Fitting Genetic Algorithms to Distributed Online Evolution of Network Protocols☆

Sara Alouf, Giovanni Neglia*

INRIA, 2004 Route des Lucioles, B.P. 93 – 06902 Sophia Antipolis, France

Iacopo Carreras, Daniele Miorandi

CREATE-NET, via Alla Cascata 56/c, IT – 38100, Trento, Italy

Álvaro Fialho

Microsoft Research-INRIA Joint Centre, 28 rue Jean Rostand – 91893 Orsay, France

Abstract

In this work, we introduce a framework for enabling the on-line evolution of network protocols. The proposed approach is based on the use of techniques and tools drawn from evolutionary computing research, and it enables embedding evolutionary features in the operation of network protocols. In this way, it becomes possible to build a system in which the operation of the network changes at run-time to adapt to the current conditions. As a case study, we apply the proposed framework to the evolution of forwarding schemes in intermittently connected wireless networks. Simulation results are reported to validate the ability of the proposed scheme to converge to the optimal operating point and to explore the various trade offs deriving from its design and implementation.

Keywords: evolving protocol, genetic algorithms, wireless network, delay tolerant networks
PACS: 89.20.Ff, 89.20.Kk
2010 MSC: 93A14, 93C41, 93E35

1. Introduction

In general, a "network protocol" is understood as a set of rules and conventions defining the way networked devices can communicate with each other. Network protocols represent the common grammar and syntax necessary for enabling computing devices to communicate and perform complex tasks. The success of the Internet and of Web-based applications is rooted

☆Part of this work has been presented at IEEE MASS (Bionetworks workshop), Pisa, Oct. 2007 (cf. [1]) and has been accepted for publication as a chapter of a book on Autonomic Computing and Networking, published by Springer (cf. [2]).

*Corresponding author. Phone: +33.4.92.38.79.06. Fax: +33.4.92.38.79.71.

Email addresses: `name.surname@sophia.inria.fr` (Sara Alouf, Giovanni Neglia), `name.surname@create-net.org` (Iacopo Carreras, Daniele Miorandi), `name.surname@inria.fr` (Álvaro Fialho) in the widespread diffusion of a set of protocols (the TCP/IP suite) enabling communications among remote devices.

Network protocols are commonly implemented as a set of software (or firmware) routines performing a set of given tasks following some internal algorithms. Conventionally, network protocols are designed in a *static* way. They are designed, implemented, tested, and then released for deployment. As network protocols are at the basis of communications, network entities need to be aligned in terms of network protocols in use. Migration between different network protocols has always been an issue, involving problems related to backward compatibility and incremental deployment. The IPv4-IPv6 migration is an example of the complications involved in such processes.

Dynamic and flexible network protocol stacks have been a subject of research since the early 90's, e.g., [3, 4]. Notwithstanding, they did not make their way into widely deployed network software. This is rooted in a series of factors, including, e.g., the technological problems related to the increased computational burden and the penalty in performance (especially relevant for wired networks) and the lack of a suitable programming model for enabling dynamic composition of protocols. Recently, the arising of large-scale highly dynamic networked systems, such as peer-to-peer systems, pervasive computing environments, wireless sensor networks etc., has raised new interest in such approaches [5]. In such scenarios, indeed the penalty to pay in terms of performance is highly rewarded by the increased level of flexibility and adaptability provided.

One of the main concerns when dealing with dynamically changing network protocols relates to interoperability issues. In reality, however, it is possible to distinguish between functional and non-functional aspects of network protocols. If an appropriate architecture is used, non-functional aspects could be changed on-the-fly without disrupting the system's operations. In particular, the algorithmic aspects underpinning many protocol operations (e.g., forwarding, congestion control, medium access control, etc.) could be changed at run-time without the need to specify them completely during the design phase. The resulting flexibility could result extremely useful in enhancing the robustness of the global networked system and its ability to adapt to unforeseen conditions and usages.

In this paper, we aim at introducing a framework for enabling the on-line evolution of network protocols. We focus on those network protocols which might be represented as atomic stateless services. We first introduce the elements of the framework, including modules for estimating the performance of the protocol currently in use and a repository of possible solutions and their respective estimated performance levels. We then discuss techniques to be used for updating the protocol in use based on the information present in the local repository.

The framework in itself is quite general. However, the real challenges are related to its implementation and tailoring to a specific network protocol. In order to provide insight into such issues, we present a case study, namely epidemic-style forwarding and its application to intermittently connected wireless networks [6]. We present a possible representation of the non-functional aspects of the protocol (in this case: the forwarding probability) and show how evolutionary computation techniques can be applied in order to let the system adapt in a smart way to the current (unknown) network settings. The outcomes of an extensive experimental study are then presented, including results in terms of scalability and adaptability. The impact of various implementation choices is also discussed.

The remainder of the paper is organized as follows. In Sec. 2 the high-level framework is presented. Section 3 introduces the issues related to the application of the framework to epidemic-style data dissemination protocols. The implementation details are reported in Sec. 4. Outcomes of simulation–based performance studies are presented and commented in Sec. 5. Section 7 concludes the paper discussing potential extensions of the present work.

2. Model, Tools and Techniques

In this work, we focus on the definition of a framework for the design of network protocols able to evolve on-line in a distributed, unsupervised way. As application scenario, we focus on intermittently connected wireless networks (also variously referred to as delay-tolerant networks, DTNs, or opportunistic networks [7, 8]). In such scenarios, indeed, no centralized control of network operations is possible, due to the disconnected and potentially highly dynamic nature of the network itself [9]. One of the possible approaches for overcoming such a problem is to embed self-management capabilities (including adaptability) into the network protocols. A way to do so is to use techniques and tools based on evolutionary computation (EC) [10]. Inspired by the Darwinian theory of evolution, EC techniques are particularly suitable for obtaining systems able to self-optimize their behaviour based on the operating conditions they are in. At the same time, they represent only one of the possible approaches for embedding autonomic features in network protocols.

The work at hand is based on the following set of assumptions:

- No node has access to global information on the network status;
- Each node runs a networking service (or protocol), i.e., a routine performing functions dealing with transmission of messages from one (or multiple) source node(s) to one (or multiple) destination node(s);
- The networking routines considered represent *atomic* services, i.e., they can be carried out without requiring operations from other nodes. At the same time, their performance may be influenced by the behaviour of other nodes;
- The networking routines considered represent *stateless* services, i.e., no state information needs to be maintained between two different invocations of the same service. Within a single invocation, a service may maintain some state information needed to perform its tasks. However, this state is not conserved across subsequent service executions.
- Nodes maintain information about a possible set of services/protocols to be utilized, together with an estimate of their performance, expressed as a single numerical value;
- Nodes cooperate by exchanging information on the set of services/protocols in use and their actual performance.

It is worth remarking that our assumptions do apply only to a subset of the network services currently handled by (conventional) IP-based protocol stacks. For example the framework cannot be used to perform routing, as it configures as a *distributed* service, requiring strict cooperation and coordination of actions across multiple nodes. Still there are quite numerous scenarios where we can apply it. We hereby provide a sample list of networking services to which our framework does apply. A more detailed description of a number of them, including possible cost functions and genotype description, is provided in Sec. 6.

Example 1 (Epidemic–Style Forwarding). *In epidemic-style forwarding [6, 11] nodes, upon receiving a message not destined to them, relay it to neighboring nodes with a given probability.*

An invocation of the service corresponds to a request to send a message to a neighboring node. The service is atomic in that it does not require any coordinated action to be taken at any other node. The service is stateless (no state needs to be maintained across invocations). A set of possible alternatives for the service could correspond to a set of different forwarding probability. Our framework applies to such a service, which will be used as case study and analysed in detail in Sec. 3.

Example 2 (Network Coding in Intermittently Connected Wireless Networks). *In network coding, nodes can combine messages in order to maximize, in a multiple sources–multiple destinations scenario, the overall network capacity. In random linear coding, a source node can group a number of messages to a given destination in blocks. Relay nodes operate a random linear combination of the messages within the block and forward it together with the coefficients used. The dimension of the block can have a large impact on the resulting performance, e.g. in terms of message delay, and it depends on a number of factors (connectivity patterns, buffer size etc.) which may not be known at design time. The service is atomic in that it is performed at the source only. No state needs to be maintained across subsequent invocations, corresponding to different blocks to be delivered.*

Example 3 (Decentralized Medium Access Control). *In decentralized medium access controls, each node runs an algorithm determining the time instant at which a packet transmission has to take place. Examples include classical radio MAC protocols such as ALOHA and CSMA/CA. The service invocation corresponds to a layer–2 request to send a packet. No explicit coordination among nodes is required, hence the service is atomic. No state needs to be maintained across different invocation (i.e., subsequent requests to transmit different packets)*[1].

Example 4 (Congestion Control). *In congestion control protocols, an application–level data unit is passed to a service, which is in charge of fragmenting it into smaller packets and regulating the rate at which such packets are passed to lower level services for transmission. We consider TCP–like congestion control protocols, whereby the transmitter entity maintains a parameter (called 'congestion window') describing the amount of packets sent and not yet acknowledged. An algorithm is in charge of driving the behaviour of the congestion window value upon significant events (e.g., reception of an ACK, timeout expiration etc.). During the execution of the service a state, corresponding to the congestion window value, is maintained. Such a value is not propagated across subsequent invocations, hence the service is stateless*[2]*. The service is atomic in that it operates at the sender side only (though it requires the destination to send back acknowledgment packets, which is however independent on the specific type of congestion control algorithm used at the sender). In this case a set of alternative services could implement different flavours of congestion control mechanisms, including, e.g., TCP New Reno, TCP BIC, Scalable TCP, TCP CUBIC, Compound TCP etc. The performance metric considered could be the average per-connection throughput. The performance measured may depend on the behaviour of other nodes in the network, which may cause more or less congestion depending on the specific service implementation (i.e., congestion control algorithm) chosen.*

[1]Note that in 802.11–like protocols, a state of the form (s,b) is maintained during the operations involved in packet transmission, where s represents the backoff stage and b the value of the backoff counter [12].

[2]It is worth remarking that different contemporaneous invocations may be coupled by the network status, e.g. through buffer occupancy.

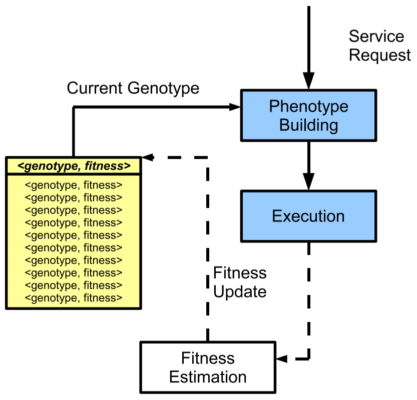


Figure 1: Architectural framework, including genotypes' pool and fitness estimation modules.

