Properties and Mechanisms of Self-Organizing MANET and P2P Systems

BARTOSZ BISKUPSKI
Trinity College Dublin
JIM DOWLING
Swedish Institute of Computer Science
and
JAN SACHA
Trinity College Dublin

Despite the recent appearance of self-organizing distributed systems for Mobile Ad Hoc Networks (MANETs) and Peer-to-Peer (P2P) networks, specific theoretical aspects of both their properties and the mechanisms used to establish those properties have been largely overlooked. This has left many researchers confused as to what constitutes a self-organizing distributed system and without a vocabulary with which to discuss aspects of these systems. This article introduces an agent-based model of self-organizing MANET and P2P systems and shows how it is realised in three existing network systems. The model is based on concepts such as partial views, evaluation functions, system utility, feedback and decay. We review the three network systems, AntHocNet, SAMPLE, and Freenet, and show how they can achieve high scalability, robustness and adaptability to unpredictable changes in their environment, by using self-organizing mechanisms similar to those found in nature. They are designed to improve their operation in a dynamic, heterogeneous environment, enabling them to often demonstrate superior performance to state of the art distributed systems. This article is also addressed at researchers interested in gaining a general understanding of different mechanisms and properties of self-organization in distributed systems.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; H.1.1 [**Models and Principles**]: Systems and Information Theory—*General systems theory*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*; *information filtering*; *relevance feedback*; *search process*; *selection process*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coherence and coordination*; *intelligent agents*; *multiagent systems*

General Terms: Algorithms, Design, Management, Performance, Reliability

The work described in this article was partly supported by the "Information Society Technology" Programme of the Commission of the European Union under research contract IST-507953 (DBE).
Authors' addresses: B. Biskupski and J. Sacha, Trinity College Dublin, Dublin, Ireland; email: {biskupski, jsacha}@cs.tcd.ie; J. Dowling, Swedish Institute of Computer Science, Kista, Sweden; email: jim.dowling@sics.se.

Additional Key Words and Phrases: Adaptive systems, Complex systems, MANET, peer-to-peer, Self-organisation

ACM Reference Format:

Biskupski, B., Dowling J., and Sacha, J. 2007. Properties and mechanisms of self-organizing MANET and P2P systems. ACM Trans. Autonom. Adapt. Syst. 2, 1, Article 1 (March 2007), 34 pages DOI = 10.1145/1216895.1216896 http://doi.acm.org/10.1145/1216895.1216896.

1. INTRODUCTION

Designers of distributed systems have recently turned to decentralized mechanisms to build large-scale systems for dynamic network environments, such as Mobile Ad Hoc Networks (MANETs) and Peer-to-Peer (P2P) networks, as traditional approaches to system design based on global knowledge or strict consensus among entities are no longer viable [Babaoglu et al. 2006; Dowling et al. 2005]. However, the complexity of decentralized control increases rapidly with system size and dynamism, and researchers have turned to self-organization as a guiding principle to construct such systems. Self-organization offers the promise of providing new ways to build distributed systems from massive numbers of low-cost hosts that interact and adapt to produce properties such as self-management capabilities, robustness and adaptability to a dynamic environment.

The properties and constraints on mechanisms that are required for a system to be described as self-organizing with emergent properties have been, in our opinion, most clearly defined in the field of biology by Camazine et al. [2001] as "a process in which pattern at the global level of a system emerges from the numerous interactions among lower-level components of the system. Moreover, the rules specifying the interactions between the system's components are executed using only local information, without reference to the global pattern." This definition captures important aspects of self-organization such as autonomous components that take decisions using only local information, and how interactions between components cause system properties to emerge. However, this definition does not say anything about how the system or its constituent components interact with the system's environment, or how interaction with an external environment can be used to guide self-organization to produce macro-level structures [Prigogine and Stengers 1984].

There has been extensive research on self-organization in biological, social and physical systems [Camazine et al. 2001; Prigogine and Stengers 1984; Heylighen 2001; Albert and Barabási 2002], producing a common set of concepts with which we can discuss these systems. We use these concepts to propose an agent-based model of self-organizing MANET and P2P systems and inform a review of three selected network systems. We use this model to describe the properties of the reviewed systems and the mechanisms used to establish those properties. As part of our model, we identify several design constraints and guidelines for engineering self-organizing applications in the target network environments. While we only analyze self-organizing MANET and P2P systems in this article, we believe that our model can be extended to a general class of self-organizing distributed systems. Our model and review contrast with

previous research that examined the application of theories of self-organization to algorithms and computer applications, such as in Serugendo et al. [2004], Collier and Taylor [2004], and Babaoglu et al. [2006], in that we evaluate how self-organization is used in these systems to improve their operation by adapting to a dynamic environment.

The rest of the article is organized as follows: In Section 2, we motivate the use of self-organization for building distributed systems by discussing the limits of existing consensus-based techniques, for increasing systems size and dynamism. In Section 3, we provide brief reviews of the systems AntHocNet, SAMPLE and Freenet that we will use in the rest of the article to show how our model is realized. In Section 4, we describe the model of autonomous agents, their partial view of the system and the influence of the environment on the system design. In Section 5, we formalize how self-organizing MANET and P2P systems, and the reviewed systems in particular, can improve desired system properties, such as network throughput, by adapting their structure and behavior to their application, network and host environments. Finally, we conclude and give some directions for future work.

2. MOTIVATION FOR SELF-ORGANIZATION IN DISTRIBUTED SYSTEMS

Traditionally, distributed systems designers have used techniques such as centralized state, group communication protocols [Hayden 1997], dynamic software architectures [Garlan and Schmerl 2002], tuple spaces [Cabri et al. 2000], and interaction protocols [FIPA 2002] to coordinate the behavior of system components to produce desirable system behavior. These techniques all rely on some type of global knowledge that can be characterized as a synchronized view among a set of participants on the value of some shared state.

In decentralized systems, global knowledge of the system includes the local state and local environment of every component in the system. Components can coordinate their behavior to improve desirable properties of the system, if they are able to reach consensus (agreement) on the state of the system and its environment. A number of different consensus models have been developed that provide components with different levels of consistency on the state of the system and its environment at any instant in time. *Strict consensus* models, based on components viewing a total ordering of the updates to the system state, provide components with strong consistency guarantees on the state of the system, but they do not scale for a large number of participants due to the number of messages that need to be sent between them. For many applications, strict consensus requirements can be loosened and techniques such as virtual synchrony [Birman and Joseph 1987], based on causal ordering of system events, can be used, enabling systems to scale better. However, Van Renesse et al. [2003] state that: "traditional consensus protocols [...] have costs linear in system size [...]. With as few as a few hundred participants, such a solution would break down." More recently, *eventual consistency* models have become widely adopted [Tanenbaum and van Steen 2001], such as those based on system-wide gossiping [Jelasity et al. 2003], and gossiping within partitions in a statically partitioned system [Van Renesse et al. 2003].

The general trend in designing larger distributed systems has been to develop consensus protocols that reduce the number of messages that need to be delivered to all components in the system to synchronize on a shared view, but at the cost of lessening consistency guarantees. However, the aforementioned consistency models still provide a system-wide view and require that messages be eventually sent to all participants, even if those messages are not relevant to all of them. For dynamic environments, such as P2P networks and MANETs, two costs associated with this approach increase with system size: the time required to propagate a changed view to all participants and the frequency with which the view is updated. As systems in these environments scale, they have increased node churn rates [Rhea et al. 2004], link state changes, and more application data to process [Sen and Wong 2004]. Other decentralized techniques such as self-stabilizing algorithms [Dijkstra 1974] define global system states (correct, safe, and erroneous states) and components attempt to stabilize the system within a bounded number of steps, that is, eventually achieve consensus that the system is in a correct state [Gustavsson and Andler 2002]. Examples of self-stabilizing algorithms include routing algorithms [Gouda 2005], for example, Routing Information Protocol (RIP) and Open Shortest Path First (OSPF), but these algorithms break down in highly dynamic environments [Perkins 2001].

In effect, as systems increase in size and dynamism it becomes increasingly difficult for systems based on either eventual consistency or self-stabilizing algorithms to adapt and improve their operation to an unpredictable environment. However, in large-scale systems operating in dynamic environments, localized groups of components are still able to establish weak consensus on the state of a relatively small part of the system. Techniques for establishing localized consensus can reduce the amount of information propagated in the systems, compared to system-wide gossiping, without the need to statically partition or group components using global knowledge.

3. A REVIEW OF SELF-ORGANIZING MANET AND P2P SYSTEMS

We focus our review of self-organizing distributed systems on systems from MANET and P2P domains. MANET and P2P systems exhibit many common properties that justify describing them in one article. Common properties include a dynamic network environment with high levels of node churn, changing network link quality and variation in the amount of resources nodes contribute to the system. These properties have led to the development of techniques that allow decentralized coordination of entities and their continuous adaptation to a changing network environment.

The criteria we used to select systems for this review includes systems that: consist of autonomous interacting entities; have no use of global knowledge (allowing for the bootstrap problem in P2P systems); have a dynamic environment; and adapt to changes in their environment to improve some desirable system property. P2P systems that do not meet these criteria, but have stimulated interest in self-organization in distributed systems, include: systems based on relatively static structures that do not evolve as the environment changes,

such as Distributed Hash Tables (DHTs) [Stoica et al. 2001; Ratnasamy et al. 2001; Rowstron and Druschel 2001; Manku et al. 2003]; super-peer architectures based on manual or centralized super-peer election approaches [Yang and Garcia-Molina 2003; Zhao et al. 2002]; peer-to-peer multicast systems based on centralised multicast tree management algorithms [Padmanabhan and Sripanidkulchai 2002]; and hierarchically partitioned gossiping systems (such as Astrolabe [Van Renesse et al. 2003]). Many of the existing MANET routing protocols, which can be characterized as reactive, zone-based and cluster-based [Perkins 2001], do not meet our criteria. In the related area of sensor networks, existing systems that describe themselves as self-organizing incorporate centralized entities such as cluster-heads or sink nodes, and are therefore not considered here [Akyildiz et al. 2002].

While much of the discussion on self-organization and our later model is applicable to distributed systems, in general, we restrict our review to three systems in P2P and MANET domains that meet our criteria, as they consist of decentralized, autonomous entities that operate in a dynamic environment, adapt their operation to changes in their environment, and have emergent properties, such as localized consensus between entities. The reviewed systems are diverse as they operate in different network environments (MANET and P2P), use different algorithms and adapt to different properties of these environments. Other existing systems that meet our criteria, but are not reviewed, include the SG-1 self-organizing protocol for construction and maintenance of super-peer overlays [Montresor 2004], the T-MAN gossip-based protocol for the topology management [Jelasity and Babaoglu 2006] and the AntNet mobile-agent-based routing protocol [Di Caro and Dorigo 1998]. SG-1 and T-MAN are not reviewed as their self-organizing mechanisms aim at achieving system-wide eventual consistency, while we are more interested in localised consistency mechanisms. AntNet is similar to MANET routing protocols reviewed in this article.

