



BISON IST-2001-38923

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

Evaluation plan

Deliverable Number: D04
Delivery Date: September 2004
Classification: Public
Contact Authors: Geoffrey Canright, Andreas Deutsch, Gianni Di Caro, Frederick Ducatelle, Niloy Ganguly, Poul Heegaard, Márk Jelasity, Roberto Montemanni
Document Version: Final (February 22, 2005)
Contract Start Date: 1 January 2003
Duration: 36 months
Project Coordinator: Università di Bologna (Italy)
Partners: Telenor ASA (Norway), Technische Universität Dresden (Germany), IDSIA (Switzerland)

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



Abstract

The BISON project aims to develop new technological approaches to network problems. These approaches must be evaluated, in such a manner that they can be compared with other approaches, and with one another. In this Deliverable we present a detailed plan for how this evaluation is to be done. Our presentation addresses three aspects of evaluation. In Section 2 we address the question of the *context* of evaluation—that is, which kinds of tests or experiments are useful for understanding and predicting the performance of a given approach. Then, in Sections 3 to 5, we discuss *figures of merit* for all of the functions in the scope of the BISON project—whether or not these functions have yet been addressed by BISON research. These figures of merit give a prescription for the quantitative evaluation of the ‘goodness’ of any given approach to performing the function; and they have been and/or will be used in the evaluation of new approaches developed by BISON. Finally, in Section 6, we offer working definitions for what we call ‘*nice properties*’. These properties (specifically, adaptivity, robustness, and scalability) are—at least in the first two cases—(i) present in biological systems, (ii) desirable for engineered systems, and (iii) (arguably) achieved more readily by engineered systems which are based on a decentralized CAS approach. Our definitions for nice properties fit neatly with the engineering figures of merit in the earlier sections; they also allow for a quantitative component in the discussion, comparison, and evaluation of the selected nice properties.

Contents

1	Introduction	5
2	Methodology for evaluation	6
2.1	Aspects of methodology: the environment	7
2.2	Aspects of methodology: evaluation criteria	8
2.3	Aspects of methodology: failures	9
3	Figures of merit: General	9
4	Figures of merit: Basic services	10
4.1	Routing	10
4.1.1	Measures of effectiveness	11
4.1.2	Measures of efficiency	12
4.1.3	MANET scenario characteristics	13
4.2	Search	14
4.3	Routing + search (path management)	14
4.4	Monitoring	16
4.5	Collective computation/aggregation	19
4.5.1	Environment	20
4.5.2	Failures	21
4.5.3	Evaluation criteria	21
4.5.4	Modeling, simulation, deployment	22
4.6	Topology management—overlay networks	22
4.6.1	Environment and Failures	22
4.6.2	Evaluation criteria	23
4.6.3	Fixed target topology	23
4.6.4	Fixed target properties of topology: the random graph	23
4.7	Topology management—wireless networks	25
5	Figures of merit: Advanced services	26
5.1	Load balancing	26
5.2	Distributed processing	28
5.3	Distributed storage	29
5.4	Distributed content	31

6	Nice properties from swarm intelligence	34
6.1	Self-organization	34
6.2	Adaptivity	36
6.3	Robustness	39
6.4	Scalability	40

1 Introduction

There is a natural tension in the BISON project, coming from its breadth of scope. The theoretical aspects of BISON are rooted in theoretical biology, the theory of complex systems, and theoretical computer science. On the applied side, BISON seeks to develop protocols and services which offer significant practical benefits for networked computing and telecommunication systems. Tension results from the scientific distance between these extremes. This tension is a positive aspect of BISON, as it stimulates ideas in a way not possible for projects with a much narrower scope.

In this Deliverable we address the very practical question of how to evaluate algorithms and protocols developed by BISON. This question has an almost purely engineering flavor: one wishes one or more figures of merit for any given approach, which are so defined as to allow for (i) quantitative measurement, from theory, simulations, or hardware implementations, and (ii) comparison with other approaches—both those coming from BISON, and those coming from the wider research community. The BISON project will use these figures of merit for evaluating promising new approaches, for selecting the best ones, and for comparing these with the best existing approaches.

The practical question of engineering performance can however also be phrased in a less nuts-and-bolts form. That is, there are significant performance benefits to be expected from using swarm-intelligence approaches. These benefits may be loosely termed “nice properties”. Nice properties are properties exhibited by biological systems—systems which have evolved, survived, and flourished, in spite of a dynamic, unpredictable, and evolving environment. Given the dynamic and evolving nature of today’s and tomorrow’s networked systems—which we can view as a form of environment—nice properties are also highly desirable for engineered, non-biological systems which shall perform needed functions in this environment.

Hence a task of BISON, also addressed in this Deliverable, is to select a working list of nice properties, and to provide working definitions of these properties. These definitions should satisfy the same criteria as the engineering definitions for figures of merit: definitions of nice properties should allow for (i) quantitative measurement, and (ii) comparison with other approaches.

Our working definitions for nice properties are formulated in such a way that they provide a natural bridge between the biological viewpoint and the practical, engineering viewpoint. In particular, we will see that the definitions of nice properties make explicit reference to other (engineering) figures of merit. Specifically, we will express the three nice properties of adaptivity, robustness, and scalability in terms of the *insensitivity* of figures of merit to various kinds of environmental challenge. Such definitions make sense both from the biological perspective and from the engineering one.

An important part of an evaluation plan is a choice of *procedure* for doing the evaluation. That is: one must address not only the question of *what* is to be evaluated, but also the question of *how* it is to be evaluated. This latter question is addressed in Section 2.

2 Methodology for evaluation

Evaluation of solutions to given problems is at the heart of all research which is oriented towards technological application. Hence the question of *how* to evaluate proposed solutions is a major aspect of the general question of methodology in applied research. More specifically: one must choose a set of *goodness measures* for any studied solutions; but in addition, one must choose a *procedure* or *context* which prescribes how these goodness measures are to be evaluated for a given solution or solutions. Such choices must be made in order that one can evaluate and compare solutions to technical problems. We discuss such methodological questions in this section.

We can identify a few key concepts that form the framework for evaluation.

environment The environment is the set of assumptions that we can use when solving a problem. That is, the environment is the 'givens' of the problem; as such, we may in fact equate the environment with the detailed statement of the problem context. For problems relevant to the BISON project, the environment will typically include a *network topology*. This topology may or may not be *dynamic*, with nodes and links appearing and disappearing according to some prescription. This prescription may be derived from the known behavior of some algorithm which builds and maintains an overlay topology; or it may come from the more or less unmanaged (but again known) behavior of networks such as ad-hoc or peer-to-peer nets. The environment also consists of *services* which are available from the network, and the definition of the *cost* of these services, perhaps along many dimensions (time, storage, money, etc). Besides having a cost, some services may also have a (hard or soft) *constraint* on usage, for example an upper bound. Also, in the case where a service is provided at differing levels of quality, there may be a constraint in the form of a lower bound on service quality. Finally, the environment will typically further include a specification of the *demand* or *load* which the problem solution must be able to handle.

problem The environment defines the context or constraints under which a problem must be solved. 'Solving' a problem then means doing (within these constraints) 'something' as well as possible. In other words, solutions seek to maximize some goodness criteria. Hence the problem can be cast as an optimization problem, where the goal is to optimize a certain set of numeric evaluation functions, which we called *goodness measures* above. (Later in this document, we will present goodness measures in terms of two sorts of criteria: *figures of merit* and *nice properties*.) In a single problem there can be many goodness criteria, reflecting diverse aspects of performance. The criteria can possibly be conflicting. Because of this, it may not be possible to state a given problem unambiguously as a single, multidimensional optimization problem: it may not be clear what weight should be given to each of several distinct and possibly conflicting criteria. Nevertheless, in principle, the problem may be defined as a set of goodness criteria which are to be maximized in the context of a given environment.

solution A solution (not necessarily optimal) to a given problem, in a given environment, is one or more algorithm implementations that make use of services in the environment (which include CPU cycles, communication, storage and other services), possibly taking

their input from the environment, and producing some output or result. The goodness criteria are interpreted over the output and the cost of the solution.

The evaluation of solutions involves considering certain environments, and evaluating the goodness measures for each solution to the problem in those environments. The performing of such evaluation may serve either or both of two, somewhat distinct, goals: (i) *Science*: we wish to *understand* why a given solution gives the performance that it does. This understanding is to be tested by its ability to *predict* performance in other experiments. (ii) *Technology*: we wish to *use* the best solutions that we find from our evaluation procedure. Of course, we also in this case want the results from various experiments to be predictors of performance in the real environment. However, the prediction need not be based on any rigorous scientific criteria; it may instead be based on a set of results, plus a significant dose of intuition or guessing. For technology, the overriding criterion is that the new solution works well. Prediction of future good performance may then be based on little more than past experience.

In the ideal case, one can find solutions to interesting problems in meaningful environments, where such solutions are *provably* optimal or which are provably *good enough*, that is, practically usable. This kind of result clearly serves both the scientific and the technological purposes. Often, however, rigorous proofs cannot be obtained, even for the case when a good enough solution is sufficient. In these cases, one can seek to produce empirical evidence in a way that allows *induction* (as opposed to *deduction* through mathematical proofs) in a scientifically justifiable and verifiable way. Then we still have predictable performance.

In many fields of applied research, none of the above criteria are fulfilled—that is, neither proofs nor verifiable inductive generalizations exist. We would say that these fields are in the *fact collection* phase. Fact collection—ie, the collection of empirical results—should ideally be carried out in such a way that the results collected can serve in the future as a basis for induction, and (possibly) ultimately deduction. Implementation of a new technology can however be carried out based solely on empirical results, if those results are sufficiently thorough, and the risk is sufficiently small. Hence fact collecting can in principle support both scientific understanding and technological decision-making; and it is after all the default approach when more rigorous methods are not possible.

2.1 Aspects of methodology: the environment

When conducting research, the environment has to be abstracted away from reality (the intended real-world application environment) because it is impossible to consider all possible details. Abstraction, however, is a sensitive issue. That is, for a great number of problems and environments, it is far from obvious how to best apply abstraction to the given environment. There arises often the well-known conflict between manageability and realism. One aims in such cases to start with manageability—a level of abstraction that allows clear results to be obtained and understood—and to subsequently move towards realism.

For some problems, the environment is simple, and therefore easy to handle. For example, in the case of a very simple problem such as the sorting of a list of numbers, the environment consists of only a few services: a read-write operation on an array of numbers, a comparison operator, and some extra working storage space. These have some fixed cost. All other costs can be safely ignored. In the opposite extreme, the environment is extremely complicated. For

example, suppose the problem is web search. Here we are dealing with a highly non-trivial and time-dependent structure of interlinked documents, where the documents themselves have rich syntactic and semantic structure, and where the link structure and the document contents are highly interrelated in complicated ways. Here, finding the right abstraction is a serious research task in itself.

Especially in the case of complicated environments, the levels of abstraction can be different. It is possible to work with *theoretical* (possibly stochastic) models that are useful for mathematical analysis. It is possible to perform empirical analysis using simulations, that work on fixed instances of the environment. These can be generated through *generators* that are themselves models of the environments, but are used indirectly. It is also possible to use *traces* or *copies* of the real environment, that are collected with an appropriate tool. This case is already too specific to be used even for induction: when individual instances of the environment are studied, the results contribute to fact collection. Finally, it is possible to actually *deploy* the solution in the real environment. The advantage of doing so is clear: one has not abstracted away *any* detail from the real environment. Hence one can use deployment results to test the performance of a solution in the real environment for which it is intended. Such results are also useful for testing the adequacy or inadequacy of results obtained from more abstract versions of the environment, such as theoretical models or simulations. However, deployment of a solution in the real environment is methodologically at the same level (fact collection) as working with traces of the environment, with the added disadvantage that such experiments are even less reproducible.

We comment briefly on the practice of *benchmarking*. Benchmarking means testing different solutions for the same problem, on fixed instances of the environment that are either generated or collected (see above). The purpose is then to compare competing solutions over a fixed combination of problem + environment. This type of comparison contributes to what we call here fact collecting. If the environment is sufficiently realistic, then benchmarking may be used as an aid in decision-making: one gets a sense of which approach (solution) may be expected to perform best in the real world. However, because of the constrained nature of benchmarking experiments, their results are unlikely to aid in further scientific understanding of the performance of the tested solutions. Hence, we believe that care should be used in choosing a suitably realistic abstraction of the environment for benchmarking experiments.