Example 5 (Connection Management in Peer-to-Peer Networks). *In peer–to–peer file sharing systems à la BitTorrent, peers having a file requested by other nodes in the system upload it to a number of peers in parallel. Each node can decide independently from the others to how many peers a requested content has to be uploaded, hence atomicity of the service. Different contents can be uploaded to different numbers of peers, therefore satisfying the stateless assumption.*

A network protocol/service is expressed in terms of a *genotype* (blueprint of the protocol/service), which can be processed by the node to obtain the corresponding *phenotype* (executable version, which might go through a developmental process). Only one phenotype is executed at a time. Each node maintains a pool of `<genotype,fitness>` pairs, i.e., information about possible protocol genotypes and their respective estimated fitness values, understood as a measure of the ability to perform the expected tasks in an effective way. Such an information is used to update the choice of the genotype to be expressed into a phenotype and executed.

The resulting system is shown in Fig. 1. Upon a service request, one genotype from the pool is selected and turned into an executable form. While in use, its fitness level is estimated and updated.

Two issues need therefore to be addressed: (i) how to build the genotype pool, and (ii) how to decide, on the basis of the information available in the pool, on the update of the service/protocol genotype in use.

Concerning point (i), an approach based on serial experiments could be used [13], according to which each node tries sequentially multiple possible genotypes and builds the genotype pool based on its own fitness estimations only. While such an approach does not require cooperation among nodes, it turns out to be extremely slow. In our framework, we opt for a cooperative solution, in which nodes may exchange information about the genotype currently in use. This is based on the assumption that the estimates performed by different nodes are (a) consistent, in that nodes communicating operate in similar conditions, and (b) trustworthy, in that nodes communicate the real value of the estimated fitness. Point (b) requires the introduction of a suitable trust and reputation framework, which is however out of the scope of this work. The exchange of `<genotype,fitness>` information and the consequent update of the genotype pool are graphically depicted in Fig. 2.

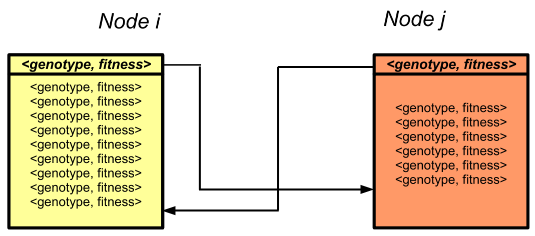


Figure 2: Exchange of `<genotype,fitness>` data among communicating node and update of the respective genotypes' pool.

Step (ii) requires to devise suitable mechanisms for making use of the information gathered on the performance of various genotypes. In general, we can formulate it as a global (multivariate, combinatorial) optimization problem. The solution space is the set of all possible genotypes; the aim is to find the best possible genotype based on the fitness values contained in the pool. Fitness values can be considered as (noisy) samples of an unknown function at given points in the solution space (the corresponding genotypes). We can therefore re-use (at least in principles) the various meta-heuristics introduced over the last decades for solving such kind of problems. These include, among others, genetic algorithms, simulated annealing, swarm optimization and ant colony optimization techniques [14, 15, 16, 17].

At the same time, such methods cannot be applied straightforwardly for the following reasons. First, an objective function of interest (e.g. a specific global performance metric of the network service) falls often in the set of so called *expensive black-box functions*, i.e. no analytical expression is available for the function nor for its derivatives. Furthermore, its evaluation is expensive, because it requires to execute on-line the corresponding phenotype, and is noisy due to the effect of a high number of unpredictable factors (e.g. random mobility, traffic variations, failures, etc). Second, the objective function depends in general on the whole set of basic networking services deployed at network nodes, hence networking services should be modified in a coordinated way in order to collectively maximize the objective function. A naive EC approach would require each node to consider directly the whole solution space, but this is often unfeasible because of the space dimensionality and because of the difficulty for each node to acquire a global knowledge. A second solution is to let nodes operate only on the local service (reducing the problem to a single agent problem), considering as target function the global objective function. This avoids the explosion of the solution space but it can lead to erroneous results because each agent is maximizing the objective function ignoring the actions of other agents. The desirable solution to this problems is to introduce a local objective function such that when each node maximizes its local evaluation function, the whole collective is maximizing the global objective function. But it is not easy in general to derive such local function from the global one (see [18] for a possible approach).

Among all the possible EC meta-heuristics, in this paper we focus our attention on the use of tools and techniques taken from the Genetic Algorithms (GAs) field [10]. In the standard case, two genotypes are selected from the pool according to some fitness-dependent mechanism (e.g., roulette wheel or tournament selection). Cross-over is then applied, and one of the two resulting genotypes is chosen at random. Mutation is then applied to the resulting genotype. A consistency check is then applied, in order to make sure the resulting genotype is a valid one (otherwise, the process is repeated until the condition is met). Such a genotype is then used to build the

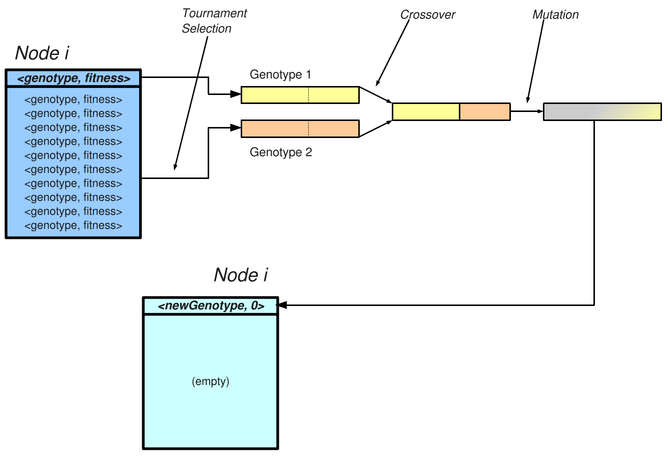


Figure 3: Graphical representation of the GA-based mechanism for updating the running protocol.

phenotype and executed. Correspondingly, the pool is emptied and the gathering/exchange of information is started again. The fitness value of the running phenotype is set to zero, until a reasonable estimate of its performance can be obtained. The whole process is illustrated in Fig. 3. GA-based techniques present the advantage of operating in a rather smooth way, making it unlikely to introduce solutions disrupting system's operations. At the same time, they are known to be slow in converging to a good solution, which represents a problem at system bootstrap (e.g., when a new node enters the system) and makes them rather inefficient in adapting to sudden changes in environmental conditions.

Even if we limit our investigation to GA approaches in this paper, we believe that the specific networking application we consider could benefit from the use of *hybrid* or *memetic* algorithms, i.e. methods combining evolutionary algorithms with one or more additional search techniques. A possible solution could be similar to that adopted in [19], which combines an evolutionary algorithm to explore the search space with another algorithm to identify clusters with potential minima and a trust-region approach, in which a local quadratic model can be used to efficiently converge to an extremum. The evolutionary algorithm is robust to the presence of noise and does not require a global model of the objective function, while the local quadratic model can locally approximate the function on the basis of a small number of points, can filter some noise thanks to interpolation, and can be efficiently optimized by a derivative-free algorithm.

3. Application to Epidemic Data Dissemination Schemes

The framework presented in the previous section is quite general and may potentially apply to a large set of problems. However, there are many design and engineering issues that have to be faced when mapping it to a real-world problem in computing and networking. In order to provide insight on the effect of such design choices and to provide guidelines to system designers, we have decided to present a case-study application of our framework. In particular, we have focused our attention on epidemic-style forwarding in delay-tolerant networks. Such a problem is appealing in that, besides being of practical interest for a quite wide range of applications, it fits well our framework of atomic stateless services and it is structurally simple enough to enable a deep study of the effect of various design choices.

Intermittently connected wireless systems represent a challenging environment for networking research, due to the problems of ensuring messages delivery in spite of frequent disconnections and random meeting patterns [20]. Many solutions have been proposed for use in such environments over the last few years. The common basis of these solutions is the observation that end-to-end routes may "be built" over time thanks to nodes mobility. Leveraging their mobility, nodes can exchange and carry other nodes' messages upon meetings, and deliver them afterward to the intended destination. This routing paradigm is referred to as *store-carry-forward*, each node in the network serving as a potential relay for all other ones. This paradigm defines a family of epidemic-like forwarding protocols, imitating the spread of viruses in nature. A node having a copy of the message is said to be *infected*, and as such, it can infect any other non-infected nodes by passing a message copy to them. If the message is copied at every meeting, the delivery delay is the shortest possible. However, this scheme is extremely wasteful of resources, like the channel capacity, the buffer space and the energy. We refer to this protocol as the unconstrained epidemic forwarding scheme.

The *epidemic routing* protocol was firstly proposed in [21]. Messages are simply flooded in the network, the only limitation being the maximal number of hops done by a message. Subsequently, [6] proposed PROPHET, a probabilistic forwarding scheme that is more sophisticated than epidemic routing. Using history of node encounters and transitivity, PROPHET achieves a performance comparable to that of epidemic routing but with a lower communication overhead. In [22], the multi-copy two-hop relay protocol was introduced, being a variant of the single-copy two-hop relay protocol proposed and studied in [23]. The infection process is largely constrained as any node can be infected only by the source and can itself infect only the destination. A new family of routing protocols was later proposed in [24]. This family, called *Spray–and–Wait routing*, can be viewed as a mix of single and multiple copies techniques, with the key feature being that a carefully chosen number of copies are generated and disseminated in the network to some relay nodes. The authors show that, if carefully designed, spray–and–wait routing incurs significantly fewer transmissions per message than epidemic routing, and achieves an interesting trade-off between efficient message delivery and low overhead.

Epidemic schemes may be combined with a so-called *recovery process* that deletes copies of a message at infected nodes, following the successful delivery of the message to the destination. Different recovery schemes have been proposed: some are simply based on timers, others actively spread in the network the information that a copy has been delivered to the destination [25].

We first observe that multiple forwarding schemes can co-exist at the same time in the network. In fact, this form of information delivery, based on the presence of multiple copies in the network, does not need nodes to be compliant with a specific behavior, enhancing system robustness with respect to conventional schemes. This flexibility comes from the completely distributed nature of the forwarding process in epidemic-style relaying, which allows node to use different policies in an uncoordinated fashion. We can therefore characterize epidemic-style forwarding as atomic services. Further, they are stateless in that no state information needs to be maintained across subsequent invocations.

