fooling the protocols and subverting mechanisms based on redundancy. The Sybil attack does not damage the DHT by itself, but can be used as a vector to create a majority of colluding malicious participants of it [15]. This attack is not specific to DHTs, but it is important because DHTs are vulnerable to it and the attack can be used to facilitate the execution of many other attacks. For example, if there are many malicious identities in the system, it becomes easier to pollute the routing tables of honest participants, and control the majority of the replicas for a given key.

A general solution to the Sybil attack is to use certification [3,15]. It assumes the existence of a trusted certification authority (CA) to make sure that one physical entity can only acquire one valid identifier. However, the proposed approaches failed to guarantee that every participant is going to check the certificate. Therefore, the malicious participants can subvert this mechanism and

create large amount of identities, even when the majority of participants are still willing to check the certificate. We are going to show, in our implementation in Sect. 4, that how our model can enforce everyone's certificate gets checked.

Another group of solutions rely on binding certain metrics to an identifier [1, 6, 16, 19]. Nevertheless, these solutions are either too rigid when relying on static information (e.g. IPs, network characteristics, geographic coordinates) or hardly working when relying on relatively dynamic metrics (e.g. network performance, social informations). Moreover, some approaches proposed to use computational puzzles [2], which try to limit the number of fake identities generated by malicious participants by having honest participants request each other to solve puzzles that require a significant amount of computational resources. The idea is to limit the capability of a malicious participant to generate multiple identities. However, similar to the certification solution, since it consumes a lot of resources, they cannot guarantee that everyone is willing to participate in this mechanism. They also need our model to enforce everyone to follow the protocol.

3 A Model of Governed Distributed Hash Table (GDHT)

We introduce here a model of a DHT that enables the regulation of a DHT via enforced protocol that can establish its overall structure and behavior. We call this model GDHT (for *Governed Distributed Hash Table*).

The model employs our previous work—the Law-Governed Interaction (LGI). LGI is a middleware that can govern the interaction (via message exchange) between distributed *actors*, by enforcing an explicitly specified law about such interaction. A detailed presentation of LGI can be found in its manual [9]—which describes the release of an implementation of the main parts of LGI.

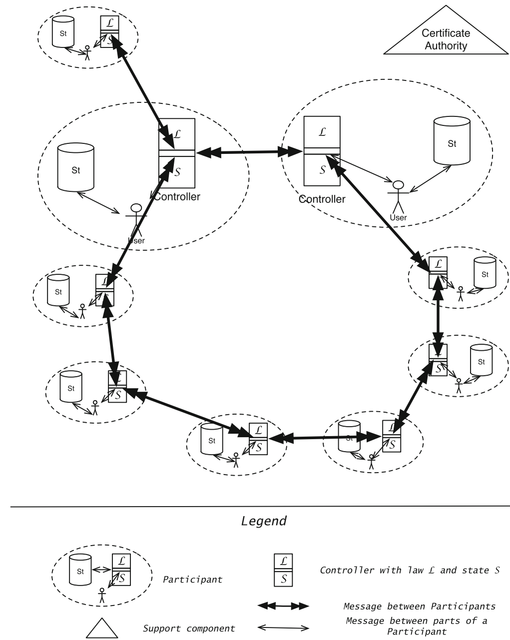

Fig. 1. The anatomy of a GDHT

The GDHT model is *generic*, and rather abstract, in the sense that it does not have any built-in communal structure. But it can support a wide range of different types of DHTs, whose structure and behavior is determined by the laws chosen for them. We do, however, present a concrete implementation of a specific Chord

protocol, in Sect. 4, to show our model can be easily used for building variety of DHTs.

This section is organized as follows. Section 3.1 is a definition of this model; Sect. 3.2 describes the launching of a GDHT; and Sect. 3.3 discusses the manner in which such a DHT operates.

3.1 A Definition of a GDHT

A GDHT-community D is defined as a 4-tuple $\langle P, \mathcal{L}, C, S\rangle$, where P is the set of participants of D; $\mathcal{L}$ is the law that governs this GDHT, and is often denoted by $\mathcal{L}_D$; C is a set of generic LGI controllers that serves as the middleware trusted to enforce any law $\mathcal{L}$ loaded into them; and S, called the *support* of D, is a set of components that provides various services to D, and is mostly specific to it. We now elaborate on this definition of the GDHT model by providing some details about its four elements, and about the relations among them. This overall structure of a GDHT is depicted schematically in Fig. 1.

The Set P of Participants of a GDHT: An individual participant x of a GDHT D is a triple $\langle user, mediator, storage\rangle$, where *user* is usually a human, operating via some kind of computational platform, like a computer or smart phone; *mediator* is one of the LGI-controllers in C that mediates all interactions between x and other participants of D; and *storage*, is the repository of resources that the participant is responsible for.