2.2 Aspects of methodology: evaluation criteria

For a well defined problem, one needs a set of goodness criteria, as described above. In many cases the criteria are quite natural and easy. For instance, when the problem is to sort numbers, one requires correctness, speed, and minimal extra storage. In general, in the field of computational complexity, a sophisticated framework of evaluation criteria has been developed, including characterization of speed and storage through upper and lower bounds in the limit as the problems size grows to infinity, and classes like polynomial (P) and nondeterministic polynomial (NP) running time algorithms. While it has its problems, much is understood about this framework.

In other problems, set in a more complex environment, evaluation criteria are less obvious, and more numerous. For example, consider again the contrasting example of web search. A web

search engine involves, among other things, a crawler, an inverted index, a text-relevance algorithm, (possibly) a link structure database and a link-analysis algorithm, techniques for parsing Boolean queries or other types of queries, a ranking algorithm, and a user interface. For each of these components, we can define performance criteria. These criteria will typically include the common ones—speed and storage—but many others will be involved. Furthermore, many of these criteria can easily be interdependent. That is, design choices about one component will affect the nature of the goodness criteria for other components. Furthermore, with the web search problem, there is an important evaluation criterion, *user satisfaction*, which is rather difficult to capture.

These comments apply with equal force to a problem in BISON's agenda, namely, a peer-to-peer distributed-content system. This problem has much in common with web search; it lacks however the hyperlinks, and has none of the centralized functions (crawling, indexing etc) which are used by a conventional search engine. Nevertheless, the problem of distributed content presents a variety of interlinked challenges, as reflected in a number of different performance measures—again including the very difficult but important measure of user satisfaction.

2.3 Aspects of methodology: failures

Failure models are part of the environment specification; however, due to their importance, we treat them separately here. Obviously, for failure models the same observations hold as for the environment in general, as discussed previously.

For the sake of organization we can classify failure into three categories: (i) catastrophic failure, where a sudden and unusual event dramatically reduces the performance of services in the environment; (ii) benign failure, where the services show some constant level of unreliability; and (iii) malicious attacks, where the environment adaptively changes in ways which explicitly work against the optimization of the evaluation criteria. These different kinds of failure will have differing degrees of relevance for various problems. Benign failure, in the form of unreliable nodes, is a common mechanism for the problems considered by BISON.

3 Figures of merit: General

In evaluating any solution for a problem, one seeks to evaluate a set of goodness criteria. Ideally, these goodness criteria will be expressed in a form that allows for quantitative evaluation. The numbers obtained for a given solution may be termed its 'figures of merit'; and, by extension, the (quantitative) goodness criteria which are defined for the problem may also be termed the problem's 'figures of merit'. In the following, we will define figures of merit (often abbreviated FOM or FOMs) for the set of BISON problems.

Figures of merit, as defined broadly in the previous paragraph, may appear to span all possible measures of goodness for any given problem. However, we will see below that, in considering the idea of 'nice properties' for solutions, we find that those nice properties on our short list all have the form of "meta-FOMs". That is, they describe the variation or sensitivity of one or more already-defined figures of merit for the problem. Hence, although in principle nice

properties (if quantified) also fit our definition of a figure of merit, we feel that the distinction between these two types of goodness criteria is conceptually useful. Hence, in this document, we retain the term 'figure of merit' for those quantities which may be obtained directly from a single test (involving a single instance of the environment), and the term 'nice property' for those goodness measures which involve sensitivity of one or several FOMs to variations in the environment.

In the following sections, we will present FOMs for all of the functions in the scope of BISON. In addition, we will sometimes (ie for some functions) discuss other aspects of evaluation. For example, for many functions we will include some discussion of nice properties. Since nice properties are only defined in rather general and abstract terms in Section 6, we have found it useful to discuss them in more specific terms for many of the functions below. We will also often include some discussion of the kinds of variations of the environment that are expected to give meaningful and useful tests of proposed solutions for accomplishing the given function. Our discussion of the environment may include a description of the relevant environmental variables; possible constraints to be applied; and failure modes to be tested.

4 Figures of merit: Basic services

4.1 Routing

Work on the routing function in the BISON project has been focused almost exclusively on one of the outstanding technological challenges for routing, namely, routing in a mobile ad-hoc network (MANET). For this reason, discussion in this section of figures of merit for routing will be couched in terms of the specific problem of routing in a MANET.

In the last 10 years, mobile ad hoc networks (MANETs) have been extensively studied, both from practical and theoretical points of view. Therefore, a number of metrics have already been identified as standard measures of protocol performance, as well as for the characterization of the networking context in which the protocol performance is measured. The 1999 RFC 2501 [5] of the Internet Engineering Task Force (IETF) MANET working group, in which routing protocol performance issues and evaluation consideration are carefully considered, has been widely accepted and followed in experimental practice. The discussion that follows is directly based on the content of that document, but at the same time takes into account some of the specific and unique aspects that are addressed by the BISON project, and also some of the developments in the domain that have occurred since 1999.

The service deployed by any network strictly depends on its context of utilization, and so do the metrics adopted to measure the performance of the routing protocol which is being used. For instance, a network that has to carry real-time voice/video communications is expected to have quite different characteristics from one which is supposed to deliver only text messages. That is, the characteristics of the expected *quality-of-service* (QoS) define in turn what is interesting to observe and measure. However, as matter of fact, so far there has been very little movement of MANET functionality from the research phase to real world deployment. In spite of the ever growing number of research papers on the subject, there are very few examples of real-world MANET implementations. Most studies rely on simulation. This lack of well-defined real-world target applications for MANET implementations, which can serve as

reference, has a twofold effect. On one side, research studies have had the possibility to span over a wide range of topics, while on the other, a significant amount of arbitrariness characterizes the scenarios that are considered to assess the quality of the algorithms (this is not usually the case for simulation studies for wired networks for instance, since most of the considered networks have a counterpart in real-world applications). Apart from a few studies that have considered the deployment of QoS and real-world physical/mobility scenarios, the majority of the studies have considered only *best-effort* traffic in *flat open spaces* for nodes that move according to quite *random, unstructured mobility patterns*. In this first phase of development and study of the algorithms we are moving along the same lines, relying on the use of Qualnet as simulation tool (see deliverable D11 [18]). Nevertheless, we are also investigating scenarios with different characteristics.

In the following we will describe different aspects of the evaluation of MANET routing protocols. We write about measures of effectiveness and efficiency, about test scenario characteristics, and about evaluating the behavior of algorithms in dynamic environments.

4.1.1 Measures of effectiveness

For best-effort traffic, the following *quantitative metrics* are usually considered to assess the general *effectiveness* of the performance of routing protocols in MANETs:

Data delivery ratio The fraction of correctly delivered versus sent packets.

End-to-end packet delay This involves a cumulative statistical measure (e.g., average, percentiles) of the delays experienced by packets traveling from sources to destinations.

Route acquisition time Here one seeks cumulative statistical measures of the time elapsing between the start of a data session and the moment the first data packet can be sent.

Proper delivery order In addition to these basic measures, also the *percentage of out-of-order delivery* is often considered. This measure is of particular interest to transport layer protocols such as standard TCP implementations which prefer in-order delivery. In fact, systematic arrival of out-of-order packets would lead to repeated re-transmissions, and thus reduce the source effective sending rate, causing a decrease in the achievable throughput. This is potentially a major problem in MANETs, in which routes between two end-points are not expected to be stable—neither in terms of the end-to-end delay, nor in terms of the number of hops or the involved nodes. On the other hand, the use of versions of TCP more tailored for MANETs (e.g., [2, 3]) can partially overcome the problem. However, how to deal with reliability at the transport level is still an open and quite controversial issue in MANETs [1]. Therefore, we have chosen not to consider this aspect as part of the metrics used to assess the quality of the routing algorithms proposed in BISON (we are using only UDP-based applications). This choice is also motivated by the fact that the algorithms developed in BISON are fully distributed multipath algorithms, so that they naturally result in out-of-order packets at the transport layer if packet reordering is not explicitly dealt with at the routing layer.

Time variations The above mentioned performance metrics are based on average and/or cumulative measures. They do not provide information about the specific *time response* of

the algorithms. For instance, the same delivery ratio D can be obtained by providing either a stationary delivery rate during the whole interval of measurement, or by providing a strongly fluctuating delivery rate that eventually averages to D . Clearly, the two situations are not equivalent. In the first case all users receive a comparable service, in the second case, users joining the network at different times will likely receive a different quality of service. This points out the need to study the response of the algorithms over different time scales. Time fluctuations, for a given quantity, may be quantified in terms of a statistical measure such as the mean square deviation from the mean.

Delay jitter In addition to the metrics mentioned so far, we also consider delay jitter, since it is usually a metric used in QoS applications. Measurement of delay jitter also provides a measure of the stability of the algorithm's response to changes in the network topology. Delay jitter measures packet delay variation, which is in turn a measure of distortion of the traffic profile of a data session. (In principle, network-induced delay jitter can always be removed at the receiver, but only at the expense of possibly large buffers and potentially increased delays). A number of definitions for delay jitter have appeared in the literature (e.g., [21, 10]), with each definition tailored to the QoS characteristics of the specific problem under consideration. We will mainly focus on a packet-level definition, which considers the average of the difference of the interarrival time between the two last pairs of received packets. That is, if the last three packets are received at respectively at t_3, t_2, t_1 , the session's jitter is calculated as the arithmetic average of the values $(t_3 - t_2) - (t_2 - t_1)$ for all triplets of received packets. This way of calculating jitter is the one implemented in Qualnet, the simulation software that we use, and also in a number of real-world routing devices.

On-board power Power consumption, for battery-powered mobile terminals, is a critical issue in MANETs; therefore energy utilization of an algorithm is an important value to track. In this case both per-node and cumulative energy utilization measures can be used. However, we do not explicitly consider this aspect at the moment in our algorithms.

4.1.2 Measures of efficiency

In addition to these general measures of performance, also internal measures of performance have to be considered. These measures indicate the efficiency of the algorithm, and have to do with the amount of overhead created, which is in turn also related to energy utilization. In other words, these measures involve the *cost* (in terms of resources used) of a solution, while the previous measures quantify the performance.

Routing overhead due to multiple hops The IETF document proposes in the first place the ratio of *the number of data bits transmitted versus the number of data bits received*. This is in fact a measure of the route efficiency: if the algorithm constructs longer paths (in number of hops), each data packet has to be transmitted more times, and more overhead is created. Favoring shortest paths might not be the best idea though (as was pointed out in [7] for example). Also, such a metric will normally favor single path algorithms compared to multipath algorithms, while multipath routing can have many advantages [19].

Control overhead Another efficiency metric is *the number of control bits transmitted versus data bits delivered*, which measures the bit efficiency of the protocol in delivering data packets.

Total routing overhead A measure combining both previous metrics is *the number of control and data packets transmitted versus data packets delivered*. This measures the algorithm's channel access efficiency, which is important in MANETs since the cost of channel access is normally quite high. Also this metric favors shortest paths, so that previous remarks apply also here.

Increasing the overhead normally improves the algorithm's performance, as more information about the network is available. In stressful situations an overload of control packets can interfere with data packets and actually decrease the performance. It is therefore important to study the efficiency metrics in a range of different situations, and see how they evolve for increasing levels of complexity of the MANET scenarios, e.g. increasing number of nodes or mobility.

4.1.3 MANET scenario characteristics

The relative performance of MANET routing algorithms depends strongly on the *network configuration*—that is, the chosen instance of the environment in which routing is to be performed. There are a large number of parameters defining the network configuration; to really get a clear picture of the performance of an algorithm, it is necessary to run experiments for different settings of all these parameters. A list of essential environmental parameters influencing the performance is as follows:

Network size This is defined by the number of nodes; it is one of the most important aspects, since scalability with respect to the number of nodes is an important issue.

Network connectivity This may be quantified in terms of the average number of neighbors per node. Sparse networks, with low connectivity, will have topological characteristics which are different from those of densely connected networks (see [8]).

Network change rate This involves some measure of the speed with which the network's topology is changing. The network change rate is related both to network density and node mobility. Possible measures for this are the link failure rate, or the average link duration.

Link capacity This is the achievable transmission bandwidth in bits per second. This is composed of two aspects: the physical capacity of the node transmission channel, and the mechanisms used at the MAC layer to access the shared medium. In many cases, the choice of MAC mechanism can dramatically affect (ie, decrease) the physical capacity.

Data traffic patterns These are the time-space characteristics of the input traffic flows that actually define the routing task, together with the physical and topological characteristics of the network. Several parameters determine the characteristics of the traffic patterns: the inter-arrival time of the sessions at the nodes, the spatial distribution of sources and destinations, the duration of the sessions, the average size of data packets, the rate of packet generation, and so on. So far mostly uniform traffic patterns have been considered in MANET studies. Nevertheless, in the perspective of a possible use of MANETs

in realistic environments (e.g., as peer-to-peer networks for content exchange) it is also important to test non-uniform situations (e.g., hot spots and variable bit rates).