Depending on the specific application scenario, different performance metrics can be envisaged. These include, e.g., the probability to successfully deliver a message to the destination, the delivery time, the total energy consumption in the system or a combination of the previous ones. For a given optimization goal, the choice of a specific forwarding scheme and the configuration of its parameters depend in general on the number of nodes in the system, on their mobility patterns and on the traffic generated in the network [26]. In many scenarios, these characteristics cannot be known at system design and deployment time and can also significantly change across time and space. For example, let us consider a Personal Digital Assistant carried by a user in its daily activities. During the day the node can travel at different speeds (e.g. from zero up to car speed), moving from highly crowded areas (supermarkets, classrooms,...) to less crowded ones, with very different trajectories (straight along a highway or following a random walk from shop to shop) and different levels of power availability. Loosely speaking, DTNs require forwarding policies to embed strong adaptation capabilities. In this sense, our framework, characterized by the ability of the protocols to self-optimize their behaviour, represents an appealing choice for such scenarios.

In this paper, we consider as performance metric for the system as a whole (also defined as "cost" in the following) the sum of the expected delivery delay of a message plus γ times the expected number of copies made by the forwarding scheme to deliver the message to the destination. The number of copies is strongly related to the total energy spent to deliver the information and to buffer requirement at the various nodes. Then the parameter γ defines how important are resource constraints in comparison to delivery delay.

With the specific implementation details being provided in the next section, we will illustrate now some of the challenges of applying our framework to epidemic data dissemination.

According to the description in Sec. 2, we want nodes to "learn" on-line what is the best forwarding policy (the local atomic service in this case) in the current scenario and change consequently the one they employ. The problem is challenging as a node cannot evaluate by itself whether its current policy fits the current scenario, because it is in general not aware of the consequences of its actions. For example a given node can never know by itself whether its decisions—according to its forwarding policy—to relay or not to relay a message were the right ones or not. Thus, a node may be relaying a message when the latter has already been delivered to its destination, hence wasting resources. On the other hand, a node may refrain from relaying a message when it happens to be the key node in the message delivery process, e.g., if it is the only node traveling between two disconnected clusters of nodes in the network. It should be clear therefore that a cooperative fitness estimation process has to be put in place in order to allow the network to show self-optimizing features. We also observe that the goodness (or fitness) of a node's policy depends on the policies implemented by the other nodes as well. This observation follows from the fact that message delivery is a collaborative process, whose performance depends on the behaviors of all nodes, so that a specific policy can be beneficial or detrimental depending on other nodes' actions. Furthermore, this observation is supported by a series of simulations that we conducted and we report on in Sec. 5.

This cooperative fitness estimation requires specific information exchange among nodes, but we observe that communication cost could become significant (nodes could end up using more energy to exchange policies information rather than to deliver the application messages), so that a communication-cost versus information-accuracy trade-off arises. Beside the communication overhead, another aspect to consider is the time needed to evaluate the cost of an infection. If information is delivered to a node long after message delivery, the node could have changed its genotype, so that the evaluation would not refer to the current genotype. In the next section we motivate specific choices related to these two trade-offs.

Another issue is related to the use of the fitness in the evolution process. Once a node gets marks for its own policy, how should these marks be used in order to change the policy? We rely on a *homogeneity* assumption, according to which (i) the nodes in the network can be partitioned in "large" groups of homogeneous ones, having similar mobility models and traffic patterns and

(ii) the optimal solution requires homogeneous nodes to adopt the same atomic service. If this assumption holds (some supporting results are provided in Sec. 5), then each node can learn from nodes in the same group: it can make its policy more similar to those policies presenting a higher level of fitness. This suggests that two other components are required in our framework: a unified description of the policies, so that each node can communicate to other nodes the one in use, and mechanisms for the generation of new (hopefully better) policies from existing ones.

Here we summarize the three fundamental components needed to apply our framework to epidemic data diffusion:

1. mechanisms for sharing with other nodes a description of the specific forwarding mechanism deployed at each node (the *genotype* associated to the forwarding policy employed at the node);
2. methods for evaluating the fitness of the schemes employed, rewarding "good" solutions in such a way to foster the diffusion of the fittest forwarding genotypes (performing a sort of *natural selection*);
3. techniques for each node to modify its forwarding scheme taking into account the schemes of other nodes (what we call, with a slight abuse of terminology, the genotype *evolution*).

4. Implementation Details

In this section, we report on a case study implementation of the proposed approach to epidemic-style forwarding in delay-tolerant networks. Our main objective with this implementation is to gain insight into the applicability of the approach proposed in Sec. 2. For this purpose, we also ran extensive simulations of this implementation which are reported on in Sec. 5.

In the following, we will detail:

- the optimization goal considered;
- the techniques used for estimation of the genotype fitness;
- the mechanisms employed for evolution of genotypes;
- the chosen genotype representation;
- the mutation process employed;
- the communication protocol used between nodes.

We would like to mention that a first effort towards evaluating the applicability of the proposed approach led to a very basic implementation that has been presented in [1]. The differences between the current implementation and that of [1] will be stressed along the text.

4.1. The Optimization Goal

In this implementation we consider, as optimization goal, the minimization of the expectation of the weighted sum of the delivery time of a message, T_D, and the number of copies of the message done in the system, C. The Cost Function (CF) is then defined as follows

$$CF = \mathrm{E}\left[T_D + \gamma C\right], \tag{1}$$

where γ defines how important are resource usage constraints in our application.

In order to minimize this cost we assume that each node can autonomously set the probability to make a copy of the message (its *forwarding probability* in what follows). Should the optimization goal be to minimize solely the expected delivery time, then the optimal data diffusion mechanism in an underloaded network—i.e., when traffic is small in comparison to the available capacity—would be message flooding in the entire network. Conversely, the presence of the number of copies in the cost function makes also an underloaded network (a realistic case) an interesting scenario to study. Such a metric is also meaningful as it is strongly related to bandwidth usage and power consumption. The evolutionary forwarding scheme will limit the number of copies to an extent that depends on the value of the parameter γ.

In our prior work [1], we were considering only the number of copies done at a subset of the infected nodes (those on the path of the first copy delivered to the destination), thus ignoring the energy consumption at other nodes. As such, (1) introduces the total energy consumption into the cost function to be minimized.

4.2. *Genotype Fitness and its Estimation*

Our definition of genotype fitness relies on the observation that a given infection will most likely involve only a subset of the nodes in the system. Conversely, a given node j will take part in only a subset of all infections. Hence, we define the fitness of the genotype at node j as:

$$\phi_j = \mathrm{E}\left[\left(1 - \frac{T_D + \gamma C}{R_{\max}}\right) \Big| \text{node } j \text{ contributed to infection}\right] \tag{2}$$

where $R_{\max}$ is set to a high enough value, so as to guarantee that fitness values are positive. Definition (2) ensures that the genotypes of nodes taking part in "good" delivery processes get on average a higher fitness than those involved in "bad" diffusion processes.

We have decided to consider as contributors to the infection of message n only those nodes that have forwarded the message copy that has first reached its destination. This set of nodes is denoted as $W^{(n)}$. This choice reduces the communication burden in comparison to considering all infected nodes.

To implement (2), a node needs to collect samples of the sum $T_D + \gamma C$ for each infection it has been involved in. In the following subsections we describe how each node can estimate the delivery time, the total number of copies and then the fitness of its forwarding policy.

4.2.1. *Delivery Time*

We assume that all nodes are synchronized and let the message header contain a field specifying the time at which the payload was generated at the source[3]. In such a way the delivery time of message n, denoted as $T_D^{(n)}$, is available at the destination once it receives the first copy of message n. The sample $T_D^{(n)}$ needs now to be conveyed to all nodes in $W^{(n)}$.

This is simply implemented by letting each node add its own identifier (ID) to the message header (this is analogous to what is done in source routing in mobile ad hoc networks). It follows then that the IDs in the header of the first copy of message n reaching the destination identify exactly the nodes in the set $W^{(n)}$. Therefore, it suffices that the destination node sends to these nodes a new *acknowledgment* (ACK) message, which specifies the delivery delay $T_D^{(n)}$.

[3]In real implementations, if local clocks are enough accurate at the message delivery time-scale, then there is another solution which does not require synchronization. Message header should have a field which indicates the time since the payload was generated. This field can be updated by each node before forwarding the message. In this case the node should keep track of the time running since it has received the message.

4.2.2. Total Number of Copies

Estimating the total number of copies is a more complex issue. In the simulation results shown later we assume that nodes know the number of copies done in the system when they receive the ACK and use this number as an estimate of the total number of copies, which will in general be larger as the infection can still be propagating.

We discuss briefly realistic estimation approaches, that we are evaluating. Each node can keep track of the number of copies it did for message n; let $C_j^{(n)}$ denote the number of copies of message n done by node j. One approach is to let each node broadcast its local number of copies and then evaluate the total number of copies aggregating the information received by other nodes, namely $C^{(n)} = \sum_{j=1}^{N} C_j^{(n)}$ (N is the total number of nodes). However, this implies a significant communication overhead.

We think it is better to rely on an estimation by adopting the "Plain Diffusion" solution proposed by [27]. In this algorithm, two meeting nodes exchange their current estimates and evaluate a new (better) estimate. The estimations at all nodes converge with probability one to the real value. However, this convergence is usually attained only some time after the end of the infection. Hence, nodes need to wait for some additional time after the reception of the ACK in order to be able to reliably update the fitness estimate.

4.2.3. Estimating the Fitness

Once node j knows the delivery time $T_D^{(n)}$ of message n, and has a reliable estimation—denoted $\hat{C}_j^{(n)}$—of the total number of copies, it can estimate the cost of spreading the message as $T_D^{(n)} + \gamma \hat{C}_j^{(n)}$, a quantity that we refer to as the reward $R_j^{(n)}$. Each reward received by a node is used to update its genotype fitness.

Consider now the set of infections a given node j has been contributing to, let M_j be this set. We can write $M_j = \{n | \ j \in W^{(n)}\}$, where n refers equally to a message and to its infection.

A naive approach to estimate the genotype fitness at node j would be to simply average the rewards over all messages in M_j, namely

$$\hat{\phi}_j = 1 - \frac{1}{R_{\max}} \frac{\sum_{n \in M_j} R_j^{(n)}}{|M_j|}.$$

In reality, this approach introduces a bias because infections with longer forwarding paths (i.e. with larger sets $W^{(n)}$}) generate a higher number of rewards in the system, namely $h_n := |W^{(n)}|$. The way to eliminate this bias is to consider the following weighted average:

$$\hat{\phi}_j = 1 - \frac{1}{R_{max}} \frac{\sum_{n \in M_j} \frac{R_j^{(n)}}{h_n}}{\sum_{n \in M_j} \frac{1}{h_n}}. \tag{3}$$

To compute (3), node j needs to know h_n, but this information is available because an ACK message needs to specify the whole set W^n in order to reward the nodes that contributed to the delivery (see Sec. 4.5).

