Dependability in Peer-to-Peer Systems

Flavio DePaoli and Leonardo Mariani • *Università degli Studi di Milano Bicocca*

Server-centric architectures such as the Web's suffer from well-known problems related to application size and increasing user requests. Peer-to-peer systems can help address some of the key challenges, but this survey of several current P2P systems shows that dependability remains an open issue. To perform in Internet-scale applications, P2P systems must address the four major properties of dependable systems: scalability, fault-tolerance, security, and anonymity.

Aiming to ease the problems associated with making a variety of items available to multiple users over the Internet, peer-to-peer (P2P) systems create fully decentralized systems. By involving lots of peers, computational and storage capabilities increase; moreover, data replication, caching, and efficient query routing enhance overall performance.

P2P systems can reduce complexity because each server deals with a smaller set of data and clients. However, P2P systems' long-running and dynamic nature creates dependability issues. In this article, we consider dependability with reference to scalability, fault-tolerance, security, and anonymity properties. Scalability and fault-tolerance depend on the number of peers and their interconnection, but security and anonymity depend on the systems' open nature and the Internet's communication infrastructure. The fact that any peer can contact any other in a network introduces data-integrity and privacy issues.

In the not-too-distant future, self-adaptable and dependable systems will emerge from the development of nomadic technologies, mobile devices, and ubiquitous computing. Recent research has used P2P systems to organize ubiquitous environments[1] and to perform worldwide computations.[2] Because most current systems were designed to address specific application domains, similar concepts have been addressed with different terminologies and models. An output of the comparison provided in this article is an attempt to move toward common terms and definitions. Because the models underlying current P2P systems must be understood to support a thorough investigation of dependability properties, we briefly examine the most popular P2P systems and then compare how these systems address dependability.

P2P-Based Systems

Most developers initially used P2P technology to implement file-sharing systems that spanned Internet-like environments, with a focus on searching and routing. More recently, they have used it to implement hash-map-like functionalities (the ability to map a key to a location) in distributed settings. P2P technology is evolving in response to peer unreliability, which continues to further the need for fault-tolerant systems. It has also become clear that effective systems require security and anonymity, so encryption and authentication techniques for large-scale decentralized systems are starting to arrive. P2P systems have further influenced research into distributed file systems, which are starting to resemble the P2P organization models.

Several organizational models with different degrees of dependability have been proposed over the past few years. Before discussing specific dependability issues, though, let's look at the most representative P2P systems. They reveal the basic mechanisms that support the system properties discussed in the rest of the article. To unify the terminology, we'll use the term *item* to refer to any general entity — a file, object, or other artifact — stored in a peer in any of these systems.

Napster

Napster (www.napster.com) lets users share audio files via network nodes. Although one of the oldest and most famous, however, Napster is a *hybrid* rather than a true P2P system (the search service is centralized). When a peer joins the network, it uploads information about locally stored files to a central server, and clients locate these files by sending requests to that server. Peer-to-peer interactions occur only during file transfer.

Gnutella

In contrast to the Napster approach, Gnutella (www.gnutella.com)[3] supplies an open, decentralized, P2P search protocol. New peers join a Gnutella network by contacting an already connected peer, which they then use to discover additional peers by broadcasting ping messages. The Gnutella approach does not require centralization: query messages are broadcast along the network through the set of known peers. Each peer receiving a query first tries to satisfy the request with its local data and then forwards the message to all known peers. In this way, a query propagates in the system until an associated time-to-live (maximum number of hops) expires.

P-Grid

The self-organizing P-Grid (www.p-grid.org)[4] system addresses responsiveness and robustness by exploiting a tree-based distributed indexing structure that maps items with nodes according to a binary-key value. Each peer p is associated with a node n of the tree; therefore, p becomes responsible for all items contained in the subtree rooted at node n. When a new peer joins the network, it computes routing data via a peer that is already part of P-Grid and negotiates item responsibility. The construction algorithm requires only local interactions and guarantees that all the tree's nodes can be reached. The tree structure ensures good search-response time — at most $O(\log n)$, where n is the number of items in the network — and moderates the amount of routing data stored on each peer.

Chord

By implementing a distributed lookup protocol, Chord (www.pdos.lcs.mit.edu/chord/)[5] addresses the problem of efficiently locating which nodes store which particular data items. This protocol maps peers and items to an m-bit identifier via a hash function and then arranges them in a "ring" network of at most 2^m peers. Each peer knows at least its successor peer.