3.1 Background on the Reviewed Systems

MANETs are a class of wireless network where mobile nodes interact and communicate with each other in an ad-hoc manner. MANETs lack any fixed infrastructure, and in routing protocols for MANETs all nodes have a routing agent that forwards packets over multiple hops, hopefully towards their destination. P2P systems refer to a general class of wide area networked systems that use distributed resources to provide some service in a purely decentralized manner.

The network environments of MANETs and P2P systems have different underlying capabilities that can be used to design self-organizing behavior. MANETs provide network broadcast functionality and promiscuous listening on the shared network medium, allowing the easy propagation of information between nearby hosts and the discovery of new nodes in a system. In contrast, P2P systems are typically built as IP overlay networks that define some communication abstraction over the IP network. IP overlay networks offer no broadcast or promiscuous listening capabilities, and centralized or well-known

name services are typically used to overcome the bootstrap problem. However, P2P systems enable an agent to directly communicate with any other agent, allowing systems to adapt their topologies arbitrarily. MANETs, in some ways, are a more challenging network environment as no single agent can be used globally to coordinate system behavior.

The reviewed systems use approximate *optimization algorithms* to solve optimal path problems (paths to destination nodes in the MANET routing and paths to files in the P2P system). The algorithms used are *approximate optimization algorithms* as they only attempt to provide a good enough solution to a problem in reasonable time, whereas *optimal algorithms* strive for convergence on a global optimal solution [Blum and Roli 2003]. However, the reviewed systems deal with problems not typically addressed by optimization algorithms, including: the concurrent solution to many different optimization problems, the loss of solution parts by agents leaving the system, the discovery of new solutions as agents join the system, and solution cost generation using measurements taken at the agents' local network environments.

We discuss the reviewed systems as examples of multi-agent systems, where an agent is an autonomous entity that is "situated in some environment, and that is capable of autonomous actions in this environment in order to meet its design objectives" [Wooldridge 2002; Jennings et al. 1998], following the Wooldridge and Jennings definition.

3.2 AntHocNet: A Self-Organizing Routing Protocol for MANETs Based on Ant Colony Optimization

AntHocNet [Di Caro et al. 2005] is a self-organizing routing protocol for MANETs. AntHocNet belongs to a group of network routing protocols based on the Ant Colony Optimisation (ACO) meta-heuristic [Dorigo and Di Caro 1999], which in turn was inspired by the observed behavior of ants foraging for food in ant colonies. Systems based on ACO are modeled as a finite set of nodes (components), connections between the nodes and a population of ants wandering between the nodes and laying a volatile substance called *pheromone* that evaporates over time. Ants wander randomly in the absence of pheromone trails at a component, whereas if pheromone trails are present on connections, ants preferentially traverse the connection with higher pheromone intensity. Ants can add pheromone to a connection as they traverse it, generally adding more pheromone if the path is shorter or of higher quality. This form of indirect communication between ants about the path quality is called *stigmergy*. When ants follow the simple behavior described above, the colony as a whole, with high probability, can solve the problem of finding the shortest path from a set of paths between a pair of nodes. This is achieved by many ants concurrently and randomly exploring their environment, and laying pheromone more frequently and in higher amounts on shorter paths, which in turn attracts more ants to those paths and creates a positive feedback loop of pheromone accumulation. The majority of initially random ants converge onto the shortest paths with the highest pheromone concentration. These paths, called *pheromone trails,* are emergent structures generated by ants exploring the environment and

indirectly interacting with each other. The pheromone trails evaporate over time, allowing the colony to explore new paths over time.

The ACO meta-heuristic was successfully applied to many routing algorithms where separate packets, called ant packets, are proactively sent into the network to continuously sample possible paths and update routing (pheromone) tables at nodes in the system. Ant-based routing protocols include ABC [Schoonderwoerd et al. 1996] and AntNet [Di Caro and Dorigo 1998] that were designed for wired networks, as well as the more recent AntHocNet for MANETs. Ant-based routing algorithms are suitable for MANETs due to their decentralized nature, high robustness to node failures, load balancing and adaptability to highly dynamic environments. However, the use of only proactive ant packets to discover optimal routes led to problems in previous work on applying ACO to MANET routing in PERA [Baras and Mehta 2003]. It has been shown that reactive protocols for MANETs, such as Ad Hoc On-Demand Distance Vector routing (AODV) [Perkins and Royer 1999], demonstrate better performance in MANETs, mainly due to the high network dynamism [Perkins 2001].

AntHocNet adapted the proactive nature of the ACO meta-heuristic to build a hybrid protocol, which combines reactive route set up with proactive route maintenance. Each node maintains a routing table, where for each known destination, an entry consists of a vector of real-valued elements, called pheromone values, with one element for each neighbor. A neighbor is defined as a node within wireless communication range. A pheromone value is an estimation of the quality of the route to the destination over a particular neighbor. Pheromone values are continuously updated according to the path quality values calculated by *reactive* and *proactive* ants. When a node wishes to send a data packet to a destination that is not in the local routing table, a reactive ant packet is sent to discover and set up an initial path to the destination. The reactive ant packet is broadcast, hence, all nodes within wireless broadcast range, that is, the node's neighbors, receive it and if their routing tables contain the destination, they forward the reactive ant packet to the next hop with a probability proportional to its pheromone value. If the destination is unknown again, a node rebroadcasts the reactive ant packet as before. This procedure is repeated until a reactive ant packet reaches the destination, whereupon it is converted to a backward ant packet that retraces its path to the source along the same path, updating pheromone values in routing tables. Routing tables are updated using a combination of the locally sensed size of the packet queue and the estimated time required to route a packet to the destination. Pheromone values are updated gradually, biasing their values towards the newly computed ones. If the backward ant cannot be delivered to the sending node, for instance due to node movements, the sending node reaches its maximum number of broadcasts and the maximum waiting time for the backward ant, assumes that the destination is unreachable, and discards all buffered packets to this destination.

Once the paths are set up by possibly different reactive ants that reached the destination, data packets can be routed to the destination (see Figure 1). Data packets are unicast probabilistically in a similar manner to the reactive ant packet, but with a lower probability of selecting suboptimal paths, that is,

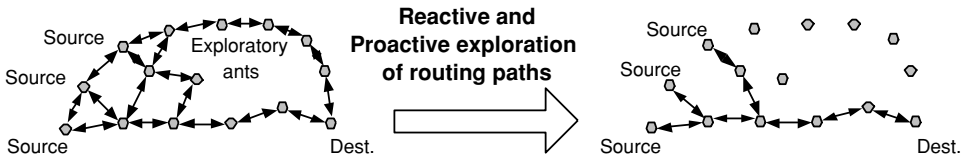


Fig. 1. Emergent "trails" (or paths) to a popular destination in AntHocNet.

every node that receives a data packet selects the next hop with a probability proportional to the next hop's pheromone value. When a data session has been established between nodes, the source node periodically sends out a proactive forward ant packet at a rate of one for every few data packets. The proactive forward ant is probabilistically unicast to the next hop, using the same formula as reactive forward ants, but there is also a small probability of a broadcast instead of a unicast in order to discover new nodes and paths in the network. As in the case of reactive ants, on the way back to the source node, proactive ants update pheromone values along the path with the quality values collected by forward ants. In addition to these proactive routing features, nodes periodically broadcast the pheromone values in their routing tables to all their neighbors. This enables the diffusion of pheromone values throughout the network, independent of ant activities. Furthermore, nodes use periodic broadcast to notify each other that they are alive.

In simulations based on Qualnet [Scalable Network Technologies, Inc. 2003], the AntHocNet algorithm has been shown to be superior to a state of the art MANET routing protocol, AODV, in terms of packet delivery ratio, average end-to-end delay and average jitter, whereas it is less efficient in terms of routing overhead since it requires many control messages to be sent. AntHocNet produces superior performance to AODV because feedback between different ant packets on route quality produces localized consensus between nodes, in the form of pheromone trails, on the best routes in the system, enabling nodes to share in the work of exploration the network environment for better routes. In contrast, AODV is a reactive protocol that does not learn route quality. The performance of AntHocNet demonstrates the potential for self-organizing approaches to building distributed systems in MANETs.

3.3 SAMPLE: A Self-Organizing Routing Protocol for MANETs Based on Collaborative Reinforcement Learning

Similar to AntHocNet, SAMPLE [Curran and Dowling 2005] is a self-organizing routing protocol for MANETs. Its design is based on Collaborative Reinforcement Learning (CRL) [Dowling et al. 2005], an extension to Reinforcement Learning (RL) [Sutton and Barto 1998] with support for online multi-agent learning. Collaborative Reinforcement Learning (CRL) is designed to coordinate the solution to discrete optimisation problems (DOPs) in a multi-agent system in order to optimize desirable system properties, such as system throughput or robustness. A CRL system is a decentralized system of partially connected RL agents, where any agent can potentially initiate a DOP that is solved by some (possibly different) agent in the system. Each agent uses a local model

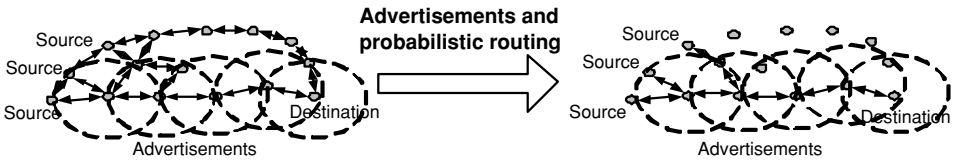


Fig. 2. Emergent stable routes to a popular destination in SAMPLE.

of both its environment and its neighbors to attempt to minimize the cost in solving a DOP. A DOP is solved by an agent either by executing an action to solve the DOP locally or, if the estimated cost is lower, an action to delegate the DOP's solution to a neighbor. Agents improve their local model using both *evaluative feedback* on the success of past actions to solve the DOP, and by acquiring *environmental feedback* from the agent's application, host and network environment, for example, using monitoring subsystems as found in autonomic systems [Dowling 2004]. Agents also provide one another with *collaborative feedback* about their estimated cost of solving different DOPs, enabling agents to coordinate the solution to DOPs and also optimize desirable system properties.

SAMPLE is based on CRL and for each different known destination agents model a routing decision as solving the DOP of finding the (neighboring) agent with the lowest estimated cost of routing a data packet to that destination. SAMPLE is a purely reactive protocol, in which routing tables with estimated costs for different destinations are updated when users provide data packets to be routed in the network. In contrast to AntHocNet, SAMPLE does not employ separate control packets for route discovery and maintenance but, instead, control data is attached to user data packets, thus eliminating the routing overhead introduced by proactive ant packets in AntHocNet.

In SAMPLE, data packets are routed using both the estimated route costs from neighbors (the next hop) to a destination and the estimated connection costs to those neighbors, where an agent's neighbors are the set of nodes within wireless communication range. Agents acquire estimated costs to destinations from their neighbors using *advertisement* of estimated route costs. The next hop for a packet is selected based on the sum of the advertised costs of each neighbor and the estimated connection cost of using a network link to that neighbor. Connection costs are calculated using the probability of a packet being successfully sent over the network link, based on a recent sample of packets sent over that link.