The Law $\mathcal{L}_D$ of GDHT-community D: This law endows a GDHT-community with its overall structure, in particular by controlling its membership, as well as the interactive behavior of its participants. The generality of LGI laws endows this model with great deal of generality regarding the nature of the GDHT governed by it. In particular, suitable laws can make a GDHT-community behave like Chord, or like Kademlia, or any other DHT.

The Set C of Controllers: C is meant to be the *trusted computing base* (TCB) of a GDHT. Every user can create its own controller, using the software provided by the released LGI middleware. But if malicious corruption of controllers by their users is of concern, then it is better for the participants to adopt controllers created and maintained by a trusted *controller service* (CoS), so that they can authenticate each other as bona fide LGI controllers. For such a CoS to be trusted to provide genuine controllers, this service needs to be managed by a trusted organization. It should be pointed out that the organization that maintains the CoS does not have the access to the data exchanged between the participants for several reasons. First each individual controller has access only to very small part of the exchanges, and even these are maintained for just a fleeting moment.

The Support S of a Given GDHT D: A GDHT D may require various services that are not themselves participant of D; and most of them are not defined by the generic GDHT model. Such services may be designed specifically for the DHT at hand, or may exist independently of it. Participant of D interacts with such services subject to law $\mathcal{L}_D$, while the services themselves may not

communicate subject to this or other LGI-law. A certification authority (CA) is such an example that may belong to S, which is used for the key authentication.

Note that the existence of central support service would not compromise significantly the scalability of a GDHT, if it is offline or used relatively rarely. And it would not compromise significantly the privacy of a GDHT, if it does not contain sensitive information.

3.2 The Launching of a GDHT

A specific GDHT D is launched by constructing its *foundation*, and then having individual participants join it incrementally. The foundation of D consists of: (1) the law $\mathcal{L}_D$ under which this GDHT is to operate; (2) the controller service CoS, whose controllers would enforce this law; and (3) the support S to be used by this particular GDHT. Each of these parts of the foundation of D can be either built specifically for it, or selected from existing such items. In particular, the controller service CoS may be managed and maintained specifically for D, but it may already exist, serving many different GDHTs, as well as other applications. And some, or all, parts of the *support* S of D—such as its CA—may have an independent existence, serving other applications.

Once the foundation of D exists, anybody can attempt to join it as a participant via the following three steps: First, the user needs to deploy its private storage. Second, the user needs to acquire an LGI-controller from the CoS used by D, and instruct this controller to download law $\mathcal{L}_D$ from the law server. Finally, the user should *adopt* this controller as its mediator.

Note, however, that the adoption is governed by law $\mathcal{L}_D$, which may require, among other things, certain certificates to be provided by the user. If the user does not satisfy the requirements of law $\mathcal{L}_D$ then the adoption will fail. This is one way to control the membership of a given DHT and defend the Sybil attack.

3.3 The Operation of a GDHT

Consider a participant x of D sending a message m, for example a lookup request, to another participant y. The message first arrives at the controller of x, which operates under law $\mathcal{L}_D$. If this controller forwards the message m to y—note that it may decide to block it—then m first arrives at the controller of y, which decides what to do with it according to law $\mathcal{L}_D$. In other words, participants of a GDHT interact with each other via their controllers, and the controllers communicate with each other subject to the law $\mathcal{L}_D$ of the DHT.

Figure 1 may help understand the situation. This figure depicts several participants, represented by ovals, each of which encloses the three components of a participant: the user, its mediator (controller), and its storage. The interaction between participants is depicted by the thick arrows. The component parts of a participant interact with each other as depicted by the thin arrows.

4 The Implementation of Chord GDHT

In this section, we are going to show our implementation of GDHT. It is an implementation of Chord, with two security enhancements—certification and redundant routing. The reason we chose Chord is for the sake of simplicity. Although it is not an optimal protocol, it is quite simple and easy to read. By demonstrating our implementation of Chord, we show that our GDHT model is capable of building arbitrary type of DHTs, as long as the protocol is specified in law. Similarly, we chose certification as an example to show our capability of implementing the security features, which are discussed in Sect. 2.

4.1 Chord Overview

Chord is one of DHT protocols, introduced by [14]. A DHT assigns keys to different nodes (participants); a node stores the values for all the keys for which it is responsible. The protocol regulates how keys are assigned to nodes and how find out the value for certain key by locating the node responsible for it. Nodes and keys are both assigned an m-bit identifier using consistent hashing, which is critical to the performance and correctness of Chord. Both keys and nodes are distributed in the same identifier space with very low possibility of collision. Moreover, it also enables nodes to join and leave the network without disruption.

