Autonomous Online Evolution of Communication Protocols

Endre S. Varga, Bernat Wiandt, Borbála K. Benkő, Vilmos Simon

Abstract—**In this paper we describe an approach for optimizing multi-hop broadcast protocols in ad-hoc mobile networks with an online, distributed machine intelligence solution. In our proposed framework not only runtime parameters of a predefined protocol are optimized, but the protocol logic itself also emerges dynamically. The model is based on genetic programming and natural selection: protocol candidates compete for being picked (natural selection), then survivors get combined with each other and/or mutated (genetic operators), forming the next generation of protocol instances. To achieve this we created (i) a genetic programming language to describe protocols, and (ii) defined a distributed, communication-wise non-intensive, stigmergic feed-forward evaluation and selection mechanism over protocol instances, and (iii) a budget based fair execution model for competing protocols. We show that the result of the online, autonomous protocol evolution outperforms traditional approaches, by adapting to the local situation, when used for multi-hop broadcast problem in ad-hoc mobile networks. Experiments confirmed 50% improvement with a random movement mobility pattern, and 66% improvement with a group based mobility pattern. The evolution also protected the system from the negative effects of initially present harmful protocols.**

I. INTRODUCTION

THE choice of communication protocol is always of high importance in telecommunication networks, typically a tradeoff between the messaging overhead and the transmitted information content needs to be found. While too chatty protocols waste resources such as bandwidth and processing power, unnecessarily tight-lipped communication strategies hinder the flow of information, and as a consequence, impede the effective operation of the system. The protocol selection problem becomes especially challenging in networks with highly dynamic structure and highly dynamic load characteristics, for example in opportunistic ad-hoc networks where mobile nodes are present. Recent studies indicate that while there is no clear answer for the protocol selection riddle in general, it makes sense to evaluate the goodness of communication protocols for a certain problem case [18], [7], [4], [1]. In this paper we focus our research on a specific subset of protocols, namely the multi-hop broadcast protocols where the aim is to efficiently distribute a set of messages to all of the nodes in the system.

The idea behind this article is to abandon the classic approach, where the communication protocol is a static building block of the system, and, instead, introduce the principle of autonomous protocol selection and protocol evolution:

• Protocol selection means that existing protocol instances compete for being selected for use.

• Protocol evolution means that new protocol instances get created systematically, by combining and modifying existing protocols, in order to enable the emergence of even more successful instances.

Our vision is a system with a high degree of distributedness and autonomy. Nodes of the system pick their protocols autonomously, based only on locally available information. Moreover, we introduce the idea of inverted decision, that enables decision making without straining the network with explicit information collection messages.

This is a drastic departure from traditional protocol engineering: clearly, the vision of communication protocols changing and shaping in an online manner during the system's normal operation is a drastic image, and rather different form the mainstream of the state of the art. However, we believe that the communication, as being a heart of the problem in many cases, is also a place for being autonomous. One advantage is that the evolution mechanism removes the burden of designing communication protocols manually. The use of machine intelligence not only reduces costs but with a suitable evolution and selection model also guarantees the emergence of successful protocols in the end. Another advantage is that the distributed, purely local selection mechanism adds flexibility and fault tolerance: enables nodes to adapt to the very local challenges, and the presence of multitudes of protocols in the system at each moment guarantee that there will always be (or emerge) suitable protocols for unseen situations.

In [14], as a precursor to the current work, we used natural selection to achieve self-adaptation of multi-hop broadcast protocols in ad hoc networks, through automatically selecting the optimal one from a predefined set of protocols, however, no protocol evolution was present in that work. In [17] we published an early version of the on-line genetic programming framework, and demonstrated that the global goodness of a protocol can be reliably approximated with pure local measurements under certain circumstances. In this paper we introduce our matured model and demonstrate its mechanisms by showing the situation dependency and dynamics of the evolution through excessive simulation.

The structure of the paper is as follows. In section II we

Manuscript received 29 February, revised 10 April, accepted for publication 15 April, 2012.

E. S. Varga is a PhD student at the Budapest University of Technology, Department of Telecommunications 1117 Budapest (email: vendre@hit.bme.hu).

B. Wiandt is a PhD student at the Budapest University of Technology, Department of Telecommunications 1117 Budapest (email: bwiandt@hit.bme.hu).