4.3. Evolution of Genotypes

In order to minimize the cost function, the system needs to promote the diffusion of the fittest genotypes. In traditional Evolutionary Computation, and in particular in Genetic Algorithms (GAs), *reproduction* is a process which selects existing genotypes to create new offsprings. In practice, to implement this, it is required that nodes have information about a pool of genotypes, not only about the one they are currently using (cf. Sec. 2). Therefore, when two nodes meet, we require that they transmit to each other their own genotype and its current fitness estimation.

Each node maintains a pool of available genotypes (including the one currently in use) and their fitness values. We use G_j to denote the pool at node j and $\hat{\phi}_{i,j}$ to denote the fitness of the genotype used by node j ($\hat{\phi}_{i,j}$ is the value of $\hat{\phi}_i$ at the time of the last meeting between node i and node j, so at a given time instant these two values can be different). We consider a synchronized reproduction phase. Every T_g seconds, the *generation lifetime*, nodes synchronously create a new offspring each, i.e., they update their own genotype. Assuming synchronized operation simplify our analysis because it allows us to clearly identify different generations during the evolution. At the same time it is not difficult to design a system where nodes change their genotypes in an asynchronous way.

At each node, e.g. at node j, the genotype to be used as a basis for the new generation is selected with a probability depending on the relative fitness value[4]. One genotype selected from the pool can be mutated with probability p_m, before becoming the active genotype/forwarding policy of the node during the next generation[5]. The mutation is applied in the form of addition of a properly defined white noise (described below). The genotype pool is emptied after every generation. If the network is large, the node's pool may not be large enough to keep all genotypes discovered in a generation. Should this be the case, a node may keep only the fittest genotypes, or alternatively select the genotypes according to a fitness-biased distribution.

4.4. Genotype Representation and Noise

We consider a simple fixed-length genotype comprising one parameter, which is the probability P to copy a message upon encountering a new node. The probability P is maintained as a double-precision number. A mutation is then performed, with probability p_m, by adding to the gene value a randomly generated "noise." In our preliminary work [1], we have considered not a continuous, but a binary representation of the genotype, and have applied bit-flip mutation, in which each bit of the binary representation is flipped with some probability.

We want the noise to satisfy the following requirements:

1. zero mean noise;
2. negative and positive noise that are equally probable;
3. finer grain for smaller values;
4. a mutation towards smaller values should have a lesser extent than a mutation towards larger values.

[4]As it is shown later in section 5, the cost function has just one minimum, that is achieved when all nodes apply the same forwarding scheme. In such a case a non-linear scaling of fitness values can be employed in order to increase the *selection pressure* without the risk of *premature convergence* [10]. We adopt a Zipf's law scaling: if we rank the genotypes from the least fit to best fit, then the i_{th} genotype is selected with probability i-times higher than the first one.

[5]We ignore *cross-over* because we consider a genotype comprising a single parameter (the forwarding probability).

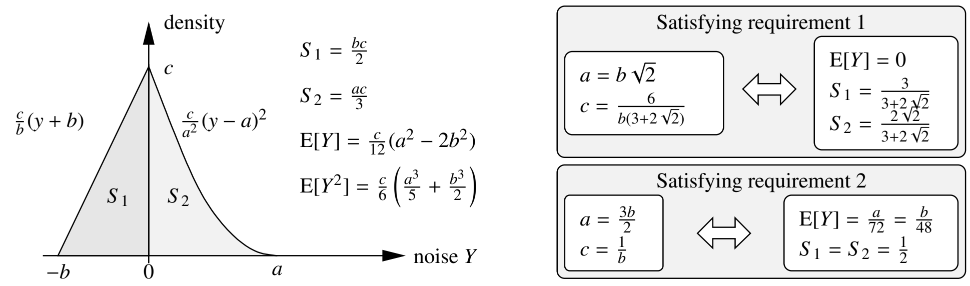

density
c
$\frac{c}{b}(y+b)$
$\frac{c}{a^2}(y-a)^2$
S_1
S_2
$-b$
0
a
noise Y
$S_1 = \frac{bc}{2}$
$S_2 = \frac{ac}{3}$
$\mathrm{E}[Y] = \frac{c}{12}(a^2 - 2b^2)$
$\mathrm{E}[Y^2] = \frac{c}{6}\left(\frac{a^3}{5} + \frac{b^3}{2}\right)$
Satisfying requirement 1
$a = b\sqrt{2}$
$c = \frac{6}{b(3+2\sqrt{2})}$
$\Leftrightarrow$
$\mathrm{E}[Y] = 0$
$S_1 = \frac{3}{3+2\sqrt{2}}$
$S_2 = \frac{2\sqrt{2}}{3+2\sqrt{2}}$
Satisfying requirement 2
$a = \frac{3b}{2}$
$c = \frac{1}{b}$
$\Leftrightarrow$
$\mathrm{E}[Y] = \frac{a}{72} = \frac{b}{48}$
$S_1 = S_2 = \frac{1}{2}$

Figure 4: Probability density function of the noise and values of parameters a, b and c for the satisfaction of requirements 3, 4 and either 1 or 2.

The first two requirements are standard and guarantee that mutations do not introduce a bias in system evolution. The third and fourth requirements are motivated from the observation that the cost function is more sensitive to smaller values of P. The fourth requirement ensures that the mutated genotype will incur sensibly the same deviation in the cost function regardless of whether the noise was positive or negative.

It turns out not to be possible to satisfy all four requirements simultaneously, as requirements 1, 2 and 4 are antinomic (one may ensure at most two out of three). Therefore we have decided for a distribution that satisfies requirements 1, 3 and 4. This may be obtained by multiplying the value of the genotype P by $(1 + Y)$, where Y has the probability density function depicted in Fig. 4, and $a = b\sqrt{2}$ and $c = 6/(b(3 + 2\sqrt{2}))$ in order to satisfy the first requirement. More details about the noise are provided in Appendix Appendix A.

4.5. Message Structures and Communications

The evolving protocol makes use of two types of messages to be exchanged over the network: DATA messages and ACK messages. DATA messages are those carrying the payload transmitted by any mobile node to a specific destination, whereas ACK messages are used for the following purposes:

- to acknowledge the successful delivery of the message at its intended destination;
- to feed back the reward to the nodes along the successful path from source to destination;
- to serve as anti-DATA, by blocking the diffusion of already delivered messages and removing them from nodes buffer.

The fields common to all message headers are (i) [message ID], which is the (unique) identifier (ID) of each message and also specifies whether it is a DATA or an ACK message (ii) [GenTime], which describes the time at which the message has been generated. Furthermore, DATA messages include the identifiers of all nodes which forwarded the message along the path from the source to the actual node. ACK messages, on the other hand, include the complete set of nodes involved in the forwarding path and the DATA message delivery time T_d (cf. Sec. 4.2).

Each mobile node maintains a unique internal data structure dedicated to the storage of both DATA and ACK messages. Each entry storing a DATA message additionally stores the current estimation of the total number of copies of that message.

Two nodes are able to exchange messages when they get within mutual communication range. Once it happens, they perform the following steps:

1. exchange node IDs;
2. exchange IDs of stored messages;
3. each node decides which messages should be forwarded to the other node;
4. messages are exchanged;
5. each node can drop some messages from its internal structure (if full) in order to free space for new messages.

For any DATA message to be relayed, a node adds its own node ID to the header. The message is kept in the node internal data structure until the corresponding ACK message is received or its lifetime expires. Unlike the case of DATA messages, no limiting policy is applied to the forwarding of ACK messages (ACK messages are simply *flooded* into the network according to the *vaccine* recovery scheme [11]). In our previous work [1], ACK messages were forwarded only to infected nodes according to the *immunization* recovery scheme, thereby yielding an increased infection duration.

Whenever a node receives an ACK message, it first inserts it into the internal message list and removes the corresponding DATA message (if present) once delivered to the node. Second, the node checks in the ACK message header whether it is part of the successful path to the destination, in which case it updates its own fitness, according to (3).

5. Performance Evaluation

In order to evaluate the performance of the presented algorithms, we have run extensive simulations using the freely available simulation tool *OMNeT++* [28].

We consider a scenario with N mobile nodes moving at constant speed v over a $L \times L$ square playground according to the Random Direction (RD) mobility model [29]. Each mobile node, when reaching the boundary of the simulation area, pauses for a randomly chosen time and, afterward, chooses uniformly an angular direction in the range $[0; \pi]$ radiants and starts to move. In our RD implementation, we considered mobile nodes to be moving at a constant speed with no pausing. The nodes' initial locations are sampled from a uniform distribution. As the uniform distribution is the stationary one under this mobility model, this characterizes a *perfect simulation* [30].

Two nodes are able to communicate if the mutual distance falls below the communication range r. No interference is considered, as we target sparsely deployed networks. Each mobile node generates a DATA message in a time interval which is uniformly distributed between 0 and T_s seconds, with a destination chosen uniformly among the nodes in the system. Each message generated is stored in the outgoing queue of the generating node. The position of every mobile node in the simulation is updated every T_{step} seconds. Each generation lasts for T_g units of time. Unless otherwise specified, the simulation settings and parameters used are those reported in Table 1. The default values of the mutation process are $p_m = 0.2$, and $b = 0.5$ ($c = 6/(b(3 + 2\sqrt{2}))$, $a = b\sqrt{2}$). All the confidence intervals are 95% ones.

Table 1: Simulation parameters

$L = 2500$m	$\gamma = 1800$s	$T_g = 1000000$s
$r = 25$m	$T_s = 10000$s	$p_m = 0.2$
$v = 1$m/s	$T_{\text{step}} = 2$s	$b = 0.5$

5.1. Sanity Check of Assumptions Made in Sec. 3

Given that message delivery is a collaborative task, we expect the fitness of a given node to be affected by the genotypes of other nodes in the network. In order to verify this, we have conducted a series of simulations with $N = 100$ nodes, having each node adopt a fixed forwarding probability throughout a simulation run. In the first experiment one node uses a forwarding probability equal to p_1, while all the others adopt a forwarding probability equal to p_2. We have considered different values of p_1 and p_2 in the range $[0, 1]$ and evaluated the average fitness of the single node using p_1. The fitness landscape is reported in Fig. 5. For a given value of p_1, it is clear that the node's fitness varies as a consequence of having all other nodes in the network change their forwarding probability. Similarly, even though to a lesser extent, if all nodes maintain their forwarding probabilities unchanged, a given node observes a change in its own fitness by changing its own forwarding probability. This observation suggests that other nodes' policies weight more in one's fitness than own policy.