Agents advertise their routing costs reactively by piggy-backing estimated route costs to both the packet's source and destination nodes inside each data packet. Neighboring nodes receive all data packets sent over the shared wireless communication channel by promiscuously listening to all data packets, and use the advertised route costs to update and improve the quality of their local routing tables and connection cost models (see Figure 2). In order to remove stale routing information from the routing tables, SAMPLE uses *route decay*, where estimated costs in routing tables for non-advertised routes grow steadily higher over time, hence, gradually eliminating these routes from consideration

from routing decisions. This is similar to the idea of pheromone trail evaporation used in ACO [Dorigo and Di Caro 1999], and it means that agents adapt to recent routing traffic, thus preventing convergence on stale routes.

SAMPLE supports implicit exploration of the network for better routes by routing packets probabilistically. SAMPLE also includes a discovery action, implemented as a broadcast packet, to find new nodes and routes in the network. The discovery action is always executed when there is no routing entry available for a destination. However, as Boltzmann action selection [Sutton and Barto 1998] is used to select actions, discovery actions can also be executed, with low probability, during packet forwarding to attempt to discover new nodes, and possibly more optimal routes, for example, to adapt to network congestion by discovering uncongested routes.

SAMPLE has been specifically designed for a type of MANET scenario where Internet access is provided to mobile nodes in a metropolitan area. In this type of network, a majority of traffic terminates at the few servers in the network that provide the Internet access; there also are a group of stable, fixed nodes with stable routes to the servers. Experimental results of SAMPLE in the NS-2 network simulator [Breslau et al. 2000] showed that more popular traffic destinations have higher quality routes, since routes to these nodes are more explored and advertised. An emergent property of traffic flowing over the stable routes to the servers can be observed as more traffic flows from mobile nodes to the servers. This emergent property results from many agents using feedback from their local environment and feedback from neighbors to collectively adapt their routing behavior to favour stable network links. In the presence of stable links, high congestion and a high level of agent dynamism, SAMPLE has been shown to have significantly better network throughput and packet delivery performance than both AODV and Dynamic Source Routing (DSR) [Johnson et al. 2001] protocols [Curran and Dowling 2005; Dowling et al. 2005].

3.4 Freenet: A Self-Organizing Overlay P2P File Storage and Retrieval System

Freenet [Clarke et al. 2000] is a self-organizing P2P system that allows for publication, replication, and retrieval of data, while protecting the anonymity of both authors and readers. Freenet can be seen as a distributed storage application in which every peer contributes some storage space to the system, and which supports two operations: data insert and data retrieval. Deletion of data is implicit, that is, items that are not accessed for a period of time are removed automatically. Freenet is a P2P system where the global structure is not predetermined, but emerges in a bottom-up manner. Each peer in the system reactively updates its connections to neighbors using a local routing table adaptation algorithm. A peer's routing table contains addresses of neighboring peers and the data keys it believes they hold. Data keys are used to identify data items; these location-independent identifiers allow for the insertion, removal and discovery of data items in the system. In the current Freenet implementation [Clarke et al. 2000], keys are obtained by applying the 160-bit SHA-1 hash function to a data item. Every peer maintains a local, encrypted data repository where other peers can insert a data item indexed by a key. Peers know only

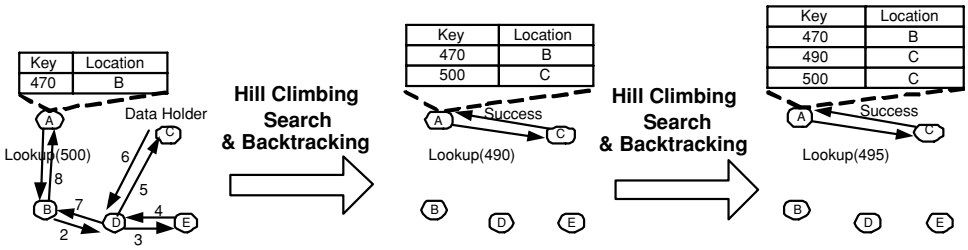


Fig. 3. Emergent key clustering in Freenet agents.

the identifiers of the locally stored data, which is sufficient for answering user requests.

Freenet's routing mechanism is especially important from the perspective of self-organization. User requests are routed from one peer to another using a *hill-climbing algorithm with backtracking* [Russell and Norvig 2003] to decide the location of the next hop. When a peer receives a query, it first checks its own repository, and if it finds matching data, sends it back along the request path. Otherwise, the peer forwards the request to the peer in its routing table with the closest key (determined by lexicographic distance) to the one requested, that is, hill-climbing search. If a peer sends a query to a recipient that is already on the request path, the message is bounced back and the peer tries to use the next-closest key instead. If a peer runs out of candidates to try, it reports failure back to its predecessor on the path, which then tries its next best choice, and so on, that is, backtracking. In order to limit resource usage, each request is given a *hops-to-live* limit that is decremented at each peer and when it reaches zero, the request fails. If a request is ultimately successful, it returns the data back to the upstream requester, where the data is cached locally, and a new entry is added in the requester's routing table associating the actual data source with the requested key. This way, subsequent requests for the same key will be immediately satisfied from the local cache, whereas requests for "similar" keys will be forwarded to the previously successful data source (see Figure 3).

The presented routing table adaptation mechanism leads to large improvements in routing performance over time. Peers use feedback from successful user requests to adapt their routing tables to specialise in handling clusters of similar keys (determined again by lexicographic distance) [Zhang et al. 2004]. Since each time a peer succeeds in handling a request, it is entered in another peer's routing table, this increases the probability that it will receive requests for keys that are similar to the key it handled. As the peer gains more experience in handling queries for those keys, it will successfully answer them more often and, in turn, get asked about them more often, in a kind of positive feedback loop. Similarly, peers' repositories will specialize in storing clusters of data items with similar keys. The creation of new entries in the routing table, after successfully answered requests, is the crucial aspect of the neighbor selection algorithm that leads to the emergent property of clustering of network connections to peers with popular data items. However, one problem with Freenet

is that multiple clusters can form in the network containing similar keys, leading to performance problems when the hill-climbing search algorithm causes a request to search clusters (hill tops) that do not contain the data item (as it may be in a different cluster containing similar keys), and ultimately time out. The emergent clusters are localised, not global structures. In general, there is a lack of published material on the performance of Freenet, mainly due to the difficulty in performing experiments due to the strong support for anonymity in Freenet. However, experiments that simulated Freenet's performance suggest that its performance is acceptable for highly replicated data, but poor for difficult-to-find data [Zhang et al. 2004].

4. AGENT MODEL

The review and analysis of AntHocNet, SAMPLE and Freenet show that these systems have many structural features and mechanisms in common. One essential component of these self-organizing distributed systems is the autonomous agent that cooperatively solves distributed problems.

In this article, the reviewed systems are considered multi-agent systems [Wooldridge and Jennings 1995; Wooldridge 2002; Jennings et al. 1998; Ferber 1999], as they meet general characteristics of multiagent systems [Sycara 1998]: agents have incomplete information or capabilities for solving the problem; there is no system global control; data is decentralized; and computation is asynchronous. The agents in the reviewed systems respond in a timely fashion to changes in their local environments, proactively and reactively exchange messages with one another, maintain models of their local environment, and take actions in order to satisfy both individual agent problem solving and system improvement goals. Of the three general classes of interactions in multiagent systems: cooperation, coordination, and negotiation; the agents in the reviewed systems demonstrate *cooperation* in routing packets and finding files and *coordination* in adapting and improving collective routing behavior in a dynamic environment. There is no *negotiation* between agents since our agents are not self-interested.

This section describes the environment in MANET and P2P systems and how agents maintain partial views of the system in order to improve performance and adapt to a changing environment.

4.1 Local Agent Environment

An agent in a distributed system executes in a *local environment* consisting of its local host, operating system, network, and local applications or users interacting with the agent. A local environment is defined as everything that is external to the agent and the system (other agents are not external to the system), that has both a direct impact on its operation and can be directly sensed or manipulated by the agent. Thus, from the agent's point of view, users and external applications are part of the agent's local environment (see Figure 4). An agent has interfaces to its local environment that determine the nature and scope of what the agent can sense and manipulate in its local environment. Agents may also be mobile, changing their local environment.

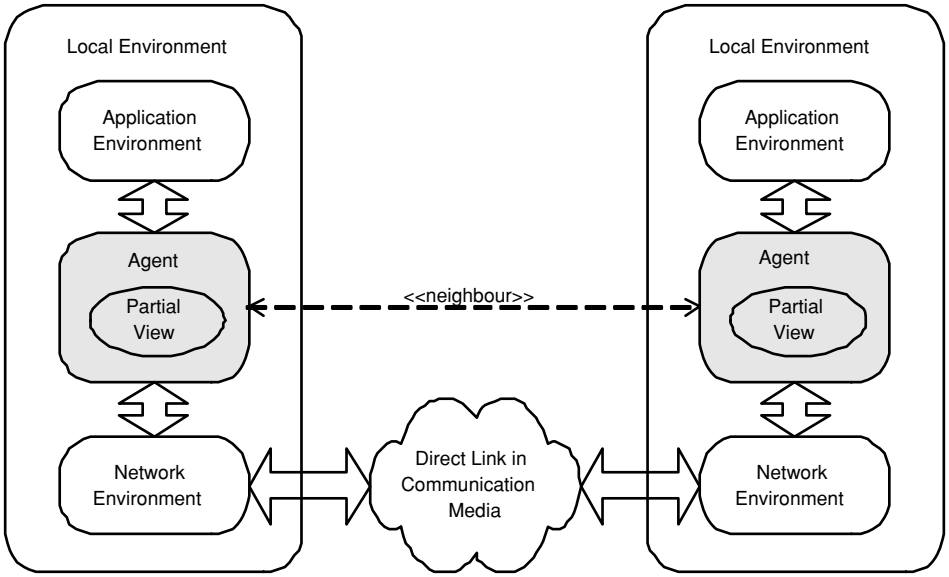


Fig. 4. Two neighboring agents and their local environments. The local environment of a single agent consists of applications that provide it with local data, a host and network infrastructure enabling message passing between agents.

Most environments in which self-organizing distributed systems operate are *inaccessible*, *nondeterministic*, and *dynamic*. In the particular case of MANET and P2P network environments, the state of network links is inaccessible in practice as network links must be tested to establish whether they are working (an expensive operation); routing actions are nondeterministic, having unpredictable outcomes; and the network state is dynamic, as external factors such as wireless interference and network congestion affect link quality.

Formally, every agent i in the system operates in a local environment, where the local environment's state at time t is denoted as

$$e_t^i \in E,$$

where E is the set of all possible local environment states, and $i \in N_t$, with N_t defining the set of agents in the system at time t. The total system environment $e_t \in \mathcal{E}$ comprises the union of the local environments (potentially overlapping) of all agents in the system at time t

$$e_t = \bigcup_{i \in N_t} e_t^i,$$

where $\mathcal{E}$ denotes the set of all possible system environment states.