Node mobility This is a very important parameter. Nodes can in fact move according to different mobility models, and depending on this there can be more or less correlation between subsequent network topologies. Most simulation studies use the random waypoint mobility model [16], in which there is very little structure in the movements after the pausing points, but we are also using other models (e.g., Gauss-Markov [4]) in order to study the response of the algorithm to models with different degrees of correlation in node mobility. Each mobility model also has its own parameters, such as the minimum and maximum node speed for example, that define in turn the link failure rate that in a sense characterizes the mobility/difficulty of the scenario (see, e.g., [22, 17]).

Note that studying the variation of our various performance measures, as a function of variation of the above environmental parameters, allows one to evaluate nice properties such as scalability and adaptivity.

It is clear that the number of possible scenarios and metrics is quite extensive. In a sense, it is not reasonable to assess the general superiority of a protocol over another just according to few simulation results. On the other hand, if a precise context of application is defined this might reasonably reduce the range of the possible scenarios and allow for more meaningful comparisons.

4.2 Search

In the BISON project, work on the “basic” search function has been tightly integrated with work on the “advanced” service termed distributed content. Searching is one of the main challenges in a distributed content system. Furthermore—as we realized early on, when we did state-of-the-art assessments for the various functions—how searching is done, and also how well it is done, depends strongly on several other functions (replication, indexing, etc) which are implemented to support and enhance searching.

For these reasons, we have not considered search in isolation. Hence, discussion of performance criteria for search will be placed in the section on distributed content (below), under advanced services.

4.3 Routing + search (path management)

As noted just above, we have found it advantageous to consider the search function as a part of a distributed content system, rather than considering search in isolation from other functions. Another combination considered by BISON [25] is the combination (search + routing). Another term used to describe this approach is *resource path management*.

Resource path management involves the establishing and maintenance of an explicit route from a terminal entity to an end-system entity (e.g., peripheral equipment or information/content) in a network, while simultaneously considering the constraints and quality of *both* the route and the end-system entity. This integrated approach is in contrast to the traditional approach,

which is to first do the search for the best possible end-system entity, and then find the best hop-by-hop path from a given source to the location of this entity. This two-step approach, in which the search and routing functions have minimal interaction, can sometimes result in a bad path to an excellent end-system entity, when a good route to a good end-system entity is available.

Resource path management addresses the medium-term temporal characteristics of the path resources, i.e., how to use the available network resources (e.g. bandwidth, processing power, battery power, storage capacity) to establish paths that best meet the requirements of the offered traffic, on a time scale in the range of 100ms to some hours. The requirements include those cited in Section 4.1 for routing—for example, data delivery ratio, packet delay, delay variations (i.e. jitter), path setup time—and also the resource requirements for the end-system entity (for example, if the end-system resource is a printer, the requirements include bandwidth, delay, and type). These requirements, or quality measures, must then be combined into a single function which gives a quality measure for the total *resource path*. The resource path [25] includes all resources relevant to the transaction: client resources (describing those resources used by the terminal entity—eg, a client application such as a browser), transport resources (addressed by the routing function) and peripheral resources (characterizing the end-system entity).

A set of performance criteria for resource path management is given below. These are obviously interrelated and partly conflicting.

Path finding ability This is the obvious objective: if a path exists between a source and a destination in a network it should be found, and within given time constraints. There may be additional requirements, such as to find multiple node disjoint, or link disjoint, paths, for better resource utilization and resilience. A quantitative measure is the probability of convergence to an acceptable (or more restrictive: the optimal) solution, within given time constraints. The optimal value is 100%.

Resource utilization The network resources should be used efficiently. That is, the goal is to meet the given traffic and QoS requirements with the least possible resource consumption. This aim (of maximizing resource efficiency) should be maintained in the face of dynamic conditions, such as changes in the loading of the network, in the topology, and in the capacities of the network elements. A quantitative measure of resource utilization is the deviation between the optimal resource utilization (when the optimum is known) and the actual utilization given by the path management method. However, not all path management tasks are polynomial problems; hence such tasks will not reach the optimal solution within the given time constraints. (The latter are determined by the service requirements, and by the dynamics in the network environment.) Hence, an alternative, and less restrictive, quantitative measure of resource utilization efficiency is simply a modified version of path finding ability: the probability of finding a feasible solution, within the time constraints, such that the utilization of any resource pool, relative to its capacity, does not exceed a given upper bound.

Robustness and adaptivity In some services, temporal aspects such as continuity of service or negligible down times are of great importance. That is, the service quality should remain high at all times—even in the case of failures. As noted in Section 6, maintaining quality in the face of environmental failures (such as node failures) is defined as adaptiv-

ity for BISON purposes, while if the failure affects the CAS itself, then consistent service quality is called robustness. Loss of management information—ie, information which is propagated by the CAS system which is responsible for resource path management—is an example of “damage” to the CAS; such losses may occur due to failures or overload of network elements. A quantitative measure of robustness for the path management system is variation of the path finding ability, given a management information loss rate. One can for example compare the path finding ability with a given information loss to that for the no-loss case. The optimal value is then 100%, if the management system is totally insensitive to loss of information.

Priorities and fairness The path management system should provide a fair service, i.e. all users (or traffic flows) sharing the same service and dependability requirements (and so belonging to the same service class) should statistically receive the same QoS. A quantitative measure of the fairness is the ratio between the best and worse fit to prescribed service quality requirements for the user in the same service class. The optimal value is 100%.

Minimum overhead All management functions will consume some capacity from the resources that are managed. Such consumption includes the transmission of monitoring data, the collection and processing of measurement data, the processing of path management decisions, and the transmission of path updates. Hence overhead resource consumption is added both to processing units (network nodes) and to the transmission systems (bit transport over the network links). Quantitative measures for overhead include the number of management bits relative to information bits, and CPU cycles used for processing of management information. The ideal values of these overhead metrics are 0—obviously, we want to minimise the overhead.

4.4 Monitoring

Monitoring is an important tool used in the management of resources, and in the control of service levels (as specified, for example, in Service Level Agreements or SLAs). These two main purposes of monitoring reflect two different points of view: that of the service provider, and that of the service user. From a service provider point of view, monitoring is the supervision of *resource utilization and availability*. Examples of resources in a BISON context include information content (documents, video, music, etc.), processing power, link capacities, node bandwidth, and battery power. From a service user point of view, monitoring is the supervision of *service quality level*. Examples of services include transport of bits over a link, download of a file from a server, and audio or video interaction between peers. The definition of service quality depends on the specified service, and is typically measured relative to the user’s demand (including willingness to pay).

Monitoring is a general term which is also applied in many other fields. For example, a general definition of monitoring may be found in the field of environmental protection [24], where monitoring is defined as *the periodic oversight of a process, or the implementation of an activity, which seeks to establish the extent to which input deliveries, work schedules, other required actions and targeted outputs are proceeding according to plan, so that timely action can be taken to correct the deficiencies*

detected. Here the crucial term which captures the nature of monitoring is “oversight”. Monitoring involves the gathering of information. The “timely [corrective] action” in the above definition is triggered or guided by the results of the monitoring activity; but corrective action is not considered to be a part of the monitoring function. The “processes” which are monitored may be in principle any other function; and (again) the performance of this function may be evaluated in terms of resource utilization, and service level provided. Another important term in this definition is “deficiencies”. The monitoring activity is expected to detect and report deviations from a set of prescribed limits or constraints; these deviations are the deficiencies, ie, they represent failure of the monitored system to stay within prescribed limits. When the monitoring system reports such failures, we will say that it gives an *alarm*.

As noted above, the monitoring of system performance may be viewed from both a user perspective and a provider perspective. The user (consumer) is interested in maximizing the quality of the delivered service, while the provider is concerned with maximizing resource utilization, while delivering service within the given quality guarantees. The service quality constraints are part of a Service Level Agreement (SLA) between consumer and provider. In the SLA, the monitoring function should be clearly and unambiguously described, in order for the consumer and provider to have a common understanding of how the service quality is defined, and how this quality level should be measured.

Monitoring may be applied to any of a wide variety of functions. The definition of precise figures of merit for monitoring will then depend on which function is to be monitored. Nevertheless, we can define a set of general performance criteria for monitoring. These general criteria come directly from the general goals of monitoring: to *provide information* about the monitored system; and specifically to give *alarms* when a deviation is detected. Based on these simple objectives, we can define the following performance measures for monitoring:

True alarm reporting ability The monitoring function should report changes in network conditions, when these conditions fall outside prescribed limits. That is, it should report alarms, when alarm conditions occur. A straightforward quantitative measure of this criterion is simply the percentage of actual alarm conditions which are reported as alarms. One might assume that 100% is the optimal value for this criterion. However (see below), there may be cases where it is *not* desirable to report all true alarms.

True alarm promptness It is not sufficient that true alarms are reported; it is also important that they be reported without undue delay. In the simplest of situations, there is only one type of alarm, with a specified tolerable (threshold) delay for reporting. One may then seek to minimize the percentage of true alarms reported for which the delay exceeds this threshold; or one may take the average delay over threshold as the “cost” which is to be minimized. In more realistic situations, there are multiple types of alarms, each type having its own acceptable threshold delay, and each type having a different penalty for exceeding the threshold.

True alarm location In many cases, alarms apply to local rather than global conditions. Here, by “local”, we mean, pertaining to a region of the network which is bigger than a single measuring point, but smaller than the entire network. Hence alarms (as with health indices, see below) may have location information associated with them. It may be desirable to know the location(s) of any true alarms, so that the management function (which is

not part of the monitoring system) can react properly. A quantitative measure for this criterion depends on defining a distance function for locations on the network. Given such a distance function, one seeks to minimize the average distance of the reported alarm location from the true location.

No false alarms The monitoring function should not send false alarms. This is a measure of correctness: a false alarm reports a deviation, when in fact the deviation has not occurred. A simple measure of this criterion is the percentage of total alarms reported which are false. The optimal value is zero.

Meta-alarm reporting A good monitoring system will also monitor itself. One important example of this is to monitor the frequency of reported alarms. If this frequency (measured over some time interval) exceeds some limit, then we might want the monitoring system to emit a “meta-alarm”. That is, alarms serve a useful function when they report relatively rare exceptions to normal, healthy operating conditions. When alarms are generated too often, they are no longer useful. This can occur, for example, if a system operates approximately at the threshold condition for the alarm, with constant fluctuations above and below the threshold. More serious conditions may drive the system well beyond the threshold, giving rise to continuous alarm reporting. In any case, the emitting of meta-alarms (reporting a too high frequency of alarms) may be a task of a monitoring system. The same quantitative criteria as for normal alarms (high percentage of true positives, no false alarms) can apply here.

Network health reporting Besides reporting alarms, the monitoring function system should make available other, more general types of information about the performance and functioning (“health”) of the monitored system. Such information may include variables such as the current level of resource utilization, the current distribution of resources, any recent transient failure of resources, and all kinds of service performance levels. The reporting interval should be flexible, dependent on the need of the management function. A quantitative measure of this health-reporting function is the instantaneous deviation of the reported value h_{rep} for health parameter h from its true value h_{true} ; or, a time average of $|h_{rep} - h_{true}|$ may be used.

We note that this reporting function (like the alarm function) involves more than simply performing local (point) measurements, and conveying the results to some management function. That is, monitoring requires the processing of many such point measurements $m_i(t)$ to obtain useful health indices $h(t)$. This processing is a form of aggregation—typically, $h(t)$ is a function of several local measurements, taken from many nodes, and so represents the health of a local region (perhaps also averaged over some time interval), or even the entire network.

Hence the BISON form of monitoring involves a decentralized form of aggregation, with the aggregated health indices (including alarms) reported to a management function. This latter function may itself be centralized—as in the “standard” picture—or it too may be implemented by a decentralized CAS system. In the latter case, it may be difficult to practically distinguish the monitoring system from the management system (even though the two *functions* are still in principle distinct).

Non-intrusiveness Monitoring, like any other function, has a cost associated with its performance. This cost should not be so large as to interfere with the resource utilization and service quality itself. For example, reporting an observed bottleneck should not contribute to making the bottleneck even more severe. This requirement (non-intrusiveness) may come into conflict with the requirement of giving a prompt and correct alarm on high resource utilization. A possible quantitative measure for intrusiveness is the difference between health indices $h(t)$ with and without monitoring.