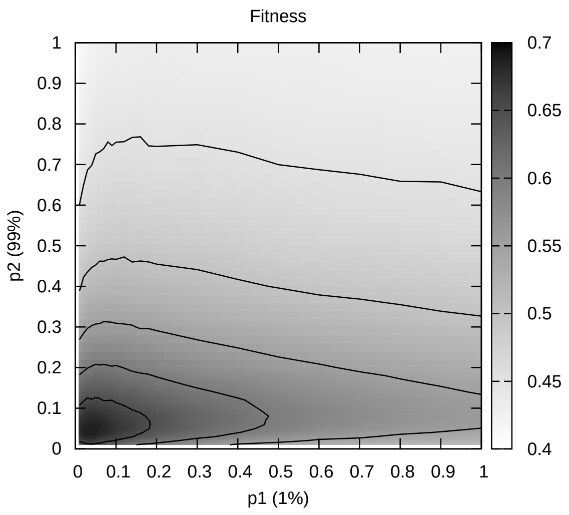


Figure 5: Fitness landscape of a node using a fixed probability p_1 while all other nodes in the network adopt the fixed probability p_2. Isolevel curves correspond to a given measured average fitness level.

We next assess the validity of the homogeneity assumption introduced in Sec. 3. According to this assumption, nodes can be classified according to their characteristics such that all nodes within the same class are homogeneous and are expected to have the same optimal behavior. Therefore, nodes that belong to the same *class* can learn from each other. In order to verify that the optimal behavior should be the same for all nodes within a class, we conducted a second series of experiments with $N = 100$ nodes in which message delivery was achieved through pure probabilistic forwarding, with all nodes adopting fixed forwarding probabilities, half of them adopting the forwarding probability p_1 and the other half adopting p_2. We have considered different values of p_1 and p_2 in the range $[0, 1]$ and evaluated the resulting cost function. The network's optimal operating point is represented by the minimum of the cost function across all the possible values of the forwarding probability. Figure 6 presents the results of this experiment. We observe that the configuration with the minimum cost (the isolevel curves help to identify it) correspond to a symmetric situation where all nodes adopt the same forwarding probability (around 0.03). This result supports the homogeneity assumption.

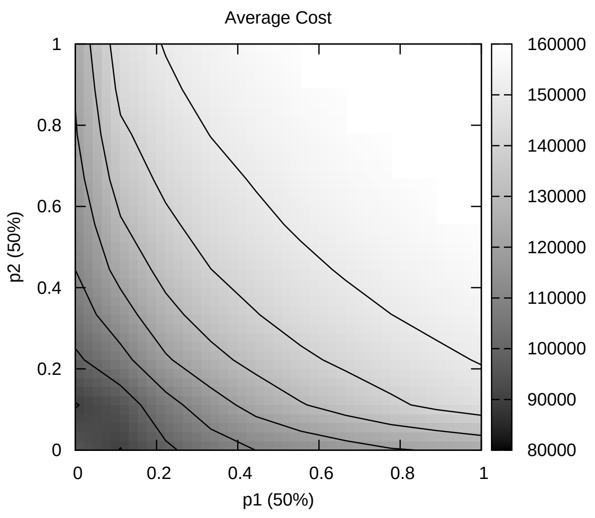


Figure 6: Average cost in the presence of heterogeneous policies (bimodal forwarding probability distribution). Isolevel curves correspond to a given measured average fitness level.

In most of the following experiments we will assume that all nodes in the network belong to the same class and should thus converge to the same optimal behavior. The case of two different classes of nodes having each a different communication range will be addressed in Sec. 5.7.

5.2. Convergence Time Evaluation

Our first objective is to assess the ability of the evolutionary forwarding mechanism to reach the network's optimal operating point. In our scheme, at each generation instant the system changes its behavior updating forwarding probabilities toward those values minimizing the cost function. This effect is evident in Fig. 7, which illustrates the distribution of the forwarding probabilities employed by each of the 100 nodes and the cost function CF over the generation number. As it may be seen, after approximately 10–15 generations, the cost seems to have reached a constant value and nodes use only a subset of the available probabilities. We refer to this steady state as a convergence state.

We can compare the performance of our scheme once convergence has been reached with the performance of a statically configured probabilistic scheme (shown in Fig. 6). For the sake

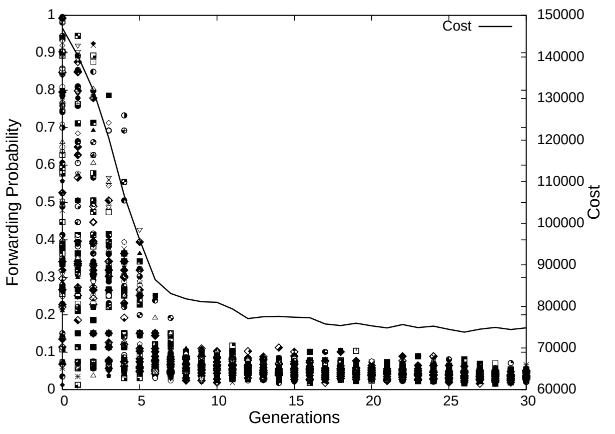


Figure 7: Cost function and forwarding evolution as a function of the generation number, $N = 100$ nodes.

of better readability Fig. 8 shows a curve representing the cost achievable when all the nodes in the network use the same forwarding probability, together with the average cost and probability achieved by the evolutionary scheme (evaluated from 60 runs). The figure clearly shows that our scheme is able to converge around the optimal operating point.

Given the implementation differences presented in Section 4, in [1] we show similar results for different values of the parameter γ and of the number of nodes N. In this paper we want to study systematically the effect of different parameters on the convergence process.

5.3. Impact of Noise

In the proposed scheme, the noise is exploited to gradually explore the genotype space: at every generation, new genotypes are generated, starting from those that were present in the nodes pool. The noise parameters regulate then *how far* are the newly generated genotypes from those existing before the evolutionary step. As customary in GA applications, the noise has a twofold effect on the system's behaviour. On the one hand, it helps in exploring wider portions of the genotype space, but on the other hand, it may induce oscillations around the optimal configuration once it is discovered by the algorithm.

This trade-off needs to be carefully considered when tuning the noise parameters. Indeed, the larger the noise, the faster the exploration of the genotype space, but also the larger the oscillations in steady–state. In order to investigate the impact of such an issue, several simulations have been performed, varying the noise parameters.

Figure 9 shows the average cost for different noise settings ("p_m – b" in the legend) obtained in the steady state versus the convergence time (100 simulation runs have been performed). The procedure used to automatically identify the steady state and determine the convergence time is described in Appendix B. For $p_m = 0.1$, it appears that the performance is not very sensitive to the parameter b and more simulations would be needed to draw conclusions. On the contrary, results for $p_m = 0.2$ confirms the expected trend: the larger the multiplicative noise (i.e. the higher b), the shorter the convergence time but the higher the cost in the steady state.

In what follows, unless otherwise specified, we consider $p_m = 0.2$ and $b = 0.5$.

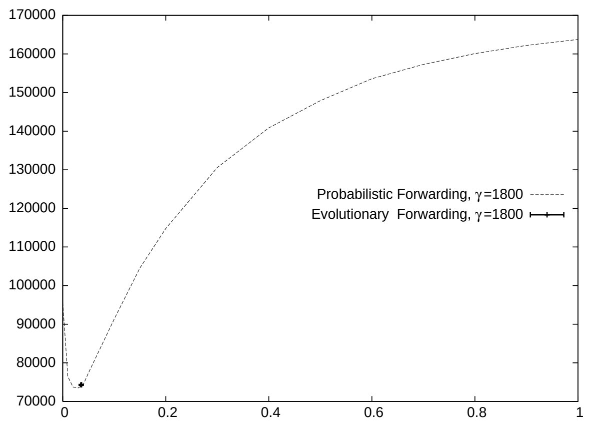


Figure 8: Cost function value as a function of the forwarding probability (statically set) and comparison with the results from the proposed evolutionary forwarding scheme.

5.4. Impact of Generation Time

The generation time T_g, described as the time elapsed between two subsequent reproduction processes run on a given node, is a critical parameter of our algorithm, since the accuracy of the fitness estimation process depends heavily on its duration. The longer the generation time, the larger the amount of feedback on different infections that each node can collect and then the more accurate its fitness estimation. There is also a more subtle effect to take into account. A node can receive, at a given time, feedback related to messages it has forwarded in the previous generation. This feedback cannot be referred to the genotype currently used by the node, and so it is ignored. But we observe that such feedbacks correspond to messages whose average delivery time is longer than the delivery time of other messages, and they introduce therefore a bias in the samples considered in (3). The shorter the generation time, the higher the impact of such a phenomenon.

Table 2: Convergence state for different values of the generation time

Generation time	Probability	Cost	Gen. # until Conv.	Time until Conv.
$5 \cdot 10^5$	0.0405331	$[74940, 75269]$	$[19.30, 26.47]$	$\approx 11 \cdot 10^6$
10^6	0.0358154	$[74161, 74406]$	$[14.30, 17.16]$	$\approx 16 \cdot 10^6$
$2 \cdot 10^6$	0.0350766	$[74107, 74282]$	$[14.35, 18.35]$	$\approx 32 \cdot 10^6$

Table 2 reports the performance attained for three different values of the generation time ($T_g \in [5 \cdot 10^5; 10^6; 2 \cdot 10^6]$). Such results were evaluated from 60 runs with 100 nodes. We observe that the performance in the steady state (forwarding probability and cost) and the number of generations until convergence are similar for the two largest values of T_g. Hence, increasing the generation time from 10^6 to $2 \cdot 10^6$ does not lead —for the specific settings considered— to any significant improvement in the fitness estimation process. Clearly, the same number of

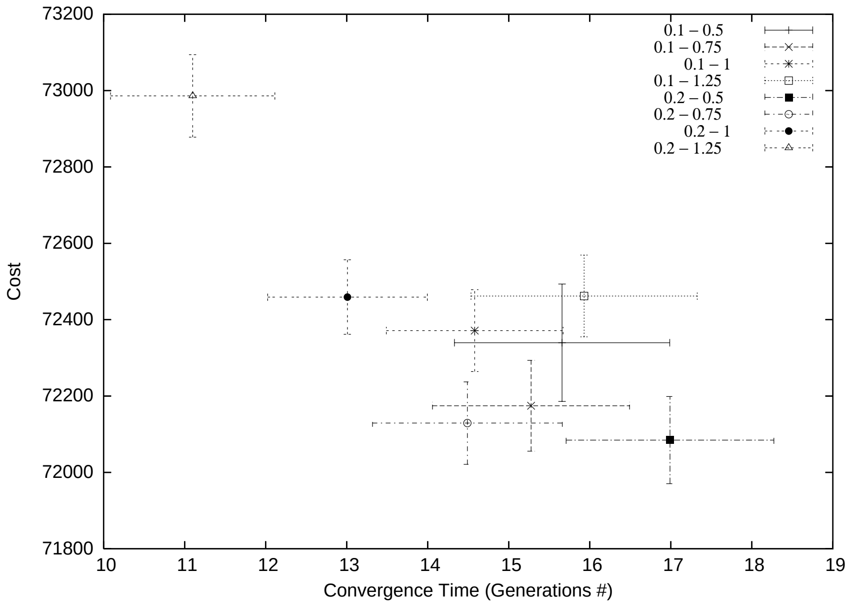


Figure 9: Cost function value vs. convergence time for various values of the noise parameters, 95% confidence interval.

generations corresponds to a convergence time larger for the system with $T_g = 2 \cdot 10^6$ and we can conclude that $T_g = 10^6$ is preferable in order to have a more responsive system.