The state of the agent's local environments is total with respect to the agent and independent of the state of the agent. The local agent environment can be modified by actions of the agent itself and actions of other agents operating in it. However, there is an uncertainty in the outcome of agent actions on the state of the environment due to a lack of knowledge of other agent actions as well as nondeterminism and dynamism of the environments.

4.2 Agent Partial View

In highly distributed systems, no agent possesses a global view of the entire system and its environment as this is not feasible in any system with high complexity (see Section 2). Similarly, an agent's view of its local environment can be limited and inaccurate due to inaccessibility, nondeterminism and dynamism. Consequently, an agent typically maintains a *partial view* that encompasses a model of the system and of its local environment. An agent's partial view is constructed using information received from other agents and information observed from its own local environment.

In order to solve distributed problems, a partial view should contain models for estimating the state of relevant, inaccessible parts of the system or its environment. These models allow agents to take actions based on the state of their local partial view, without the need to first communicate with other agents, which introduces scalability problems and is generally not feasible in real-time environments. However, if agents are to provide good solutions to distributed problems, they must coordinate their behavior and this can be achieved by the convergence of agents' partial views. Agents typically adapt and converge their partial views using feedback from their local environment and feedback from other agents. The adaptation and convergence of partial views allows agents to coordinate their action selection and to improve problem solutions. The agent's partial view includes:

(1) a neighborhood
(2) a model of the local environment
(3) partial knowledge from neighboring agents
(4) estimated knowledge over groups of agents or the whole system (that can be deduced from 1, 2 and 3) that we call localized system properties.

An *agent's internal state* consists of all components in the agent's partial view, that is, its set of neighbors (neighborhood), the model of its local environment, partial knowledge from neighboring agents and localized system properties. The internal state of agent i at time t is denoted as

$$s_t^i \in I, \tag{1}$$

where I is the set of all possible internal states of the agent $i \in N_t$. It should be noted here that while s_t^i includes an agent's model of its local environment at time t, e_t^i represents the actual state of the environment at time t. An agent's model of the local environment can deviate from the state of the local environment, and should be kept accurate by frequent feedback from the local environment.

The following subsections describe each element of the agent's partial view (internal state), and relate the reviewed systems to this model, which is summarized in Table I.

4.2.1 *Neighborhood.* In decentralized systems, an agent has a neighborhood, that we define as the set of agents with which it can communicate (i.e.,

Table I. Comparison of Agent Partial Views in AntHocNet, SAMPLE and Freenet

	AntHocNet	SAMPLE	Freenet
Neighborhood	agents in wireless communication range	agents in wireless communication range	most recently used agents
Local Environment Model	estimated packet transmission time over local links and packet queue size	estimated probability of successful packet transmission over local links	local data storage and estimated connection quality for each neighbor
Partial Knowledge from Neighbors	costs of routes to destinations calculated by neighbors	costs of routes to destinations calculated by neighbors	keys that each neighbor can locate
Localized System Properties	estimated best next hop for destinations	estimated best next hop for destinations	estimated best next hop for finding files

send messages) directly at a particular moment in time. An agent may be aware of other agents outside its neighborhood, for example, an agent may know the address of some remote agent and communicate with it through other agents in a multi-hop manner; but if it cannot directly communicate with them, they do not belong to its neighborhood. The neighborhood is dynamic with respect to its size and membership, and it must be managed and updated over time. In systems based solely on stigmergy, such as AntNet [Di Caro and Dorigo 1998], where communication between agents is indirect and mediated by the environment, an agent's neighborhood is considered to be empty, by our definition. In AntHocNet, however, we consider the routing programs on the hosts to be agents and the reactive and proactive ants to be messages. AntHocNet's routing programs are autonomous programs that actively maintain a view of their neighboring nodes.

The agent, in order to interact with other agents and hence participate in the system, needs to initialise its own neighborhood by discovering other agents in the system and informing them about its existence. This process is often called *bootstrapping* or *discovery* [Milojicic et al. 2002; Sycara et al. 2003]. Once the agent discovers a connected agent in the system, it can find other agents and build its own neighborhood. However, as systems grow in size, agents are not able to maintain and constantly update information about all potential neighboring agents. Agents often refine their neighborhood to some limited number of agents by evaluating the relative "fitness" of their neighbors and potential neighbors. The agent's neighborhood needs to be also continuously updated, by agents removing neighbors that leave the system. Agents leaving the system might notify neighbors before they leave, but a notification mechanism is not sufficient in the case of arbitrary failures. The only indication to other agents of such a failure is absence of any activity at the failed agent.

In AntHocNet, the agent's neighborhood consists of all agents in the agent's wireless broadcast range. These are the only agents that it can directly communicate with. It can reach other agents only in a multi-hop manner. However, the agent may not know about all its direct neighbors. The agent's knowledge about

neighbors is reflected by its routing table, where each entry corresponds to a direct neighbor. Wireless broadcast and promiscuous listening enable agents to learn about neighbours and bootstrap their routing tables. The routing table entries need to be continuously updated by inserting neighbors that join the system and removing neighbors that fail or leave the system. For failure detection, AntHocNet uses a common *heartbeat mechanism* [Huhns et al. 2002], where hello messages are periodically sent to neighbors to determine their availability. Unresponsive neighbors are removed from routing tables.

Similarly in SAMPLE, the agent's neighborhood consists of all agents within the agent's wireless broadcast range, and it is maintained in the agent's routing table. Again, wireless broadcast and promiscuous listening are used to bootstrap an agent's routing tables. In contrast to AntHocNet, SAMPLE uses a *decay* mechanism for agent failure detection, where the routing table entries (corresponding to a neighbor) are degraded over time, in the absence of receiving messages from neighbors. If a routing table entry corresponding to a neighbor reaches some "staleness" threshold, the neighbor is removed from the agent's routing table.

In Freenet, the agent's neighborhood consists of a limited number of agents that have been most recently used for routing. The Freenet protocol does not specify a bootstrap mechanism, but Freenet implementations support centralised seed nodes that can be initially added as neighbors. Once an agent makes an initial connection and issues requests for data, it learns about other agents that successfully answer requests and inserts them in its neighborhood. A maximum neighborhood size prevents unbounded neighborhood growth, set at 250 neighbors in simulation [Clarke et al. 2002], and when the agent completes a subsequent user request, the Least Recently Used (LRU) entry in the routing table (i.e., a neighbor) is removed to make way for the new entry. This way, assuming enough user requests are satisfied by neighbors over time, inactive neighbors are eventually removed from the neighborhood.

In other systems, agents sometimes attempt to optimize their set of neighbors using some evaluation function that measures the relative "fitness" of their neighbours and potential neighbors. Distributed Hash Table P2P systems select neighbors based on their unique identifiers [Stoica et al. 2001; Rowstron and Druschel 2001; Rhea et al. 2004] and additionally refine them using some proximity metrics such as latency. In super-peer networks, agents adapt their neighbors based on agent resources and capabilities (e.g., high bandwidth, low latency, large storage space or high uptime) [Yang and Garcia-Molina 2003]. In multi-agent systems, where agents may have different functionality, agents select neighbors based on their capabilities to solving required tasks [Decker et al. 1997].

4.2.2 *Local Environment Model.* In dynamic network environments, observing the state of local network links is an expensive operation. The reviewed MANET systems overcome this problem by maintaining a model of their network links, that they use to make decisions about the probability of transmission over a link succeeding, see Table I. Similarly, aspects of the agent's application and host environments can be modeled to help agents

estimate their state. Local environment models are stored in the agent's partial view.

Local environment models are typically estimators for aspects of the environment that are relevant to the agent for distributed problem solving or for improving behavior. As agents often take decisions based on the state of the local models, instead of the state of the actual environment, the models need to be continuously updated by gathering information from the local environment. However, local models are not sufficient for agents to take actions that improve system performance. For instance, a routing agent could select a locally optimal action of forwarding a network packet to a neighbor that has the lowest latency, but this locally optimal decision may not be globally optimal if the selected neighbor has only poor quality paths to the destination. Thus, the agent needs to learn about remote regions of the environment through the exchange of partial views with other agents.

In AntHocNet, an agent builds a local model of its network environment by monitoring the size of its local queue of packets that are to be sent at the MAC layer, and by measuring the average time between the arrival of a packet at the MAC layer and the time needed for successful transmission. This information, together with estimated route costs to destinations received from neighbors, is used by the agent to estimate the total cost of routing to known destinations.

In SAMPLE, an agent builds a local model of its network environment by storing a sliding window of the observed number of successful transmissions to failed transmissions to each of its neighbors, at the MAC layer. The sliding window, for each neighbor, is used to estimate the probability of a successful packet transmission over the link to its neighbor. This model is then combined with advertised route costs to destination to estimate the total route cost of delivering a packet to a destination.

Freenet's representation of the environment contains information about the locally available data items and their keys as well as the reachability of the neighboring hosts. The information about the locally available data items is propagated to neighboring agents, enabling them to locate these data items. Recently, in other P2P systems, estimated models of network links have been used to demonstrate improved system performance for DHT-based systems [Rhea et al. 2004].

4.2.3 *Partial Knowledge from Neighbors.* Agents that solve distributed problems need to acquire knowledge about remote regions of the system (beside their own local knowledge) from agents that explore these regions. The problems that agents solve are known as tasks, and are defined by some specification [Wooldridge 2000]. Agents cooperate in task solution and each agent typically has a model of the estimated costs of solving tasks by each neighbor. Agents use local models to make decisions about task delegation to a neighbor, since the cost of communicating with neighbors before deciding on the best neighbor for a task is often prohibitive, especially for increasing numbers of neighbors. The first version of Gnutella is a well-known example of how flooding neighbors with queries, without using a knowledge from other agents, can severely impact system scalability [Ripeanu et al. 2002].

In self-organizing distributed systems, information contained in partial views is shared among agents to promote convergence between agent partial views. There are different methods for sharing knowledge, but the basic design choice is between proactive and reactive mechanisms [Wooldridge 2002]. Proactive knowledge exchange can be initiated by agents at any time, typically periodically, whereas reactive knowledge exchange is only triggered when some external source (such as the agent's environment) introduces new information to the system. The choice of proactive and/or reactive mechanisms should take into consideration the expected rate of interactions between agents due to the external environment.

The sharing of knowledge introduces a trust problem into the system, and the model of self-organization presented in this article assumes that agents interactions are trusted and cooperative, as is the case in the reviewed systems. Agents are trusted if they behave according to the system's rules and do not try to disrupt the correct functioning of the system. Agents cooperate in that they contribute their own resources (e.g., bandwidth, storage space) in order to increase the system utility, rather than maximise their individual utility. The issue of support for trust models is considered outside the scope of this article and has been addressed elsewhere [Axelrod 1997; Rapoport and Chammah 1965; Cahill et al. 2003; Cohen 2003].

In AntHocNet, the knowledge that neighbors exchange consists of the estimated costs of routing to selected destinations. AntHocNet uses both reactive and proactive mechanisms for sharing this knowledge, including reactive ants to setup routing paths, and proactive ants to periodically sample routing paths during data traffic.