B. K. Benkő was with the Budapest University of Technology, Department of Telecommunications 1117 Budapest (email: bbenko@hit.bme.hu).

V. Simon is an assistant professor at the Budapest University of Technology, Department of Telecommunications 1117 Budapest (email: vendre@hit.bme.hu).

discuss the background of the problem as well as related work. Section III describes the vision of autonomous protocol selection and its main ideas. In section IV we focus on the genetic programming framework created for the multi-hop broadcast problem. Section V presents the results of the experimental evaluation where we analyzed the course of evolution under different environmental conditions. In section VI we give our overall conclusions.

II. Background and Related Work

This section discusses the background and related work in connection to our problem. We define multi-hop broadcast and present the difficulties implementing such protocols, then we investigate the known directions for optimization.

A. Multi-Hop Broadcast

It is a common task in ad hoc networks to distribute messages globally to all, or almost all, participants. This is basically an extension of local broadcast, usually referred to as multi-hop broadcast. By nature, this kind of service consumes a significant amount of resources (channel usage, collisions), therefore optimization is of high importance.

Channel usage is just one of the difficulties that present themselves when one implements global scale broadcast protocols. One dangerous phenomenon is the so called Broadcast Storm [13] that happens when multiple nodes start forwarding a message simultaneously after receiving it from a common source node, leading to excessive collisions. The common presumption used in protocol design, that traffic patterns of neighboring nodes are uncorrelated with each other, is not valid for this case. In the case of multi-hop broadcast, several nodes may decide to transmit at the same time, and collide even after several backoff events. To avoid this outcome, protocols have means to de-correlate the traffic of neighbors, for example by waiting for a random time before forwarding the message.

Multi-hop broadcast algorithms typically exploit the local broadcast channel to reduce channel usage and the number of collisions in the system. This way, as one transmission may be overheard by multiple devices, it is possible to drastically reduce the amount of transmissions. The essence of this optimization is to identify or approximate a Minimal Connected Dominating Set (MCDS) [10], [7] in the network, and broadcast the message once per set.

A set for a graph $G(V, E)$ is a Connected Dominating Set if M is a connected subgraph of $G(V, E)$ and for each vertex either or there exists an edge so that . A Connected Dominating Set M is a minimal CDS (MDCS) if $|M|$ is minimal. An additional constraint in multi-hop broadcast is that if B is the set of vertices containing the nodes that initially possess the payload to be broadcasted, then must hold. If the vertices V of the graph $G(V, E)$ stand for nodes in the network and an edge $e = v, w$ represents that v and w are in radio range, then an MCDS gives the smallest set of nodes needed to accomplish a successful global broadcast.

The identification of an MDCS raises several questions. First of all, it is an NP-complete problem. Another problem is that the MCDS model describes a static scenario, where the connection between nodes do not change. Clearly, if the network is distributed and dynamic (nodes move, disappear or new nodes appear), and changes may occur much faster than they can be discovered, then a centralized model, such as a centralized MCDS solver is not practically possible. Instead of tackling with real MCDSs, broadcast protocols typically use some kind of approximation based on simple heuristics and local knowledge. These heuristics range in sophistication from simple counter based solutions to probabilistic methods and complex graph theoretic approximations [19], [4]. A common point in these approximations is that they concentrate on the closest part of the network rather than dealing with the whole topology. We also followed this approach in our system.

B. Protocol Optimization Directions

Various literature sources investigate possible protocols for multi-hop-broadcast and their performance characteristics, a few examples are [18], [7], [4], [1]. Authors in [9] also give a theoretical upper bound for the worst-case performance of their algorithm.

Results suggest that there is no general winner; instead, the performance of a protocol heavily depends on volatile attributes of the environment. These attributes include mobility patterns, node speed, node density, transmission technology, and traffic models. Selecting the suitable protocol, therefore, requires deep and exact knowledge about the actual environment. However, that is generally hard to acquire, given the complex factors involved, such as human behavior influencing the mobility pattern and the load characteristics. Worse, the environment will change over time, through appearance and disappearance of nodes, technology turnovers, or changes in the usage practice, i.e. human habits; therefore any static off-line design is just a compromise.