Nodes and keys are arranged in an identifier circle that has at most 2^n nodes. (n is large enough to avoid collision.) Each node has a successor and a predecessor. The successor is the next node in the identifier circle in a clockwise direction. The predecessor is counter-clockwise. If there is a node for each possible ID, the successor of node 0 is node 1, and the predecessor of node 0 is node 2^n - 1; however, normally there are "holes" in the sequence. The concept of successor can be used for keys as well. The successor node of a key k is the first node whose ID equals to k or follows k in the identifier circle, denoted by *successor(k)*. Every key is stored at its successor node, so looking up a key k is to query *successor(k)* [14].

4.2 The Law of Chord GDHT

Rule $\mathcal{R}1$ shows how a user joins a GDHT. There are two cases of joining a DHT: (1) if a participant x knows some other participant y in the DHT already, then x will query for its successor from y; (2) otherwise, x is the creator of this DHT, then it will set its successor as itself. x, whether it is the creator, will also set obligations to periodically check the liveness of its successor and predecessor, and update its finger table. We will show later how to achieve when time is due.

The key of a participant is determined by the hash of its name. Here we assume that each participant has unique name. If the uniqueness of names cannot be guaranteed, the use of other unique ID (e.g. social security number or driver license number) should be applied. To prevent malicious participants providing multiple fake names to launch the Sybil attack, the controller will check the certificate which certifies the authenticity of the name (Fig. 2).

```
R1.
    UPON adopted(Arg, X, cert(issuer(CA),subj(X),attr([name(X)])))
            DO[add(Key(hash(X)));
            imposeObligation(stabilize, 1, min),
            imposeObligation(fix_fingers, 1, min),
            imposeObligation(check_predecessor, 1, min)]
            IF Arg = nil
            DO[ add(Predecessor(nil)),
            add(Successor(Self))]
            ELSE
            DO[ add(Predecessor(nil)),
            forward(Self, find_successor(Key@CS, Self, Successor@CS), Arg)]
R2.
    UPON arrived(X, find_successor(id, callback_id, callback_var), Y)
            IF id in (Self, Successor@CS]
            DO[forward(Self, found_successor(Successor@CS, callback_var), callback_id)]
            ELSE
            DO[next_hop = closest_preceding_node(id),
            forward(Self, find_successor(id, callback_id, callback_var), next_hop)]
R3.
    UPON arrived(X, get_predecessor, Y)
            DO[forward(Self, predecessor(Predecessor@CS), X)]
R4.
    UPON obligationDue(stabilize)
            DO[forward(Self, get_predecessor, Successor@CS),
            imposeObligation(stabilize, 1, min)]
R5.
    UPON arrived(X, predecessor, Y)
            IF predecessor in (Key@CS, Successor@CS)
            DO[add(Successor(predecessor)),
            forward(Self, notify(Key@CS), Successor@CS)]
R6.
    UPON arrived(X, notify(id), Y)
            IF Predecessor@CS = nil
            or id in (Predecessor@CS, Key@CS)
            DO[add(Predecessor(id))]
R7.
    UPON obligationDue(fix_fingers)
            DO[imposeObligation(fix_fingers, 1, min),
            update_finger_table(next++, find_successor(Key@CS + 2^next-1))]
R8.
    UPON obligationDue(check_predecessor)
            IF Predecessor@CS is Unreachable
            DO[add(Predecessor(nil))]
```

Fig. 2. Law $\mathcal{L}_D$: Implementation of chord GDHT

Once a participant is asked about the successor of a key, according to Rule $\mathcal{R}2$ it will first check whether its successor is responsible for that key. If so, it will return its successor, otherwise, it will search its local finger table for the highest predecessor of that key, and then ask it for the successor of that key.

Rule $\mathcal{R}3$ simply shows when a participant receives a query about its predecessor, it will respond with it.

Every participant runs a function called *stabilize* periodically to learn about newly joined participants. Rule $\mathcal{R}4$ and Rule $\mathcal{R}5$ show that a participant x calls its successor y periodically about its predecessor, if y's predecessor is not x but some participant z between them, meaning z is a newly joined participant, x will notify z that it is z's predecessor. In the case that x is a newly joined participant, it will notify its successor y that it is y's predecessor. Rule $\mathcal{R}6$ shows that when

notified by another participant that it is its predecessor, mostly because that participant just joined, it will update its predecessor in control state.

Each participant periodically calls a function named *fix fingers* to make sure the finger table entries are updated. This is how new joined nodes initialize the finger tables, and how existing nodes add new nodes into their own finger tables. Rule $\mathcal{R}7$ shows that a participant will periodically check each entry in its finger table and update accordingly.