Similarly, in SAMPLE, neighbors exchange estimated costs of routes to destinations. However, only reactive mechanisms for exchanging routing knowledge are used during both route setup and normal routing operation. Routing cost and node availability information is piggybacked in routing packets, which neighboring agents promiscuously receive and use to adapt their routing tables.

In Freenet, the knowledge that agents exchange is the agents ability to locate particular keys. A reactive mechanism for knowledge exchange is used; when a data item is successfully located, agents along the request path update their local sets of keys to reflect that they can now locate that item.

Other mechanisms seen in different multi-agent systems for propagating and adapting partial views of agents include proactive gossiping protocols in Newscast [Jelasity et al. 2003] and Astrolabe [Van Renesse et al. 2003], reactive event-based notification in Chord [Stoica et al. 2001] and Pastry [Rowstron and Druschel 2001], stigmergy mechanisms in AntNet [Di Caro and Dorigo 1998], and knowledge and information exchange languages for MAS [Finin et al. 1994; FIPA 2002].

4.2.4 *Localized System Properties.* From the combination of local environment models and knowledge received from neighboring agents, an agent can reason about localized properties of the system, such as a next hop on an optimal routing path to a desired destination. These estimated properties enable

coordinated agent actions that are globally near-optimal and result in the system behaving as a coherent whole.

In AntHocNet and SAMPLE, routing costs are aggregated over the set of agents in a routing path, and thus make up a property of the system that is localized to a group of agents. Agents use the search algorithms ACO and CRL, respectively, to make routing decisions to destinations in remote parts of the system. These search algorithms can generate good global decisions by basing their decisions on both local environment models and the estimated cost of routing via a neighbor to a known destination.

In Freenet, when an agent needs to locate a data item that is not available locally, it selects a neighbor that specialises in locating the closest key. As agents that provide similar keys tend to cluster, the local knowledge of key clustering helps solve the global problem of finding good paths to a destination.

5. SYSTEM MODEL

One of the main goals when engineering self-organizing systems is the improvement of *desired system properties*, given the state of the system and its environment [Collier and Taylor 2004]. Many different properties of distributed systems can be recast as problems that can be quantified and subsequently improved or optimised. Examples in the reviewed systems include maximising routing performance, minimising packet loss, and maximizing the clustering of similar data items. Related concepts for evaluating the performance of multi-agent and distributed systems include Wooldridge's system utility [Wooldridge 2000, 2002], Wolpert's world utility that is used to rate the collective behavior of agents in his Collective Intelligence (COIN) model [Wolpert and Tumer 1999], and Babaoglu's figure of merit (FOM) that is used to rate the sensitivity of the system to a dynamic environment [Babaoglu et al. 2006].

In this section, we introduce a formal model of self-organization in distributed systems as the adaptation of the system to improve desired system properties in the current system environment. No implementation or formal validation of the model is provided, but we show how the model is realized in the reviewed systems. Our approach is based on models proposed for multi-agent systems [Wooldridge 2000, 2002; Genesereth and Nilsson 1987]. Our model should be useful in aiding system designers understand how techniques such as feedback models and evaluation functions can be applied to build self-organizing distributed systems with desirable system properties.

5.1 System State

The set of all agents' internal states comprises the *system state*. The system state at time t is denoted as s_t,

$$s_t = \left(s_t^i\right)_{i \in N_t} \in \mathcal{S},$$

where N_t is the set of agents in the system at time t, $s_t^i \in I$ are the internal states of each agent, as defined in Formula 1, and $\mathcal{S}$ is a set of all possible system states.

Similar to Wooldridge's multi-agent model [Wooldridge 2002], we can model *system behavior* in a given environment as a sequence of pairs of the system state and the total environment state (i.e., sequence over $\mathcal{S} \times \mathcal{E}$),

$$(s_0, e_0) \rightarrow (s_1, e_1) \rightarrow (s_2, e_2) \rightarrow (s_3, e_3) \rightarrow \cdots$$

Each system state transition is caused by transitions of individual agent internal states, which are, in turn, caused by agent action execution, receiving feedback from neighboring agents and the local environment, and decay mechanisms. We introduce all these concepts in the rest of Section 5. The transitions of the environment state are caused by agents modifying their local environments and dynamism of the environment, see Section 4.1.

5.2 Problem Statement: Improving System Utility in a Dynamic Environment

Desired system properties in a multi-agent system can be optimized if they can be recast as approximate optimization problems that can be subsequently maximized or minimised. Formally, we introduce an abstract *system utility* function that measures the quality of some desired system property for a particular state of a system and the system's environment

$$util : \mathcal{S} \times \mathcal{E} \longmapsto \mathbb{R}.$$

We can also say that the utility function measures how well the system state is matched to the environment (where higher values are preferable). self-organizing systems should be engineered to adapt towards states that increase the system utility. We define the set of optimal system states for a given environment e as

$$\mathcal{S}^\star(e) = \{s \in \mathcal{S} : \ util\ (s, e) = \max_{s' \in \mathcal{S}} util(s', e)\}.$$

Hence, at time t, the set of optimal system states is $\mathcal{S}_t^\star = \mathcal{S}^\star(e_t)$. Given system behavior as a transition of system and environment states, we say that the system state converges toward the optimal states if

$$\lim_{t \to \infty} \left(util\ (s_t, e_t) - \max_{s' \in \mathcal{S}} util(s', e_t)\right) = 0.$$

In order for agents to adapt to an optimal state, they require a relatively stable environment; an unrealistic assumption in MANET and P2P networks. For systems in dynamic environments, their goal is to reach good enough, near-optimal states in reasonable time. Systems have to trade off their speed of adaptivity for their ability to find near-optimal states.

The presented utility function gives some high-level intuition in how a self-organizing system behaves, however, it does not provide any insights on how to realise this behavior. This is because agents have only limited view on the system and do not have access to the system utility function. In the reviewed systems, the utility function could measure the quality of routing tables, that is, how precisely routing tables reflect the current environmental conditions. In optimal system states, $S_t^\star$, the routing tables should enable all agents to select routing paths for packets (or requests) such that they optimize some

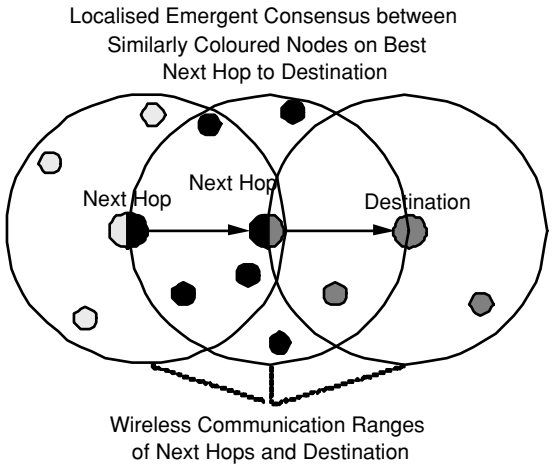


Fig. 5. In SAMPLE and AntHocNet subgroups of agents can establish consensus on the best Next Hop to a Destination. Here, identically colored agents agree on the best next hop, while the larger agents with two colors are agents on the "best" path to a destination.

routing metric, such as maximizing network throughput or packet delivery ratio (Figure 5).

The challenge in building systems that improve system utility in a dynamic environment is to coordinate agent actions that are based only on the agents' local states and to adapt agent states to a changing environment. This is difficult for the following reasons:

—uncertainty in the actions and state of other agents in the system
—uncertainty in the level of system utility
—uncertainty in the state of the environment
—uncertainty in the effect of actions on the environment.

The first three uncertainties result directly from the agent's lack of global system knowledge. The last two uncertainties are caused by inaccessibility, dynamism and nondeterminism of the environment.

In the following sections, we show a model for how agents in self-organizing systems select their local actions based on local state, and how they use feedback and decay to adapt their local state to their environment and neighbors. We then discuss, using our model, how systems can increase their utility in a dynamic environment.

5.3 Localized Consensus

Some form of consensus on the state of the system and the environment, between a subset or all of the system's agents, is required in order to coordinate agent behavior, and, consequently, improve desired system properties [Parunak et al. 2005]. Agents proactively or reactively exchange selected parts of their state, promoting convergence between neighboring agents on their view of the system, thus enabling a form of consensus to emerge. Agents share the

minimal, but complete, state necessary to cooperate and solve their distributed problems, for example, routing agents only share best estimated routing distances to destinations rather than all estimated distances using all of their neighbors.

The model of consensus we are interested in is not system-wide, but localized to groups of agents in the system, and there is no management of group membership. Agents can adapt their state to become closer to one another using feedback mechanisms, introduced later in Section 5.5. We start defining consensus between a set of agents by first introducing an abstract *distance metric* that measures the difference between internal states of two agents:

$$dist : I \times I \longmapsto \mathbb{R}.$$

A system is said to have reached *strict global consensus* at time t when the distance between any two agent states in the system is zero:

$$\forall_{i,j \in N_t}\, dist\left(s_t^i, s_t^j\right) = 0,$$

where N_t is the set of all agents in the system at time t. However, this form of consensus is not possible as it would require that all agents possess synchronized global knowledge of the system. Thus, agents only strive to achieve *localized* forms of consensus between smaller groups of agents $N_t' \subset N_t$. This consensus is also *weak,* as agent states are not necessarily strictly consistent; their distance is limited by some constant, ε:

$$\forall_{i,j \in N_t'}\, dist\left(s_t^i, s_t^j\right) < \varepsilon.$$

The consensus between localized groups of agents in self-organizing systems is often *eventual* (see eventual consistency models in Section 2), meaning that in a stable environment agents' internal states converge over time:

$$\lim_{t \to \infty} \max_{i,j \in N_t'} dist\left(s_t^i, s_t^j\right) = 0.$$

In the reviewed systems, a form of localized weak consensus emerges between the agents. However, due to the dynamism of the environment and nonlinear adaptations produced by interagent feedback, see Section 5.5.1, it is mathematically intractable to derive all the values of $i, j \in N_t'$ and ε using an analytical approach. Nevertheless, given a reasonably accurate model of the system's environment, these values can be observed empirically through experimentation. It can be shown that statistically, within a given confidence interval, that agents maintain localized weak consensus on some state of the system.

In AntHocNet, proactive ants improve consensus between agents that lie on their paths between the source and destination. Since ants are forwarded with higher probability to agents situated on the high-quality paths, more effort is dedicated to improve consensus on high-quality paths, which are also more likely to be exploited for routing. Furthermore, routing information is gradually diffused between agents by periodic broadcasts. Through these mechanisms, localized groups of routing agents, delimited by their physical proximity, can establish weak consensus on the best quality routes to popular destinations in the network, given a stable network environment.

In SAMPLE, localized weak consensus between agents is achieved through advertisements and promiscuous listening. When a data packet is sent by an agent, all neighboring agents (i.e., agents within the wireless communication range) receive information about the cost of the path from that agent to the packet's original source and destination (see Figure 5). Furthermore, routing costs are advertised through broadcasts. Advertisements can propagate over multiple hops in the system when neighbors adapt to advertisements and re-advertise their new paths. Since popular destinations are advertised more often, better consensus emerges on the cost of paths leading to them, and consequently lower cost paths are discovered to popular destinations.