When a peer joins the network, it takes the position in the ring corresponding to the assigned hash key and assumes responsibility for items according to the formula `itemID mod 2m = peerID`. The same formula assigns new items to peers; if the computed peer doesn't exist, the item is transferred to the next available peer with a greater identifier. Ring-structure navigability increases by making each peer hold a finger table (a shortcut that reduces the numbers of hops) with at most m entries. As a result, both lookup operations and table maintenance can be performed efficiently — $O(\log n)$ and $O(\log^2 n)$, respectively, where n is the number of items in the network.

CAN

The Content-Addressable Network (CAN)[6] is a self-organizing P2P system that offers a hash-map-like functionality on Internet-scale systems. CAN conceptually organizes the search space into a d-dimensional Cartesian coordinate space partitioned among peers (each area of the space corresponds to a peer). The items stored in the network are `<key, value>` pairs in which keys are computed via a hash function that associates values to areas. To enable message routing, each peer must be aware of peers responsible for neighboring areas; in a d-dimensional space, a peer therefore stores references to $2d$ peers.

A node joins the system by computing a random entrance point (a random number in the coordinate space). The peer responsible for the entrance point splits its search space by selecting one of the available dimensions and assigning half the search space to the new peer, which receives items, creates the new neighboring table, and then discloses its existence to all neighboring peers. To perform lookup operations, any peer forwards requests along a straight line in the coordinate space from the request's source to its destination. Therefore, the cost of a lookup is $O(dN^{1/d})$, where n is the number of items in the network.

Farsite

Farsite (http://research.microsoft.com/sn/Farsite/)[7] is an organization-scale distributed file system that addresses information dependability and availability even if it's built on mistrusted peers. The operations it considers are large-scale read-only item sharing and small-scale read–write item sharing.

Each peer joining the system makes part of its storage capability available for a distributed directory service to manage. A set of selected peers (Farsite's core group) runs a placement-and-retrieval algorithm; this group's presence makes Farsite a hybrid system. The core group forms a Byzantine-fault-tolerant group[8] that overcomes the lack of trust and supplies the requested authentication and security properties by increasing intragroup communication. Message-authentication code (MAC) authenticates communication between clients and server groups, whereas certificates and certification authorities validate authority.

Freenet

Freenet (http://freenet.sourceforge.net/)[9] is a decentralized storage system that focuses on anonymity, but at the cost of unpredictable retrieval times. Each peer stores a routing table that contains its neighbors' addresses along with numeric keys corresponding to items stored in them. This table also helps route requests to the peers storing nearest key values. The search works as a steepest-ascent hill climbing with a backtracking algorithm that forwards messages for a maximum number of hops and then backtracks to follow a different path if it collects no positive answer. Peers add a new entry in the routing table whenever they receive either an announcement message disclosed from a joining peer or a query message sent from an unknown peer. In Freenet, nodes can freely join and leave the network: file persistence is not requested, and files are replicated on multiple machines.

Oceanstore

Oceanstore (http://oceanstore.cs.berkeley.edu/),[10] a global-scale utility infrastructure for raw data management, is designed to handle billions of users and exabytes of data. The system is composed of mistrusted servers; it uses redundancy and client-side cryptographic techniques to protect items. The routing model used in Oceanstore is Tapestry,[11] a routing infrastructure focusing on fault tolerance, automatic repair, and self-organization that enables item retrieval with $O(\log_b N)$ hops. It does this by maintaining a table of $O(b * \log_b N)$ entries on each peer (N is the number of peers, and b is a Tapestry parameter).

Oceanstore associates each item with a responsible party (a group of hosts) that splits the item into several fragments and then sends them across the network. A client must collect the fragments to reconstruct the original item. Any peer can perform lookups by contacting the inner ring that gathers the item's building blocks via the Tapestry routing infrastructure.

Scalability

Scalability is the degree of adaptability that a system exhibits with respect to increasing-load situations. P2P systems are highly dynamic and unpredictable in size (growing or shrinking), topology (connecting or disconnecting peers), and activity (number or kind of requests). To ensure high quality of service, issues such as adaptability with respect to the number of peers, items, and messages, as well as the presence of any hot spots (peers hosting popular items), must be addressed.

