



## **BISON** **IST-2001-38923**

*Biology-Inspired techniques for  
Self Organization in dynamic Networks*

### **Structures and Functions of Dynamic Networks**

**Deliverable Number:** D01  
**Delivery Date:** June 2003  
**Classification:** Public  
**Contact Authors:** Geoffrey Canright, Andreas Deutsch, Mark Jelasity,  
Frederick Ducatelle  
**Document Version:** Final (January 30, 2004)

**Contract Start Date:** 1 January 2003  
**Duration:** 36 months  
**Project Coordinator:** Università di Bologna (Italy)  
**Partners:** Telenor Communication AS (Norway),  
Technische Universität Dresden (Germany),  
IDSIA (Switzerland),  
Santa Fe Institute (USA)

**Project funded by the  
European Commission under the  
Information Society Technologies  
Programme of the 5<sup>th</sup> Framework  
(1998-2002)**



### **Abstract**

BISON's task is to seek ways of implementing various functions on various network structures, using complex adaptive systems (CAS) to give distributed solutions for dynamic structures. In this Deliverable we describe and specify two of three of these elements: structure and function. D02 will then address what we mean by a CAS. We divide structures into two types: overlay networks and wireless ad-hoc networks. Overlay networks are logical networks, which are built on an underlying physical infrastructure that supports easy connectivity and routing. Ad-hoc networks represent the opposite case, that connectivity and routing are challenging and difficult. Hence, for overlay networks it is possible to consider using topology management to maintain a target topology chosen from a wide variety of possibilities (discussed here). The nature and constraints for a dynamic, ad-hoc network are also discussed. Then the functions to be implemented are defined and discussed. These functions are classified as 'basic' and 'advanced'; the former are simple enough to be defined in terms of a flow diagram, while the latter are described with words. In each case, our goal is to seek as much precision as possible in specifying the function — removing the ambiguity that tends to build up around the terms used. Finally, the notion that a function can support a topology (as well as the converse) is discussed. In the resulting ('modular') picture, basic functions (carried out on a given topological structure) can enable desired overlay topologies, which in turn allow for highly efficient implementation of other functions.

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Structures</b>	<b>5</b>
2.1	Overlay Networks . . . . .	5
2.1.1	Informed and Uninformed Structure . . . . .	6
2.1.2	Engineered Topologies for Search from Computer Science . . . . .	6
2.1.3	Randomness for Information Dissemination . . . . .	9
2.1.4	Empirically Observed Topologies . . . . .	10
2.1.5	Summary and Conclusions . . . . .	11
2.2	Ad-hoc networks . . . . .	13
2.2.1	Characteristics of mobile ad-hoc networks . . . . .	14
2.2.2	Topology management in MANETs . . . . .	18
<b>3</b>	<b>Functions</b>	<b>20</b>
3.1	Basic functions . . . . .	21
3.1.1	Routing . . . . .	21
3.1.2	Search . . . . .	21
3.1.3	Discovery . . . . .	22
3.1.4	Monitoring . . . . .	22
3.1.5	Collective computation . . . . .	23
3.2	Advanced functions . . . . .	24
3.2.1	Distributed processing . . . . .	25
3.2.2	Distributed storage . . . . .	25
3.2.3	Distributed content . . . . .	26
3.3	Topology management is a function . . . . .	28
<b>4</b>	<b>The Interplay of Structure and Function</b>	<b>28</b>
4.1	Conceptual Framework . . . . .	29
4.2	An Illustrative Example for Modularity . . . . .	30
4.3	Related Work . . . . .	31
<b>5</b>	<b>Conclusions</b>	<b>32</b>

## 1 Introduction

The BISON project seeks to enable a variety of functions on dynamic networks, using decentralized, robust, and adaptive mechanisms like those found in many biological systems. The goal of deliverable D01 is then to describe — and, as far as possible, define — the dynamic network structures on which the functions are to be implemented, as well as the functions themselves.

We interpret the “structure” of a network essentially as its topology. BISON studies networks whose topology is time-dependent. The most strongly time-dependent networks are those which are self-organized, in a way which allows nodes to enter and leave, and to make and break connections, without any central coordination. Such strongly dynamic networks require high adaptivity and robustness on the part of any mechanisms which are to implement the desired functions. The dynamic nature of the network topology may occur on two distinct network layers: the physical layer, and the application layer. In the former case, the dynamic connections are wireless links, and the network is an ad-hoc network. In the latter case, the connections are virtual or logical links, built atop a working wired network, such as the Internet. In this case the network is an overlay network.

As discussed in detail below, these two distinct types of dynamic networks present distinctly different constraints on the topologies that may be, and typically are, realized. Hence the subsequent discussion of network structures (Section 2) is broken into two parts: one for overlay networks, and one for wireless ad-hoc networks.

In principle, network functions may be defined independently of network structure. The BISON project has identified a short list of functions to be implemented using decentralized ‘swarm-intelligence’ techniques. These functions are classified as ‘basic’ or ‘advanced’ — each category represented by a corresponding Workpackage. The aim of Section 3 of this Deliverable is then to define these various functions, as precisely as possible. Such definitions represent part of a larger goal, that is, the goal of describing behavior as precisely as possible. Precise specification of the desired behavior on a network, coupled with precise (via abstraction) description of the behavior of biological multi-agent systems, will aid in the identification of promising swarm-intelligence methods for bringing about the former.

Finally, we observe that network structure need not be viewed as a given. Instead, topology can be actively managed in order to aid in the realization of a desired network function. Hence we add ‘topology management’ to our list of functions. Effective topology management can in itself however require the support of some other network function. For example, implementation of a ‘superpeer’ topology requires the gathering of information about bandwidth (and other properties) of some subset of nodes in the network. Hence, in seeking to implement topology management, one abandons the simplistic view that functions are built on top of a given topology. Rather, functions can be built on top of a managed topology, which in turn is built upon some other (set of) functions. These latter must be built on some topology — but it is not the same one which they enable the management of. The resulting picture is thus not ‘chicken-and-egg’; instead it is a ‘bootstrapping’ picture, which has a well-defined starting point, from which one builds up successively both functions and managed topologies. This picture is presented in detail in Section 4. The possibilities for topology management are considerably more limited for an ad-hoc net than for an overlay network, since the former does not allow truly long-range links. Hence the discussion in Section 4 is focused primarily (but

not exclusively) on overlay networks.

## 2 Structures

As already mentioned, this document focuses on decentralized, fully distributed systems, and in particular on overlay networks and ad hoc networks. The structure of these systems can be described in terms of the structure of the underlying networking infrastructure and the structure of the distributed system itself which deploys the infrastructure.

In this section we describe both structures using graph theoretic terminology. In the case of the networking infrastructure the nodes of the graph are the nodes in the network that implement the communication protocol layers of the infrastructure and the edges are defined by the “could send information to” relation. In other words, if the infrastructure allows sending information from A to B then the edge (A,B) is part of the graph. In the case of the virtual or overlay structure, the nodes are the components of the application (processes, agents, etc) and the edges are defined by the relation “can send information to”. Note the difference between “can” and “could”. The former implies the “knows about” relation as well. Obviously, the structure of the underlying infrastructure acts as a constraint on the overlay infrastructure.

The difference between overlay and ad hoc networks will become clear in the remainder of this section. Here we simply mention briefly that, in the case of overlay networks, message routing is understood as part of the networking infrastructure, and considered relatively cheap; while in the case of ad hoc networks, routing is considered very difficult and expensive. This makes the two cases radically different. For overlay networks the more interesting part is the structure of the application, because routing makes the infrastructure practically fully connected. For ad hoc networks the more interesting structural part is the infrastructure itself.

This observation also reveals a little inconsistency in the terminology, in that “network” is used in two different senses. In “overlay network” it refers to the structure of an application, while in “ad hoc network” it refers to the structure of the networking infrastructure (mobile devices, sensors, etc.).

### 2.1 Overlay Networks

In the case of overlay networks the networking infrastructure is typically the Internet or any similar large network where in principle any pair of nodes can communicate with each other. The structure of this infrastructure, w.r.t. connectivity is therefore quite boring: the relation “could possibly communicate” defines a complete undirected graph over the nodes of the network. Note that a fully realistic account should take into consideration the cost of routing (which can be quite different for different pairs of nodes), and also partitioning induced by firewalls and similar solutions, which often renders communication between a pair of nodes unidirectional or even impossible.

Even though in overlay networks research the fine details of the structure of the infrastructure are playing an increasingly important part [52, 15, 36], in our approach we focus more on the virtual, overlay structure.

In the following we first examine the usual choices of topology in overlay networks. These topologies we classify according to the function they support. One class is the search-like functions, like routing, and lookup in distributed hash tables. The so called third generation of P2P systems (also called structured overlays) maintain several kinds of topologies which are supposed to allow an effective implementation of these functions. The other class is information dissemination: broadcasting, multicasting, flooding. The so called second generation of P2P systems used these mechanisms also for search, a choice which turned out to be very inefficient. The prominent example of these systems is Gnutella [25]. As we will see these functions can also be implemented on top of more sophisticated structures; they pose somewhat different research questions which are related to the randomness of the structure. Finally, we discuss naturally occurring (self-emerging) topologies from biology, sociology, and even computer science and relate them to the engineered topologies.

### 2.1.1 Informed and Uninformed Structure

Before going into the details of the different topologies an observation must be made. In the most general setting, a graph is defined by a set of nodes and a set of edges over those nodes. When talking about topology, one can refer to structures such as circle, star, etc.

However, in overlay networks, nodes and edges have certain properties. Nodes can hold some content and possess different resources, edges have capacity, reliability, etc. It is usual that topologies correlate with these properties, and for some functions like routing, this correlation is heavily exploited.

When a topology is defined not only in pure structural terms but also in terms of the properties of the nodes and links, we say the topology is *informed*. Otherwise we say it is *uninformed*. An example is a directed circle (uninformed) vs. a directed circle in which node IDs are in increasing order (informed). Another example is a star (uninformed) vs. a star in which the center is the node with the highest bandwidth (informed).

### 2.1.2 Engineered Topologies for Search from Computer Science

**2.1.2.1 Star** The star topology is usually considered to be very vulnerable and not scalable. Still, some of the first generation of partially centralized P2P systems like Seti@home [47] and Napster [41] were very successful. Scalability is a clear problem, but the star topology should not be underestimated. While vulnerable to a directed attack against, or failure of the central node, the star is extremely robust to random node removal. Putting it more strictly, independently of system size, the preservation of star structure has a constant probability when removing a given proportion of nodes (it does not increase with increasing system size). It also makes very effective implementations of search possible. This is the reason why different generalizations of the star topology are emerging, like super peer networks [54]. In a certain sense scale free networks are robust versions of the star topology [2] and even trees are very similar to it in many senses, especially if non-leaf nodes have a large degree (in fact, a star is a tree as well).