We thus see that monitoring (as with other functions), even when viewed at this general level, involves a set of performance criteria that may easily be conflicting. Hence we do not propose a single, combined performance function to be optimized.

4.5 Collective computation/aggregation

'Aggregation' is the standard term in computer science for what we sometimes call 'collective computation' in a BISON context. We have used the latter term in those cases where the *computational* phenomenon of aggregation needs to be distinguished from the *physical* phenomenon of aggregation, as exhibited, for example, by slime mold colonies. In this section, there is no likelihood of confusion; hence we stick to the more standard term here.

Aggregation is not a single problem. Instead, it is a type of problem, or a set of similar problems [23]. In general, the goal is to compute some numeric function of the state of a distributed system, such as the average or maximal value, or the number of certain types of components. For such well-defined computational tasks, there is always the simple evaluation criterion of *correctness*. For our purposes (looking ahead to dynamic problems), we state the correctness criterion as demanding that the difference between the ideal and the computed value should be minimized. Other criteria describing cost are also applicable, as always.

We can identify two major dimensions along which we can separate different aggregation problems. The first one involves the proactive vs. reactive distinction. In the proactive problem, we want the aggregate value to be known at all times, independently of any events from the environment. In the reactive case, we want the aggregate value to be computed only as a response to a request from the environment. The second distinction is defined by whether or not we require the aggregate value to be known by all processes (nodes), or only by one. These two binary distinctions give rise to four sets of aggregation problems.

All of the four possible types of problems can have practical applications. In our research we focus on the case of proactive aggregation where all nodes know the result of the aggregate continuously. Clearly, this makes the evaluation criteria depend on time as well. For example, when the environment is changing fast, the cost may (or may not) increase; and the quality of the estimate of the aggregate value may decrease.

Since even the seemingly 'simple' problem of aggregation involves multiple, and likely conflicting, goodness measures, a full problem specification also requires defining which function of the various measures is to be optimized. This provides possibilities for even more variation in the set of aggregation problems. Focusing on time-dependent criteria as outline above, we can trade off cost for quality, we can try to minimize the peaks of cost, we can go for optimizing average quality or we can try to avoid valleys of the quality criterion. These choices can be

cast in the form of secondary criteria, over which a simple (possibly multicriteria) optimization problem is defined.

4.5.1 Environment

The services of the environment consist of computation and communication services. The computation services are instantiated by a possibly changing set of CPUs that can run one or more processes. Each process has an address. Assuming the existence of a suitable solution to the routing problem, the communication services can be assumed to allow any process to send a message to any other process, provided only that the address is known. In addition, for all processes, an additional input service is available that returns a numeric value whenever called. Aggregation will be interpreted over the return values of this input service over the set of active processes.

All the services have some cost and performance characteristics. The level of abstraction of the environment that instantiates these characteristics can be varied. As mentioned before, the crucial requirement is that the abstraction does not remove or ignore important features. We will use two models of the environment, for evaluation purposes.

Theoretical model. For the purpose of theoretical investigation and simulations, we will assume that the cost of sending a message successfully is constant, independently of the sender and recipient. In one version of the model, we assume that messages are instantly delivered. In a modified version, we additionally assume that messages can also be lost with some nonzero probability. The assumption of instant message delivery may seem to be an overly strong assumption; but in fact we argue that it is not. From the point of view of our specific solution (presented in deliverable D05 and in [14]), what is required is that a very small number of messages are delivered within a relatively long time interval. When applying a sufficiently short timeout for sending messages, we can very closely approximate the conditions expressed in this theoretical model. This reasoning is justified further through more realistic simulations, as will be mentioned later. Note however that for other solutions this assumption can be extremely unrealistic.

We will not consider the cost of the input service nor the computations. Again, this is justified by our specific solution, where the computational requirements are very small. Hence the only cost considered is that of communication.

Towards a real environment. For justification of the theoretical assumptions, we will implement our solutions and deploy them in a real wide area networking environment, PlanetLab. PlanetLab is “real” in the sense that actual Internet connections are used for transmitting messages in a planetary scale testbed consisting of a few hundred nodes. It is certainly not real in the sense of workload, churn, failure scenarios and malicious attacks, just to name a few factors. In other words, working with PlanetLab still needs a fair amount of modeling if these factors have to be considered.

The real environment is the actual deployment of the solution in systems used by real people to solve real tasks on the real Internet. This level is unfortunately not reached by academic institutions very often.

4.5.2 Failures

A key aspect of the environment is the nature and pattern of failures. The aggregation problem makes very few assumptions about the environment; it allows for basically any network that is capable of providing some message delivery service. Since the area of possible applications is large, the analysis of failures also has to be generic enough to be applicable to a wide range of situations.

One way of achieving this is to look at some artificial failure patterns that in a sense represent the worst case, so that they cover most real scenarios. For example, if we examine churn rates where, say, a randomly selected 50% of the processes are replaced in some short unit of time, then we in fact perform a worst case analysis over all scenarios where *at most* a randomly selected 50% is replaced, which greatly simplifies the discussion of the matter. The same observation holds for all sorts of failures discussed previously: catastrophic, benign and malicious failures.

If a certain solution cannot be proven to work well in these worst case scenarios, then it does of course not mean that the solution is not good. It only means that a more detailed specification of the environment is necessary. If the solution does work well, however, then no more details are needed.

4.5.3 Evaluation criteria

In this section we give specific details about the problems we plan to evaluate, except the choice of the aggregation function, which will simply be a black box numeric function over the state of the distributed system.

As mentioned before, we are focusing on the proactive aggregation problem where all processes want to know the aggregate value continuously. The criteria we define are the following.

communication cost To be minimized over an appropriate unit of time. In more detail, since the protocols we propose are based on passing messages of constant size, the simplest notion of communication cost is the number of messages passed. Another aspect of communication cost is its distribution over the nodes (processes). Maybe the simplest way of capturing this is the maximal number of messages processed (sent or received) by any process in a unit of time over the set of processes in the system.

local quality The difference of the estimate of the aggregation from the correct value (defined for each process). To be minimized. For example, in the case of averaging, we need the difference between the true average over the entire system and the local approximate value.

quality The average difference of the estimate of the aggregation from the correct value (defined for each process). To be minimized. This is in fact local quality averaged over all the processes.

quality gap The difference between the maximal and minimal estimations over the set of processes. To be minimized.

convergence speed The speed at which the estimates follow the correct value after the correct value has changed in the system. To be maximized.

These criteria define a multi criteria optimization problem over the set of solutions, where we can possibly apply weights to emphasize one criterion or another.

4.5.4 Modeling, simulation, deployment

Our approach emphasizes working towards analytical, quantitative results to describe the evaluation criteria (cost and quality). Since the environment specification for the aggregation problem is rather generic (that is, aggregation can be deployed in a wide range of rather different systems), we work towards results that are equally generic and thus fit many special cases. In other words, we have as few assumptions as possible about the environment. As a result, our model of the environment is simple and highly abstract; hence modeling the environment is not very difficult.

Simulation results at different levels of realism are still desirable to justify any simplifying assumptions made during theoretical analysis.

4.6 Topology management—overlay networks

In the topology management problem we are given a set of nodes (processes), that have an address. This address is sufficient to send the node a message; that is, we assume that the environment provides a routing service. We are also given a target graph over the nodes, which defines target links between the nodes. This graph can be defined directly [12], or indirectly through an evaluation function [13]. The problem is to make sure that each node knows the set of other nodes that are defined by the target links (perhaps along with some additional links), starting from some specified initial configuration.

In the most general sense, the topology management problem is a complex set of problems. It can be classified into topology construction and topology repair. Some solutions will solve both simultaneously.

According to the type of topology and intended application, we can apply different evaluation criteria. If the target topology is uniquely defined, we can apply some distance measure (eg missing target links), while if the topology is defined through an evaluation function, then we can use the evaluation function value itself. An example of the latter type is when we want to create a random topology, defined by eg low diameter, low clustering and possibly some other measures, where these measures can be applied to evaluate the solution.

4.6.1 Environment and Failures

The environment is very similar to that of the aggregation problem. We rely on the same communication service and computation service, and we apply the same abstractions of this environment to evaluate solutions. Accordingly, the notes on the failure models also apply for topology management as well, and the simulation and emulation methodology is also very similar.

4.6.2 Evaluation criteria

We define evaluation criteria for two kinds of problems. The first is the case when the target topology is given as an explicit graph, so that it is explicitly known which nodes each node should have as its neighbors. That is, when the target topology is an explicit graph, all the links are specified. The other kind of target topology is not explicitly defined; instead, only some of its properties are fixed (such as low diameter, etc). Here we will focus on the random topology. Before giving specific criteria for these two cases, let us discuss the common ones.

communication cost To be minimized over an appropriate unit of time. Similarly to aggregation, since the protocols we propose are based on passing messages of constant size, the simplest notion of communication cost is the number of messages passed. Another aspect of communication cost is its distribution. Again (as with aggregation), unevenness in the distribution of communication costs over the nodes may be measured in terms of the maximal number of messages processed (sent or received) by any process in a unit of time over the set of nodes in the system.

convergence speed The speed at which we improve the target topology. This is a derived criterion which assumes one or more other criteria, discussed soon, and is based on the change of these other evaluations in unit time. To be maximized.

dead links The number of links that point to nodes that are no longer in the system. This measure is suitable to capture the self healing properties of a protocol: after damage many links become dead, and we want to remove these. To be minimized.

4.6.3 Fixed target topology

In this case the main criteria we focus on are the following.

missing links The number of target links that are not included in the overlay topology. The missing target links are uniquely defined and therefore can be counted. To be minimized.

time to completion Beside the speed, we can also look at the time it takes to reduce the number of missing links to zero. Hopefully this time is finite. To be minimized.

This is a rather simple case, and evaluation is easy, due to the clear definition of the problem.

4.6.4 Fixed target properties of topology: the random graph

This case is somewhat more controversial. It might seem that a natural way of defining the target topology is requiring that each node links to a randomly drawn sample of a fixed size from the set of all nodes. Such a property would be very useful, especially when it comes to prove properties of applications that rely on such a random overlay network. There is a major problem however: if we can in fact draw random samples, then we know that the criteria for randomness are automatically fulfilled; but, in the likely case that we must *approximate*

randomness by some rule-driven procedure, we need measures of randomness that are often very indirect. What can be done is to specify statistical measures of randomness. Working evaluation criteria for randomness can then simply be the distance of the observed values of the statistics from the expected values for the random graph. Another way to generate evaluation criteria for this case is to evaluate how likely the measured statistics are if the graph is random, using statistical likelihood measures.

Another issue to keep in mind is that when an application uses the random overlay topology, eg to broadcast messages, or for aggregation, different aspects of randomness might become relevant. The study of which aspects are important for given applications is a separate research problem that we do not tackle here. We focus on statistics that are general and important enough for many different applications.

average path length Between two nodes, the shortest path is defined as the minimal number of links one has to traverse to reach one node from the other. Calculating the average of this length over all pairs of nodes gives the average path length. In random graphs this length is well described. The property of small path length is essential for many functions, such as search, or information dissemination. For a measured graph which is a good approximation to a random graph, the difference between the measured path length and the random-graph path length should be small.

clustering coefficient The clustering of one node is the number of links between any pairs of its neighbors, divided by the number of all possible such links. The clustering of a graph is the average clustering of nodes. The clustering of certain random graph models is well understood; hence (as with path length) the goal is to minimize the difference between the random graph value for clustering, and the measured value.

node degree auto-correlation Since our protocols not only construct but also constantly maintain (and thereby also change) the topology, it is of interest to examine how the topology changes through time, even in a constant environment. The autocorrelation of node degree is the correlation of the degree of a fixed node between different points in time, as a function of the time lag. It expresses whether node degree, for a given node, is stable, or fluctuating over time. This criterion is not strictly specific to random graphs, although in the case of random graphs it is especially useful. For example, measuring the node degree autocorrelation for all the nodes allows one to detect whether an unbalanced (and therefore undesirable) node degree distribution is caused by a fixed set of nodes, or is a global distributed property of the system, where no individual nodes are responsible for the pattern.

mean and variance of the degree distribution Node degrees define a random variable that is given by the degree of a random node. This random variable is of interest because recent result from the field of complex networks suggest that many properties such as robustness to failure and search performance heavily depend on it. This random variable is well understood in the case of different random graph models. In our case (equal number of random neighbors) the target distribution is a Gaussian distribution which defines the target values of the statistics of the degree distribution. We can measure the difference of the empirical values from these target values.