In Freenet, the key clustering mechanisms group together agents specialising in locating and storing data items associated with similar keys. Agents located within the same clusters have similar routing tables and store similar keys, and hence, are close to each other in terms of the *dist* metric and reach a localized weak consensus. The consensus can be seen to emerge between agent routing tables on the best agents to use for different keys. For each successful request, agents on the request path improve their consensus on how to locate the requested key. These agents also improve consensus on the location of other data items that have similar keys since agents specialize in routing to clusters of similar keys.

5.4 Action Selection

Agents need some model for selecting actions that both solves agent-level tasks and strives to improve system utility in a given environment. Individual agent behavior can be abstractly defined by three functions that describe how agents evaluate their available actions in the current state, select an action, and execute the selected action. First, we introduce an *eval* function that is used by agents to estimate the utility of the possible actions, given the current agent state

$$eval : Ac \times I \longmapsto \mathbb{R},$$

where Ac is the set of all actions available at the agent. In systems where agents share a common goal, as in the systems reviewed in this article, the evaluation function should be designed in such a way that action utility calculated by *eval* corresponds to the action's estimated effect on system utility. This is a challenging problem that we return to in Section 5.6.

Subsequently, an action is selected based on the output of the evaluation function. Since agents only have a partial view of the system, estimations of action utility may be inaccurate, as the local state may not accurately model the current system and environment state. Thus, an agent needs to trade-off the exploitation of the knowledge of its current state (executing locally higher utility actions) with exploration for new states (executing lower utility actions). Formally, we define an action $\alpha \in Ac$ executed by an agent i at time t as *exploratory* if:

$$eval\left(\alpha, s_t^i\right) < \max_{\alpha' \in Ac} eval\left(\alpha', s_t^i\right).$$

Otherwise, the action selected *exploits* knowledge in the current local state if

$$eval\left(\alpha, s_t^i\right) = \max_{\alpha' \in Ac} eval\left(\alpha', s_t^i\right).$$

This leads us to define a probabilistic model for action selection that allows for the selection of both exploitative and exploratory actions as

$$select : I \longmapsto \mathcal{P}_{Ac}$$

where $\mathcal{P}_{Ac}$ is the set of all probability distributions over the set of actions. Thus, the *select* function returns a discrete probability distribution $P_{Ac} \in \mathcal{P}_{Ac}$, such that $\sum_{\alpha \in Ac} P_{Ac}(\alpha \mid s_t^i) = 1$. Typically, such a probability distribution is designed so there is a higher probability that actions with higher estimated utility are selected, over actions with lower estimated utility. An action selected from the probability distribution is executed by the agent, where an execute function is defined as

$$execute : I \times E \times Ac \longmapsto I. \tag{2}$$

The *execute* function captures how an action may involve sensing the agent's local environment, and how it results in a new agent state.

In the reviewed systems, AntHocNet agents evaluate actions, including data routing and sending a reactive or proactive ant, based on the availability of routes and estimated route costs for neighboring agents. If routes are available to a destination, routing actions for data traffic are selected using a probabilistic model that is tuned in experiments to favors better routes. If no route is available, an action to send a reactive forward ant is selected that uses a probabilistic model to discover a route that is tuned to favor exploration. Actions to send proactive ants for route maintenance are triggered after n data routing actions.

SAMPLE uses CRL to evaluate agent actions based on the local environment model, the availability of routes and the estimated route costs for neighboring agents. If routes are available to a destination, SAMPLE selects routing actions using a probabilistic policy called Boltzmann action selection. If no route is available, a broadcast action is selected that floods the network to discover a route.

Freenet evaluates actions based on the distance of the required key to each neighbors' advertised keys. In Freenet, actions are limited to unicast routing. Freenet uses a deterministic, hill climbing search algorithm when handling user requests for keys. Initially, agents have no information about keys associated with the neighboring agents, thus routing involves much exploration that gradually reduces as agents specialize in locating clusters of similar keys.

5.5 Agent Adaptation

Agent adaptation involves the updating of agents' internal states, and consequently their behavior, as a result of action execution (Formula 2), receiving feedback from neighboring agents and the local environment, and decay mechanisms. Agents encode information from local and remote parts of the system locally in their state or in the system topology by reconfiguring their neighborhoods. Adaptation can help reduce the uncertainties that prevent agents

from selecting actions that improve system utility. We describe here how agents adapt their behavior using two mechanisms: feedback and decay.

5.5.1 *Feedback.* Feedback is a form of communication of information between agents in a self-organizing system about the state of a part of the system or its environment. It is a mechanism that enables groups of agents to reach some form of consensus on their state and on their environment.

Two possible sources of feedback for an agent are its local environment and its neighbors. Agents use feedback from their local environments and neighbors to update their local state. Formally, we can define feedback from an agent j to an agent i produced at time t as a feedback message, $f \in \mathcal{F}$, that contains a part of agent j's state

$$f \subseteq s_t^j \in I.$$

Feedback messages are generated by agent actions

$$feedback : I \times E \times Ac \longmapsto \mathcal{F}$$

where $\mathcal{F}$ is the set of all possible feedback messages, I is the set of all possible agent states, E is the set of all possible local agent environment states and Ac is the set of all possible agent actions. A feedback message for an agent can also be produced by the agent's local environment. It is then represented as a subset of the state of the local environment in which agent i operates, $f \subseteq e_t^i \in E$. Agents that receive a feedback message from the local environment or neighboring agents, use it to adapt their local state that we formally represent by an *adapt* function:

$$adapt : I \times \mathcal{F} \longmapsto I.$$

Feedback is the main mechanism used for the adaptation of agent behavior; by adapting an agent's state, feedback can modify an agent's action selection policy. *Environmental feedback* enables an agent to learn about the state of its local environment, and agents that share the same local environment (such as agents deployed on the same host) can share the same local environment model. *Interagent feedback* (feedback between agents) is generated by the proactive or reactive sharing of selected agent state and enables the convergence of agents' state. Agents should not share their entire partial view, but only information relevant to neighbors, for example, information common to problems they are solving.

Two variants of feedback are possible in dynamical systems: *positive* and *negative feedback* that, respectively, amplify or decrease the selection of actions by agents. Formally, we say that positive feedback $f \in \mathcal{F}$ amplifies the use of action $\alpha \in Ac$ by agent i if the updated agent state, s_t^i, improves the utility of that action:

$$\frac{eval\left(\alpha, adapt\left(s_t^i, f\right)\right)}{\sum_{\alpha' \in Ac} eval\left(\alpha', adapt\left(s_t^i, f\right)\right)} > \frac{eval\left(\alpha, s_t^i\right)}{\sum_{\alpha' \in Ac} eval\left(\alpha', s_t^i\right)}.$$

Similarly, negative feedback can be defined for an action as one that decreases the probability of an action being selected. Positive feedback loops can form at

the system-level, where some agent adapts its local state and produces feedback to neighboring agents, which in turn adapt themselves, affecting their neighbors, often resulting in system-wide adaptations.

Interagent feedback, whether positive or negative, improves consensus between agents. When agent j sends a feedback message, f, to agent i and agent i adapts its local state, the distance between the internal states of agents i and j decreases

$$dist\left(adapt\left(s_t^i, f\right), s_t^j\right) < dist\left(s_t^i, s_t^j\right).$$

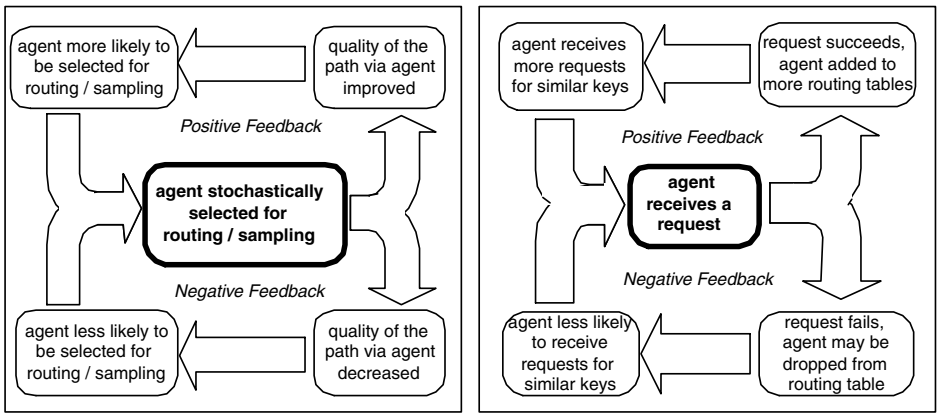
Similarly, when agent i adapts its state s_i to the feedback received from the environment, s_i becomes more consistent with e_i (which can be formalized by introducing appropriate metrics).

In the reviewed MANET systems, interagent feedback consists of estimated route costs for destinations. Feedback is considered positive or negative, according to whether it increases or decreases the cost of a particular routing path in the agent's routing table. Positive feedback processes cause more data packets to be attracted to a path, resulting in more feedback being generated for that path. Negative feedback, on the other hand, decreases the attractiveness of the routing path and consequently reduces the amount of feedback generated for the path (see Figure 6(a)). Only local adaptations that change an agent's best estimated route cost to a destination are further propagated to neighbors. As such, cascading updates in a system are often the result of positive feedback processes where new better paths are discovered. The presence of positive and negative feedback loops leads to nonlinear system behavior, where small causes can have large effects, making the systems less amenable to formal analysis. Nonlinear interactions between agents can be observed in SAMPLE and AntHocNet when an agent on a stable path or trail to a popular destination unexpectedly fails. This relatively small event (from a system perspective) can trigger a feedback process whereby the routing policies of a large number of agents are adapted until agents converge on a new route to the popular destination.

In AntHocNet, interagent feedback about estimated route costs is carried by reactive and proactive ants. The local network environment provides feedback to agents about the size of local packet queues. The implementation of the ACO algorithm uses both forms of feedback to adapt routing table entries.

In SAMPLE, interagent feedback is provided by unicast and multicast packets that carry route cost advertisements and that agents receive by promiscuously listening. The local network environment provides feedback about the success or failure of packet transmissions. The CRL algorithm uses inter-agent and environmental feedback to adapt the routing table entries and the local network link models, respectively.

In Freenet, feedback is supplied to agents when they handle a user request. Agents receive feedback on whether the request has been successfully handled. If it has, the agent associates the requested key with the agent that fulfilled the request in its routing table. This adaptation of agent routing tables causes more requests for similar keys to be sent to this agent, triggering a positive feedback loop (see Figure 6(b)). The positive feedback loop causes agents to specialize in



(a) Feedback in SAMPLE and AntHocNet (b) Feedback in Freenet

Fig. 6. Feedback in the reviewed systems.

both routing for clusters of similar keys and, as agents cache each reply locally, storing clusters of data with similar keys. However, Freenet's performance suffers due to excessive positive feedback, where agents over-specialise in clusters of keys. Excessive clustering adversely affects the efficiency of Freenet's request routing performance for requests that are not in the agent's local neighborhood [Zhang et al. 2004]. Zhang improved Freenet's neighbor reconfiguration algorithm by allowing for probabilistic selection of long-range random links. Random links act as negative feedback on excessive clustering. Zhang showed how this negative feedback can help stabilise an agent's routing table and improve system performance for request searching.