For $T_g = 5 \cdot 10^5$ we can observe a different effect: the number of generations needed to achieve convergence is higher than in the other two cases. This is due to the higher noise in the fitness estimation process. Nodes tend therefore to take "wrong" decisions more often, leading to a longer convergence time in terms of number of generations. Nevertheless, as each single generation lasts less, the convergence time (in seconds) is shorter than in the other cases. In any case we observe that, due to the inaccuracies in the fitness estimation process, the convergence state is different and the corresponding average cost results higher.

5.5. Impact of Number of Nodes

In this section we evaluate the impact of the number of nodes on the convergence time. We observe that this is *a priori* different from considering how the population size affects the convergence of a traditional genetic algorithm. In fact, in our case (i) the solution space dimensionality nominally increases with the number of nodes N (namely the solution space is $[0, 1]^N$) and (ii) the same objective function is not fixed but depends on the number of nodes (e.g. cost surface in Fig. 6 would have a different shape with a different number of nodes).

We performed 60 runs, considering a number of nodes varying from 25 up to 400. The playground size has been varied accordingly, in order to keep a constant node density. The results, presented in Tab. 3, show that, as the number of nodes increases, the number of generations needed to achieve convergence first increases and then slightly decreases. Interestingly, despite the differences observed above, these results seem to agree with the *piecewise convergence-time model* [31] for standard GA. Such a model combines two known results:

Table 3: Convergence state and corresponding cost for different values of the parameter N (number of nodes in the system).

Node #	Probability	Cost	Gen # until Conv.
25	0.120	[23017, 23659]	[6.16, 13.91]
50	0.061	[41319, 41726]	[10.55, 15.65]
100	0.036	[74118, 74401]	[13.40, 17.73]
200	0.022	[124515, 124877]	[18.22, 21.05]
400	0.014	[208579, 210445]	[14.44, 15.84]

- for large populations, convergence time results can be obtained by considering the selection pressure induced by the fitness–dependent selection phase of the GA [32];
- for small finite size populations, the convergence can be determined by the *genetic drift* [33], i.e. a random fluctuation of gene frequencies can lead to a steady state even in absence of a selection pressure in the GA. The expected time for this to happen is linear in the population size.

The authors observe that in presence of fitness pressure, the fitness-based selection strategy leads normally to convergence on a shorter time scale than that of genetic drift, but for small population size the drift could be dominant. It can be expected then that the convergence time first increases almost linearly (being mainly determined by the genetic drift factor) and then saturates to an almost–constant value. As the population size grows, larger portions of the solution space can be explored in parallel; coupled with the presence of selection pressure in the reproduction process, this can lead to a reduction of the time needed to achieve convergence. The model in [33] predicts, for some objective functions, a non–monotonic behaviour for the convergence time as a function of the population size, similar to what we have found in our case. We conjecture that this similarity is in part caused by our homogeneity assumption. Indeed, even if the search space nominally grows with N, the real space in which the nodes look for the symmetrical solution is (almost) uni-dimensional, as homogeneous solutions are the optimal ones.

5.6. Adaptation to Changing Conditions

We investigated the ability of our scheme to adapt to a dynamic environment, in which network settings change over time. We considered a scenario in which the number of nodes changes abruptly, starting from 25 nodes, up to 100 nodes, then up to 300 nodes and then it reduces again to 100 and finally to 25 nodes. Differently from the previous section, here we do not rescale the playground size, but we keep it equal to a $2500\times2500\ m^2$ area. For this simulation we considered $p_m = 0.1$.

Figure 10 depicts the trend of the value taken by the cost function and of the average forwarding probability versus the generation number. The cost plot presents spikes whenever N increases. The abrupt change is mainly due to the arrival of new nodes, whose initial forwarding probability is set uniformly at random. During the transient following a change, genotypes better fitted to the new scenario are identified and the cost drops. As an example, the system is able to let the forwarding probability decrease when the number of nodes increases and vice versa.

The two time intervals when there are only 25 nodes (generations $1-20$ and generations $70-90$) illustrate interesting effects. First, it appears that in the first 20 generations the average

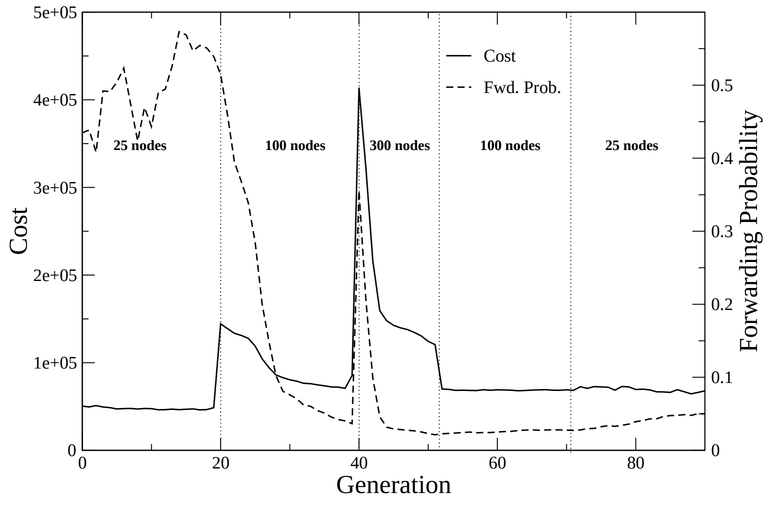


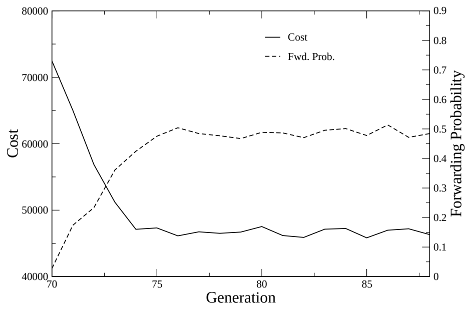
Figure 10: Dynamic scenario: cost function value and average forwarding probabilities versus generation number.

forwarding probability changes significantly across time, without showing any clear convergence trend. The constant cost value reveals that in such conditions the cost function is not very sensitive to changes in the forwarding probability. At the same time the extremely low density of the network causes a high variability of infection costs (i.e. the delivery time can change significantly from message to message), resulting in highly noisy fitness estimates. These, in turn, can hide the potential improvements coming from the adoption of specific genotypes. A different behaviour can be observed when the number of nodes decreases from 100 to 25. In this case, indeed, the system tends to slowly increase the forwarding probability. There is in fact a difference in the two possible change directions related to the *diversity* of the resulting population. When new nodes are added, the newcomers draw at random a seed forwarding probability, as they do not have any knowledge of the network scenario. This randomness injects a sufficient level of genotype diversity, allowing our scheme to explore, in parallel, a wide range of possible forwarding probabilities. Conversely, once the system has converged to the optimal value for a specific setting and the number of nodes is reduced, the system has a very low level of genetic diversity, and it results therefore extremely slow in the search for a better solution.

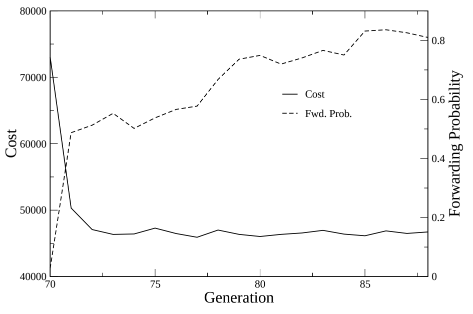
In order to deal with such limited diversity in the population, and to have a larger variance of the noise to boost the convergence towards fitter genotypes, we have modified the original scheme as follows (we will refer to such variant in the following as the "modified scheme"). When an abrupt change in the environment conditions is detected, nodes select with probability p_{rand} the forwarding probability uniformly in $[0,1]$. The occurrence of abrupt changes in the network settings is detected by monitoring variations in the number of nodes met during subsequent generations. If such a parameter varies by more than 10%, then a random forwarding probability is used with probability p_{rand}.

The performance of the modified scheme has been evaluated with reference to the previous scenario, when the number of nodes reduces from 100 to 25. In these new simulations the 25 nodes start with the same forwarding probabilities that they had at the end of the 70^{th} generation and the evolution of the system is then analyzed.

Figure 11 presents the results of the modified scheme in the case of $p_{rand} = 0.5$ and $p_{rand} = 1$. As it can be observed, in both cases the cost reduces over the generations. In particular, the cost is lower than with the original scheme (Fig. 10). The case $p_{rand} = 1$ achieves a faster convergence. In such a case, indeed, the system re-starts from scratch; the introduced genetic diversity enables the scheme to converge fast to the actual optimal operating point for the system.



(a) $p_{rand} = 0.5$.



(b) $p_{rand} = 1$

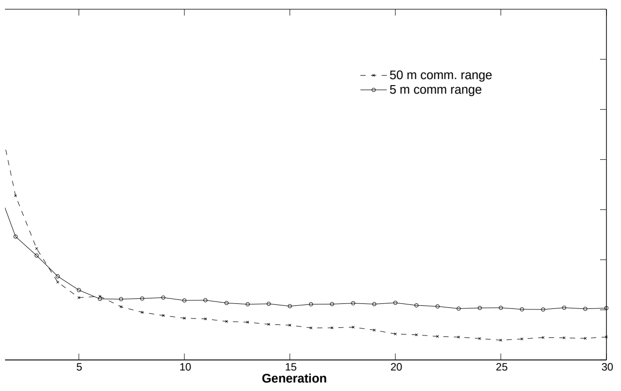
Figure 11: Performance of the modified scheme (in terms of cost function value and average forwarding probability) in a dynamic scenario, $p_{rand} \in \{0, 5, 1\}$, N dropping from 100 to 25.