Finally we discuss briefly the question of visualizations. A visualization is not usually an evaluation; yet visualization can be a useful tool for the researcher. Visualization makes it easier for a human observer to estimate (possibly) many evaluations intuitively, and simultaneously, without defining an explicit number. While such estimation (via visualization) cannot serve as a substitute for explicit evaluation of quantitative goodness criteria, it can still be useful to give ideas towards that goal. In our case, one example of a visualization is plotting histograms of the node degree distribution of the topology. Another, even more indirect, approach is to visualize the network topology itself, perhaps after appropriate pre-processing.

4.7 Topology management—wireless networks

We consider wireless networks where individual nodes are equipped with omnidirectional antennae. Typically these nodes are also equipped with limited capacity batteries and have a restricted communication radius. Topology control is one of the most fundamental and critical issues in multi-hop wireless networks, since the topology that is established by the wireless nodes will directly affect the network performance. In wireless sensor networks, topology control essentially involves each node choosing the right transmitter power, so as to maintain adequate network connectivity. Incorrectly designed topologies can lead to higher end-to-end delays and reduced throughput in error-prone channels. In energy-constrained networks where replacement or periodic maintenance of node batteries is not feasible, the issue is all the more critical, since it directly impacts the network (battery) lifetime.

Unlike in wired networks, where a transmission from i to m generally reaches only node m , in wireless sensor networks it is possible to reach several nodes with a single transmission (this is the so-called *wireless multi-cast advantage*). This property is used to minimize the total transmission power required to connect all the nodes of the network.

A set of performance criteria for minimum power connectivity in wireless networks is the following one:

Connectivity This criterion indicates whether the methods proposed are experimentally able to provide a fully connected network or not. In case the topology provided by some algorithm does not have all the nodes in a single connected component, we would have a failure, since the target is to find a fully connected topology.

Power consumption This is the key measure for the algorithms we will propose. It indicates the total power consumption over the network in the power configuration found by the algorithms. Smaller overall power consumptions over the network can be associated with better algorithms, since the target of the optimization is to keep the total power dissipation at the minimal level.

Computation time Another important factor for minimum power topology algorithms is the computation time they require to provide an optimized power configuration. This figure of merit is important when the methods are to be used within a dynamic environment, where quick responses are required.

5 Figures of merit: Advanced services

5.1 Load balancing

Load balancing is a rather elementary function, which may be used in support of all of the advanced functions on our list. For this latter reason, we include load balancing in this section on advanced services.

The nature of the load varies among the three advanced services: for distributed processing it is work, measured in cpu time; for distributed storage and distributed content it is data, measured in bits or bytes. However for the discussion of load balancing, abstracted from its possible applications, the nature of the load is unspecified. We will call the load at node i ϕ_i .

The task of load balancing is of course to establish and/or maintain a uniform distribution of load over a set of nodes in a network. In a realistic scenario, the capacity of the nodes can vary. We take then the load ϕ_i to mean the load *relative* to capacity, and continue to call this quantity “load”, hence keeping the notation and the terminology compact.

Now we offer a list of figures of merit (FOM) for load balancing.

Convergence An obvious FOM is convergence. That is, given a static problem (fixed network with an initial load distribution, no injection or removal of load thereafter), a given approach should, eventually, converge to the correct (uniform) distribution $\bar{\phi}$. This is a binary (yes/no) criterion. The criterion of *correctness* may be defined to be the same as convergence (as defined here). That is, an algorithm which has the correct answer as an unstable fixed point—or which, for any other reason, can ‘recognize’ the correct distribution, but cannot find the correct distribution from a given starting point—will not be deemed to be ‘correct’.

Approach to convergence For dynamic problems, we want our approach to move towards convergence. That is, suppose an algorithm requires time t_c (on a given network) to converge; and suppose load is changed (injected or removed), and/or small topology changes take place, on a timescale less than t_c . This algorithm will of course not converge; but the question is, will it, on these shorter timescales, move towards the uniform state?

It is of course reasonable to assume that an algorithm which converges over time t_c will move towards convergence on a timescale $< t_c$. However it is quite possible that a given method’s rate of approach to converge is effectively zero over some shorter time scale. If this shorter time scale is comparable to the dynamic changes in the problem setting, then such an algorithm is useless.

Hence it is important to measure an algorithm’s *rate of approach to convergence* over the timescale t_{dyn} of interest. A simple way to do so is as follows. First define $\Delta_i \equiv |\phi_i - \bar{\phi}|$, and then let Δ be the average over nodes of Δ_i . At convergence, $\Delta = 0$. Hence, for a given start condition, $R(t_{dyn}) \equiv \Delta(t = 0) - \Delta(t_{dyn})$ gives a measure of the rate of convergence over a timescale t_{dyn} .

Convergence time The time to converge (to within some margin $\Delta \leq \epsilon$) is an inverse FOM, ie, a cost.

Total load moved The total load moved is also a cost.

Here an observation is in order. Suppose a system has a load distribution $\{\phi_i\}(t)$ at time t , and a new distribution $\{\phi_i\}(t+1)$ at time $(t+1)$. The problem is then that there is no unique answer to the question of what set of net load movements $\{\delta\phi_{net,i\rightarrow j}\}$ for each link ij has given rise to the change from $\{\phi_i\}(t)$ to $\{\phi_i\}(t+1)$. (Consider for instance moving load around a loop. Arbitrary amounts may be moved around the loop, without changing the load distribution at all.)

Nevertheless, in a real system, load is changed by transferring it over links. Hence any algorithm must specify not only the change in load distribution from one time to the next, but also how it was transferred. With this requirement satisfied, the quantities $\{\delta\phi_{i\rightarrow j}\}$ are nearly determined. Here we say ‘nearly’ because, even given, for a given link ij , the net amount transferred $\delta\phi_{net,i\rightarrow j}$, there are many ways to achieve this net transfer. A “smart” algorithm will only transfer this net amount, and no more. Nevertheless, there may be “less smart” algorithms, which transfer some load over a given link ij in *both* directions, in some time steps. For such algorithms, $\delta\phi_{net,i\rightarrow j} = \delta\phi_{i\rightarrow j} - \delta\phi_{j\rightarrow i}$. Then, allowing for the existence of “less smart” algorithms, we define the cost function for load transferred for link ij , in a given unit time, to be $C_{ij} \equiv |\delta\phi_{i\rightarrow j}| + |\delta\phi_{j\rightarrow i}|$. For “smart” algorithms, one of these terms is always zero, so that $C_{ij} = |\delta\phi_{net,i\rightarrow j}|$.

The cost to convergence is then $C_{conv} = \sum_{t=0}^{t_c} \sum_{links\ ij} C_{ij}(t)$.

Load moved in an interval For cases in which dynamics prevent convergence, we still must measure the cost in terms of load transferred. If the timescale of interest is again t_{dyn} , then the cost is simply $C(t_{dyn}) = \sum_{t=0}^{t_{dyn}} \sum_{links\ ij} C_{ij}(t)$.

(Convergence rate)/(cost) It is of course always important to compare benefits to costs. Hence a useful FOM, for the non-converging case, is to compare the rate of convergence to its cost: $R(t_{dyn})/C(t_{dyn})$. The same comparison may be made for convergence in the static case, substituting t_c for t_{dyn} .

Now we have a satisfactory set of FOMs for load balancing. They are however quite narrow, in the sense that each applies only to a given problem setting (topology + initial distribution + dynamics). It is of course important to know how well a given approach works for a *variety* of problem settings. Hence we define further performance measures, in the form of ‘nice properties’, in the following. That is, these measures assess the sensitivity (or insensitivity) of various specific FOMs to environmental change, damage, and growth in system size.

Sensitivity to dynamics For any of the above FOMs, one can define their sensitivity to some change in the problem setting. For example, if the rate of change of topology, or of injection and removal of load, is changed, how is the FOM of interest affected? Assuming we can parameterize the rate of dynamic change with parameter d , then measuring sensitivity amounts to measuring the FOM under differing values of d , and computing $\Delta(FOM)/\Delta d$. It is of course preferred that an algorithm be *insensitive*; hence the inverse of this measure is a good performance measure.

Sensitivity to topology Here we have in mind a “reference” topology, such as scale-free, random, ad-hoc, etc. Dynamics are assumed to take place in a way which preserves the

reference topology. Then the question is how well a given approach works over different reference topologies. Here we doubt that one can find a reasonable scalar measure for variation of reference topology. One can however define a standard set of reference topologies, and report the variation of any given FOM over this standard set.

Sensitivity to noise Noise is present in any real system—for example, in the form of dropped packets. Of course, error-correcting protocols will act to reduce such noise; but they require time. Hence it is a reasonable approximation to assume that, in a given “time step” of simulation time, some messages which are sent by the load-balancing protocol are lost. These messages may be load transfers, or communication. For either case, one can assume a percentage p that is dropped per time step, and measure the variation of any FOM as a function of p .

Robustness Robustness is on BISON’s list of nice properties. The working BISON definition equates robustness with insensitivity to damage—where here “damage” means damage to the CAS (which is responsible for accomplishing some function), and not damage to the environment.

Now we assume we have a promising distributed, agent-based load balancing system, and ask, How can we damage it? The agents in this case are simply the communication messages (eg, ants, or chemotactic signalling messages). Damage, in the form of loss of agents, is included in the previous item (sensitivity to noise). Hence, our working definitions of adaptivity and robustness give us the following distinction (which is not large): insensitivity to noise in the form of lost load packets is adaptivity; and insensitivity to noise in the sense of lost load-balancing communications packets is robustness. (In our ant-based load-balancing scheme, the ants perform both functions. However, noise in the form of lost ants may clearly be considered to be damage.)

Scalability A straightforward measure of the size of the load-balancing problem is the number of nodes N . It is common practice, and important, to measure the performance of an algorithm as N grows large. Scalability then refers to how the performance varies with N . Typically one seeks a simple scaling law which holds at large N . Both convergence time and the total load moved may be expected to grow with N . Comparing different algorithms then amounts to comparing such rates of growth.

5.2 Distributed processing

The BISON project is working actively on CAS solutions for the problem of load balancing. Load balancing is a particularly important function for both distributed processing and distributed storage. However, to date, of our three advanced functions, only the third—distributed content—has been addressed in its full form by BISON research. Hence, our discussions of both distributed processing (this section) and distributed storage (in the next section) will be somewhat abbreviated.

In this section, we consider the problem of distributed processing. As we have discussed earlier (D08), the BISON view of the problem of distributed processing is distinct both from grid computing and from traditional parallel processing. The difference is that, in the BISON case,

the participating peers are generally much more unreliable. Therefore, unlike traditional parallel processing where speed-up of execution time is important, here simply getting the job done is much more important. This is because participating machines are supposed to donate 'free' time for such work, and it is not their 'real' work. Keeping this perspective in mind, we define figures of merit for distributed processing. (Note: the FOMs of load balancing are also automatically applicable to distributed processing; hence we don't repeat them here.)

Multiple administrative domains and autonomy An important criterion for any distributed processing algorithm, operating over a resource-sharing peer-to-peer network, is that it respects the autonomy of resource owners. That is, the P2P distributed-processing protocols should not in any way override or violate each node's local resource management and usage policies. This criterion is in nature more of a constraint than a figure of merit, since we will reject algorithms which do not satisfy this criterion.

Heterogeneity Distributed processing should be platform independent, that is, it should be able to utilize peers involving a multiplicity of resources that are heterogeneous in nature, encompassing a vast range of technologies. This criterion may be either handled as a constraint—ie, a requirement of compatibility with a list of local systems—or as a figure of merit. In the latter case, one could study (for example) degradation of performance as a function of the various choices of differing types of local systems which the P2P protocols must work with.

Success rate per job assigned Due to the unreliability of the peers, typically each job is assigned multiple times. So, we have to calculate both the average case (the average number of repetitions required to ensure that the job is executed) as well the worst case, to evaluate the performance of the system. The number of repetitions is a cost; the success rate is then (roughly) its inverse, ie, (useful results)/(number of repetitions).

Response Time Although we noted above that expectations for response speed are lower for P2P systems than for grid or parallel processing systems, it is still of interest for evaluating purposes to compare differing P2P approaches according to an FOM such as response rate. This rate may be defined as the inverse of the average response time (a cost). We note in this context that it is also useful for the system itself to have an estimate of the average response time—either in terms of purely local estimates, or averaged over the system using aggregation. The point is then, given the average response time, a 'time-out' time can also be guessed—that is, the time when a node can conclude that the job sent has very little chance of being processed. Then a copy of the job can be resent. Thus a knowledge of the average response time can be used to avoid unnecessary repetitions, thus increasing the average success rate as defined above.