The issues above raise the question whether an automated, online, adaptive approach could solve the matter of obtaining the best protocols for a given situation. The use of online, adaptive techniques for protocol optimization (i.e. fine tuning of operational parameters on-the-fly) is a known, but not widely used practice. For protocols, even if machine learning is applied, this step typically happens during the manual design phase, and not as part of the operation of the actual system. An exception is [5], where authors used online machine learning to approximate the behavior of sophisticated broadcast algorithms and found that simple heuristics were able to reproduce the sophisticated decision with 87% accuracy. This result indicates that in practice small but powerful heuristics could provide good approximations instead of sophisticated calculations. Note that Colagrosso's work uses predefined (fixed) protocol bodies, and aims to optimize the runtime parameters of these protocols. Our approach goes one step further: in our work the protocol body itself is also an emergent, ever-changing element.

The idea of on-the-fly protocol selection or protocol switching has been present for many years in other areas, such as in cryptography. In the field of telecommunication protocols, [5] proposed the idea of using machine learning to switch

between a small set of predefined protocols in order to accommodate to the recent topology changes. In [14] we proposed stigmergic communication and natural selection for online, automatic protocol replacement. Natural selection as a tool is not unheard of in this area, in [2] authors applied a form of natural selection for parameter optimization of ad-hoc network protocols, using explicit feedback from neighboring nodes. Our selection approach is different; our approach does not require explicit feedback, resulting in a significantly smaller communication overload, furthermore, the mechanism works with arbitrary broadcast protocol without modification.

C. Genetic Programming

The basis of our work is genetic programming. Genetic algorithms are biologically inspired random search algorithms directed towards a global optimum, based on generations of solution candidates (individuals). In each round, the algorithm evaluates individuals with a fitness function; then, the next generation gets produced by applying genetic operators (crossover and mutation) on the selected individuals of the current generation. Genetic programming (GP) is a form of genetic algorithm, where individuals are programs composed of instructions in a particular programming language. When using GP we generally distinguish on-line and off-line approaches. 'Off-line' means that solutions are generated during a design phase, and the result is then used unmodified in the operational phase of the system; while in the 'on-line' case the evolution itself is part of the system and new solution instances are generated continuously, during the system's normal operation. According to our knowledge, on-line genetic programming has not been applied in the area of broadcast communication protocols before.

The variety of challenges present in multi-hop broadcast protocols for mobile ad hoc networks, such as the unpredictability of the position or speed of the mobile node, changing topology and the diversity of devices, makes it an ideal target for genetic programming. Protocols need to cope with numerous and more or less distinct aspects of the problem; hence, these aspects can be freely blended by the genetic algorithm so that the blend is still likely to make sense. Human-designed protocols typically have distinct "sweet spots": working best under different conditions. By increasing the diversity of the protocols and enabling them to adapt freely to the current environmental conditions through evolution, it is possible to always maintain a protocol or a family of protocols that works well in the current environment. This approach can also be considered as an automated protocol design tool.

While it provides many benefits, the use of GP also has certain design consequences. Fine-tuning the system becomes problematic, as evolution tends to produce individuals that circumvent human imposed rules and design patterns. GP also generates a large amount of individuals, meaning that obtaining insight becomes harder (it is impossible to inspect all the generated individuals manually). However by using data mining techniques, getting an insight into the mapping relations becomes easier, as demonstrated later in this chapter.

The success of any genetic algorithm relies on a carefully designed fitness function. In ad hoc systems the unavailability of global, system-wide data demands a careful approach to fitness calculation, collecting any kind of performance metric may easily result in a prohibitive amount of channel usage.

III. The Model of Autonomous Online Protocol Evolution

This section describes the model of online, automated protocol evolution, comprising of four main building blocks:

1) natural selection with decision inversion to select which protocols survive for the next round
2) a budget based execution model for the current protocol instances
3) a genetic programming language to a describe protocol instance
4) genetic operators in order to combine and mutate protocols.