**2.1.2.2 Ring (with shortcuts)** Maintaining a ring in an overlay network is a very popular idea in the field of distributed hash tables [48] (and implicitly [45]) and occurs in general low diameter networks too [35]. Almost all known distributed hash table approaches apply one or two rings to support their protocols, as we will see. We have to add, the main (otherwise justifiable) reason for that could be that this topology is easy to grasp.

For routing and search a ring is quite ineffective however, requiring  $O(n)$  communications (where  $n$  is the number of nodes in the ring). Therefore all applications of the ring extend it with some other idea to reduce costs. The usual idea is to add shortcuts, in an informed way, that is, in a way which is correlated to the ID of the nodes (or content). Shortcuts are necessary for the other reason of providing robustness through redundancy. A ring alone is extremely vulnerable to partitioning as a result of node removal. With different techniques for maintaining the topology, both Pastry and Chord can offer search with time complexity  $O(\log n)$  on average.

We can conclude that even though it is a general tendency to focus on the ring as the core idea in the above mentioned systems, in fact the core idea is the shortcuts, and the ring can be replaced by any convenient regular or non-regular structure which provides sufficient and predictable local connectivity. We will return to this observation later in connection with the small world graphs.

**2.1.2.3 d-Dimensional grid** This topology (in fact a torus) is applied in the distributed hash table system CAN [44]. This structure has the drawback that it is both hard to maintain (being highly regular); and the search time that can be expected is  $O(d\sqrt[n]{n})$ , which grows much faster than  $\log n$ . The advantage is that the degree of each node is fixed, so with increasing system size the number of known nodes does not have to be increased. Other topologies discussed below have this property also however. Furthermore, it has been proven in [30] that in a constant degree network the vulnerability to partitioning as a result of random node removal is an increasing function of  $n$ .

**2.1.2.4 Butterfly** The butterfly topology is well known and often used for implementing shared memory multiprocessors (see Figure 1). The role of the nodes in the network is not symmetric. There are targets (right column), switches (middle) and processors (left column). The switches have two inputs  $i_1$  and  $i_2$  and two outputs  $o_1$  and  $o_2$ . They can choose between two states: they can either forward  $i_1$  to  $o_1$  and  $i_2$  to  $o_2$  or they can forward  $i_1$  to  $o_2$  and  $i_2$  to  $o_1$ . The processors want to reach the targets quickly through the switches.

The advantages of the topology are many-fold. It has a diameter  $O(\log n)$  and the nodes are of constant degree: the switches are of degree 4, and the other components have degree 1. Routing is always done from the processors towards the targets. It is based on the binary IDs of the targets: the process starts from level 0 (processors, left side) and each bit of the ID determines the state the switch chooses to reach another switch at the next level of the butterfly. It also scales well, because for  $n$  targets the number of switches is  $O(n \log n)$ —since, due to the routing algorithm, the number of layers is exactly the number of bits in the target IDs.

Viceroy, a distributed hash table approach [39], implements an approximation of such a topology in a distributed fashion. Another difference from the above-described ‘classic’ butterfly is

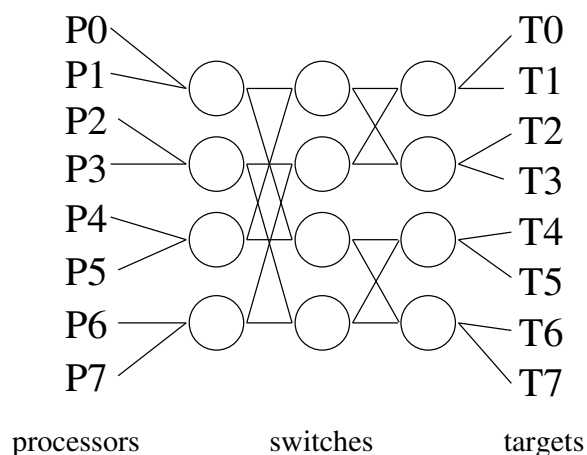


Figure 1: A butterfly network

that, in the Viceroy network, all nodes can take the role of targets. These two differences make the building and maintenance of the network considerably more challenging. The protocol is rather complicated, as is the topology that is actually generated and maintained, and as is the three phase routing algorithm. The basic idea is to maintain several rings, one for the whole graph, and one for each level. Local mechanisms are also needed to allow nodes to decide which level to choose. Each node has several types of links: some belong to the rings, and some to the embedded butterfly topology. Routing is three phase: the first phase targets a node from level 0, then using the butterfly the message goes to a node with maximal level and finally, using the global ring, the target is reached.

We also have to note that Viceroy also faces the problem of robustness, as we mentioned in connection with CAN, due to the constant degree construction. A separate mechanism is therefore needed to make sure the topology is robust. And to ensure this, the degree of at least some nodes have to be at least logarithmic. This fact seriously questions the main motivation behind Viceroy, which was developing a constant degree solution. On top of this, the Viceroy protocol needs an approximation of system size to operate correctly.

**2.1.2.5 de Bruijn graphs** A remarkable approach to implement distributed hash tables is Koorde [30]. Like many other approaches, Koorde is also based on a ring, in fact on top of the Chord system discussed earlier. Beside running the Chord protocol to maintain the ring, a de Bruijn graph is also embedded and used for routing. In the binary case, a de Bruijn graph is defined as follows. The nodes of the graph are  $1, \dots, 2^b$ . From a node  $m$  there is a link to nodes  $2m \bmod 2^b$  and  $2m + 1 \bmod 2^b$ . Note that this means  $m$  has links to numbers which have the binary representation of  $m$  shifted to the left by one position, dropping the highest order bit, and adding both 0 and 1 as lowest order bit. This observation serves as a basis for the routing algorithm. The idea is that it is trivial to find a path which is always 1 bit closer to the target, by shifting in the bits of the target one at a time.

The lookup cost is therefore  $O(\log n)$ . Generalization of de Bruijn graphs is straightforward for different bases than 2. In particular, choosing the base  $\log n$  we have lookup complexity



$O((\log n)/\log \log n)$  and also robustness due to the logarithmic degree.

**2.1.2.6 Tree** The tree topology is attractive for several reasons. It is ideal for broadcasting, because, from a given node, each other node is accessible through exactly one path, so that no unnecessary messages are generated. It is also ideal for aggregation (node counting, average value of some property, etc) also because of the lack of redundancy.

However, exactly because of the lack of redundancy, a tree is very fragile. Removing any non-leaf node results in partitioning. For the sake of completeness we have to mention that if non-leaf nodes have out-degree larger than 1 then there can be significantly more leaf nodes than non-leaf nodes.

In some overlay systems a tree is maintained or generated dynamically on demand. In Astro-labe [52] a tree is maintained with the help of an epidemic protocol which reflects the hierarchy of participating groups of nodes. In [20, 5] a spanning tree is dynamically generated for the purposes of broadcasting and aggregation.

**2.1.2.7 Specialization** Although specialization is not strictly a topological issue, it is clearly a structural one. Specialization means that in the network some nodes will specialize in a service or a function. An example of specialization is the several forms of super-peer networks [31, 54] where ordinary nodes with above average capacity are temporarily delegated to be servers for the other peers.

Specialization, like topology, can also be informed or uninformed. An example is when 10% of the nodes are specialized (uninformed) vs. when 10% of the strongest nodes are specialized.

### 2.1.3 Randomness for Information Dissemination

A topology, which for the moment we could loosely call a random topology, is not a traditional choice of computer architecture. However, for long it has been an interesting research topic in mathematics; and random graph results have found their way into computer science, in the form of randomized algorithms.

Recently, random graphs have emerged as topologies for P2P overlay networks as well. By definition, random graphs are uninformed topologies, so they can be applied to unstructured tasks like epidemic broadcasting, flooding, or other information dissemination mechanisms. The reason is that random graphs have a small diameter and good mixing properties, i.e. a random walk will quickly reach its stationary distribution. This means one can get a random sample from the graph by performing only a short random walk on it.

Intuitively, one could think that random graphs are trivial to maintain. This turns out not to be true however. In fact quite sophisticated mechanisms are necessary to make sure that joining and leaving nodes do not induce undesirable patterns. Hence developing a protocol for maintaining randomness is just as much a nontrivial research question as maintaining any other topology. SCAMP is one example [23].

**2.1.3.1 Relevant mathematical research** The characterization of randomness and the properties of random graphs have an extensive mathematical literature which is relevant to the field of overlay networks.

The theory of random walks on graphs [37, 38] addresses the question of the possibility and speed of getting random samples from the graph using random walks. Mathematical notions like mixing and conductance play an important role. Mixing characterizes the speed of reaching a stationary distribution with a random walk. Conductance characterizes the reachability of one part of the graph from another.

The field of pseudo-random graphs also addresses the issue of randomness, in particular the question: how can we characterize the randomness of the graph [34]. Instead of random walks, pseudo-randomness focuses on the edge distribution. The edge distribution is a probability distribution over the number of links that lie within fixed-size subsets of the nodes of the graph. In case of randomly generated edges one can describe the distribution of the number of edges that are within a given subset of a fixed size. For a fixed given graph it is possible to compare the actual distribution with this random distribution. The intuition is that if the edge distribution is flat enough (there are no extremely connected or disconnected subsets) than the graph can be called pseudo-random.

In all the fields mentioned above the spectral theory of graphs plays a key role. This involves eigenvalues of the adjacency matrix of the graph, and in particular, the spectral gap, the difference between the largest and the second largest eigenvalue. Roughly speaking, the larger the gap, the more random the graph is.

## 2.1.4 Empirically Observed Topologies

In the previous part we discussed topologies which need explicit effort to maintain. In a sense, all protocols mentioned (including SCAMP, that maintains a random topology) have to invest significant effort to fight the forces of emergence. The main problem is always handling node failures, nodes leaving the system and joining nodes.

But what happens if we just let it loose and allow the topology develop? In fact the situation was exactly this in the development of the world wide web, and also Gnutella, which originally did not include any efforts to manage topology.

But not only computer science produces emergent topologies. In fact the social, physical and biological world is full of structures which emerge through local interactions [2]. In the following we describe two main notions from the field of empirically described topologies: small worlds and scale free graphs.