5.3 Distributed storage

In a distributed storage system, the contents (data) of a node are stored by other peers. A detailed discussion of the distributed storage problem can be found in Deliverable D08. Here we offer a list of figures of merit for distributed storage. A well designed distributed storage system should perform well according to the following basic criteria.

Availability In a system with perfect availability, information can be accessed 24 hours a day, seven days a week. Peer-to-peer distributed-storage systems will tend to fall below perfect availability, due to the unreliability of the participating nodes. A quantitative measure of availability can then be

$$\text{Fraction of Blocks available} = \frac{\text{number of blocks recovered within time } \tau \text{ from request}}{\text{number of blocks requested}}$$

Durability We wish that information entered into the system will last virtually forever. However—again due to the unreliability of the nodes—it is possible for information stored in the P2P system to be lost. A possible measure of durability can be

$$\text{Fraction of Blocks Lost} = \frac{\text{number of blocks not recovered within time } T \text{ from request}}{\text{number of blocks requested}}$$

Note that, although the above two equations are similar, availability and durability have two different connotations in the case of a transient P2P system. The availability FOM measures to what degree the data requested are readily available. It may for instance happen that the data requested reside on a peer which is presently down—giving a low availability—but this peer will rejoin the network in some future time. So these data are still considered to be durable. When the peer has no possibility to join in future time, or the data are not retrievable at any future time due to other kinds of failure, we declare that the data are lost. Therefore, typically $T \gg \tau$.

Access control Information must always be protected against unauthorized access. Access control includes considerations of *privacy* (ensuring that unauthorized entities cannot read stored information) and *write-integrity* (ensuring that unauthorized entities cannot change information). These criteria are probably best implemented as constraints, rather than as FOMs.

Authenticity Adversaries must not be able to substitute a forged document for a requested document. Authenticity can then be measured, from experiments involving malicious attacks, as the percentage of ‘good’ files remaining after the attack.

Denial-of-service (DOS) resilience It should be difficult for an adversary to compromise availability, by for example flooding the system with spurious data. DOS resilience may be measured by tracking changes in the availability which are due to a DOS attack. Hence, this is a form of adaptivity.

In addition to the more or less standard goodness criteria listed above, several interesting desirable goals, which are particularly relevant for P2P systems, have also been discussed recently:

Massive scalability One might expect of a P2P system that it should spread over the entire Internet, and still function well.

Anonymity One important purpose for building P2P systems is to provide anonymity. For an anonymous system, it should be impossible, or very difficult, for an outside observer

to ascertain who has produced a document, and who has examined it. Clearly, one can quantify the loss of anonymity by measuring the amount of information that a given observer is able to extract. Equivalently, the ‘anonymity probability’ can be defined as 1 - (probability of success for a receiver of a packet to trace the origin of the packet).

Deniability Users should be able to deny knowledge of data stored on their machines. This can probably be handled as a constraint.

Resistance to censorship It should be impossible, or as difficult as possible, to censor information, once it is written to the system [20]. Note that resistance to censorship may be measured similarly to both durability and authenticity. In all of these cases, the cost to be minimized is the fraction or number of lost files.

Replication This is a technique where a file is broken down into parts, and each part is copied multiple times, in order to circumvent the unreliability or the lack of trustworthiness of the participating peers, and also to allow for better load balancing. Hence replication is a design technique, rather than a FOM. However, replication is relevant for several FOMs: (a) Degree of load balancing (termed ‘convergence’ above). Here, small blocks allow for better load balancing. (b) Availability. Replication is used in order to enhance availability. However whole-file replication has a high cost (see below). Hence files are broken into blocks, which allow for replication at lower cost. If the block size is too small, it is likely to increase the response time for retrieval, and thus to negatively affect the availability. (c) Authenticity and anonymity. Replication, especially with small blocks, gives more points of attack for an outside agent, who may wish to extract information, and/or to corrupt the files. (d) *Replication overhead*. We italicize this since it is new to our list. This is clearly a cost, which may be quantified as (for example) (total information stored)/(amount of unique (original) information stored). When the replication overhead is expressed as a ratio, its inverse is a FOM.

5.4 Distributed content

We include the basic service *search* along with the advanced service of distributed content, because one of the principal challenges for a distributed content system is to enhance the quality of the search operation. Also, the design and performance of search algorithms are normally strongly affected by the nature of other functions which are implemented in the building of a complete distributed-content system. These other functions may include data replication or redistribution, indexing, and topology management.

Distributed content differs from distributed storage in that the data are not “owned” by any user. Hence considerations of retrievability (by the owner) are irrelevant, while considerations of durability play a lesser role. Instead, the retrieval problem is replaced with the problem of *finding* content that is stored in the network—that is, *effective search* becomes of primary importance.

The design of search algorithms varies widely, depending on which other functions listed above (replication, indexing, etc) are implemented, and on how they are implemented. Nevertheless, since the goal of search may still be broadly stated as finding hits at low cost, one can define a number of FOMs for search which are independent of how the search (and the other

functions) are implemented. Not surprisingly, these FOMs are numerous, and potentially conflicting. We believe that research on search over distributed content has not yet reached a stage whereby one can quantitatively define a uniquely best meta-evaluation parameter, which combines, in some way, all of the relevant FOMs into one.

We now offer a list of figures of merit.

Hits per search The most obvious FOM is certainly the number of hits per search—that is, the average number of items found when a search is initiated.

Hit quality Besides the number of hits, the *quality or relevance of hits* is an important FOM. In search, relevance is a measure of the quality of a search result, indicating how close the search result is to the information being sought. There exists however a wide variety of methods for measuring relevance. Relevance is often measured, for instance, by the number of times a search term occurs in a document relative to the length of the document. Many other factors may be considered, such as: frequency of synonyms and related terms, overall popularity of the document, links to the document from other “relevant” documents, expert ratings of relevance, or explicitly defined keywords in the document that match the search term [6]. Some of these relevance measures may be evaluated by a machine, while others require user participation. Unfortunately, the ‘truest’ measures of relevance are those using humans—since, implicitly, relevance is defined as being relative to the human searcher. This makes relevance hard to measure.

Query expressiveness A similar FOM for a search system may be termed its *query expressiveness*. That is, the query language used for a system must be able to express the information desired by the user in sufficient detail. Expressiveness is closer to being specifiable in a logical sense—in which case, it may be implemented as a constraint, rather than as a FOM. In any case, query expressiveness is a property of the design of a system, and as such (for most definitions at least), may be measured without resorting to experiments.

Query packet overhead The cost criterion which corresponds to the number of hits FOM is the *number of query packets* generated per search. An illuminating combined FOM is then the obvious one: $(\text{hits per search})/(\text{query packets per search})$, averaged over many searches.

Query path length Another cost can be defined from the nature of the route followed by the message packets. An important design choice for search is the routing of queries—where here “routing” is intended in a broad sense, to include such strategies as proliferation [9]. Many different routing strategies are possible. Hence, the *average number of hops* required by a message packet to reach a hit can be measured, and is another cost for the system.

Search time A closely related cost measure is the *response time* for a search. This may be defined as average time to first hit, or as the average time to completion of the search. (Other measures are of course also possible, such as time to obtain a minimal number h of hits, or time to obtain $p\%$ of the total hits eventually obtained.) Response time is relevant for P2P content systems, as users usually prefer an “instant” response—while, if the system is large, the actual response time may be significantly longer than “instant”, even on a human scale of time perception.

Data replication Replication of data is an important technique which is used to improve search performance. However, replication schemes can enhance the possibility of finding popular articles, while making rare articles even more rare. So, in order to measure the effectivity of replication, in addition to the above mentioned FOM (number of hits) (which normally doesn't provide any information about the underlying distribution), distribution specific evaluation criteria should be developed, so that enough importance is given to rare articles. For example, one can define "rare" articles according to their placement in the tail of the distribution, and then define *retrievability of rare documents* in some way—for example, percentage of successful searches for rare documents, number of hits obtained, etc.

Superpeers and indexing Indexing schemes are often used in P2P content systems—for example, in systems with a peer-superpeer architecture. In the distributed form of indexing, peers get associated with superpeers, such that the index over each peer's content is written to the associated superpeer. The efficiency of indexing depends directly upon the nature of formation of these peer-superpeer clusters. Therefore—for those schemes using a peer/superpeer topology for indexing—one can define performance criteria for the peer/superpeer topology management subcomponent. (Note that these criteria are not direct measures of search performance; instead they quantify the performance of a different, optional, component of a distributed-content system.) We will mention one such measure here: the size of the index (or number of peers), assigned to any given superpeer, should be adjusted according to the bandwidth of the given superpeer. That is, these two quantities should be managed so as to give, as nearly as possible, an ideal target ratio between the two. Deviations from this ratio are then an inverse FOM. Also, there should be a constraint on this ratio (in terms of an upper bound), to avoid congestion at any superpeer.

The above list of FOMs for search are of the form that can be measured for a single experiment. Now we discuss measures—nice properties—which quantify how well a given approach works for a variety of problem settings. These are (again) expressed in the form of sensitivities of previously-defined FOMs.

Sensitivity to topology This goodness measure may be defined essentially in the same way as it was for load balancing—with the one change that the FOMs whose sensitivities are to be measured are those listed here. Note that this item, and the next one, represent a kind of (inverse) adaptivity.

Sensitivity to dynamics Again we focus only on the differences from the definition in the load-balancing case. Here, the dynamics comes from fluctuations in topology (as with load balancing), and from steady variation in the content which is present on the system. Each of these types of dynamic variation can be simulated and parameterized, allowing the measurement of the sensitivity of FOMs such as number of hits, or of cost criteria, to the dynamics parameter(s).

Sensitivity to noise Here the definition is essentially the same as that given for load balancing. The packets which may be lost, in this case, are the query packets. Loss of query packets,

by our definition, is “damage” to the CAS which is responsible for search. Hence the *insensitivity* of the system to this kind of noise is a kind of robustness.

Scalability Here, we have at least two different ways of parameterizing system size—the usual one being the number of nodes N , and another one representing the size (number of distinct documents) or diversity (measured in terms of breadth of information content represented) of the total content on the system. Clearly, the difficulty of search will grow with increasing values for either of these measures of system size.

6 Nice properties from swarm intelligence

The aim of this section is to develop working definitions of what we call “nice properties” for technological systems. We will take our working definitions to the point that they are capable of being quantified; but we will not give specific, quantified definitions. Our reason for holding to a more general form of definition is that the evaluation of nice properties may be performed for a large variety of problems, solutions, and environments. For different environments—and perhaps also for different problems—different specific definitions of nice properties will be appropriate. So we confine ourselves to ‘generic’ definitions here, providing a framework which brings us close to the point of having precise, quantitative definitions.

Not all of the items in this section are nice properties. We will see however that the three properties which we deem to be ‘nice’ are all of the form of an *insensitivity* of one or more figures of merit to environmental challenges of various types. Hence the biological motivation of the notion of nice properties remains clear throughout, while at the same time practical engineering quantifiability comes within reach. Note that all of our ‘insensitivities’ have then the dimensions of an inverse derivative—a change of FOM divided by a change in environmental variable.

6.1 Self-organization

The property of being “self-organized” may be defined as follows. We assume a BISON context, with agents moving and acting on a network of nodes and links. We also imagine that the system—network plus agents—*starts* at some time (or perhaps more than once). Then the system is self-organized if, in the starting state, no node (or agent residing at the node) has any knowledge beyond local knowledge (such as a neighbor list). Furthermore, in subsequent time development of the system, all interactions are local.

The point here is that there should be no global knowledge “for free”. There can be no Godlike node which, at any time, has an instantaneous view of the entire system, and which can communicate such information to any other chosen node. Global knowledge *can* of course be built up in the course of development—using local interactions, and building from initially purely local knowledge. For example: the system can choose a “leader” node for some function; or it can compute the average of some quantity over the whole network. It can build up a hierarchy. But it cannot *begin* with such information; it must acquire or create such information, using only local rules.

This definition appears to have a great deal of overlap with the term “decentralized”. This latter term can however be misleading: it seems to rule out situations such as a self-organized Napster (star) topology, which has a single node chosen as a center. This is a centralized topology, built up (by assumption) in a self-organized fashion from a decentralized starting state. To point out the obvious: a self-organized system can be organized!—but it must organize itself, without help from any “outside” source. Hence such a system may change from decentralized to centralized (and back, perhaps many times) in the course of its history. A good example from biology is the slime mold colony, which can move from a highly unorganized state of individually grazing amoebae, to a highly organized (and centralized) state, with a single mobile sluglike body, which subsequently sprouts a fruiting stalk. The stalk then gives off spores, which scatter in many directions, and hatch out to give new amoebae, in a new place, in the disorganized grazing state. In this way, the colony moves over time between an unorganized, decentralized state, and organized, highly centralized states; but it is always self-organized. Technological examples of (partially or fully) centralized, but still self-organized, states are also numerous—for example, the election of superpeers in a peer-to-peer network, or the formation of routing hierarchies in an ad-hoc network.