5.7. Heterogeneous Case

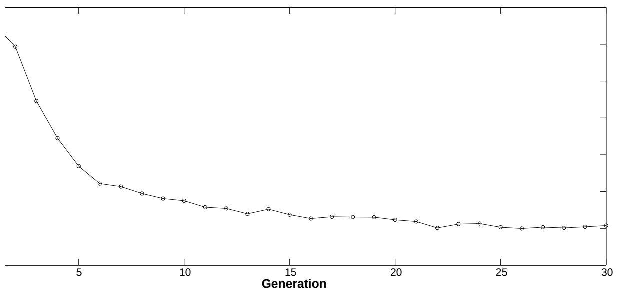
In this last experiment, we evaluated the system performance in the case of a heterogeneous scenario, in which different classes of nodes are characterized by different communication ranges. All nodes participate equally to the data forwarding process, meaning that whenever in mutual communication range they exchange data applying their current forwarding probability. At the same time, we assumed that only nodes belonging to the same class participate to the evolution process. This translates in the different classes of nodes evolving their forwarding strategy in parallel. We have then evaluated (i) the convergence of the forwarding probability for each class of nodes and (ii) the overall cost function.

In Fig. 12, we reported the performance attained in the case of $N = 100$ nodes that are split into two classes of 50 nodes. The first class is charaterized by a communication range of 5 m, whereas it is 50 m for the second class of nodes. As it is possible to observe from Fig. 12(a), each class of nodes converges to a different forwarding probability: as expected, nodes with a lower communication range converge to a higher one. The larger forwarding probability compensates for the smaller number of meetings due to the reduced communication range. Figure 12(b) shows the average cost versus the number of generations. As it can be observed, the cost decreases with a dynamic that is very similar to the homogeneous case, although this is the result of the two evolution processes occurring in parallel.

We have run various other simulations with different settings, varying the gap between the communication ranges of the different classes of nodes. In all cases, we obtained a similar pattern where nodes with the lower communication range converge towards a higher forwarding probability. This confirms the ability of the proposed evolutionary approach to accommodate heterogeneous scenarios.



(a) Forwarding probability over generations.



(b) Cost over generations.

Figure 12: Performance evaluation in the case of two classes of nodes characterized by a communication range of 5 and 50 meters.

6. Other possible applications

In this section we provide a sample list of other networking services to which our framework can be applied.

6.1. Network Coding in DTNs

Network coding, i.e. the combination of different messages inside the network, has been shown to be able to achieve maximum information flow in a network [34]. Recently, Random Linear Coding (RLC), a special form of network coding, has been proposed to be applied also to unicast transmission in Intermittently Connected Networks in order to reduce information delivery delay [35]. In particular [35] considers that a source can group its messages to a given destination in blocks of size K. Relay nodes may then generate random linear combinations of messages in the same block. Such combinations, together with the corresponding vectors of coefficients, are going to be forwarded to other relay nodes and to the destination. The destination can decode the entire block of messages when it has received K independent combinations. In [35] the authors show that the time to receive K independent combinations is shorter than the time to receive all the K messages separately in absence of coding. In particular the smaller the node buffer size, the larger the potential improvement coming from RLC. But if buffer sizes are large enough to accommodate all the messages each relay node receives, the improvement from RLC can be marginal, and RLC cannot be an advantageous solution, considering the increased overhead due to the transmission of coefficients and the decoding complexity (both increasing with K). Hence, depending on the specific network scenario and in particular on its traffic load, a source may choose to apply or not network coding, or, with a more granular tuning, may select an appropriate value of K (where $K = 1$ corresponds to the non-coding case). The following global cost function can be considered:

$$CF = \mathrm{E}\{T_D + H\},$$

where T_D is the delivery time of a message in the block[6] , H is a measure of the overhead to be incurred when applying network coding. Both T_D and H depends on the block sizes selected by *all* the nodes in the network. The optimization of this cost function appears then similar to our case study. If nodes are quite homogeneous (or they can be split in homogeneous groups), then they can perform a cooperative optimization, where the genotype of a node j is the value K it has adopted and the corresponding fitness can be defined as

$$\phi_j = \mathrm{E}\left\{\left(1 - \frac{T_D + H}{R_{max}}\right) | \text{ node j is the source of the message}\right\},$$

where R_{max} is simply a value high enough to guarantee that fitness values are positive. T_D can be estimated thanks to some feedback from the destination, while H is uniquely determined from a the size of the block at node j. We note that RLC based block delivery is an atomic and stateless service. In fact each node can select its value of K independently from any other node, and it can change it at the transmission of the following block.

[6]If T_B is the time after which a block of size K can be decoded, $T_D = T_B/K$.

6.2. Decentralized Medium Access Control (MAC) in Dense Ad-Hoc Networks

We consider decentralized medium access control techniques, such as the CSMA/CA technique used in the IEEE 802.11 family of standards for wireless local area networks. In this case, the invocation of the service corresponds to a request to layer 2 to send a packet to a nearby wireless node. A critical parameter to determine the performance of the access control technique is the size of the maximum back-off window W. When a node wants to transmit, if it senses a collision on the channel, it draws a number w uniformly at random between 0 and W and will wait w time slots before trying to access the channel again. As it can be easily understood, the value of W strongly affects the performance. In fact, if it is too small, nodes continuously experience collisions and frames might not be forwarded, while if it is too high, transmissions are uselessly postponed. In both cases queue sizes increase and larger delivery delays are experienced. Let us consider an ad-hoc network (not necessarily mobile) where nodes are implementing decentralized MAC and the maximum back-off window value can be independently tuned by each node. The following cost function could be a natural optimization target:

$$CF = \mathrm{E}\left\{\sum_i Q_i\right\},$$

where Q_i is the backlog of frames to be forwarded at node i. It depends from the maximum back-off window at the node (W_i), but also on window values at other nodes, because, for example, a neighbor with a very small window may monopolize the channel and impede any transmission from node i. We observe that the MAC service is atomic in that it does not require for example coordination about the back-off window values across the network. It is also stateless because no state needs to be maintained between two requests to transmit different messages, and in particular W can be changed for each message. In order to apply our cooperative optimization framework, we need nodes to work in conditions similar to their neighbors. This is the case if for example 1) the network is very dense and its node density changes quite smoothly in the considered area, and 2) the amount of traffic generated or terminated at the node is small in comparison to the traffic it receives to be relayed. These conditions guarantee that a node and its neighours have almost the same type of connectivity and the same amount of traffic to be forwarded. Clearly in this case the genotype of node i would be its current value W_i. A possible fitness function could be:

$$\phi_j = \mathrm{E}\left\{1 - \frac{Q_i + \sum_{j\in N_i} Q_j}{R_{max}}\right\},$$

where N_i is the set of neighbors of node i. This requires neighbors to periodically exchange their queue sizes, beside their genotypes and the corresponding fitness values.

6.3. Connection Management in File Sharing P2P Networks

In BitTorrent networks [36] each peer uploads in parallel to N other peers. In the original protocol specification, N was selected to be equal to 5, but in a newer version of the mainline client N is proportional to the squared root of the client upload capacity. This value is a critical parameter to determine the speed of content spreading in the group of interested peers (the swarm). First, if N is too small, then a client may waste its upload capacity. At the same time, the larger N, the longer the upload time of a file piece[7] and then the slower the replication of

[7]Each file is divided in pieces. Peers do not need to have the whole file to start uploading pieces.

the piece in the network. From these two remarks it seems that N should be set equal to the minimum value that guarantees full utilization of the upload capacity. In reality there are other aspects to consider. In fact, due to BitTorrent Tit-for-Tat incentive mechanism, a peer mainly uploads pieces (reciprocates) to its best uploaders, then the larger N, the higher the number of potential uploaders, but at the same time the lower the probability to be reciprocated by them. Then we can expect that there is an optimal number of parallel uploads for each peer, but this is difficult to determine analytically and depends in general on three characteristics of the peer itself, but also of all the other peers: the capacity, the content already downloaded, and the arrival time in the swarm.
Similarly to the other examples above, let us consider that a generic peer, say peer i, can change its number of uploads N_i. This does not require any coordination with other peers and the choice can be changed dynamically. It is natural to consider as Utility Function (UF) the total uploading rate in the network (equal to the total downloading rate) at time t:

$$UF = \mathrm{E}\left\{\sum_i r_i(t)\right\},$$

where $r_i(t)$ is the uploading rate of peer i at time t, depending on the vector of values N. We can apply our framework considering N_i as the genotype of node i and

$$\phi_i(t) = r_i(t),$$

as the fitness of node i. Genotypes and corresponding values can be exchanged among homogeneous nodes. Here two nodes are homogeneous if they have similar capacity values, they have joined the system in close time instants and have a comparable share of the file.

7. Conclusions

In this paper, we have presented a framework for the on-line evolution of network protocols. The framework in itself is quite general and relies on evolutionary techniques in order to optimize, in a fully distributed fashion, the communication protocol being run by the nodes of a network. In particular, the framework applies to services that are *atomic and stateless*, though their performance may depend on the interactions with other nodes.

As a case study, we have applied the proposed framework to the evolution of forwarding schemes in intermittently connected wireless networks. We have analyzed the various aspects involved in its implementation, and performed an extensive simulative study for evaluating its performance. Results show how the proposed scheme, independently from the number of nodes, is able to autonomously evolve its operational parameters in order to optimize the considered performance function. At the same time, our study clearly shows that the design space typical of such framework presents many degrees of freedom (in terms of parameters involved in the application of the genetic algorithm as well as in terms of the design of appropriate mechanisms for evaluating the fitness level of the solution in place). The performance of the resulting applications can be strongly influenced by such design choices. Further, the arising of trade-offs between convergence time and stability in steady–state requires careful considerations. One research direction of particular interest appears the application of techniques for the automatic tuning of (some of) the design parameters, along the lines of meta–evolutionary algorithms [37].