**2.1.4.1 Small world** The notion of small worlds originates from sociology [40] and refers to the fact that in spite of the enormous size of human population, an arbitrary pair of people have a short chain of friends (less than 10) which connects them. Mathematically speaking, the graph defined by friendship over the set of people has a very low diameter, almost as low as that of a random graph.

In fact, the random graph should be called small world as well, but historically the notion of small world also became associated with high clustering coefficient. The clustering coefficient

of a graph is the probability that two neighbors of a node are also neighbors of each other. In social networks clustering is high, so a small world graph is defined as a graph which has low diameter and high clustering. The random graph is not a small world according to this definition as random graphs have very low clustering.

One key fact is that small worlds usually *emerge* spontaneously, ie, without any central planning or global knowledge. For example, social networks are not designed by anyone, instead being maintained by strictly local operations. Another key fact which makes small world graphs relevant is that they can be used for routing [32]. In fact, there are attempts towards actually using small world topologies for routing and thereby implementing a distributed hash table [13].

From a mathematical point of view, one of the most popular way of looking at small worlds is assuming that they have a regular component like a grid or a ring which is responsible for clustering and some shortcut links which are responsible for low diameter. One such model is the Watts-Strogatz (WS) model [53, 42].

**2.1.4.2 Scale free** Scale free topologies are characterized by their power-law degree distribution. In other words  $\Pr(d(a) = i) = i^{-k}$  where  $d(a)$  is the degree of node  $a$  and  $k$  is the parameter, usually around 2.

Many naturally emerging topologies show the power-law degree distribution including the world wide web, the co-authorship graph, chemical reaction networks, food chains, etc. [2]. The intuitive meaning of the power-law distribution is that there are very few nodes with a large degree (many neighbors) and almost all the nodes have a very low degree. This makes this topology very similar to the star, discussed above. We can consider it as a sort of fuzzy star which, most importantly, emerges based on simple (but not necessarily local) rules.

Models for generating a scale-free graph are known. One such model is the preferential attachment rule [4]; other approaches can be found in [17].

An interesting question arises however. In contrast with the small world networks, scale-free networks are yet to find a computer science application. Their robustness to random node failure is often cited; but they are very sensitive to targeted attacks—similarly to the star. Another open question is the explanation of emergent scale-free graphs in a dynamic environment (such as Gnutella). Current models describe the growth of scale-free graphs, but do not address fluctuation of nodes. Also, models like the preferential attachment rule assume global knowledge about the network.

**2.1.4.3 Newscasting** We briefly refer to our own work here [29] which provides evidence for a naturally emerging third type of topologies (and probably there are many more types). The topology which emerges as a result of the simple local epidemic-style operations of the newscast protocol is a small world, which does not have a clearly identifiable regular and random component, and it is not scale free either.

## 2.1.5 Summary and Conclusions

Let us summarize the main structural concerns in overlay networks that were mentioned scattered in this section. We will list a number of design goals and the corresponding desired

properties of the topology. We will also mention a possible optimal choice for all design goals, pretending there are no other goals to satisfy. This illustrates nicely how these design goals contradict to each other.

**robustness** From a pure structural point of view we can interpret robustness as the ability to preserve structure in the face of node removal (failure). This can mean the strong requirement of preservation of a specific topology (like ring or tree) or the weaker requirement of pure connectivity which still makes it possible to at least re-build the topology. *Optimal choice: fully connected network (clique).*

$\Omega(\log n)$  **degree** As shown in [30] a necessary condition for connectivity with constant probability after removing half the nodes from a graph is that there are at least some nodes with  $\Omega(\log n)$  degree. Probably much stronger results could be proven (for example it is evident that at least a constant proportion of nodes have to have  $\Omega(\log n)$  degree) but in any case this means that a constant degree graph does not suffice.

**high path-redundancy** Obviously, topologies with minimal redundancy like a tree or a ring are not robust, not even in the weak sense. This implies that if some system is to make use of such topologies, the only option is to embed them in some more redundant structure which guarantees at least connectivity. This topic of stacking topologies will be revisited in the present document in detail.

**statistical definition** Being in search of topologies which provide robustness in the strong sense we have a larger chance of success with the ones that are defined in statistical terms. Even though we are not aware of such research, it seems intuitively likely that removing random nodes from a scale free or small world graph will give a modified graph which is still very similar to scale free or small world. And random graphs (slightly depending on the definition we choose) are of course robust to node removal in the strong sense. A remarkable exception is the star, which is not statistically defined but which is robust even in the strong sense with a constant probability, independently of graph size.

**maintenance costs** For a given topology the costs of the protocol that maintains it depends not only on the topology but also on the implementational choices of the particular protocol. Nevertheless, from a structural point of view it seems possible to identify some guidelines that will be likely to always correlate to the costs. *Optimal choice: edge-less graph (isolated nodes, zero costs).*

**node degree** Approaches which require a high node degree cost more, and scale more poorly than those that need only a small constant degree. The reasons for this include the higher costs to update links (in the case of newly joining nodes), and the higher costs of ping messages that make sure a node has a sufficient number of living connections. Some approaches, like Viceroy or Koorde, were even motivated specifically by the need for constant degree solutions. Note however, that this requirement is in contradiction with the robustness requirement mentioned above, so these approaches have to develop dedicated mechanisms to make sure the system is robust.

**emergence** Intuitively, structures like small worlds and scale free topologies that tend to emerge in nature are easier to maintain, because natural systems normally do

not rely on central services or coordination. This provides motivation for discovering the utility (if any) of these structures and understanding better the mechanisms through which they emerge.

**efficiency (for routing)** Independently of dissemination algorithm it is rather difficult to identify good properties. *Optimal choice: search tree.*

**low diameter** This allows an algorithm to reach a node from any other node quickly—ie, in a small number of hops.

**informed topology** There has to be a correlation between distance in terms of topology and in terms of content space or address space.

As pointed out by Kleinberg [32], efficient routing requires both short paths *and* efficient ways of finding these short paths. Low diameter gives the first requirement (short paths)—however, random graphs (for example) have low diameter, but do not allow any efficient way of finding the short paths that are present in the graph. Hence our second requirement, which provides a form of ‘guidance’ through the graph. Distributed hash tables are explicitly built to meet both requirements, and so support efficient routing.

Note that clustering is not required to be high. For example, the optimal topology (the search tree) has zero clustering.

**efficiency (for dissemination)** Again, it is rather difficult to identify good properties independently of dissemination algorithm, but the following ones are potentially rather algorithm independent, assuming that in a decentralized network dissemination is done using some form of flooding or epidemic-style protocol. *Optimal choice: algorithm dependent but e.g. regular tree is very good.*

**low diameter** As in routing.

**low clustering** Clustering slows down dissemination by generating redundant messages.

**low path-redundancy** Redundant paths can result in a large number of redundant messages which is expensive. Note that path-redundancy does not imply clustering (eg the random graph is redundant, but has very low clustering), although the opposite implication is true.

## 2.2 Ad-hoc networks

Mobile ad hoc networks (MANETs) are the second kind of dynamic network environments we will investigate in this project. Like overlay networks, nodes can enter and leave the network at any moment, and links are added and deleted continuously. An important difference with overlay networks, however, is that overlay networks are logical networks which are implemented on top of a fixed wired infrastructure, while for MANETs it is the physical network itself which is dynamic. More concretely, a MANET consists of a set of mobile nodes which communicate with each other through wireless connections. Nodes which are in each other’s transmission range can communicate directly with each other, while communication over longer distances is established in a multi-hop fashion: intermediate nodes serve as relay points which help forwarding the data from the source to the destination. Due to the mobility

of the nodes, available links change constantly, and the resulting graph is very dynamic. In this section we will go into some detail about the communication network formed by a MANET. In the first subsection we will describe some properties of MANETs, especially concerning capacity, structure and mobility. In the second subsection we will investigate how we can perform topology management in MANETs. We will describe how existing algorithms try to manage the MANET environment.

### 2.2.1 Characteristics of mobile ad-hoc networks

In this subsection we give an overview of characteristics of MANETs. First we will describe the general properties which define what a MANET is. Then we will look at some important other characteristics which are consequences of these basic properties. We will talk about network capacity, about network connectivity, and about the influence of mobility and data traffic.

**2.2.1.1 General characteristics** A first characteristic of MANETs is that they are wireless networks. The wireless communication is usually obtained through omnidirectional antennas, which means that nodes can reach all nodes in a circular area around them. It also means, however, that they cause interference for all nodes communicating in this circular area. Whether nodes can communicate with each other in the presence of interference depends on the signal-to-noise ratio: if the signal is strong enough compared to the interference, it will be received. If there is too much interference, it is possible that the channel becomes completely blocked. Since the interference is caused by the traffic generated by the nodes, the MANET topology becomes in fact dependent on the data traffic pattern.

A second important property of MANETs is that nodes are mobile and that they can enter and leave the network at any given moment. They move around and the set of neighbours which they can reach with their antennas changes constantly, giving rise to a continuously changing communication graph. The way nodes move, and how correlated their movements are depends on their own mobility patterns. In paragraph d we will go into more detail about this.

A third characteristic is that there are usually no fixed parts in a MANET: the only elements in a MANET are the mobile nodes. This characteristic could be relaxed though. Although most research concentrates on the case without fixed infrastructure, there is also some interest in combining a pure MANET with some fixed base stations, e.g. in order to improve connectivity or network capacity.

A fourth property is that all nodes are (typically) equal. That is, the typical (research) ad hoc net is 'flat': there should be no nodes which have more responsibilities or privileges than other nodes. Some topology management protocols (see below) deviate from this flatness criterion.

Finally, a last characteristic is that there is no central control or overview. So any task (eg, routing) needs to be performed in a distributed way, using only local information.

**2.2.1.2 MANET network capacity** In [27] the authors calculate an upper bound for the transport capacity of a wireless multi-hop network in bit-meters per second per node. They consider MANETs where mobile nodes transmit with omnidirectional antennas in a circular area

around them. They find that the maximum transport capacity per node is inversely related to the square root of the number of nodes. So, the more nodes on the same surface, the lower the throughput per node. Intuitively, this result is a formalization of the following trade-off. Depending on the range of the hops, a bit of data generated at one node has to be retransmitted several times, and therefore causes a certain amount of bit transport in the network. One option could be to increase the transmission range, so that the source node would be able to reach the destination in one hop. This would reduce the number of retransmissions. But since omnidirectional antennas are used, the total area which is blocked during the transmission (due to interference) scales with the square of the distance between source and destination. So no nodes in this area can send data, and therefore their capacity goes to waste for the period of the transmission. This puts a serious limitation on the total capacity of the network. If we choose to use a short transmission range instead, using multiple hops between source and destination, the area blocked during transmission is much smaller: it scales only linearly with the distance. It is in fact optimal to take the shortest transmission range possible, and therefore the highest number of retransmissions. However, since the distance between source and destination in number of nodes to hop over scales with the square root of the number of nodes, the total channel blocking scales in the same way. This means that the network capacity scales with the inverse of the square root of the number of nodes.