Now we come to a question that the reader may also be asking: is the property of being self-organized a “nice property”? It certainly has connotations of “niceness”: we imagine that such a system can *re-organize* itself, if its organized state is damaged in some way. However, this property—of being able to repair damage and so restore functioning—is not identical to the property of self-organization as defined here. Instead, the former—which we will identify with “robustness” below—is a likely, but not guaranteed, consequence of the latter (self-organization). That is: self-organization is a promising *means* to achieve robustness; but the two properties are not the same.

Furthermore, we would claim that the property of being organized at all (self- or not) is not in itself intrinsically desirable. That is, if the term “organization” may (as seems reasonable) be taken to mean something distinct from the property of functioning well, then it is truly the latter property that is of interest for performance evaluation, and not the former. One can easily imagine for example a set of agents which have arranged themselves into a highly organized state, which however happens to give very poor performance for the function(s) of interest.

Thus we do not place self-organization on our list of nice properties. Instead, we view this property as describing the kind of systems that the BISON project should study. The logic of BISON is: Biology is self-organized. Living systems have nice properties. Therefore, we will study self-organized systems, in order to realize the nice properties.

Because of this logic, it is very much of interest to define and study self-organization. But any measure of self-organization will be a measure, not of any nice property, but rather of the appropriateness of the system under study for the scope of BISON.

For these reasons we have not attempted a quantitative definition of self-organization. Our verbal definition implicitly suggests some ideas however. For instance, one could measure deviation from self-organization in terms of the amount of global information which is input to a (not completely self-organized) system.

6.2 Adaptivity

Consider a system that is operating under more or less steady environmental conditions, and yielding a (more or less steady) FOM f_0 . Now the environment changes. We say that the system is adaptive if (after some time) it restores (more or less) the same FOM.

There are clearly two aspects here to the adaptation. One is the *response time*: the time that the system takes to achieve a new, roughly steady-state, performance, after a change in the environment. The second is the *insensitivity* of the steady-state FOM to the environmental change.

These two aspects are clearly distinct: the speed with which the system finds a new steady FOM f' says nothing about the relationship between the new FOM f' and the old one f_0 . And yet both quantities (the response time, and the change in FOM) tell us something about the ability of the system to adapt.

So as to avoid the inconvenience of having two working definitions for one word, we make here the explicit choice that the word “adaptivity” will be used to refer to the insensitivity of the steady FOM to environmental changes; and we keep the term “response time” as a separate FOM. This choice is consistent with our general picture: that speed of response is a FOM, measurable in a single experiment, with units of inverse time; while adaptivity is a nice property, with units of insensitivity, ie, $\Delta(\text{environment})/\Delta(\text{FOM})$. Also, this choice allows us to reject as non-adaptive the case where the new FOM f' is achieved quickly, but is in many cases much worse than the old FOM f_0 . For these reasons, we focus here on adaptivity as defined in terms of ability to return to a good FOM after an environmental change. Our goal is to render the concept quantitative.

First we consider variation of a single environmental variable EV. This quantity can be any of a number of things: topology (changes in nodes and/or links), traffic, a malfunctioning node, etc. The point is that the environment in which our CAS is to implement a service changes; and the CAS must adapt to this change. The EV may or may not be representable in terms of a single real variable. Nevertheless, for illustrative purposes, we imagine that we can plot FOM vs EV (Figure 1).

The important thing however is to vary the EV, and measure the resulting (steady) FOM that results. If we again think in terms of a plot (by again assuming that the EV can be varied continuously in one dimension), we get a curve showing FOM vs EV. One then must quantify the amount of variation in this curve. This variation measures sensitivity of the system (CAS, working in an environment) to changes in the EV. Adaptivity is then the inverse of this sensitivity: it is insensitivity to changes in EV. We might also think of adaptivity in terms of the “stiffness” of the curve of FOM vs EV (cf. Figure 2).

This definition is not sufficiently detailed to be quantitative. For one thing, a quantitative definition requires a choice of range for the environmental variable. Furthermore, a quantitative measure of the stiffness of the FOM variation, over the chosen range of variation of the EV, is also needed.

These two issues are in fact interrelated. For example, it may make sense to choose a single operating point for the EV, and then measure changes in FOM due to small changes in the EV around this operating point. That is, if the definition is the (inverse of the) derivative, then the

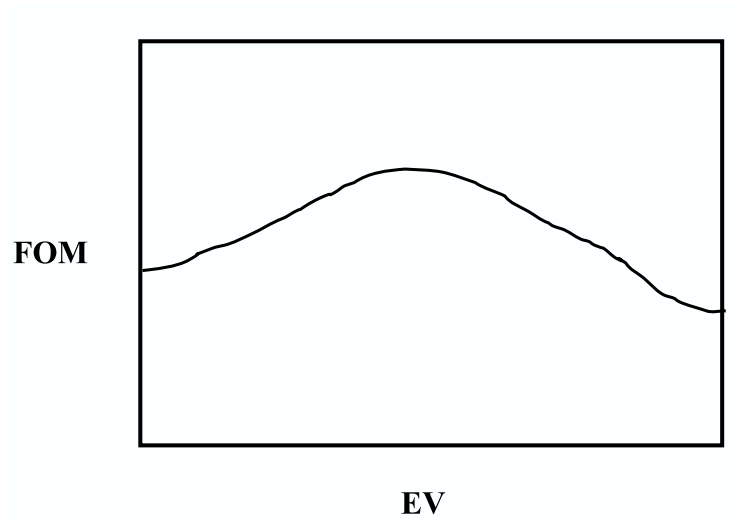


Figure 1: A schematic plot showing variation of a figure of merit FOM with environmental variable EV.

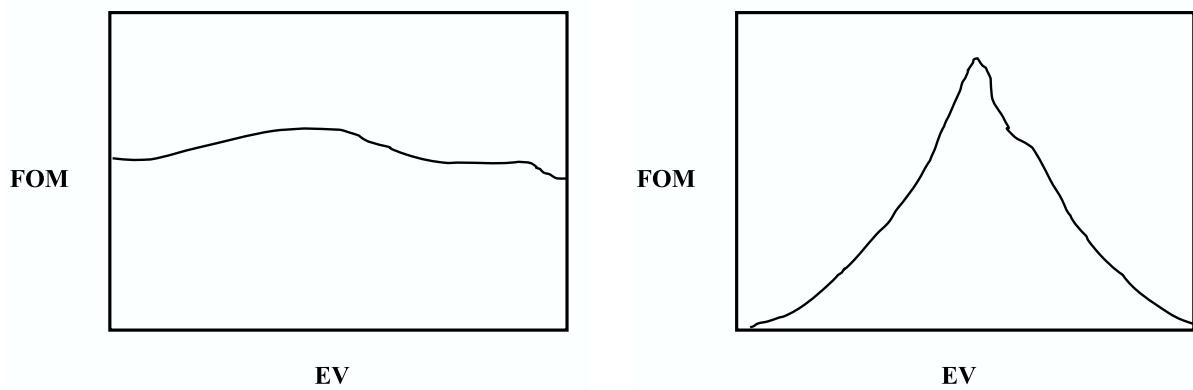


Figure 2: A schematic plot showing a relatively "stiff" behavior of a FOM (left). We call such stiffness adaptivity. The right plot shows a FOM which is strongly sensitive to changes in the environmental variable. Such a system has a low degree of adaptivity.

range is a point.

On the other hand, one may choose a finite range for the EV, and then choose some stiffness measure defined over this whole range. The result then clearly depends on the range chosen; and still there are multiple possibilities. For example, if there is an optimal operating point with a single maximum of the FOM there, one may choose as a measure of stiffness the range of EV required to reduce the FOM by a given percentage. If the whole range ΔEV is of interest, then the total variation ΔFOM may be used—with the stiffness then being $(\Delta EV / \Delta FOM)$.

Yet other choices are possible. Adaptivity is a broad term. We believe that it is most useful in the BISON context simply to define adaptivity as the inverse of the (change in steady FOM)/(change in EV). This allows for many specific definitions, all having the same units of measure.

Furthermore, it is clear that each choice of FOM or EV gives a new kind of adaptivity. We can call all of these choices “single-FOM, single-EV adaptivities”. More sophisticated measures are also possible, in which one considers more than one EV, more than one FOM, or both, and still defines a global measure of adaptivity for the CAS, in terms of changes in the set of FOMs, versus changes over the whole domain of EVs.

Finally we comment on the relationship of the definition presented here to biology. Clearly the term FOM can be replaced with “fitness”, while environmental variation retains its role as tester of adaptivity. The CAS itself may be any living subsystem for which the fitness criterion makes sense: an organism, a subsystem of an organism, a species, an ecosystem.

If we focus on the organism as our adapting unit, we note that it can adapt in several ways, on several timescales. On a relatively short timescale, it can change its behavior. For example, a bird can learn to use a new material in nest-building, when the preferred material becomes scarce, and/or the new one becomes abundant.

The phenotype is also subject to adaptive change. For instance, callouses on hands and feet develop in response to stress on the skin; or the skin tans as protection against solar radiation; or muscles grow in response to use. All of these changes are reasonably viewed as adaptive.

Finally, on the longest timescale, the organism can evolve, ie, it can change its genotype. The interesting point here is that the mechanism of change in this case is not intrinsically adaptive—it is random mutation. Hence mutation is only adaptive when supplemented with the mechanism of selection. We note here, in this context, that selection has had a feedback effect on the mutation mechanism itself. That is, it is likely that selection has tuned the mutation *rate* of organisms to a value that is neither too high (giving too many errors in phenotype), nor too low (giving a lack of adaptivity).

How are these three biological adaptation mechanisms reflected in BISON systems?

It seems clear that there is little difference between phenotype and behavior for a simple, agent-based CAS on a network. That is, for such systems, the behavior is the phenotype; and it is determined by the encoded rules (the genotype).

Hence these three mechanisms in biology become two in BISON systems. We examine each—behavioral adaptation, and evolutionary adaptation—in turn.

It seems completely straightforward that technological systems can exhibit adaptive behavior with a fixed set of rules or laws for that behavior. Examples are numerous and obvious.

Evolutionary adaptation is more subtle. We suppose for example that we have a set of ants which have two different kinds of routing rules: one for high node mobility, and one for low node mobility. We can then design “smart ants” which can (i) estimate the node mobility, and (ii) change from one routing rule set to another, based on that estimation. These ants are thus *changing the rules* which encode their behavior; hence we can claim that they are adapting via evolution.

The validity (or lack of it) of this claim is probably a matter of semantics. However we want to point out that this kind of rule change is not the same as that (random mutations) used in biology. Instead, it is a *rule-based rule change*, where some “meta-rule”—which is partially or wholly deterministic—invokes the change in routing rules. The smart ants carry this meta-rule in their genome. Hence the change is very much like the biological mechanism of gene activation and deactivation. Such a change is not considered evolution in biology; instead (when it is not random) it is a normal aspect of the functioning of the phenotype.

It is not the purpose of this discussion to favor one semantic choice over another, with respect to the words (evolution, or phenotype change) to be assigned to the mechanism of rule-based rule change. Instead we wish to point out that such changes are possible, and of interest in building adaptive CAS. Furthermore, such rule-based rule change represents a mechanism for adaptation which is in some sense distinct from either the adaptation arising from a simple fixed set of rules, or that coming from “pure” evolution, consisting of random mutation plus selection.

6.3 Robustness

We have mentioned above that we expect robustness to be a possible consequence of self-organization. An even closer term is “self-repair”. However self-repair has a focus on structure; and in BISON we tend to focus more on function than structure. The two are not in any case mutually exclusive. For example, it may be the job of a topology management system to maintain a structure. If this system functions well, then the structure itself will be self-healing or self-repairing (we use the terms interchangeably). Hence we can subsume self-healing into the broader property of robust functioning.

We get another clue into the meaning of robustness from the idea of being self-healing. (For a broad discussion of notions of robustness, see for example Refs. [11] and [15].) That is, healing is a response to damage. Similarly, and in the broader sense, we view robustness as a response to damage: robust functioning means the ability to restore functionality after damage has occurred.

Now we see that our working definitions of robustness and of adaptivity are quite similar: in each case, we ask the CAS to restore functionality, in response to some change. For adaptivity, the change comes from the environment. For robustness, the change is called “damage”. Now we must seek some precision in the meaning of this word “damage”.