Gnutella and Napster systems behave efficiently when the number of items or peers increases. Napster reacts by increasing the amount of data maintained on the central server (administrators can increase computational resources to efficiently face data growth), and Gnutella simply increases its size by including new peers. As the number of requests increases, however, Napster and Gnutella suffer from scalability limits: Napster's central server easily becomes overloaded with requests, whereas Gnutella's network tends to fill up with messages. In both systems, a hot spot's presence has no influence on routing performance (because searches are not goal directed), but a peer storing a popular item might have to transfer it continuously.

One way to solve the problem of increasing requests is to implement goal-directed searches. P-Grid, Chord, CAN, and Oceanstore have different complex infrastructures that enable goal-directed logarithmic cost searches. Logarithmic searches scale quite gracefully with respect to increasing requests, but they fail to deal with hot spots — peers hosting popular items cannot be protected from floods of requests. Chord can reduce a search's logarithmic cost by first increasing the amount of data the peers maintain and then increasing the cost of peers joining or leaving. CAN allows for similar results by implementing multiple dimensions.[6]

Many systems address the problem of frequently requested items by implementing caching and replica-management mechanisms. For caching, peers maintain local copies of popular items; for replica management, a peer loaded with several requests distributes copies of the most popular items among its neighbors to share the load. Farsite implements the caching mechanism for pathname: peers store responses obtained from the core group that maps pathnames to physical locations, and Freenet intensively caches requested items so that peers become specialized for families of similar keys. Oceanstore maintains multiple item fragments spread throughout the network, whereas CAN implements both caching and replica management.

For mutable content — items that can change dynamically — the introduction of cached items and replicas requires management mechanisms to deal with any synchronization problems. Updates on a single item can introduce system inconsistency if copies of that item are not promptly updated, but immediately updating all copies might be infeasible because of how many resources system-scale searches for copies consume. Existing approaches to managing mutable items are often inaccurate and sacrifice correctness in favor of efficiency. Among the systems we surveyed, we found that P-Grid can run multiple depth-first searches, a single breadth-first search, or searches based on "buddy lists" (lists of peers sharing the same key) to locate and update copies;[4] CAN executes multiple searches for the same item and selects a valid response by looking at the number of identical answers. The realization of a reliable, efficient replica management technique remains an open issue.

Homogeneous distribution of items among peers (to reduce the probability of unbalanced bandwidth and storage consumption) is also an issue. Several P2P systems run placement algorithms to decide where to store items and zone-reassignment algorithms to redistribute the search space's responsibilities,[6] but current approaches do not guarantee balanced distribution. Moreover, research in semantic-oriented P2P systems concerning the creation of semantic models and algorithms for placement, searching, and updating is still in its infancy.

Existing systems pay logarithmic costs for join, leave, and search operations, but in some applicative domains, privileging efficiency of some operations in favor of others could be more effective. Chord and CAN partially explore an interesting research direction in defining configurable scalability: both offer the possibility of choosing at deployment time the value of some of the parameters that affect search, join, and leave operations.

Fault Tolerance

Fault tolerance is a system's ability to supply regular service operations in the presence of hardware or software faults. In principle, the absence of centralized control and coordination makes P2P systems robust with respect to failures that might occur at any peer. Faults at client peers don't usually affect system behavior (because only private information gets lost), but faults at server peers can result in data loss and denial of service.

Napster's centralized nature makes it fragile with respect to faults: if the central server stops working, all services become unavailable. To restore this type of system, server peers might need to upload data to the centralized server again. Farsite and Oceanstore eliminate the single failure point by substituting a Byzantine server group for the single server.

Napster's centralized nature makes it fragile with respect to faults: if the central server stops working, all services become unavailable.

The pure P2P organizational model implies full decentralization of services — a radical solution to failing servers. We can distinguish two meaningful types of pure P2P systems with respect to fault tolerance: those that implement goal-directed searches and those that implement generic routing strategies.

Pure P2P systems with generic routing strategies (Gnutella and Freenet, for example) are the best choice for fault tolerance because no one peer should be more important than any other. Each implements the same set of functionalities, so a failing peer loses only locally stored items. In practice, though, peers in uncontrolled environments tend to organize themselves as in the small-world network model:[12] a few nodes become strategically more important than the others because of the high number of connections they manage. If a peer with many connections fails, serious consequences for routing efficiency occur because several peers can become difficult or impossible to reach.