This result means that MANETs with large numbers of nodes on a fixed surface are in fact impossible. It should also be clear from the description above that enlarging the surface does not make any difference: the same reasoning holds. In fact, when one enlarges the surface, this makes the upper bound of the capacity increase with the square root of the surface, but since also the average transport distance between source and destination scales with the square root of the surface, this means that for a fixed number of nodes the maximum throughput in bits per second per node will stay more or less the same. So a MANET with a large number of nodes  $N$  has a low capacity, and in the limit  $N \rightarrow \infty$  this capacity goes to zero. It must be pointed out that the work described above is purely analytical, and only considers the signal interference between nodes. The authors assume that there is perfect coordination among the nodes, so that as soon as one is finished sending, another one can use the channel (this is why the capacity result is an upper bound). In realistic MANETs, this issue of channel access coordination among nodes is an important problem known as medium access control (MAC). Since there is no central control in a MANET, and all nodes are equal, there is no central mechanism which can decide which node is allowed to send over the wireless medium. The nodes have to coordinate among themselves, and this is a very difficult process.

Unfortunately, there has not been much progress in the field of MAC protocols for MANETs. In simulations, an adaptation of IEEE 802.11 is commonly used. This protocol was originally designed for cellular networks or one-hop ad hoc networks. In this algorithm, nodes which want to send a message broadcast a request-to-send to their neighbours, and when they receive a clear-to-send back, they start sending. If they do not receive a clear-to-send, they wait for a random amount of time to try again. After transmission they expect an acknowledgment about the arrival of the message. It should be clear even from this rough description that this algorithm causes a lot of overhead per message. Due to the random backoff time after a failed request-to-send, it is in periods of high congestion even possible that some nodes never manage to obtain access to the channel. It is to be expected that more research in this area will increase the MAC efficiency, but it will always remain a difficult problem which causes a lot

of overhead. So, clearly the network capacity is in practice even much lower than the upper bound proposed in [27].

**2.2.1.3 Connectivity in a MANET** In [18] and [19], some interesting work is presented about network connectivity in MANETs. In these papers, the authors make a strong mathematical analysis of the structure and connectivity of MANETs. It is mainly theoretical work, in which nodes are randomly distributed over an area according to a Poisson process (although the results were backed up with some simulation analysis based on real population density data). In [19], it is assumed that two nodes can communicate as soon as they are within a certain distance of each other (interference is not taken into account). When a snapshot is taken of a MANET and connections are drawn between every pair of nodes which are close enough to each other to communicate, this gives rise to a graph. The authors investigate the properties of this graph, with respect to connectivity. They draw their inspiration from percolation theory. They find that there is a critical node density, below which the network is made up of scattered islands of interconnected nodes. Nodes are only connected with close neighbors, and there is no global network connectivity. The probability that there exists one big cluster of interconnected nodes which covers the whole network area is zero. Only when the node density goes above the critical point do the islands of interconnected nodes merge, and start to form a large network which covers the whole area. Even then, it is still possible for one node or a group of nodes to be outside this large network and therefore have no connectivity with faraway nodes. The probability that this happens for any one node (1 minus the percolation probability) decreases as the node density increases.

The most interesting point when it comes to network structure is when the node density is just above the critical point. It was found that in such a situation, even though there is a large cluster of interconnected nodes which covers the whole area, the network formed by this cluster is sparse and contains a lot of bottlenecks. This means that there will normally be a route between most pairs of nodes, but this route might be very far from the geographically shortest route, due to the sparse network structure. The route might also be quite vulnerable, as it goes through several bottlenecks. And when it breaks, an alternative route might only be found going over completely different regions of the network. This kind of situation, where the density is just above the critical density, might not be so rare: real MANET node densities will usually not be constant, and are bound to have regions where the density is in the range just above the critical density.

In [18], the authors investigate the same issues, but in a more realistic environment. They incorporate radio interference in their model, so that connectivity between two neighboring nodes not only depends on how close they are to each other, but also on how close other nodes are. They use code division multiple access (CDMA), and find that if the codes are sufficiently orthogonal the same results as above hold. Then they propose to combine CDMA with a simple form of time division multiple access (TDMA): each node individually (without central control like in normal TDMA systems) chooses one of  $k$  time slots to send in. Due to the distributed choice of time frames, some nodes will choose a time frame which was chosen by very few other nodes, while others will meet a lot of other nodes in their time frame. Nodes which are in low populated time frames will get little interference and will therefore be able to send quite far. Most other nodes will get a lot of interference and they will only be able to reach nearby nodes. The result of this scheme is some kind of small world graph at the physical level: many nodes



have short connectivity, while a few nodes manage to transmit very far. This should lead to good network connectivity (a low critical node density) even in case of imperfect orthogonality of the codes. It must be said, however, that this is a theoretical result and that this mechanism has not been tested in realistic simulations (as was mentioned in paragraph b, people usually use IEEE 802.11 to solve MAC issues, CDMA and TDMA are very rarely used).

**2.2.1.4 Mobility and traffic patterns** In paragraphs b and c we have described some structural properties of MANETs. These properties were studied in a static setting however: they are properties of MANETs at a certain point in time. It is clear that in order to fully understand the MANET environment, one also has to take into account dynamic properties. One has to understand how MANET structures evolve over time. It is quite clear that a sparse network with a lot of bottlenecks as described above is much less of a problem when mobility is low for example, or if mobility is high but nodes move in highly coordinated ways. The most important factor defining the dynamic properties of MANETs seems to be the mobility model, which describes how nodes move through the MANET area. Apart from that, the data traffic model also has influence.

In a realistic environment, nodes could move in many different ways. Nodes could e.g. be people at a concert, in which case they would move quite randomly with respect to each other. Or they could be soldiers in a platoon, so that the movements would be well coordinated, and the relative positions between nodes would change very little. Many other mobility patterns are possible, and each one of them gives rise to a different kind of MANET environment.

Unfortunately, real MANET implementations are non-existent, so that realistic mobility patterns (as traces) are in fact not available. Therefore researchers usually use artificial mobility models in simulation studies. Many different ones are possible (see [14] for an overview), but the most often used is the random waypoint model. In this model, nodes randomly pick a point in the MANET area to move to, and a speed. They move to the new point and wait there for a randomly chosen pause time before choosing a new destination. It is clear that this represents quite an artificial situation, and that it is dangerous to generalize results obtained for this mobility model, especially when the dynamics of the model are not fully understood. In [11], for example, it was found that if pause times are longer than 20 seconds, the number of neighbour changes for nodes is very low, even at high speeds, giving in fact a very stable network. Other studies indicate that the spreading of nodes in a random waypoint model is not uniform (see for example [7]: the node density tends to be larger in the center of the area), so that if simulations are initialized with a uniform node distribution, there is in fact a long initial phase with high variability before stable behaviour is reached, which can lead to unrealistic performance evaluations in simulation studies.

It is clear that different mobility patterns lead to different dynamic properties for the network. In order to understand these properties, it is important to have good mobility metrics, and to understand what they mean. Most MANET simulation studies use parameters of the mobility model (e.g. the pause time or the node speed) as mobility metrics, and compare performance measures for different values of these parameters. These parameters are however completely artificial (there is often no equivalent in realistic MANET implementations), and they might often not give real information about dynamic properties of the network, especially if their effect is not fully understood. (See e.g. the remark about pause times above: since pause times

over 20 seconds give very stable networks, there is no point in varying pause times between 30 and 600 seconds). In [12] the average link duration is proposed as a mobility metric. This metric is independent from the mobility model and from any network protocol, and seems to correlate well with performance measures, showing that it is a good indicator of the degree of difficulty presented by different dynamic properties. Other possible metrics are the average relative speed between nodes and the link change rate.

Apart from the mobility model, also data traffic patterns have an influence on the dynamic properties of MANETs. Data traffic will define which areas of the network are congested. Due to limitations at the MAC layer (see paragraph b), it is possible that competition between two connections in each other's vicinity completely starves one of the connections (see [6]), in fact altering the actual available network topology. Also in less extreme cases, it is clear that different traffic patterns and rates have a significant influence on the actual MANET environment. Unfortunately not much can be said about traffic patterns in MANETs in general. Most simulation studies use quite random traffic patterns, and some adjust the traffic rate in order to evaluate different situations. As for mobility models, it would be useful to develop traffic load metrics. One relevant remark here is the fact that, in MANETs, TCP cannot be used. This is due to congestion control mechanisms and timers used in TCP which are not compatible with the highly dynamic and unreliable MANET environment (see [3, 49]). This means that UDP has to be used, which has for data traffic the consequence that traffic will typically only go in one direction (no acknowledgments are sent back). This gives a constraint on the traffic pattern which needs to be modeled realistically.

## 2.2.2 Topology management in MANETs

As was described in subsection 2.2.1 the topology of a MANET can possibly have a very sparse connectivity, and is subject to continuous changes due to node mobility. In such an environment it can be very difficult to perform functions like routing or search, and therefore some algorithms try to perform topology management. It is extremely difficult however to change the actual (physical) network topology, since this is the result of uncontrolled node mobility, traffic, and interference. Instead, topology management algorithms for ad hoc networks try to build an overlay network on top of the dynamic MANET, and perform topology management in this overlay network. In what follows we will mainly focus on topology management for routing algorithms, because routing is the most studied function in MANETs. Each of these methods should also be useful for other functions, like search for example. In the first part we will look at partitioning protocols, which try to impose a global structure onto the MANET. In the second part we will look at neighbour selection protocols, in which individual nodes make decisions about which nodes they consider near or far. In this way each node maintains a very simple structured view of the network, but the combined effect of their individual efforts provides an overlay structure which can be quite powerful.

**2.2.2.1 Partitioning protocols** Partitioning protocols try to create a global overlay structure in the network. Often they will build a hierarchy so that some nodes get a central role and become more important than others. Obvious examples are cluster-based algorithms, like Clusterhead Gateway Switched Routing (CGSR, [16]). In this kind of algorithms, nodes are grouped

into clusters, usually with a clusterhead. Ordinary nodes forward their packets to their clusterhead, and clusterheads route the packets among themselves, towards the destination cluster. Clustering and the selection of clusterheads in CGSR is performed either with the lowest-ID algorithm ([21]) or the highest-connectivity algorithm ([24]). For the maintenance of clusters, the Least Cluster Change algorithm is proposed.