A. The overall picture

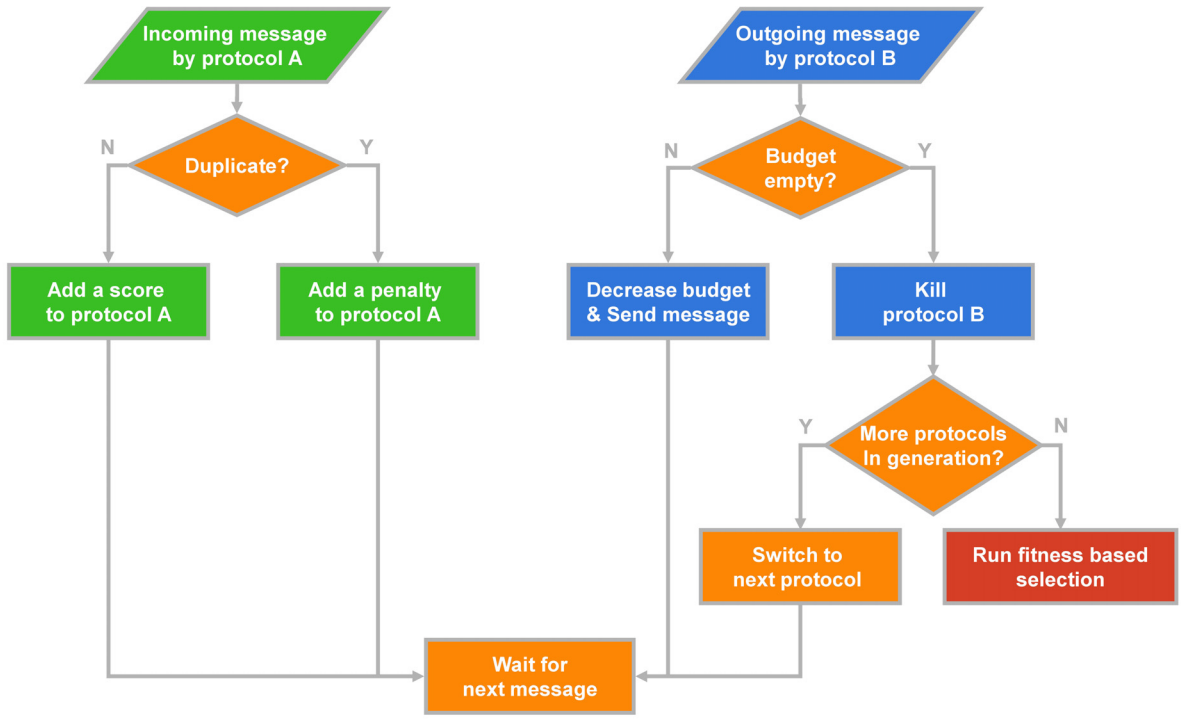
The protocol evolution is a fully distributed, asynchronous mechanism; each node selects and generates protocols on its own agenda. The core process is a loop, as visualized in figure 1b.

The evaluated protocols undergo a selection step, deciding which protocols survive and which end their life time. Once the surviving generation is selected, genetic operators, i.e. crossover and mutation, are used in order to introduce further new protocols by combining and/or modifying survivor instances. The program code of the new protocols gets compiled, resulting in a new generation of executable protocol instances. Then each of the protocols is executed, sequencially, using a budget-based execution scheme, giving equal opportunity to each individual of the generation to live. Finally, the loop starts over.

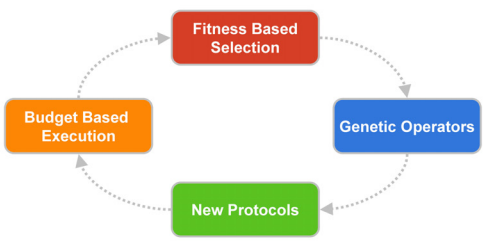
Protocol evolution runs on each node of the network, in parallel, without any explicit synchronization with other nodes. No global clock is assumed. On the other hand, an implicit synchronization is indeed present in the system. Neighbors, as part of the inverted decision making mechanism, discover each other's protocols; thus, a successful protocol instance may spread over the network from hub to hub. This fully distributed scheme also brings fault tolerance to the system.

A distributed protocol evolution model needs to answer the following questions.