An alternative solution to uncontrolled topology is to manipulate the system-construction process to support goal-directed searches, as Chord and CAN do. Chord achieves fault tolerance by storing references to r nearest successors in each peer instead of only storing the next one. Moreover, each peer runs a stabilization algorithm that maintains correctness on references by checking if the successor's predecessor differs from itself. CAN addresses fault tolerance with several techniques directly derived from the system model. One technique is the definition of *multiple realities* (multiple coordinate spaces): each item is stored once for each reality, so that if it is no longer available (because a fault has occurred in the storing peer), a requestor can still retrieve it from another reality. Another way to implement fault tolerance in CAN is to overload coordinate zones — each zone can be shared among several peers (up to a MAXPEERS parameter before splitting). This approach reduces the average path length (placing multiple peers on the same zone is similar to reducing the number of peers) and per-hop latency (each peer has multiple choices at each step). It also improves fault tolerance because no vacant zones exist (unless all peers responsible for the same area crash simultaneously, which is highly unlikely). Whenever a peer is deemed to have died, neighbors execute a TAKEOVER procedure that reassigns the responsibility for the vacant search space to the neighboring peer with the smallest volume of data.

Early P2P systems didn't implement security mechanisms, but the research community is increasingly paying attention to security issues.

Replica and caching mechanisms provide basic fault-tolerance support independently of the topology-management strategy.[13] P-Grid replicates items among peers with the same key value, whereas CAN addresses replica management by *multiple hash functions* that map every item to multiple places. This latter approach enables replica location, but at the cost of the hash function's computation. Farsite implements replication at both the server and client sides, so metadata can migrate to another machine whenever a machine is unavailable for a certain period of time. Oceanstore addresses fault tolerance in several additional ways. In one, it uses the *erasure code technique*, which consists of storing items by partitioning them in n parts and replicating each part in an arbitrary number of peers, such that the total number of copies m should be $m > n$. In another, the Tapestry routing infrastructure[11] exploits redundant neighbor pointers to increase reliability. Finally, Oceanstore implements four repair mechanisms that guarantee extreme durability: disk failure prediction, local server sweeps, distributed detection and repair, and global sweep.[10]

Security

Security is defined as a system's ability to manage and protect sensitive information. Early P2P systems didn't implement security mechanisms, but the research community is increasingly paying attention to security issues, especially for P2P systems that implement file-system-like features. P2P systems introduce several challenges for security professionals because a malicious peer can act as either a client or server (mutual trust among computers should not be assumed). Moreover, malicious peers could cause a variety of attacks, from supplying wrong answers to stopping the entire system.

Hybrid P2P systems that implement the Byzantine Fault Tolerant Protocol[8] (Farsite and Oceanstore, for instance) ensure a certain level of security and trust because the protocol won't allow the entire group to misbehave or fail unless malicious or faulty peers make up one third of the group's members. A drawback is that the Byzantine Fault Tolerant Protocol is effective only for small groups of servers, not the large groups of decentralized servers typically found in pure P2P systems. Farsite's security is even greater because MAC authenticates the messages the core group receives.

Both hybrid and pure P2P systems address security with encryption techniques. Farsite implements the *convergent encryption technique*,[14] which encrypts identical files into the same cipher text regardless of the user key. It works by computing a one-way hash of each block of the file and then encrypting the block with this hash. In turn, the public keys of all the users with permission to access the file encrypt the hashes. Farsite also increases trust by using MAC and certificates. Oceanstore uses public-key encryption techniques to protect client-side data. Freenet implements data-encryption techniques to ensure privacy, rather than to protect against malicious access. Accordingly, dictionary attacks can be particularly effective in discovering and collecting passwords on Freenet's encrypted files.

Freenet explicitly addresses several types of attacks. For local *eavesdroppers* (peers that can observe activities of another peer), it assures no protection if the user contacts a malicious eavesdropper on the first node of the network, but in all other cases, encrypted messages reduce eavesdroppers' effectiveness. Freenet reveals several denial-of-service attacks; in particular, it takes into consideration attempts to fill the network with junk files and key collisions. (A key collision is caused when two different files correspond to the same key value.) To deal with junk file injection, Freenet can divide the data store into two parts: one for new files and another for established files (those with many requests). The rule is that new files can only displace other new files: an attack can affect insert operations, but it can't displace established files. Another approach is the Hashcash technique,[15] which slows down normal service execution to limit any denial-of-service attack's effectiveness.