5.5.2 *Decay.* One of the limitations of the feedback model is that when agents leave the system, their state may be still included in neighboring agents' partial views. Failed agents cannot produce feedback messages to inform their neighbors of their departure from the system. A similar problem can be found when an agent stops receiving feedback from the local environment; this may lead to divergence between the state of the agent's local environment model and its real environment. To handle these cases, agents can decay their partial view over time. Decay models are useful in ensuring that agents' partial views do not diverge too much from the actual state of the system's environment, as they require a constant flow of feedback from the environment and other agents to maintain stable agent states. Decay can be described by a function that each agent applies to its partial view:

$$decay : I \longmapsto I.$$

In AntHocNet, pheromone values in route tables decay over time in a process known as evaporation. In the absence of routing traffic, route costs to neighbors are gradually increased. However, routes are only removed when a neighbor's availability changes, and this is monitored by proactive hello messages.

In SAMPLE, decay is a form of time-driven, single-agent negative feedback, that is, it does not cascade to other agents or produce feedback loops. Agents decay the estimated route costs in the routing tables at every discrete time unit, that is, they increase the cost of routing actions. In the absence of routing packets from the application environment, the stable paths that are formed through positive feedback processes gradually degrade in quality until the routing actions for those stable paths are removed from routing table entries.

In contrast, Freenet's decay model is not based on updating agent state at discrete time steps. Routing tables are decayed when user requests are fulfilled; the decay model causes the LRU entry in its routing table to be removed. However, as decay is not based on elapsed time, the absence of user requests can lead to stale routing tables that can contain entries for agents that have left the system.

5.6 Discussion

The presented model describes the self-organizing behavior of MANET and P2P systems, as improving system utility through agents using local information to adapt to a dynamic network environment. A summary of the model is presented in Figure 7. It explains how agent-level mechanisms such as inter-agent feedback, environmental feedback, decay and agent adaptation enable agents to coordinate their behavior to improve system utility. Agents in our model use only information from their local state, without reference to any global knowledge, to improve system utility.

We believe that our model is also applicable to understanding and building self-organizing multi-agent systems, in general. It allows us to ask questions such as: given a group of agents, an environment representation, and a problem to solve, how do we construct:

—feedback and decay functions to allow an agent maintain an accurate model of its local environment
—feedback and decay functions to allow an agent to propagate relevant state changes to relevant neighbors
—evaluation functions that enable agents to select globally good actions based only on their local states, improve a system's utility, and effectively solve a distributed problem.

The main challenges in building self-organizing distributed systems using our model are the design of: environmental feedback models that match an agent's internal models to real environment; inter-agent feedback models that promote localized consensus between agents; and action evaluation functions that improve system utility in a stable environment. For local environment feedback, information gathered from the agent's local environment must somehow be transformed into an efficient representation, stored in the agent's partial view. For interagent feedback models, distributed problems must somehow be represented in a form where less than strict consensus between partial views is acceptable for applications.

Properties

—*Decentralization*: The lack of a single entity with complete, system-wide knowledge about the state of the system and the environment, or an entity that controls and coordinates the behavior of the entire system. The system consists of interacting, autonomous agents with partial views of the system.

—*Localized consensus*: A property of decentralized systems where localized groups of agents maintain weakly consistent views on the state of the system and their local environments. Agents establish localized consensus in order to coordinate their actions. Strict global consensus cannot be maintained in a decentralized system of any reasonable complexity.

—*Utility optimisation*: The goal of the system is to continuously optimize its structure and behavior in a dynamic environment, in order to improve (ideally maximize) the value of a system utility function or to satisfy certain utility constraints.

Mechanisms

—*Partial view* : Every autonomous agent maintains a partial view of the system that includes its neighborhood, a model of its local environment, partial knowledge of the system acquired from neighbors, and estimated properties of localized parts of the system.

—*Feedback*: A form of communication in self-organizing MANET and P2P systems that results from agents taking actions. Environmental feedback allows an agent to update its internal model of the local environment. Interagent feedback enables a group of agents to establish localized consensus on the state of the system and the system's environment. Positive feedback increases the probability of the action that caused the feedback being selected, while negative feedback decreases the likelihood of this action's selection.

—*Decay*: A mechanism that degrades the quality of information stored in an agent's partial view until the information becomes stale and is removed from the partial view.

—*Evaluation function*: A heuristic that allows an agent to estimate the utility of available actions, given the current agent state. The evaluation function should be engineered in such a way that when an agent selects actions that maximize its local evaluation function, the system optimises its global utility.

—*Action selection*: The algorithm performed by an agent in order to select the next action for execution. Exploitative action selection policies favor higher utility actions, while exploratory policies may select suboptimal actions that enable the discovery of previously unknown states and actions.

Fig. 7. Summary of the Properties and Mechanisms of Self-Organizing MANET and P2P systems.

The relationship between action evaluation functions and system utility is a particularly challenging problem. In cooperative systems, where agents share a common goal (as in the reviewed systems), the evaluation function can be designed as a utility function. However, if there is no feedback between agents or feedback between agents and the environment, agents have high uncertainty as to the utility of available actions due to their lack of global knowledge or the state of the environment. Probabilistic action selection is one approach that enables agents to explore their environment to find better solutions to problems (helping improve the system utility). However, models where agents only use exploratory actions to improve their estimation of the action utility often decrease system utility, due to exploratory actions increasing usage of the system's resources. Feedback models, in contrast, are more efficient at allowing agents learn about the state of the system and the environment, as a feedback message can be generated by a single agent executing an action, but received by many agents. Feedback enables the collective, asynchronous improvement in

the quality of partial views at many agents, helping improve agents' ability to estimate the utility of actions. The convergence of agent state through feedback can help agents to coordinate their behavior to improve system utility, provided that local environmental models accurately model the real environment.

The use of feedback to improve system utility can be observed in congestion scenarios in the MANET systems. When the level of traffic over a path reaches a state of congestion (decreased system utility), agents with a new routing problem avoid using the congested path through feedback from the local environment model and feedback from remote agents. The agent does not need to route packets on the congested path in order to discover its state of congestion, which would decrease system utility. Instead, feedback enables a localized group of agents to reach consensus that the path is congested, and agents then take actions using local state to avoid that path. In this way, evaluation functions that use state models updated by environmental and interagent feedback, can be based on approximate optimisation algorithms that maximise the agent's local utility, while simultaneously improving system utility.

In the reviewed MANET systems, designers favoured experimentation and simulation to evaluate system utility. This is due to the difficulty in formally validating convergence properties of multi-agent systems in dynamic environments with non-linear system behavior [Wolf et al. 2005; Babaoglu et al. 2006]. Experiments require a realistic model of the system's real environment, provided by the network simulators used by SAMPLE and AntHocNet, where system utility values such as network throughput and packet delivery ratios can be measured over different network setups and experimental runs.

6. CONCLUSIONS AND FUTURE WORK

In this article, we presented an abstract agent-based model of self-organizing MANET and P2P systems and showed how it is realized in the reviewed systems. The model describes how agent behavior, based on local state models of the environment and neighbors, can be adapted to improve overall system behavior, through feedback generated from a dynamic environment. Agents continuously use feedback from neighboring agents and their local environments to improve the quality of their partial view of the system, allowing localized groups of agents' partial views to converge, thereby enabling coordinated agent behavior. Coordinated agent behavior, in turn, can help improve desired system properties. In general, feedback models can reduce the need for agents to take actions to explore the system's environment and can reduce the amount of message passing required to coordinate agent behavior; both helping to increase system utility. The agent-level mechanisms that our model covers are action evaluation functions, action selection policies, feedback, and decay.

We believe that our model can be used to inform the construction of distributed systems with improved performance in dynamic environments. For example, in existing state of the art DHT-based P2P networks, system structure is determined by node identifiers, rather than the state of the environment. We have been investigating a Gradient topology where system structure captures information about node uptime and performance characteristics [Sacha et al.

2006]. We are also using our model to help inform other research in building P2P multicast streaming protocols that adapt to heterogeneity in their network environment [Biskupski et al. 2006], and self-organizing traffic lights that optimise vehicular traffic [Cunningham et al. 2006].

ACKNOWLEDGMENTS

We would like to thank Giovanna Di Marzo Serugendo and the anonymous reviewers for their valuable comments and suggestions to improve the quality of this article.

REFERENCES

AKYILDIZ, I., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. 2002. Wireless sensor networks: a survey. *Comput. Netw. 38*, 4 (Mar.), 393–422.

ALBERT, R. AND BARABÁSI, A.-L. 2002. Statistical mechanics of complex networks. *Rev. Mod. Phys. 74*, 1 (Jan.), 47–97.

AXELROD, R. 1997. *The Complexity of Cooperation*. Princeton University Press, Princeton, NJ.

BABAOGLU, O., CANRIGHT, G., DEUTSCH, A., DI CARO, G., DUCATELLE, F., GAMBARDELLA, L., GANGULY, N., JELASITY, M., MONTEMANNI, R., MONTRESOR, A., AND URNES, T. 2006. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst. 1*, 1, 22–66.

BARAS, J. AND MEHTA, H. 2003. A probabilistic emergent routing algorithm for mobile ad hoc networks. In *WiOpt 2003: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks* (Sophia-Antipolis, France). IEEE Computer Society Press, Los Alamitos, CA.

BIRMAN, K. AND JOSEPH, T. 1987. Exploiting virtual synchrony in distributed systems. In *SOSP '87: Proceedings of the 11th ACM Symposium on Operating Systems Principles*. ACM, New York, 123–138.

BISKUPSKI, B., CUNNINGHAM, R., DOWLING, J., AND MEIER, R. 2006. High-bandwidth mesh-based overlay multicast in heterogeneous environments. In *Proceedings of the International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications* (PISA, Italy). ACM, New York, to appear.

BLUM, C. AND ROLI, A. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35*, 3, 268–308.

BRESLAU, L., ESTRIN, D., FALL, K., FLOYD, S., HEIDEMANN, J., HELMY, A., HUANG, P., MCCANNE, S., VARADHAN, K., XU, Y., AND YU, H. 2000. Advances in network simulation. *IEEE Comput. 33*, 59–67.

CABRI, G., LEONARDI, L., AND ZAMBONELLI, F. 2000. Mobile-agent coordination models for internet applications. *Computer 33*, 2, 82–89.

CAHILL, V., GRAY, E., SEIGNEUR, J.-M., JENSEN, C., CHEN, Y., SHAND, B., DIMMOCK, N., TWIGG, A., BACON, J., ENGLISH, C., WAGEALLA, W., TERZIS, S., NIXON, P., SERUGENDO, G., BRYCE, C., CARBONE, M., KRUKOW, K., AND NIELSEN, M. 2003. Using trust for secure collaboration in uncertain environments. *IEEE Perv. Comput. Mag. 2*, 3, 52–61.