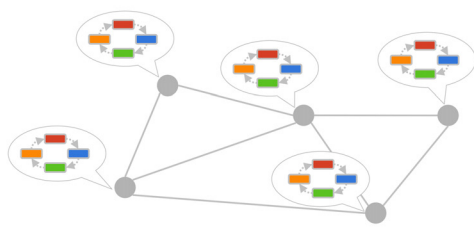
- How to measure the goodness of a protocol. The decision must be made locally, without the aid of global or wider area help, so only local or nearly-local metrics are acceptable. We defined a set of locally available evaluation factors to estimate the overall goodness of a protocol. In [17] we showed that these local metrics are linearly dependent, strongly associated with the global metrics, and are very good approximators for it.
- How to avoid the communication overhead of the measurement. We propose an inverted decision making model



(a) Inverted selection mechanism executed at the receiver node, assuming that the sender at the moment executes protocol A, and the receiver executes protocol B.



(b) Flow of protocol evolution at a node.



(c) Network-wide view of protocol evolution.

Figure 1: The Model of Autonomous Online Protocol Evolution

which avoids the sending of feedback messages, saving bandwidth and also helping the system stay asynchronous.

- How to describe protocols in a way that is robust enough even for crossover and mutation operators. We do not only need to avoid syntactic errors, but also need to facilitate the emergence of presumably sensible new protocols (semantic correctness). We propose a genetic programming language.
- How to evaluate a set of protocols at runtime, during the system's normal operation. Clearly, the evaluation should happen in a fully online manner, i.e. by sending real messages rather then through some kind of internal simulation. We propose a budget based daisy chain model to give equal chance to each protocol to show its strength.

B. Natural Selection with Decision Inversion

Natural selection is a fundamental principle in evolutionary systems. In our case, when the system is equipped with several protocol candidates, in order to select the most suitable ones, we need to measure the performance of each candidate and push the system towards the use of them.

To realize a lightweight but efficient natural selection mechanism is highly non-trivial in our case. Protocols, in order to be successful, need to address conflicting requirements, i.e. maximal coverage versus minimal duplication count. These two requirements not only make the choice of evaluation metrics hard, but also pose measurability problems, such as: (1) Only the sender node is able to reliably measure the real cost of a successful message transmission. Lost messages, by definition, could not be seen by other nodes, so the amount of work (total sent messages) is only known by the sender (2) Only the receiver node is able to reliably measure the number of redundantly received messages (we call these *duplicates*). (3) Each receiver node can measure the number of local duplications (i.e. the duplications they personally received), but they can not measure the number of total duplications in the system. (4) In order to collect the measurement results

at a designated place, message passing is needed over the same channel as normal (payload) messages use. Measurement messages, just like payload messages, may get lost.

To get a clearer picture of the measurement problem, consider the case when a node sends out the same message ten times. Although naively we could consider this as ten duplicates, in reality this is not necessarily the case. If during the timeframe of the broadcasting of the ten messages ten nodes pass by, each hearing the message only once, then there are no duplicates at the receivers. Now consider the case, when the sender sends out exactly one message, with ten other nodes in transmission range. If all the neighbor nodes already received this message in the past then this transmission alone generates ten duplicates at the receiver nodes. From these examples it should be clear, that the amount of copies sent and the amount of duplicates are not the same, and the former is known by the sender, and the latter is known collectively by the neighbor nodes. To centralize all this data at a designated place, further communication is needed. Unfortunately this generates a large overhead, especially if the reliability of this procedure is considered.

The above factors imply [14] that implementing a centralized (even locally centralized) protocol selector criterion is impractical, because the reliable collection of performance data is both technically challenging and wasteful in terms of channel usage. Instead, we propose a feed-forward selection method using stigmergy and natural selection.

The solution is based upon the idea of decision inversion. The naive implementation of natural selection would be that every sender collects its performance metrics from the surrounding receivers and creates the next protocol generation according to this information. However, as explained above, this can not be efficiently implemented in general. Instead of trying to select locally at the sender, we delegated the task of decision making to the receivers, because they are in the optimal position to observe the performance of a protocol. Nodes score and evaluate all protocols that successfully sent messages to them. Then, the next generation of the node's protocols will be comprised of the highest scoring sender protocols. This essentially means, that no performance metrics are disseminated back to the sender, instead, the evaluation is delegated to the set of receivers, collectively.