Freenet also deals with key-collision attacks. If a malicious peer tries to insert hash-colliding items, the system checks for the existence of an item with the same key within a certain scope. This check makes it very difficult to insert colliding items. Moreover, if Freenet adopts a content-based encryption technique, successfully performing such an attack becomes impossible.

When attacking a P2P system, knowledge of the actual topology can increase the attack's effectiveness. In hybrid systems, attackers focus on centralization points; for decentralized systems, attackers can exploit the small-world network model[12] to target the peers that manage either numerous or important wide-range connections. Only a homogeneous distribution of connections provides no reference points for attackers. Collaboration among malicious peers can facilitate discovering both system topology information and single-peer activity. A group of malicious peers can also execute very effective coordinated attacks.

Anonymity

Anonymity is the degree to which a software system allows for anonymous operations. In P2P systems, anonymity is addressed by enabling anonymous searches, decoupling service providers and consumers, and hiding the peers that store data.

Freenet is the only P2P-like storage system that implements anonymity — of the sender and of the searched items. In general, any peer can identify a request's sender by looking at the message's depth value (the maximum number of hops). In Freenet, this information is obscured by random selection of the initial depth value and by probabilistically incrementing it at each step. Freenet achieves sender and key anonymity by performing a prerouting procedure. Once prerouting is finished, a peer that receives the message won't know the original sender's identity. Anonymity of both sender and content is also achieved during prerouting because nodes along the prerouting path forward the message by using a succession of public keys (which also encrypt the message) rather than by the key value. Once the prerouting phase is completed, it is impossible to maintain key anonymity because the key value is used to route messages. Freenet also protects the data source (the peer responding affirmatively to a request) by occasionally updating the data source field contained in a reply: in so doing, any peer receiving the message can't distinguish between a forwarding peer and the peer that really stores the searched item. Anonymity is further improved by forwarding with finite probability any message with a hops-to-live equal to one.

P2P systems that do not implement any anonymizing technique could take advantage of anonymizing network layers, such as Tarzan.[16] Tarzan is a P2P-specific anonymizing network layer that addresses anonymity by implementing encryption techniques and multihop routing. Technically, a message is initially routed among secure Tarzan hosts and then sent over the Internet network. Tarzan self-checks for correctness, and if prerouting fails, it automatically tracks for the failing (or malicious) peer.

Conclusions

Most P2P systems are experimental developments that focus on specific aspects without aiming to deliver real systems. The systems we examined are at different stages of maturity. Several of them address scalability and fault tolerance, but only a few of them address security and anonymity — and only partially, at that. Table 1 (next page) summarizes the results of our inspection.

None of the systems address all four of the dependability properties we considered, which implies that further developments are necessary to deliver dependable systems that effectively support real Internet-wide applications.

In the case of security, we expect next-generation P2P systems will be based on mecha-

Table 1. Peer-to-peer system comparison.

System	Application domain	Scalability	Fault-tolerance	Security	Anonymity
Napster	Sharing of audio files in Internet-like environments	Low: centralized server	N/A	N/A	N/A
Gnutella	File sharing in Internet-like environments	Low: broadcast	Broadcast	N/A	N/A
P-Grid	Access structure for Internet-like environments; compatible with Gnutella	High: logarithmic costs	Replica management	N/A	N/A
Chord	Hash map functionality in Internet-like environments	High: configurable logarithmic costs	Extra routing data	N/A	N/A
CAN	Hash map functionality in Internet-like environments	High: cheap join and leave; logarithmic lookup; multiple dimensions	Extra routing data; ring searches; multiple realities; overloading coordinate zones; multiple hash functions; `TAKEOVER` procedure	N/A	N/A
Farsite	Organization-scale distributed file system	Medium: centralized server group, but with pathname caching, leasing mechanisms, event packaging	Byzantine fault tolerant group; client- and server-replica management	Byzantine server-side protocol; convergent encryption; certificate authority; message authentication code	N/A
Freenet	Anonymous file-sharing systems for Internet-like environments	Medium: steepest-ascent hill climbing with backtracking; caching	N/A	Data encryption techniques against junk-file and key-collision attacks	Anonymous data provider and receiver
Oceanstore	Data management on global-scale environments	High: logarithmic costs	Byzantine fault tolerance; erasure codes; Tapestry; four automatic repair techniques	Byzantine server-side protocol; client-side cryptographic techniques	N/A

nisms for protecting sensitive content via encryption and authentication techniques, rather then by blocking incoming or outgoing communication. Firewalls are ineffective, for example, because it's impossible to identify static trustable perimeters on P2P systems. An even greater challenge is effectively developing fully decentralized authentication and encryption techniques.