In general one could say that partitioning protocols offer an advantage in large networks by providing a structure which helps to efficiently perform tasks in MANETs. However, an important drawback of partitioning protocols is the overhead created by the partitioning algorithm. Especially in highly mobile networks this will be a problem, since the overlay structure will have to be reconsidered every time the topology changes too much. Also the size of the network could pose difficulties, as cluster membership changes often have to be propagated over the whole network. So, although partitioning algorithms are supposed to help routing algorithms scale, they do not scale very well themselves. Apart from that, other disadvantages of partitioning algorithms include the increased dependence on certain vital nodes and the fact that some nodes will have to do more routing work. Finally, although partitioning leads to faster route discovery in large networks, the routes discovered are not always the shortest: nodes could be close to each other in the network, but still have to communicate through their respective cluster heads.

**2.2.2.2 Neighbour selection protocols** A good alternative in large networks seems to be the group of neighbour selection protocols. In these protocols, each node makes its own decisions as to which other nodes are considered near or far; thus, they each construct their own structured view of the network. And since every node does this, an overall structure emerges for the network. This is different from the partitioning algorithms, where a structure is imposed top down, and the nodes have to decide according to certain rules where they fit into the given structure. So like in partitioning protocols there is some structure in the network, which makes it easier to perform tasks over large distances, but there is no overall structure which has to be maintained. Each node makes a hierarchy from its own point of view.

An example is the Zone Routing Protocol (ZRP, [28]). In this protocol nodes keep up-to-date routing tables for nodes within a certain hop distance by constantly exchanging routing updates. If routing is needed to nodes further away, a reactive protocol is used: a route request message is broadcasted over the network in order to find the destination. This simple way of structuring the network leads to quite a powerful mechanism however: since nodes know their  $k$ -hop neighbourhood perfectly, they can send the route request directly to the edges of this area, rather than just to their direct neighbours. So the broadcasting can go in large jumps. Also, routes can be expressed as a list of the nodes on the edges of the  $k$ -hop neighbourhoods, rather than a full list of nodes between source and destination. This should allow more flexibility in the routes, which should make them more robust. So, due to the individual behaviour of the nodes, the network is divided into cells of  $k$  hops radius, which makes the routing task easier.

Another good example of neighbourhood selection protocols is the friends mechanism of Terminodes Routing ([8, 10, 9]). Each node proactively maintains a list of nodes which it considers to be "friends". This list of friends includes all nodes within a  $k$ -hop neighbourhood (as in ZRP), plus a certain number of distant nodes which are considered useful for the discovery of

good routes (the algorithm could easily be modified for other functions like search). The friend management algorithm constantly evaluates the current friends with respect to their usefulness, and indicates when it is time to drop a friend or to accept a new one. The nodes make sure that they have at all times good routes available to their friends. Since all nodes follow the scheme described here, the resulting overlay friendship graph has high local clustering (due to the  $k$ -hop neighbourhood) and a few long range links, and should have properties of small world graphs. Hence, path lengths to any destination should be short in the friendship graph. So, as with ZRP, the structuring behaviour of individual nodes provides an overlay structure in the MANET, which makes tasks like routing easier.

**2.2.2.3 Conclusion** The neighbourhood selection protocols seem to offer a good alternative to partitioning protocols: they do not depend on a hierarchy, and they should scale better, since no global structure has to be maintained. Nevertheless, neighbourhood selection protocols also create overhead. The friends mechanism for example needs constant updating in order to maintain the friendship graph in the continuously changing MANET environment. This is because they try to place a more or less static structure on top of a highly dynamic network. Herein lies an important difference with the overlay networks which were described in section 2: for those networks, at least the underlying infrastructure was fixed. This is not the case in MANETs, so that maintaining an overlay graph is quite difficult.

In general we can say that topology management can make tasks easier in MANETs, but it is a difficult task in its own right. A better idea might be to try to observe the given MANET environment and try to learn about it, rather than to try to actively manage it. One could learn about the actual topology at hand, or one could learn about general properties of the network, like node density or mobility. Information obtained in such a way could be a part of an adaptive algorithm, which might give good performance without managing topology.

### 3 Functions

In this section we seek to define a set of functions which are of interest for running networks. These functions are chosen because of their relevance for real needs of real networks. We seek sharp (and somewhat abstract) *definitions* for these functions, because we wish to view them as instances of “swarm intelligence”. That is, the BISON project seeks to use large numbers of microscopic “agents”, moving and acting on the network structure, in order to realize useful network functions as collective behaviors of the “swarm”. Hence the function needs to be expressed in a way which is as independent as possible from any notions of the underlying network structure.

We divide our functions into “basic” and “advanced”. The distinction is not precise; rather it is a useful way of organizing the work, involving many different functions, in the BISON project. Basic functions are viewed as ‘simpler’ than advanced functions. Also, they may reasonably be expected to be realized by a single, suitably crafted, swarm. Advanced functions are best viewed as being built from the coordinated operation of more basic functions.

### 3.1 Basic functions

#### 3.1.1 Routing

The task in routing is to find "good enough" paths. We assume that there is a source  $S$  which initiates the request for a route, and one or more destinations  $D$ . Then the routing function is to find one or more good enough paths from  $S$  to  $D$ . Typically there is some goodness criterion (small delay, low number of hops, etc) for the path or paths — that is, not just any path is acceptable.

By this definition, routing is distinct from transport. Of course, the path-finding function is most commonly used to enable data traffic, ie transport. However, the two are distinct. For example, there are routing schemes which first find a path, and then send the data; and there are other schemes (for instance, ant-based or stochastic routing schemes) which allow  $S$  to send data before finding an entire path. For example, with stochastic routing, at each node, and for each destination, all neighbors have a weight, which may be interpreted as a probability. Data can then be routed according to these probabilities — and no node knows a whole path. A more extreme case of the decoupling of routing (finding paths) and data-sending may be found in [26]: here the data are sent without *any* knowledge of paths; and the node mobility is used to find the destination  $D$ . Thus, in general, there is some flexibility in the timing relationship between routing and transport; and the two functions are considered to be distinct.

We summarize this basic function using a flow diagram:

$$(S, D) + \begin{pmatrix} \textit{goodness} \\ \textit{criteria} \end{pmatrix} \Rightarrow \begin{pmatrix} \textbf{routing} \\ \textbf{function} \end{pmatrix} \Rightarrow \textit{path(s)} \quad (1)$$

This defines the routing function in terms of inputs and outputs. Note that the means by which the function is accomplished is not part of the definition.

#### 3.1.2 Search

Consider "resources" distributed over a network. This term can include many things: people, files, storage capacity, information, devices, etc. Again there is a source  $S$  which initiates the search, based on a need to find the resource or resources. When the search function is successfully performed,  $S$  should receive a listing of network resources meeting  $S$ 's criteria, along with their locations (one or more network addresses).

The specification of the resource can be exact (eg, a specific person or file), or not (files with information about avocados). For an exact search, a single location is normally wanted. For a more generic search,  $S$  usually wants many resources (hits). Hence for the generic case  $S$  must also specify termination criteria for the search — typically, a number of hits.

Again we represent the function with an input-output diagram:

$$\begin{pmatrix} \textit{resource} \\ \textit{specification} \end{pmatrix} \Rightarrow \begin{pmatrix} \textbf{search} \\ \textbf{function} \end{pmatrix} \Rightarrow \begin{pmatrix} \textit{resources,} \\ \textit{locations} \end{pmatrix} \quad (2)$$

Here “resource specification” includes not only criteria for a hit, but also criteria for terminating the search.

### 3.1.3 Discovery

Discovery is a special case of search. Consider a person walking in a city. This person may be in search of some resource, such as a shoe store. Yet at the same time, all humans are constantly receiving input about what lies in their immediate environment. They also — invariably — filter this input, keeping some information from reaching the conscious “foreground”. This combination of receiving and filtering unsolicited input is what is meant here by discovery. We believe that this function, as defined here, will be an important part of future mobile wireless devices, supplementing the five human senses with digital radio sensing.

Hence discovery differs from the generic search function, defined above, in two ways: (i) the resources of interest are specified very broadly — perhaps by stating what is not of interest, ie, what is to be filtered out; (ii) the locations of interest are “local” ones. The point is, the searcher in this case simply wants to know what is available in his/her near neighborhood. So discovery, defined in this way, is a search that is (i) extremely broad in the resource specification, but at the same time (ii) narrow in the set of locations considered.

The input-output diagram:

$$(\text{filter}) + \left( \begin{array}{c} \text{neighborhood} \\ \text{specification} \end{array} \right) \Rightarrow \left( \begin{array}{c} \text{discovery} \\ \text{function} \end{array} \right) \Rightarrow (\text{resources} + \text{locations}) \quad (3)$$

Note that some generic searches also have a “neighborhood specification”, in the form of a time-to-live (TTL) — to be applied in the event that the requested number of hits is not found. Thus both functions (search and discovery) have the same logical elements. Furthermore, the discovery function is one of the less challenging instances of searching, since one is more or less guaranteed results, and a simple flooding-like mechanism is quite suitable. Hence — although discovery promises to be an important function in future, wireless, context-sensitive systems — it will not be viewed as a separate function in the BISON project. That is, further work and discussions will simply refer to the search function, and discovery will only be mentioned specifically where it is relevant as a special case of search.

### 3.1.4 Monitoring

The common understanding of monitoring involves two functions: measurement and processing. The purpose of monitoring is to “measure” the state of health of the network and of its functioning. Here the word “measure” is in quotation marks, because one does not usually have an instrument which directly measures health — however it is defined. Instead, one makes some “raw” measurements, and then processes those measurements in some way so as to come up with one or more health indices. These indices in turn can be either local, or global. BISON is interested in distributed or decentralized monitoring. In this context, distributed monitoring means the production of local or global health indices, obtained from local measurements, without recourse to a centralized mechanism. Here “local” health indices are

those pertinent to a neighborhood: not necessarily just one node, but smaller than the entire network.

In short, a distributed monitoring function must embody a rule for "interpreting" (processing) local measurements, so as to give one or more health indices as output. This rule must be implemented in a distributed fashion. There can be multiple health indices, not just because there are multiple neighborhoods, but also because there can be more than one index for each neighborhood.