A crucial aspect of the needed definition is the ability to in fact distinguish the CAS from its environment. If we consider the whole system (CAS + environment), we can imagine several types of change. We can have change in the environment—to which the CAS must respond if it is adaptive. We can have “intended” changes to the CAS itself, such as state changes, proliferation, or planned death. These changes are *part* of the response of the CAS, and hence do not

require any further response by the CAS. Finally, we can imagine “unwanted” changes to the CAS, caused by the environment—for example, loss of agents, or state changes due to noise. These unwanted CAS changes then are the most natural interpretation of the term “damage”. However, in order to distinguish unwanted CAS changes from general environmental change, we must be able to distinguish the CAS from its environment. That is, we need a clear defining boundary between the two. There is likely to be some degree of arbitrariness in the placement of this boundary, in many cases. Nevertheless, in our proposed scheme of definitions, the difference between the meaning of the two terms ‘robustness’ and ‘adaptivity’ resides entirely in the placement of this boundary.

The CAS, plus its environment, constitute a complete system, which is in principle well defined (although, in practice, there are no real boundaries separating ‘system’ from ‘universe’). Hence the most straightforward way to distinguish the two is to define the CAS—which is finite, and capable of clear definition—and then let the environment be ‘everything else’. Examples of damage to a CAS can be: loss of agents by ‘unnatural death’ (eg, due to dropped packets, or to loss of a node); and errors in the agent state, either during transmission over a link, or simply over time at a node.

This rounds out our verbal definition of robustness: it is the ability of a CAS to restore functioning in response to damage to the CAS.

Now, to render our definition of robustness in a more quantitative form, we use ideas very similar to those used for adaptivity. That is, where before we measured response to changes in an environmental variable EV , now we measure changes in response to damage D . The changes are measured as before in terms of functioning, or more specifically, in terms of one or more FOMs. Hence robustness will be measured in units of $(\Delta D)/(\Delta FOM)$.

This definition then simply amounts to defining robustness as another kind of stiffness. Changes in FOM may be measured in any of the ways discussed in the context of adaptivity (previous section). Similarly, multiple FOMs, and more than one kind of damage, may be included in the definition.

We note that, as defined in this way, robustness may be thought of as a form of (generalized) adaptivity, since it is simply an insensitivity of one or more FOMs. Also (as we will see below), our definition of scalability may be thought of as a kind of adaptivity, for the same reason. Nevertheless—as a matter of convenience in definition—we retain the terms ‘robustness’ and ‘scalability’ as nice properties which are distinct from adaptivity.

6.4 Scalability

Scalability is very roughly seen as the ability of a system to function well, even though its size, as measured by some parameter N , may become extremely large. As such, scalability (unlike adaptivity and robustness) is not typically considered to be a property of biological systems. In biological systems, notions of stability—a concept seemingly at odds with unlimited growth—are typically more important. For instance, the standard picture of a stable ecosystem is one in which no species is able to dominate. Instead, in a stable ecosystem, there is a high level of diversity of species in a local area, and there is considerable variation as a function of geographic distance.

Here it is useful to distinguish two different aspects of the term scalability. Our definition of scalability—as given in the first sentence above, and applicable to either biological or nonliving systems—is the ability to function well, even in the face of extreme growth. Hence, this definition calls for only the *potential* for growth, and need not involve growth itself. However, in the absence of a sufficient theoretical basis (as is common in biological systems), the surest way to confirm the potential for growth (scalability) is to observe the growth itself. That is: the empirical observation of extreme growth allows one to deduce that the growing subsystem has scalability. Furthermore, there is always the danger that a system with the potential for growth (ie, a system with scalability) will in fact grow to a size that is highly undesirable for some reason. Thus, for these (rather obvious) reasons, there are close connections between scalability (the potential for growth) and the growth itself.

Unconstrained growth is not normally considered a ‘nice property’ for biological systems. To the contrary, it typically implies instability. For example, invasive species in an ecosystem (such as rabbits in Australia) demonstrate scalability by growing to huge numbers and destabilizing the ecosystem. Similarly, cells in an organism must know when to stop growing; when they do not (cancer), the organism is destabilized—it dies. That is, ‘biological scalability’—combined with not knowing when to *stop* growing—is viewed as having negative consequences for the host organism (in the case of cancer) or for the ecosystem (invaded by rabbits).

In contrast, for networked systems, scalability of an algorithm or protocol is typically viewed as a desirable thing. We want systems which will work for N nodes, whether N is 10 or 10 million. And the intent is often that such systems *will* in actuality grow to enormous size—say, to the scale of the entire planet (World Wide Web, Internet). Hence the growth itself is also viewed as desirable—rather than as an unwanted instability.

It is of interest to try to understand why the two sides of our favorite BISON analogy (network systems \approx ecosystems) appear so different in this regard. We suggest two possible reasons.

First, for typical network ‘organisms’ (for example, a peer-to-peer protocol), too little growth is often more of a problem than too much growth. Early P2P file-sharing protocols such as Gnutella were in fact unable to ‘scale’—that is, they lost functionality when the number of users N became too large. As another example, we cite routing protocols for ad-hoc networks. All such protocols struggle with severe scaling difficulties, arising from the simple physical constraint of having no long-range links. For these reasons, wireless ad-hoc networks are not expected to span large areas without any help from infrastructure.

Exceptions to this rule (that typical network organisms have trouble scaling) are often (unfortunately) malicious agents such as viruses and worms. Actually, there are no doubt many such pests that fail to scale; but it is the ones that do scale (sometimes spectacularly) that are noticed.

In any case, we believe that one reason that engineers often work hard to achieve scalability is that it is not (yet) easily done. Certainly it cannot be taken for granted.

This however begs the question of why scalability is *desirable*. Our answer lies in our second reason for understanding the popularity of scalability in engineered network systems. That is, good engineering practice includes a measure of *cost* in the FOMs that are used to evaluate a technological system. Here, by ‘cost’, we mean cost to the *whole system*—not simply cost to the organism (subsystem) itself (as is the unwritten rule in biology). Once whole-system cost is included—as a cost, ie, as something that cannot be allowed to grow too large—in the

definition of 'good functioning', then the ability to function well in the face of growth includes the property of knowing when to stop growing.

In other words, scalability is a desirable property when an appropriate whole-system cost measure is included in the FOMs which measure the functioning of the subsystem as a function of size parameter N .

It is only implicit, in the previous discussion of this section, that such a cost measure should be included in a good set of FOMs. Now we make that prescription explicit. We note that, even in measuring robustness or adaptivity, the cost of the good performance should be included in the performance measures. Now we see that the same prescription holds—perhaps with even more force—when the nice property to be measured is scalability. Obviously, a CAS system which scales in terms of good performance, but only under the assumption that its resource usage can explode as a function of N , will not be defined as having scalability. (Nor, by our definition, will the cancer cells, or the invasive species.)

With this point clear, we have a verbal definition of scalability; and again the definition is in terms of insensitivity or stiffness. We define scalability as the insensitivity of a FOM (including an appropriate cost measure) to increases in the system size.

To make such a definition quantitative, we need a measure of the system size, and (as usual) a measure of the sensitivity of the FOM or FOMs. System size, for network systems, is most typically measured in terms of number of nodes, N . Other measures are however possible—for example, number of documents stored, at fixed N , in a distributed-content system.

Choice of FOM depends of course on the system studied. It is however common in studying scalability to require that *all* relevant FOMs (each including an appropriate cost function) should be sufficiently insensitive to growth in system size N .

We have one final comment about scalability—namely, that it is possible to apply this criterion too uncritically. It is always interesting to let $N \rightarrow \infty$ in theoretical studies, and deduce the consequences. However there can be applications of distributed systems for which the behavior at $N \rightarrow \infty$ is not of interest—because there is a natural cutoff at some finite N instead. Examples include: an ad-hoc network which is not expected to span the globe, but only to cover a given area; a routing system which must only handle a given administrative domain; or a peer-to-peer system which is intended for a finite, and in some sense exclusive, set of nodes. In cases such as these, scalability should obviously be defined as good functioning (per unit cost) up to the cutoff value for N .

References

- [1] V. Anantharaman, S.-J. Park, K. Sundaresan, and R. Sivakumar. TCP performance over mobile ad hoc networks: a quantitative study. *Wireless Communications and Mobile Computing*, 4:203–222, 2004.
- [2] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka. TCP-PR: TCP for persistent packet reordering. In *In Proceedings of the IEEE 23rd International Conference on Distributed Computing Systems*, pages 222–231, May 2003.

- [3] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka. A new TCP for persistent packet reordering-TCP-PR. *Accepted for Publication in Transactions on Networking*, 2004.
- [4] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications & Mobile Computing: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2002.
- [5] S. Corson and J. Macker. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. Network Working Group, Request for Comments 2501, January 1999. <http://www.ietf.org/rfc/rfc2501.txt>.
- [6] N Daswani, H Garcia-Molina, and B Yang. Open Problems in Data-Sharing Peer-to-Peer Systems. In *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings*, volume 2572 of *Lecture Notes in Computer Science*. Springer, 2002.
- [7] D.S.J. De Couto, D. Aguayo, B.A. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. In *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, October 2002. ACM SIGCOMM.
- [8] 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.
- [9] N Ganguly, G Canright, and A Deutsch. Design Of An Efficient Search Algorithm For P2P Networks Using Concepts From Natural Immune Systems. In *8th International Conference on Parallel Problem Solving from Nature*, September 2004.
- [10] F. Guillemin and W. Monin. Management of cell delay variation in ATM networks. In *Proceedings of IEEE GLOBECOMM'92*, Orlando, FL, USA, 1992.
- [11] Santa Fe Institute. Robustness in natural, engineering, and social systems. <http://discuss.santafe.edu/robustness>, 2003.
- [12] Márk Jelasity and Ozalp Babaoglu. T-Man: Fast gossip-based construction of large-scale overlay topologies. Technical Report UBLCS-2004-7, University of Bologna, Department of Computer Science, Bologna, Italy, May 2004. <http://www.cs.unibo.it/techreports/2004/2004-07.pdf>.
- [13] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based unstructured overlay networks: An experimental evaluation. Technical Report UBLCS-2003-15, University of Bologna, Department of Computer Science, Bologna, Italy, December 2003. an extended version appears in the Proceedings of MIDDLEWARE 2004.
- [14] Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of The 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pages 102–109, Tokyo, Japan, 2004. IEEE Computer Society.
- [15] Erica Jen. *Robust Design: A Repertoire of Biological, Ecological, and Engineering Case Studies*. Oxford University Press, Oxford, UK, 2004.
- [16] D.B. Johnson and D.A. Maltz. *Mobile Computing*, chapter Dynamic Source Routing in Ad Hoc Wireless Networks, pages 153–181. Kluwer, 1996.

- [17] B.-J. Kwak, N.-O. Song, and L.E. Miller. A canonical measure of mobility for mobile ad hoc networks. In *Proceedings of IEEE MILCOM'03*, Boston, USA, 13–16 October 2003.
- [18] A. Montresor, G. Di Caro, and P. Heegaard. Architecture of the simulation environment. Internal Deliverable D11 of Shared-Cost RTD Project (IST-2001-38923) *BISON* funded by the Future & Emerging Technologies initiative of the Information Society Technologies Programme of the European Commission, 2003.
- [19] S. Mueller, R. Tsang, and D. Ghosal. Multipath routing in mobile ad hoc networks: Issues and challenges. In *Performance Tools and Applications to Networked Systems*, volume 2965 of LNCS. Springer-Verlag, 2004.
- [20] A. (Ed) Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O Reilly Books, 2001.
- [21] J. Roberts and F. Guillemin. Jitter in ATM networks and its impact on peak rate enforcement. *Performance Evaluation*, 16(1–3), November 1992.
- [22] N. Sadagopan, F. Bai, B. Krishnamachari, and A. Helmy. PATHS: analysis of PATH duration statistics and their impact on reactive MANET routing protocols. In *Proceedings of MobiHoc'03*, pages 245–256, 2003.
- [23] Robbert van Renesse. The importance of aggregation. In 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, pages 87–92. Springer, 2003.
- [24] WHO: Protection of the Human Environment. Water and Sanitation: Water supply and sanitation sector monitoring. Available online (last visited 2003-02-24), January 2003.
- [25] Otto Wittner, Poul E. Heegaard, and Bjarne E. Helvik. Scalable distributed discovery of resource paths in telecommunication networks using cooperative ant-like agents. In *Proceedings of Congress on Evolutionary Computation, CEC2003*, Canberra, Australia, December 2003. IEEE.