CAMAZINE, S., FRANKS, N. R., SNEYD, J., BONABEAU, E., DENEUBOURG, J.-L., AND THERAULA, G. 2001. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ.

CLARKE, I., HONG, T. W., MILLER, S. G., SANDBERG, O., AND WILEY, B. 2002. Protecting free expression online with Freenet. *IEEE Internet Comput. 6*, 1, 40–49.

CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. 2000. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the International Workshop on Designing Privacy Enhancing Technologies*. Springer-Verlag, New York, 46–66.

COHEN, B. 2003. Incentives build robustness in BitTorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems* (Berkeley, CA). 251–260.

COLLIER, T. AND TAYLOR, C. 2004. Self-organization in sensor networks. *J. Parall. Distrib. Comput. 64*, 7 (July), 866–873.

CUNNINGHAM, R., DOWLING, J., HARRINGTON, A., REYNOLDS, V., MEIER, R., AND CAHILL, V. 2006. Self-optimization in a next-generation urban traffic control environment. *ERCIM News—Special: Emergent Computing 64*, 55–56.

CURRAN, E. AND DOWLING, J. 2005. SAMPLE: Statistical network link modelling in an on-demand probabilistic routing protocol for ad hoc networks. In *Proceedings of the 2nd Conference on Wireless On Demand Network Systems and Services*. IEEE Computer Society Press, Los Alamitos, CA, 200–205.

DECKER, K., SYCARA, K., AND WILLIAMSON, M. 1997. Middle-Agents for the Internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)* (Nagoya, Japan). 578–583.

DI CARO, G. AND DORIGO, M. 1998. AntNet: Distributed stigmergetic control for communications networks. *J. Artif. Intell. Res. 9*, 317–365.

DI CARO, G., DUCATELLE, F., AND GAMBARDELLA, L. 2005. AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *Europ. Trans. Telecom. (Special Issue on Self-Organization in Mobile Networking 16*), 443–455.

DIJKSTRA, E. W. 1974. Self-stabilizing systems in spite of distributed control. *ACM Commun. 17*, 11, 643–644.

DORIGO, M. AND DI CARO, G. 1999. The ant colony optimization meta-heuristic. In *New Ideas in Optimization*. McGraw-Hill, London, U.K., 11–32.

DOWLING, J. 2004. The decentralized coordination of self-adaptive components for autonomic distributed systems. Ph.D. dissertation, Dept. Computer Science, Trinity College, Dublin, Ireland.

DOWLING, J., CURRAN, E., CUNNINGHAM, R., AND CAHILL, V. 2005. Using feedback in collaborative reinforcement learning to adapt and optimise decentralized distributed systems. *IEEE Transactions on Systems, Man and Cybernetics (Part A), Special Issue on Engineering Self-Organized Distributed Systems 35*, 3, 360–372.

FERBER, J. 1999. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Addison Wesley Longman, New York.

FININ, T., FRITZSON, R., MCKAY, D., AND MCENTIRE, R. 1994. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM '94)*, N. Adam, B. Bhargava, and Y. Yesha, Eds. ACM, Gaithersburg, MD, 456–463.

FIPA. 2002. FIPA Interaction Protocol Library Specification. Available at http://www.fipa.org.

GARLAN, D. AND SCHMERL, B. 2002. Model-based adaptation for self-healing systems. In *Proceedings of the 1st Workshop on Self-Healing Systems*. ACM, New York, 27–32.

GENESERETH, M. R. AND NILSSON, N. J. 1987. *Logical Foundations of Artificial Intelligence*. Morgan-Kaufmann, San Francisco, CA, USA.

GOUDA, M. G. 2005. Guest editorial on special issue: Self-stabilizing systems, Part 1. *J. High Speed Netw. 14*, 1, 1–2.

GUSTAVSSON, S. AND ANDLER, S. F. 2002. Self-stabilization and eventual consistency in replicated real-time databases. In *WOSS '02: Proceedings of the First Workshop on Self-Healing Systems*. ACM, New York, 105–107.

HAYDEN, M. 1997. The Ensemble system. Ph.D. dissertation, Dept. Computer Science, Cornell University.

HEYLIGHEN, F. 2001. The science of self-organization and adaptivity. *Encyclop. Life Supp. Syst. 5*, 3, 253–280.

HUHNS, M. N., HOLDERFIELD, V. T., AND GUTIERREZ, R. L. Z. 2002. Achieving software robustness via large-scale multiagent systems. In *SELMAS* (Orlando, FL). Springer-Verlag, New York, 199–215.

JELASITY, M. AND BABAOGLU, Ö. 2006. T-man: Gossip-based overlay topology management. In *Engineering Self-Organizing Systems*. Lecture Notes in Computer Science, vol. 3910. Springer-Verlag, New York, 1–15.

JELASITY, M., KOWALCZYK, W., AND VAN STEEN, M. 2003. Newscast computing. Tech. Rep. IR-CS-006, Dept. Computer Science, Vrije Universiteit, Amsterdam, The Netherlands.

JENNINGS, N. R., SYCARA, K., AND WOOLDRIDGE, M. 1998. A roadmap of agent research and development. *J. Autonom. Agents Multi-Agent Syst. 1*, 1, 7–38.

JOHNSON, D., MALTZ, D., AND BROCH, J. 2001. DSR: The dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*. Addison-Wesley, Reading, MA, 139–172.

MANKU, G. S., BAWA, M., AND RAGHAVAN, P. 2003. Symphony: Distributed hashing in a small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*. USITS, 127–140.

MILOJICIC, D. S., KALOGERAKI, V., LUKOSE, R., NAGARAJA, K., PRUYNE, J., RICHARD, B., ROLLINS, S., AND XU, Z. 2002. Peer-to-peer computing. Tech. rep., HP Labs.

MONTRESOR, A. 2004. A robust protocol for building superpeer overlay topologies. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing*. IEEE Computer Society Press, Los Alamitos, CA, 202–209.

PADMANABHAN, V. N. AND SRIPANIDKULCHAI, K. 2002. The case for cooperative networking. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. (London, UK). Springer-Verlag, New York, 178–190.

PARUNAK, H. V. D., BRUECKNER, S. A., SAUTER, J. A., AND MATTHEWS, R. 2005. Global convergence of local agent behaviors. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems*. vol. 1. ACM, New York, 305–321.

PERKINS, C. E. 2001. *Ad Hoc Networking: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

PERKINS, C. E. AND ROYER, E. M. 1999. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd Workshop on Mobile Computer Systems and Applications*. IEEE Computer Society Press, Los Alamitos, CA, 90–100.

PRIGOGINE, I. AND STENGERS, I. 1984. *Order Out of Chaos*. Bantam, New York, NY.

RAPOPORT, A. AND CHAMMAH, A. M. 1965. *Prisoner's Dilemma*. University of Michigan Press, Ann Arbor, MI.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. 2001. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, New York, 161–172.

RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. 2004. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference*. USENIX, 127–140.

RIPEANU, M., IAMNITCHI, A., AND FOSTER, I. 2002. Mapping the gnutella network. *IEEE Internet Comput. 6*, 1, 50–57.

ROWSTRON, A. I. T. AND DRUSCHEL, P. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms* (Heidelberg, Germany). Springer-Verlag, 329–350.

RUSSELL, S. J. AND NORVIG, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.

SACHA, J., DOWLING, J., CUNNINGHAM, R., AND MEIER, R. 2006. Discovery of stable peers in a self-organizing peer-to-peer gradient topology. In *Proceedings of the 6th IFIP International Conference on Distributed Applications and Interoperable Systems*. Lecture Notes in Computer Science, vol. 4025. Springer-Verlag, New York, 70–83.

SCALABLE NETWORK TECHNOLOGIES, INC. 2003. *QualNet Simulator, Version 3.6*. Culver City, CA, USA. http://www.scalable-networks.com.

SCHOONDERWOERD, R., HOLLAND, O. E., BRUTEN, J. L., AND ROTHKRANTZ, L. J. M. 1996. Ant-based load balancing in telecommunications networks. *Adapt. Behav. 5*, 2, 169–207.

SEN, S. AND WONG, J. 2004. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. Netw. 12*, 219–232.

SERUGENDO, G. D. M., FOUKIA, N., HASSAS, S., KARAGEORGOS, A., MOSTÉFAOUI, S. K., RANA, O. F., ULIERU, M., VALCKENAERS, P., AND AART, C. V. 2004. Self-organizing applications: Paradigms and applications. In *Proceedings of the Engineering Self-Organizing Applications Workshop (ESOA '03)*. Springer-Verlag, New York.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Comput. Commun. Rev. 31*, 4, 149–160.

SUTTON, R. S. AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

SYCARA, K. 1998. Multiagent systems. *AI Mag. 10*, 2, 79–93.

SYCARA, K., PAOLUCCI, M., VELSEN, M. V., AND GIAMPAPA, J. A. 2003. The RETSINA MAS infrastructure. *Autonom. Agents Multi-Agent Syst. 7*, 1/2 (July), 29–48.

TANENBAUM, A. S. AND VAN STEEN, M. 2001. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Upper Saddle River, NJ.

VAN RENESSE, R., BIRMAN, K. P., AND VOGELS, W. 2003. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Systems 21*, 2 (May), 164–206.

WOLF, T. D., SAMAEY, G., , AND HOLVOET, T. 2005. Engineering self-organizing emergent systems with simulation-based scientific analysis. In *Proceedings of the 4th International Workshop on Engineering Self-Organizing Applications* (Hakodate, Japan). Lecture Notes in Computer Science, vol. 3910. Springer-Verlag, New York. 138–152.

WOLPERT, D. AND TUMER, K. 1999. An introduction to collective intelligence. Tech. Rep. NASA-ARC-IC-99-63, NASA.

WOOLDRIDGE, M. 2000. On the sources of complexity in agent design. *Appl. Artif. Intel. 14*, 7, 623–644.

WOOLDRIDGE, M. 2002. *An Introduction to MultiAgent Systems*. Wiley, Chichester, England.

WOOLDRIDGE, M. AND JENNINGS, N. R. 1995. Intelligent agents: Theory and practice. *Knowl. Engin. Rev. 10*, 2, 115–152.

YANG, B. AND GARCIA-MOLINA, H. 2003. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering* (Bangalore, India). IEEE Computer Society Press, Los Alamitos, CA, 49–60.

ZHANG, H., GOEL, A., AND GOVINDAN, R. 2004. Using the small-world model to improve Freenet performance. *Comput. Netw. 46*, 4, 555–574.

ZHAO, B. Y., DUAN, Y., HUANG, L., JOSEPH, A. D., AND KUBIATOWICZ, J. D. 2002. Brocade: Landmark routing on overlay networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems* (Cambridge, MA). Springer-Verlag Heidelberg, Germany, 34–44.

Received September 2005; revised September 2006 and October 2006; accepted November 2006