Measurements may also be nontrivial. For instance, one may wish to know the delay incurred in sending a packet over a link. This is not a measurement that can be localized at a single node. It is then perhaps useful to define "raw" data as single-node quantities; everything else, all the way to the desired health indices, is processed data. Then one might view the monitoring function as simply the processing part:

$$\begin{pmatrix} \text{raw} \\ \text{data} \end{pmatrix} \implies \begin{pmatrix} \mathbf{monitoring} \\ \mathbf{function} \end{pmatrix} \implies \begin{pmatrix} \text{health} \\ \text{indices} \end{pmatrix} \quad (4)$$

Note that distributed monitoring then becomes a form of distributed processing — but not in the same sense as that implied by "grid computing". It is in fact a form of "collective computation", as described below. In spite of this, we will continue to list the monitoring function as a distinct function. Our reasons are that (unlike for discovery) monitoring is both challenging, and highly important for all types of network operation.

We note that the BISON project will explore a "synergistic" approach to monitoring. Here the idea is that a CAS, implementing another function (eg, routing), will in the course of its activities generate data which are either directly useful as health indices, or useful after further processing. For instance, certain properties of the pheromone traces left by routing ants may be used to yield indices indicating the local 'health' of the routing function. This approach is logically consistent with the picture given here: it is a form of distributed collective computation (by, eg., the ants), yielding local or even global health indices.

### 3.1.5 Collective computation

The task is to compute some function  $f(\dots)$  whose inputs are the states of all the nodes, and whose output(s) are the function  $f$ . The standard solution is to pick a node, send all the state information to that node, and let it do the computation. But this obvious solution is a centralized solution, requiring global coordination. Here we use the term "collective computation" to denote solving the same problem in a distributed fashion.

$$\begin{pmatrix} \text{node} \\ \text{states} \end{pmatrix} \implies \begin{pmatrix} \mathbf{collective} \\ \mathbf{computation} \end{pmatrix} \implies f(\text{node states}) \quad (5)$$

The "link-state" routing algorithm [50] qualifies as an example of collective computation. The task (computation) is to find a complete set of shortest paths; and it is accomplished without the singling out of any central node. The nodes' (initial) states are their knowledge of their neighbors. Then all the nodes circulate the information they have until they converge to a common map of the entire network. That is, the converged state is a state in which all nodes have

knowledge of all the other nodes. From this, each node can (individually) compute shortest paths from it to all other nodes. So this is a collective computation — but a rather “greedy” one, as it depends on all state information reaching all nodes. It is also slow, and so not well suited for a dynamic network.

Another example is the Byzantine agreement problem [43]. Here the function  $f$  is a global function: the correct (binary) signal, representing a consensus among the ‘honest’ nodes, to be determined even though an unknown fraction of the nodes are ‘dishonest’, and perhaps even conspiring to defeat the honest nodes. The problem is collective computation, because no node has global knowledge — the solution is to be obtained by message passing among equal nodes.

Note also that one can take the node states to be measured raw data, and the function  $f$  to be the health indices. It is then clear that distributed monitoring (as defined above) is simply a type of collective computation.

Finally we note that ‘collective computation’ is often called ‘aggregation’ in computer-science literature [51]. The meaning is essentially the same as that defined here. We suggest the use of a less traditional phrase here, simply to avoid the collision with another meaning of ‘aggregation’ which is central to biological modeling, and a main theme of our project. In biology, aggregation refers to the physical gathering of objects; this is (for example) what amoebae in slime mold colonies do when they have too little food.

Many common forms of collective computation do require a form of aggregation (of information) — for example, the computation of local averages, or voting. Hence it is not surprising that the computer-science meaning of the word collides with the meaning from biology. Since there is little chance of confusion, in this document we use ‘aggregation’ to mean the same as ‘collective computation’.

### 3.2 Advanced functions

The “advanced” functions to be studied by BISON differ from the “basic” functions in their complexity. They involve the integrated performance of multiple, more basic functions. Hence they are not well suited for expression in terms of input-output diagrams. Instead — in addition to a description, which seeks to define these advanced functions with some precision — we offer a preliminary ‘decomposition’ of the advanced functions, in terms of more basic functions.

One function belongs on the ‘decomposition list’ for all of the following, namely, topology management. This function however resists categorization as “basic” or “advanced”: it requires some basic functions in order to be implemented, and yet it may be used to support other functions, which may be either basic or advanced. It seems in fact that virtually any non-trivial function to be realized on a network may be enhanced by the possibility of choosing (at least partially) the network topology. Hence we defer our discussion of topology management to Section 4.

We note that the three advanced functions discussed below have historically been considered almost solely in the context of overlay networks built over the Internet. The reason for this is that these functions involve the sharing, and hence more effective utilization, of various types of capacity (processing, storage, bandwidth) present at the participating nodes. The collective



capacity (processing, storage, bandwidth) of all the PCs collected to the Internet is enormous. In contrast, ad-hoc networks tend to be small (and confined to the research stage) — and the typical mobile terminal participating in an ad-hoc net has (again, historically) had relatively limited capacities of all three types.

### 3.2.1 Distributed processing

Here the capacity to be shared is processing power. Sharing processing over an overlay network is also called “grid computing”. This name evokes an analogy to an electrical power grid: one connects to the grid and taps off the power one needs.

This picture (distributed processing = “grid” computing = a utility) also helps to clarify the fundamental distinction between collective computation and distributed processing. That is, in the latter case there is little coordination between the processes going on at the different nodes — it is close to “brute parallelism”, and so best suited for processing tasks that can be broken into nearly-independent pieces and so run in this fashion. In other words, the emphasis is on the utilization of highly distributed processing power, with the attendant practical constraint that the resulting parallel processing must seek to minimize the interprocessor communication needed to support the computation. In contrast, with the “basic” function termed collective computation, local processing capacity is not significant: the computation is done primarily via messaging, and typically using small messages.

It is clear from the above that several functions are necessary to implement this advanced function.

- (i) Whole jobs are generated at participating nodes, and must be subdivided into workable pieces (termed here partial jobs) before they can be distributed.
- (ii) A load balancing function is crucial: one wants to effectively utilize available capacity by holding the local load as close to capacity as possible. This load balancing function, in turn, must take into account not only the discrepancy of the current state from uniform utilization, but also the cost (in time) of transporting partial jobs over the links of the network.
- (iii) Results for partial jobs (belonging to a single whole job), obtained at a large number of nodes on the network, need to be gathered in some fashion that eventually transports them back to the initiating node.

Function (i) here is carried out at a single node. Function (ii), in contrast, is a good candidate for application of CAS-like mechanisms. Function (iii) is likely best coordinated with (ii): the gathering function basically must invert the distribution that was done in (ii).

### 3.2.2 Distributed storage

Here the capacity to be shared is storage capacity. This is the principal difference from the previous function. Distributed storage may also be thought of as a utility, in this case analogous

to distributed warehousing: participating nodes seek to utilize efficiently their shared pool of data storage capacity.

The more basic functions needed to implement a distributed storage system differ somewhat from the previous list. One major reason for this is that, with distributed storage, the gathering step is initiated by the user, rather than by the nodes providing the capacity. And the user will likely not want to wait to retrieve the warehoused data. This means that the *reliability of on-demand retrieval* will be an important consideration here — in contrast to the grid computing case (where retrieval is typically not on-demand by the source).

Some basic functions involved are:

- (i) Subdivision (of files, data sets, etc) is still needed, although here the task is relatively simple.
- (ii) Load balancing is again the crucial, and challenging, task.
- (iii) Because of the need for reliable on-demand retrieval — along with the unreliability of the individual nodes — there needs to be some strategy and mechanisms for replication of the stored results. This strategy may seek to employ information, in the form of (for example) statistical correlations of downtimes among the various nodes. Gathering such information is a distinct function (in fact, an example of collective computation).
- (iv) Retrieval of the distributed data is also of course a needed function.

It seems clear that functions (ii)–(iv) need to be coordinated, while (i) is again a single-node function. Load balancing and reliable replication are good candidates for swarm-intelligence techniques.

### 3.2.3 Distributed content

The distributed capacities utilized here are storage and bandwidth. The utility analogy is that of a distributed library, rather than a distributed warehouse. That is: the stored material is not “owned” by any one node — instead, the most essential sort of sharing involved is that of content or information. Content may be injected into the library by any participating node, and requested by any other. Hence the distribution mechanism is quite different from that in the previous two functions. It follows that the “retrieval” task is also quite different, and in fact much more challenging: a given node, seeking content, must search through resources which the seeking node itself played no role in placing on the network. Hence the seeking node normally has no historical connection to the sought resources.

Furthermore, the specification of the sought resources is typically much broader in the case of the library than is the case for the warehouse. In the latter case the seeking node wants a specific object (whose placement was initiated by that node). In the former (library) case, one may seek a specific object (which other nodes have placed on the net). Or, one may seek many objects (hits), specified in a broad way by criteria such as keywords.

For these reasons, search over a distributed information base, built up via decentralized contribution, is a challenging task. Furthermore, it is of great practical interest. Because of this

combination of interest and challenge, a great deal of work has been addressed to this problem. Also, a number of supporting functions have been invoked.

Some of the basic functions which may be used to support this advanced function are:

- (i) Indexing. It is likely to be useful to store, at some or all nodes, not only content, but also information about where other content may be found. Indexing is frequently used in current peer-to-peer content-sharing systems — both working systems (eg, Gnutella [25]), and those under study in research.
- (ii) Again, a replication strategy may be employed. A good strategy will place copies of content in a way that facilitates subsequent search. A working example of an effective replication strategy is that used in the Freenet [22] system.
- (iii) One needs to route queries (Q routing) from an initiating node, in a way that efficiently finds satisfactory content without overloading the network's links.
- (iv) Each node must test for search criterion satisfaction, so that it can decide how many hits (if any) are found in its content (or index).
- (v) Information about hits must be routed back to the source of the query ('Reply routing' or R routing).
- (vi) Testing for the termination of the search (via sufficient number of hits, or expiration) must be done. This function is easily implemented if the queries do not branch. If they do, some coordination is required. This function, plus (iii) through (v), make up the "basic" search function.
- (vii) Retrieval of a selected subset of the hits is also needed.

It seems clear that functions (ii), (iii), and (v)–(vi) need to be coordinated, while (i) and (iv) are single-node functions, and (viii) may be accomplished using a point-to-point routing mechanism. Load balancing and reliable replication are good candidates for swarm-intelligence techniques.

Note that not all of the above functions are necessary. For example, the early Gnutella system used no indexing, nor did it have a replication strategy.