To make this possible, receivers must know the protocol that generated the given payload message. This is achieved by senders attaching a representation of the protocol itself to every payload. Such compound packets act as a virtual seeds where the "nutritional" part of the seed is the payload and the genetic material is the code of the sender protocol. Nodes (as receivers) collect seeds from surrounding nodes and assign scores to the protocol instances they carry. Every payload that is useful to the receiver node means a score for the sender protocol. Every unnecessary message (duplicate) means a penalty to the sender protocol. Seeds with useful nutritional parts (high scores) will survive on the receiver node. The main advantage of the inverted selection is that performance results do not need to travel back over the network to the sender: instead, the receiver will utilizes them during the creation of its own next protocol generation. Of course in the next round, the sender may meet the offsprings of its own good protocol. With this approach, both the measurement overhead and the need for synchronization is minimal.

It is important to see, that this procedure optimizes protocol instances locally, using local information. This is similar to the way how static, human-made protocols work, but instead of only optimizing protocol parameters, our framework is capable to "invent" new heuristics as well.

C. Budget Based Protocol Execution Model

As the selection of new algorithms happens at the receiver it is impossible to implement explicit cost calculation without expensive control overhead. To overcome this, we adopted a stigmergic solution: by assigning a limited transmission budget (called *quota*) to each protocol instance at the sender, protocols are forced to make good use of the channel resources. Any lost or duplicate message is a lost opportunity for reproduction; therefore transmission has an implicit cost function, even if it is not expressed directly. Similarly, we added a timer that upon firing, removes the current protocol, and replaces it with the next instance, forcing protocols to use the available time efficiently, we call this the *time limit* of a protocol

D. A Genetic Programming Language for Protocols

Natural selection implemented by decision inversion, along with the budget based execution mode, answers the question how protocols should be executed, evaluated and selected for survival in a distributed fashion. In this section we introduce our choice protocol representation and the tools used in protocol composition.

In order to make Genetic Programming and evolution possible, the representation of protocols needs careful consideration. As the protocols in the system are no longer engineered by humans, a lightweight, flexible and robust formal description is needed which suits genetic operators.

In a GP environment protocols must be represented by their program code. However, to use a general purpose programming language as the protocol representation would be problematic because the application of genetic operators could result in frequent syntactic errors and uninterpretable code. To avoid these issues several GP specific languages were designed. We selected the Push language [15] as a starting point for the design of our own language, called GPDISS.

Push is a natural choice, as it is a widely known and used language in GP related research, and Push code is relatively easy to interpret by humans. Artificial chemistries were also considered [8], particularly the Fraglets language by [16], which was used to conduct experiments in protocol evolution in [11]. However, artificial chemistry based languages are notoriously hard to analyze manually, therefore we decided to exclude these languages from our experiments.

Push is a stack based language, meaning, that its instructions do not have explicit arguments, instead, they are taken from the corresponding stack. For example, an 'if' instruction takes its argument from the top of the Boolean stack. This enables a mutation and crossover friendly, flexible syntax.

When defining our custom programming language, GPDISS, we borrowed ideas from Push, but augmented it with new instructions and introduced new concepts in order to provide a better fit for our particular problem. Our modifications include:

- Extended instruction set in order to match the multi-hop broadcast problem.
- A new data type (relation type) to describe complex data relationships. This is particularly useful for modeling graph-like structures
- The concept of typed event handlers that simplify the crossover of message handling routines

1) Extended instruction set: The most important instructions of GPDISS are shown in table I. The set of operations include control instructions, basic arithmetic and logic operators, control flow, messaging, timers, and the handling of complex data using relations. Note that this particular instruction set was designed for the multi-hop broadcast problem. For a different problem a slightly different instruction set may be required but without affecting the basic principles.

Table I: Most important instructions in GPDISS.

Stack	Instruction	Description
*	dup, drop, swap,rotate, hold, release	Common instructions available on all stacks implementingcommon stack manipulation operators.
number	add, div, mult, random	Simple floating point arithmetic and random number generation.
bool	and, or, not, if, while	Boolean logic (usually for control flow).
list	additem, nth, remove_first, delete_duplicates	Typed list handling. Common operations are available.
relation	addpair, union, join, remove_first, invert, intersect	Typed relations are like two column tables. They can be filtered, joined, intersected, etc.
messages	send, sender	Common instructions for handling all types of messages.
timers	id, start_timer	Timers can be used to schedule different tasks at different points in time.
-	else, endif, do, return	Control flow constructs.