The discussion on the case study implementation clearly shows that our framework introduces additional complexity with respect to statically designed network protocols. This refers in particular to (i) the need to introduce appropriate signalling mechanisms for enabling nodes to exchange information on the current set of solutions in use (ii) the potential difficulties in tuning some of the parameters that drive the adaptation at the single node level. This seems to be the unavoidable price to pay for having a system that is able to optimise its behaviour at run–time, a feature that is of paramount importance in situations where working conditions cannot be identified (or span a too wide range) at design time.

Acknowledgements

This work has been partially supported by the EC within the scope of the BIONETS Project EU-IST-FP6-027748 (`www.bionets.eu`).

Appendices

Appendix A. On the Distribution of the Mutation Noise

Among the noise requirements specified in Sec. 4.4, requirements 1 and 2 are consistent with the well-known additive Gaussian noise. Actually, any symmetric, zero mean noise satisfies the first two requirements. However such type of noise does not satisfy the fourth requirement, which is needed because of the increased sensitiveness of the cost function to small values of P. It turns out not to be possible to satisfy all four requirements simultaneously. Therefore we have decided for a distribution that satisfies requirements 1, 3 and 4.

One simple way of achieving the third requirement is to consider the additive noise to be *proportional* to the value P. This may be obtained by multiplying the value of the genotype P by $(1 + Y)$, where Y is an opportune random variable. If $\mathrm{E}[Y] = 0$ then $\mathrm{E}[(1 + Y)P] = \mathrm{E}[P]$ and the first requirement is also satisfied.

Satisfying the fourth requirement can be achieved by having for Y a distribution with smaller support at negative values. The specific probability density chosen is shown in Fig. 4.

Another issue to address is related to the bounds of the gene values, e.g. the forwarding probability P is in the range $[0, 1]$. In our case, we have to deal only with the boundary 1, as the genotype can get only infinitely close to 0 but can never go below 0. Three methods to deal with boundaries are considered in the literature:

1. iterate until the value falls within the boundaries;
2. try once and bounce the value between the boundaries until it falls between them[8];
3. try once and truncate the value to the boundary.

All three methods introduce a bias towards smaller values in the distribution of P. We did not perform an extensive study to evaluate which method is most suitable. We decided to use the first method relying on the following intuitive explanation. We expect the third method to have the smallest bias as it modifies the least the original value. However, given the structure of the noise, the exploration of the state-space of the probability is much slower towards the left than towards the right. Having then a bias towards smaller values is not as an important issue as one

[8]When bouncing a value at a boundary, we retain the excess value that lays beyond the boundary and fold it back to the other side of the boundary, e.g. the value 1.2 bounces on the upper boundary 1 so that the value after bouncing becomes 0.8.

may think at first. As for the second method, we conjecture that it will be more suited when the noise distribution is symmetric, which is not the case here.

In practice, we use the first method with a pre-defined (high) limit of trials in order to avoid the system to get trapped on a long sequence of unsuccessful iterations. In case the limit of trials is exceeded, the third method is applied.

Appendix B. On the Computation of the Convergence Time

We define the convergence time as the time needed to reach the steady state. It is clear that in order to identify this time, we need first to characterize the steady state. We have adopted a simulation-based approach similar to the control charts used to identify when a system is out of control [38], i.e. when the characteristics of the process become really different from the standard ones.

We have selected the average system cost for each generation as a synthetic metric characterizing the system. For a given setting we have launched R independent simulations of length equal to G generations, where G is high enough so that we can be confident that the system has reached the steady state in the last g generations (this has been established by visual inspection of cost time evolution). Then, we have processed the data in the last g generations of the R independent runs, estimating the confidence interval for the expected cost in steady state. In particular, we have first obtained the average cost and its standard deviation at each of the L generations, and then we have averaged such values across all the g generations. Once we have obtained this characterization of the system in steady state, we can quantify the convergence time as the time until the cost stays within the confidence interval. In particular, we take as convergence time the time instant after which the cost can exceed the boundaries of the confidence interval only occasionally for no more than one generation (i.e. if it exceeds the boundaries at one generation, then it falls back in the interval at the following one). This approach is similar to Shewhart chart [38], widely used in industrial applications to detect when a system does not operate normally. In fact we have simply proceeded by reverting the cost time evolution and identifying with the Shewhart chart the time the system is no more in steady state.

In the paper we have considered $G = 50$, $g = 10$ to characterize the steady state.

References

[1] S. Alouf, I. Carreras, D. Miorandi, G. Neglia, Embedding evolution in epidemic-style forwarding, in: Proc. of IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2007), IEEE Press, Piscataway, NJ, USA, 2007.

[2] S. Alouf, I. Carreras, A. Fialho, D. Miorandi, G. Neglia, Autonomic information diffusion in intermittently connected networks, in: M. K. Denko, L. T. Yang, Y. Zhang (Eds.), Autonomic Computing and Networking, Springer, 2009, Ch. 17, pp. 411–433.

[3] C. F. Tschudin, Flexible protocol stacks, in: Proc. of ACM SIGCOMM, 1991, pp. 197–205.

[4] S. W. O'Malley, L. L. Peterson, A dynamic network architecture, ACM Transactions on Computer Systems 10 (1992) 110–143.

[5] R. Braden, T. Faber, M. Handley, From protocol stack to protocol heap: role-based architecture, ACM SIGCOMM Computer Communication Review 33 (2003) 17–22.

[6] A. Lindgren, A. Doria, O. Schelen, Probabilistic routing in intermittently connected networks, in: Proc. of SAPIR Workshop 2004, Vol. 3126 of LNCS, Springer, Berlin / Heidelberg, 2004, pp. 239–254.

[7] K. Fall, A delay-tolerant network architecture for challenged Internets, in: Proc. of ACM SIGCOMM 2003, ACM, New York, NY, USA, 2003, pp. 27–34.

[8] L. Pelusi, A. Passarella, M. Conti, Opportunistic networking: data forwarding in disconnected mobile ad hoc networks, IEEE Communications Magazine 44 (11) (2006) 134–141.

[9] I. Carreras, I. Chlamtac, F. De Pellegrini, D. Miorandi, BIONETS: Bio-inspired networking for pervasive communication environments, IEEE Trans. on Vehicular Technology 56 (1) (2007) 218–229.

[10] A. Eiben, J. E. Smith, Introduction to evolutionary computing, Springer, 2003.

[11] X. Zhang, G. Neglia, J. Kurose, D. Towsley, Performance modeling of epidemic routing, Computer Networks 51 (10) (2007) 2867–2891.

[12] G. Bianchi, Performance analysis of the ieee 802.11 distributed coordination function, IEEE JSAC 18 (2000) 535–547.

[13] J. J. Ramos-Muñoz, L. Yamamoto, C. Tschudin, Serial experiments online, ACM SIGCOMM Computer Communication Review 38 (2008) 31–42.

[14] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268–308.

[15] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[16] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, 2004.

[17] A. P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, Wiley, 2005.

[18] D. H. Wolpert, K. Tumer, Optimal payoff functions for members of collectives, Advances in Complex Systems 4 (2/3) (2001) 265–279.

[19] Y. Tenne, S. W. Armfield, A memetic algorithm using a trust-region derivative-free optimization with quadratic modelling for optimization of expensive and noisy black-box functions, in: S. Yang, Y.-S. Ong, Y. Jin (Eds.), Evolutionary Computation in Dynamic and Uncertain Environments, Vol. 51 of Studies in Computational Intelligence, Springer, 2007, pp. 389–415.

[20] Z. Zhang, Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges, IEEE Communications Surveys and Tutorials 8 (1) (2006) 24–37.

[21] A. Vahdat, D. Becker, Epidemic routing for partially connected ad hoc networks, Tech. Rep. CS-200006, Duke Univ. (2000).

[22] R. Groenevelt, P. Nain, G. Koole, The message delay in mobile ad hoc networks, Performance Evaluation 62 (1-4) (2005) 210–228.

[23] M. Grossglauser, D. Tse, Mobility increases the capacity of ad hoc wireless networks, IEEE/ACM Transactions on Networking 10 (4) (2002) 477–486.

[24] T. Spyropoulos, K. Psounis, C. S. Raghavendra, Efficient routing in intermittently connected mobile networks: The multiple-copy case, IEEE/ACM Transactions on Networking 16 (1) (2008) 77–90.

[25] Z. J. Haas, T. Small, A new networking model for biological applications of ad hoc sensor networks, IEEE/ACM Transactions on Networking 14 (1) (2006) 27–40.

[26] G. Neglia, X. Zhang, Optimal delay-power tradeoff in sparse delay tolerant networks: a preliminary study, in: Proc. of ACM SIGCOMM CHANTS 2006, ACM, New York, NY, USA, 2006, pp. 237–244.

[27] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, T. Urnes, Design patterns from biology for distributed computing, ACM Transactions on Autonomous and Adaptive Systems 1 (1) (2006) 26–66.

[28] OMNeT++ Discrete Event Simulation System, `http://www.omnetpp.org` (2007).

[29] E. M. Royer, P. M. Melliar-Smith, L. E. Moser, An analysis of the optimum node density for ad hoc mobile networks, in: Proc. of IEEE ICC 2001, Vol. 3, IEEE Press, Piscataway, NJ, USA, 2001, pp. 857–861.

[30] J.-Y. Le Boudec, M. Vojnovic, Perfect simulation and stationarity of a class of mobility models, in: Proc. of IEEE INFOCOM, 2005, pp. 2743–2754.

[31] A. Ceroni, M. Pelikan, D. E. Goldberg, Convergence-time models for the simple genetic algorithm with finite population, IlliGAL Report No. 2001028, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL (2001).

[32] D. Thierens, D. E. Goldberg, Convergence models of genetic algorithm selection schemes, in: Proc. of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature (PPSN III), Springer-Verlag, London, UK, 1994, pp. 119–129.

[33] D. Thierens, D. E. Goldberg, A. Guimarães Pereira, Domino convergence, drift, and the temporal-salience structure of problems, in: Proc. of International Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA, 1998, pp. 535–540.

[34] R. W. Yeung, S.-Y. R. Li, N. Cai, Z. Zhang, Network coding theory: single sources, Commun. Inf. Theory 2 (4) (2005) 241–329. doi:http://dx.doi.org/10.1561/0100000007I.

[35] X. Zhang, G. Neglia, J. Kurose, D. Towsley, On the benefits of random linear coding for unicast applications in disruption tolerant networks, in: Proc. of the Second Workshop on Network Coding, Theory, and Applications, 2006.

[36] B. Cohen, Incentives build robustness in bittorrent, 1st Workshop on Economics of Peer-to-Peer Systems (2003).

[37] T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, New York, USA, 1996.

[38] W. A. Shewhart, Economic control of Quality of manufactured product, D. Van Nostrand, New York, USA, 1931.