Each participant also runs a function called *check predecessor* periodically, to cleanup the participant's predecessor reference if its predecessor becomes unavailable; this enables the node to accept a new predecessor when notified. Rule $\mathcal{R}8$ shows that a participant will periodically check the liveness of its predecessor.

4.3 Additional Security Features

As we mentioned in Sect. 2, there are several protective measures proposed to resolve the vulnerability of DHT from different aspects, by means of changing the DHT protocol. But since such a changed protocol is still to be executed locally, the protective measures themselves are also vulnerable to malicious attacks.

However, if certain changes of the protocol can be written into $\mathcal{L}_D$ and carried out by the controllers of GDHT, it would be a very good supplement to the Chord version of GDHT, from both security and fault tolerance perspectives. In our implementation, we employed redundant routing and certification as examples for handling routing attack and the Sybil attack. Due to lack of space, we only showed how to enforce the use of certification above. Most of proposed approaches can be implemented under GDHT and they rely on GDHT to guarantee the executions.

5 Conclusion

This paper addresses the vulnerability to security and fault tolerance posed by the nature of DHT. Namely, the correctness of the DHT relies on that all the participants follow the same protocol. Unfortunately, this cannot be guaranteed by the current approach of implementing DHTs. These risks prompted several attempts to enhance the DHT protocols. However, these solutions are to be executed locally, making themselves vulnerable to malicious attacks.

We address this issue by carrying out the DHT protocol away from the untrusted participants, entrusting it to a trustworthy middleware used to govern the interaction of the participants with the DHT. The architecture is called GDHT, for Governed DHT, which is decentralized, scalable, and stateful. As a proof of concept we implemented the Chord version of GDHT, with some security enhancements. The overhead added by employing GDHT is quite negligible, comparing to the enhancements from security and fault tolerance aspects.

References

1. Bazzi, R.A., Konjevod, G.: On the establishment of distinct identities in overlay networks. In: Distributed Computing, vol. 19 (2007)
2. Borisov, N.: Computational puzzles as sybil defenses. In: 2006 Sixth IEEE International Conference on Peer-to-Peer Computing, P2P 2006. IEEE (2006)
3. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: Secure routing for structured peer-to-peer overlay networks. ACM SIGOPS Operating Syst. Rev. **36**, 299–314 (2002)
4. Cohen, B.: Incentives build robustness in bittorrent. In: Workshop on Economics of Peer-to-Peer systems, vol. 6 (2003)
5. Dabek, F., Kaashoek, F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. ACM SIGOPS Operating Syst. Rev. **35**, 202–215 (2001)
6. Danezis, G., Lesniewski-Laas, C., Kaashoek, M.F., Anderson, R.: Sybil-resistant DHT routing. In: di Vimercati, S.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 305–318. Springer, Heidelberg (2005)
7. Hildrum, K., Kubiatowicz, J.D.: Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In: Fich, F.E. (ed.) DISC 2003. LNCS, vol. 2848, pp. 321–336. Springer, Heidelberg (2003)
8. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer information system based on the XOR metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
9. Minsky, N.H.: Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual), Rutgers, February 2006. http://www.moses.rutgers.edu/
10. Pappas, V., Massey, D., Terzis, A., Zhang, L.: A comparative study of the DNS design with DHT-based alternatives. In: INFOCOM (2006)
11. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network, vol. 31. ACM (2001)
12. Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., Harlan, Y.: OpenDHT: a public DHT service and its uses. In: ACM SIGCOMM Computer Communication Review, vol. 35 (2005)
13. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, p. 329. Springer, Heidelberg (2001)
14. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. **11**, 17–32 (2003)
15. Urdaneta, G., Pierre, G., Van Steen, M.: A survey of DHT security techniques. ACM Comput. Surv. (CSUR), 43 (2011)
16. Wang, H., Zhu, Y., Hu, Y.: An efficient and secure peer-to-peer overlay network. In: 2005 IEEE Conference on Local Computer Networks 30th Anniversary. IEEE (2005)
17. Wang, Z., Minsky, N.: Establishing global policies over decentralized online social networks. In: Proceedings of the 9th IEEE International Workshop on Trusted Collaboration, October 2014
18. Wang, Z., Minsky, N.: Regularity based decentralized social networks. In: Proceedings of the 9th International Conference on Risks and Security of Internet and Systems (CRiSIS2014), October 2014
19. Haifeng, Y., Kaminsky, M., Gibbons, P.B., Flaxman, A.: Sybilguard: defending against sybil attacks via social networks. ACM SIGCOMM Comput. Commun. Rev. **36**, 267–278 (2006)