One might embellish the above list even further. For example, we may wish that the distributed-content system shall return replies which do not satisfy the search criterion, and yet which may be judged — based on other types of criteria — to be of interest to the originator of the query. An example of this is the Amazon.com criterion of statistical correlation: "Users who sought [query Q] have also often been interested in [. . .]." Or the novel content may arise from introducing some degree of 'noise' into the search criteria, so that the search wanders in information space. This wandering may allow content which is 'similar enough' to the sought content to be captured by the search — if in fact such content is also present on the network.

We may give a name to this function as follows:

- (viii) A recommendation function may be used to allow information, which does not strictly satisfy the presented search criteria, to be presented to the searching node. The recom-

mendation may be thought of as arising from the network itself — from meta-information, in the form of structure or correlations present in the stored content.

### 3.3 Topology management is a function

In this document, we have already modified the picture presented in the Technical Annex, in a nontrivial and promising way. That is, we now cease to view structure as a given (albeit dynamic) set of constraints on the problem to be solved (function to be implemented). Instead, we will explore the possibility of *managing* — and therefore *choosing* — the topology of the network. We have discussed the possibilities for doing this, both for overlay networks and for wireless ad-hoc networks.

Furthermore, we insist that topology management be accomplished in a distributed fashion, without recourse to centralized mechanisms. We will in fact investigate CAS mechanisms for managing topology. Hence topology management itself must be added to our list of functions. This function might be considered 'basic', in that certain schemes are remarkably simple, and the topology management function is used to support other functions. Hence — recalling that the distinction is as much practical as it is logical — we include topology management in our list of basic functions.

Other implications of this new picture are discussed in the next section.

## 4 The Interplay of Structure and Function

In previous parts of this document we have seen that structure has important implications for the performance of functions. Therefore it is very important to develop protocols that can construct and/or maintain a given topology in a dynamic environment. Like in an ecosystem, the definition of the environment includes all other structures and protocols that operate in the same network.

In this section we argue that protocols for topology maintenance and algorithms implementing functions can mutually benefit from each other and can potentially form quite complex inter-related structures, defined in terms of dependence relations. These interdependent structures can result in savings in cost and improved performance when viewing the system of structures and functions as a whole.

In this approach, topology maintenance and creation becomes a function itself, at a lower level, for supporting implementations of other functions like search or broadcasting. In a truly modular system, containing a basic set of protocols for maintaining simple topologies and functions, it would also be possible to generate more sophisticated or specialized topologies, and thereby support other functions on demand.

We first present a conceptual framework in which these relationship can be expressed. Subsequently we illustrate the idea through a relatively simple, but still non-trivial example.

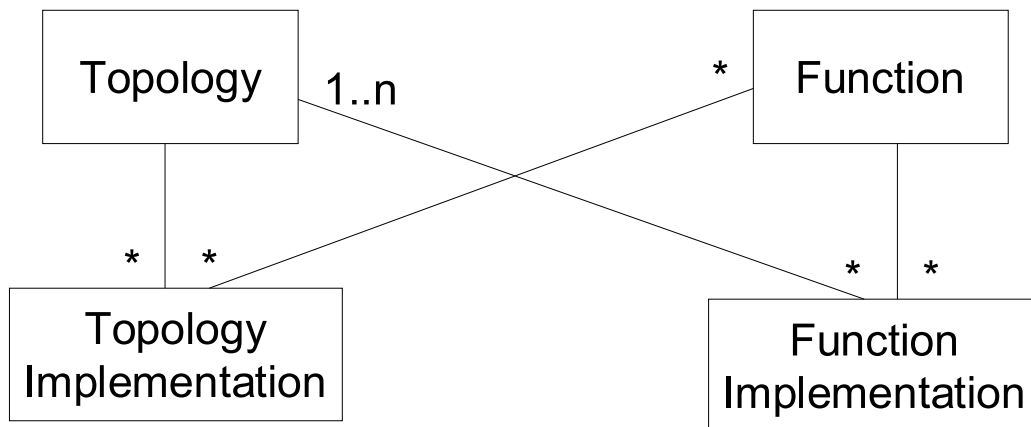


Figure 2: Conceptual structure of the relationship between topology and function.

## 4.1 Conceptual Framework

The basic concepts and their relationship are illustrated in Figure 2. In this framework, both topology and function are understood as an abstract interface which can have possibly many different implementations. Since we are talking about fully distributed systems, an interface can be expressed in terms of an API available at each component. For example, the interface of random topology would provide each node with a given number of randomly drawn neighbours for sending messages. The potentially very sophisticated way these neighbours are made available is hidden at this level.

Let us elaborate on the meaning of the symbols. The links define relationships which will allow the flexible building of complex modular systems. The link between the interface and its implementation is trivial, it represents the “implements” relationship. The other two links represent “supports” relations. In other words, a function can possibly support a topology implementation, and a topology can support a function implementation. Following the UML notation, \* means zero or more, and  $1..n$  means 1 or more classes participate at the given side of the connection. In words, a function can support many topology implementations (but possibly none) and the same topology implementation can rely on many functions (but possibly on none). The interpretation of the other link is similar.

The figure is almost completely symmetric. The only symmetry-breaking feature is that each function implementation needs at least one topology to work on. This is the only fact that indicates that topology is at a lower level than other functions. But this does not mean that topology implementation cannot build on functions—as the figure indicates, and as we will see later through examples.

**4.1.0.1 Topology Supports Function Implementation** This is the usual relationship in all structured overlay networks discussed earlier. The normal setting is that there is a protocol which maintains a topology—such as a ring with shortcuts, or a torus—and the implementation of the search or routing function makes use of this topology.

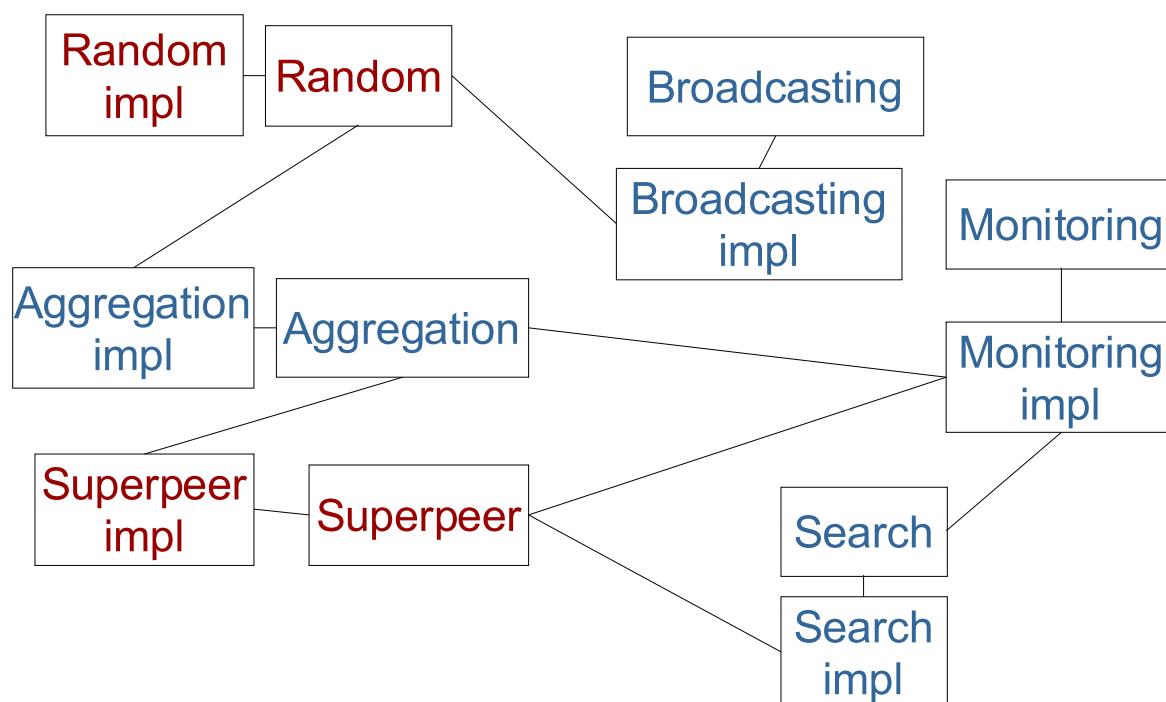


Figure 3: An example of a modular system.

**4.1.0.2 Function Supports Topology Implementation** In this direction there are much fewer examples; but, in our view, research into this problem has great potentials. In the case of the examples we are aware of, there is no clear-cut separation between function and topology implementation, and the degree of modularity is limited (see related work section).

## 4.2 An Illustrative Example for Modularity

We illustrate the framework through an example, shown in Figure 3. This figure does not represent an existing complete implemented system; but all connections are feasible, and based on research results.

The basis of the system is formed by the random topology, which does not make use of any other functions or topologies. It is used as a bootstrapping topology; also, when the system is up and running, it serves as a last safety net. The main role of the implementation of this topology is to be very stable, robust and scalable, more so than that of any other function. Even though we argued earlier that maintaining a truly random topology is nontrivial, in a practical setting any topology would suffice that can support broadcasting and aggregation, the task of the bootstrapping topology as shown in the chart. In particular, one can relax the strong constraint of randomness and concentrate on the requirements of dissemination and robustness, as discussed in the section about structure. Again, the point is that the topology of choice has to have an extremely robust and reliable implementation.

The implementation of broadcasting (perhaps in epidemic-style) is based on the random topol-

ogy. In this system this function is an “end-user”, no other components depend on it, but clearly, broadcasting could also serve as a building block for other components.

Aggregation is a key function in distributed systems [51]. Aggregation is also based on the random topology in our example system. Actual implementations of aggregation based on random topologies can be found in [33].

Perhaps the most interesting part of the example system is the fact that some functions support topology creation. In particular, aggregation is used by the implementation of the super-peer topology. One way of doing so is to use aggregation to find the best  $n$  or best  $k\%$  of the participating nodes, and promote them to super-peers. A proactive version of such a protocol could continuously maintain such a super-peer topology, which would offer robustness. The scalability is inherited from the scalability of aggregation.

The role of super-peer topologies—a traditional relationship—is to facilitate search, as we indicate in the diagram. However, the same topology can be exploited for other purposes as well, to maximize its utility. It can be used to support monitoring, one of the most important functions in overlay networks. Super-peers can collect data about the peers they serve, for example, and answer queries from the monitoring implementation. At the same time, aggregation itself can be exploited for monitoring. We can conclude that, in our example, monitoring is almost “free”—if we assume that we need the the other functions anyway.

### 4.3 Related Work

In our discussion of related work we focus on cases when topology is supported by a function which is again supported by another topology. These cases are rare because most systems follow the “one or more functions supported by a topology” pattern, which is clearly only the first step towards a really flexible and modular framework we sketched above.