Another open issue is that of protecting peers from coordinated attacks by groups of malicious peers. We envision approaches based on trustable computations; reputation systems could be the underlying technology for identifying malicious peers.

Several services including those related to fault tolerance will soon be ported from distributed systems to P2P systems. In particular, we'll probably see self-maintaining and repairing systems soon.

To make P2P a reliable end-user technology, highly dependable general-purpose P2P frameworks and middleware must be developed. Such middleware will provide developers with the APIs they need for delivering applications with a customized degree of dependability.

References

1. M. Takemoto et al., "The Ubiquitous Service-Oriented Network (USON): An Approach for a Ubiquitous World Based on P2P Technology," *Proc. 2nd IEEE Int'l Conf. Peer-to-Peer Computing*, IEEE CS Press, 2002, pp. 17–24.
2. D.P. Anderson et al., "Seti@home: An Experiment in Public-Resource Computing," *Comm. ACM*, vol. 45, no. 11, 2002, pp. 56–61.
3. RFC-Gnutella, "The Annotated Gnutella Protocol Specification v0.4," 2001; http://rfc-gnutella.sourceforge.net/.
4. K. Aberer, "P-Grid: A Self-Organizing Access Structure for P2P Information Systems," *Proc. Int'l Conf. Cooperative Information Systems*, LNCS 2172, Springer-Verlag, 2001, pp. 179–194.
5. I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, 2003, pp. 17–32.
6. S. Ratnasamy et al., "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM*, ACM Press, 2001, pp. 161–172.

7. A. Adya et al., "Farsite: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," *Proc. 5th Symp. Operating Systems Design and Implementation*, Usenix Assoc., 2002, pp. 1–14.
8. M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proc. 3rd Usenix Symp. Operating Systems Design and Implementation*, Usenix Assoc., 1999, pp. 173–186.
9. I. Clarke et al., "Freenet: A Distributed Anonymous Information Storage and Retrieval Systems," *Proc. ICSI Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009, Springer-Verlag, 2001, pp. 46–66.
10. S. Rhea et al., "Maintenance-Free Global Data Storage," *IEEE Internet Computing*, vol. 5, no. 5, 2001, pp. 40–49.
11. K. Hildrum et al., "Distributed Object Location in a Dynamic Network," *Proc. 14th ACM Symp. Parallel Algorithm and Architectures*, ACM Press, 2002, pp. 41–52.
12. D. Watts and S. Strogatz, "Collective Dynamics of 'Small World' Networks," *Nature*, vol. 393, June 1998, pp. 440–442.
13. G. On, J. Schmitt, and R. Steinmetz, "The Effectiveness of Realistic Replication Strategies on Quality of Availability for Peer-to-Peer Systems," *Proc. 3rd IEEE Int'l Conf. Peer-to-Peer Computing*, IEEE CS Press, 2003, pp. 57–65.
14. J.R. Douceur et al., "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," *Proc. 22th Int'l Conf. Distributed Computing Systems*, IEEE CS Press, 2002, pp. 617–624.
15. A. Back, *Hashcash: A Denial of Service Countermeasure*, 2002; www.hashcash.org.
16. M.J. Freedman and R. Morris, "Tarzan: A Peer-to-Peer Anonymizing Network Layer," *Proc. 9th ACM Conf. on Computer and Comm. Security*, ACM Press, 2002, pp. 193–206.

Flavio DePaoli is an associate professor at Università degli Studi di Milano Bicocca. His research interests include distributed systems architecture and languages, e-service computing, multiagent systems, user-centered cooperative systems, and multiuser interaction. He received a PhD in computer science from Politecnico di Milano. He is member of the IEEE and the ACM. Contact him at depaoli@disco.unimib.it.

Leonardo Mariani is a PhD student at Università degli Studi di Milano Bicocca. His research interests include testing and verifying component-based systems, multiagent systems, and decentralized systems. He is member of the IEEE and the ACM. Contact him at mariani@disco.unimib.it.