The examples we are aware of are the following. Koorde [30], as we already discussed, is based on a de Bruijn graph on top of ring. In fact, the ring is maintained by an “off-the-shelf” protocol, Chord [48], by the same authors. In this sense we can say that Koorde uses Chord in a modular fashion in that the ring created by Chord provides the basis for creating a more sophisticated topology.

Another interesting work which is also based on Chord is [20], where the authors describe a mechanism for creating on-demand trees over a Chord architecture, for the purpose of broadcasting. Here a topology (tree) is created using Chord search (a function) as a building block, an idea that fits nicely with our framework. We can express this approach compactly as follows: (Chord search)  $\rightarrow$  (tree)  $\rightarrow$  (broadcasting). (Here the arrow is the “support” relation, as defined above.)

We mention as a possible extension that on-demand trees can also be used for aggregation. The work [5] builds on-demand spanning trees created by a flooding-style broadcasting which, as we have just seen, can be supported by another function like effective search. The compact expression for the work in [5] is then: (broadcasting)  $\rightarrow$  (spanning tree)  $\rightarrow$  (aggregation).

Astrolabe [52] uses an automatic leader election algorithm, which is in fact very similar to the random graph based aggregation approach we sketched. The basic functional building block in Astrolabe is aggregation, which is supported by a tree topology. The tree maintenance however

is supported by random gossip on top of a local random graph which embeds the tree.

Finally, a general robustness problem raised by the need for low degree topologies for routing and search [39, 30] could be nicely addressed with our approach by adding a basic robust random layer which could serve as a safety net and a bootstrapper for more interesting, perhaps constant degree topologies.

## 5 Conclusions

This report presents the problems to be addressed by the BISON project. It presents them in considerable detail, and includes discussion of the current state of the art, as well as more abstract notions (definitions, and mathematical results) that are more or less time-independent.

All of the problems addressed by BISON may be described as: “implement function  $F$  on structure  $S$ ”. This expression, although simple, contains an implicit way of thinking, namely: take structure  $S$  as given, ie, as a boundary condition for the task to be done, and then solve the problem of implementing the function  $F$  on this given structure.

In this report we have altered and expanded this simple picture. Structure (topology) still retains a primary role, since no function can be realized without some given structure being present first. We then introduce the function *topology management*, which invalidates the simple assumption that topology is always ‘given’. Instead, *some* topology is given. But this topology may then support some functions, which in turn include or support topology management, ie the generation of new topologies, built on top of the given one.

We have looked at two different types of dynamic and challenging structures, namely logical overlay networks (assumed to be built over a stable, wired net such as the Internet), and mobile, ad-hoc networks (MANETs — which face the additional challenge of having to build and maintain their own, wireless, infrastructure). These two types of structures are distinguished by the distinct challenges they face: the former gets an infrastructure ‘for free’, which is far from the case for the latter.

Because of these distinct challenges, the possibilities for topology management are considerably more limited for the MANET than for the overlay network. Nevertheless, the basic idea is applicable in either case — as discussed in detail in this report.

One of the basic tasks of BISON is to seek to *define behavior*. The functions to be implemented are a form of behavior. This report seeks to define these functions with a level of precision that is sufficient to enable the borrowing of ideas about behavior from biological systems. That is (see Deliverable D02), we interpret network functions such as routing, search, or content sharing, as collective behaviors, which are the result of the interaction of many small, simple ‘agents’ acting on the network. An important goal of the BISON project is then to design the small agents which will bring about these collective behaviors; and biology, with its huge richness of collective, self-organized behaviors, is to be the source of inspiration. In order to see useful parallels from an examination of biological systems, it is essential to render the desired behavior in a sufficiently clear and abstract form that it can be *recognized* also in biological systems. The ‘Functions’ section of this report presents the desired functions in this abstract form.

The set of problems presented here represents a very wide range, with high relevance to prac-



tical problems in the real world. Meaningful progress on even a fraction of these problems would yield important benefits for the future operation of dynamic networks. Deliverable D02 will present the basic tools — complex adaptive systems — which will be employed to address these problems.

## References

- [1] ACM. *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, San Diego, CA, 2001. ACM Press.
- [2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002.
- [3] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of ACM SIGCOMM '96*, August 1996.
- [4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
- [5] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer network. submitted for publication.
- [6] B. Bensaou, Y. Wang, and C. C. Ko. Fair medium access in 802.11 based wireless ad hoc networks. In *Proceedings of Mobile Ad Hoc Networking and Computing (Mobihoc 2000)*, pages 99–106, 2000.
- [7] Christian Bettstetter and Christian Wagner. The spatial node distribution of the random waypoint mobility model. In *Proceedings of the 1st German Workshop on Mobile Ad-Hoc Networks (WMAN02)*, 2002.
- [8] Ljubica Blazevic, Levente Buttyan, Srđan Capkun, Silvia Giordano, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Self-organization in mobile ad-hoc networks: the approach of terminodes. *IEEE Communications Magazine*, June 2001.
- [9] Ljubica Blazevic, Silvia Giordano, and Jean-Yves Le Boudec. Anchored path discovery in terminode routing. In *Proceedings of The Second IFIP-TC6 Networking Conference (Networking 2002)*, May 2002.
- [10] Ljubica Blazevic, Silvia Giordano, and Jean-Yves Le Boudec. Self organized terminode routing. *Journal of Cluster Computing*, April 2002.
- [11] J. Boleng. Normalizing mobility characteristics and enabling adaptive protocols for ad hoc networks. In *Proceedings of the Local and Metropolitan Area Networks Workshop (LANMAN)*, 2001.
- [12] J. Boleng, W. Navidi, and T. Camp. Metrics to enable adaptive protocols for mobile ad hoc networks. In *Proceedings of the International Conference on Wireless Networks (ICWN02)*, 2002.

- [13] Erwin R. Bonsma. Fully decentralised, scalable look-up in a network of peers using small world networks. In *Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, Orlando, July 2002.
- [14] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2:483–502, 2002.
- [15] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowston. Topology-aware routing in structured peer-to-peer overlay networks. In Schiper et al. [46], pages 103–107.
- [16] C.-C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *Proceedings of IEEE SICON'97*, pages 197–211, April 1997.
- [17] Sergei N. Dorogovtsev and J. F. F. Mendes. Evolution of networks. *Advances in Physics*, 51:1079–1187, 2002.
- [18] Olivier Dousse, Francois Baccelli, and Patrick Thiran. Impact of interferences on connectivity in ad hoc networks. In *Proceedings of IEEE Infocom 2003*, San Francisco, April 2003.
- [19] Olivier Dousse, Patrick Thiran, and Martin Hasler. Connectivity in ad-hoc and hybrid networks. In *Proceedings of IEEE Infocom 2002*, pages 1079–1088, New York, June 2002.
- [20] Sameh El-Ansary, Luc Onana Alima, Per Brand, and Seif Haridi. Efficient broadcast in structured p2p networks. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, February 2003.
- [21] A. Ephremides, J. E. Wieselthier, and D. J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. In *Proceedings of IEEE*, volume 75, 1987.
- [22] Freenet. <http://freenet.sourceforge.net/>.
- [23] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), February 2003.
- [24] Mario Gerla and Jack Tzu-Chieh Tsai. Multicluster, mobile, multimedia radio network. *ACM-Baltzer Journal of Wireless Networks*, 1(3), 1995.
- [25] Gnutella. <http://gnutella.wego.com/>.
- [26] Matthias Grossglauser and David N. C. Tse. Mobility increases the capacity of ad-hoc wireless networks. In *INFOCOM*, pages 1360–1369, 2001.
- [27] Piyush Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, March 2000.
- [28] Zygmunt J. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of the IEEE International Conference on Universal Personal Communications*, 1997.

- [29] Márk Jelasity and Maarten van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, October 2002.
- [30] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, February 2003.
- [31] Kazaa. <http://www.kazaa.com/>.
- [32] Jon Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.
- [33] Wojtek Kowalczyk, Márk Jelasity, and A. E. Eiben. Towards data mining in large and fully distributed peer-to-peer overlay networks. Technical Report IR-AI-003, Vrije Universiteit Amsterdam, Department of Artificial Intelligence, Amsterdam, The Netherlands, May 2003.
- [34] Michael Krivelevich and Benny Sudakov. Pseudo-random graphs. preprint.
- [35] Ching Law and Kai-Yeung Siu. Distributed construction of random expander graphs. In *Proceedings of The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'2003)*, San Francisco, California, USA, April 2003.
- [36] Meng-Jang Lin and Keith Marzullo. Directional gossip: Gossip in a wide area network. In Jan Hlavička, Erik Maehle, and András Pataricza, editors, *Dependable Computing – EDCC-3*, volume 1667 of *Lecture Notes on Computer Science*, pages 364–379. Springer-Verlag, 1999.
- [37] László Lovász. Random walks on graphs: A survey. In Dezső Miklós, Vera T. Sós, and Tamás Szőnyi, editors, *Combinatorics, Paul Erdős is Eighty*, volume 2, pages 353–398. János Bolyai Mathematical Society, Budapest, 1996.
- [38] László Lovász and Peter Winkler. Mixing of random walks and other diffusions on a graph. In P. Rowlinson, editor, *Surveys in Combinatorics*, volume 218 of *London Math. Soc. Lecture Notes Series*, pages 119–154. Cambridge University Press, 1995.
- [39] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2002.
- [40] Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.
- [41] Napster. <http://napster.com/>.
- [42] Mark E. J. Newman. Models of the small world. *Journal of Statistical Physics*, 101(3-4):819–841, November 2000.
- [43] Alessandro Panconesi. Lecture notes in theoretical distributed computing. Technical report, Kungl. Tekniska Högskolan, Stockholm, December 1997.

- [44] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)* [1], pages 161–172.
- [45] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer-Verlag, 2001.
- [46] André Schiper, Alex A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors. *Future Directions in Distributed Computing*, number 2584 in *Lecture Notes in Computer Science*. Springer, 2003.
- [47] SETI@home. <http://setiathome.ssl.berkeley.edu/>.
- [48] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)* [1], pages 149–160.
- [49] D. Sun and H. Man. Performance comparison of transport control protocols over mobile ad hoc networks. In *The 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communication (PIMRC)*, September 2001.
- [50] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, London, 1996.
- [51] Robbert van Renesse. The importance of aggregation. In Schiper et al. [46], pages 87–92.
- [52] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2), May 2003.
- [53] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998.
- [54] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, Los Alamitos, CA, March 2003. IEEE Computer Society Press.