2) Relation type: A new data type, called relation type, and a set of accompanying instructions were devised in order to enable efficient calculations on graph structures, tabular data, or trees, as these are common in broadcast protocols. On the relation stack ordered, typed pairs of objects can be stored. Relations can be imagined as two column tables, the first column being the key, and the second column being the value. The key and the value columns are typed, which means they receive their content from the appropriate typed stack. Relations give a new dimension to stored data by describing relationships between the objects on the stacks. Relations are immutable: every operation creates a new instance on the relation stack. With the help of the operations defined on relations sophisticated data manipulations are possible, such as filtering by key or value, intersecting, joining, subtracting or creating the union of two relations. The code snippet in Figure 4 creates a new relation containing our direct neighbors, supposing that we already have a neighbor map (node-node pairs) of the network in our vicinity on top of our relation stack.

Table II: Event handlers in GPDISS

Message type	Event handler
data	The common wrapper for all data messages in the system. Data messages contain payload originating from a node with the goal of reaching all nodes in the system.
neighbor	Neighbor messages contain neighbor information of the sender nodes. This can be used to locally map the connectivity graph of the network
timer	Internal event handler. A protocol receives a timer event when one of its timers fire. One can use it to implement features such as Random Assessment Delay (a transmission delay used in broadcast protocols)
init	Internal event handler. Protocols receive this event once upon initialization, before the first message arrives to them.

3) Event handlers: Instead of defining programs monolithically, we defined multiple hooks, called event handlers, that protocols may use to implement their logic. Therefore, the code of a protocol is a list of assembly-like instructions grouped into event handlers.

Each message type has its own event handler (illustrated in table II), which gets activated when a message of that type arrives. The activation is controlled by an underlying meta-protocol, shared by all nodes. The purpose of this meta-protocol is to ensure that the result of the evolution, i.e. the messages, remain interpretable for all possible receiver nodes. The meta-protocol defines how to decode and interpret the messages sent by other nodes. The possible most basic meta-protocol simply defines the format of messages, and ensures that the incoming message is forwarded to the corresponding event handler. In GPDISS, the meta-protocol defines the format of payload messages, and a few control messages, although it does not define the ordering, timing of them. This choice reflects our conservative approach to GP, restricting the protocols to use control primitives we already know and consider to be useful. It is currently impossible for protocols to "grow" their own custom messages. It is important to note, however, that the meta-protocol does not restrict the order of messages, nor does it enforce messages to get processed (event handlers may skip any message); therefore there is still a large degree of freedom for protocols to explore. Furthermore, with a different meta-protocol, we could easily enable the emergence of new message types, as well.

E. Genetic Operators

Event handlers are the basic units used by genetic operators. The crossover operator mixes the bodies of two event handlers; while the mutation operator changes the instruction sequence of a single handler. Implementing a protocol via event handlers is a natural and practical choice because it enables the genetic

operators to modify corresponding parts of a protocol by mixing code snippets of similar task or extending protocols by adding new event handlers (or removing ones).

Although syntax is maintained, the semantic correctness of a program can not be guaranteed after the application of genetic operators. If an instruction is impossible to execute (such as pop on an empty stack), it defaults to a no-op instruction, which does nothing, and the execution of the code continues undisturbed. This results in a quasi-linear [3] genetic programming language.

IV. GENETIC PROGRAMMING FRAMEWORK FOR MULTI-HOP BROADCAST

This section demonstrates how the introduced described protocol evolution model can be used for implementing multi-hop broadcast. Note that multi-hop broadcast is especially suitable for genetic programming as it has several, more or less independent aspects, and several good and combinable solution strategies.

First we describe the initial population of protocols used as the starting point of the evolution; then, we elaborate on the details of the execution and selection mechanism and the used genetic operators. Finally, we discuss the metrics we used to investigate the course of evolution.

A. Initial Protocol Population

The initial population was selected from a small set of well-known protocols that are simple enough to be the starting point of evolution.

- Adaptive Periodic Flood (APF) is an optimization of blind flood. An APF node periodically transmits all the messages it possesses to all neighbors it encounters, after a random waiting period. However, when it detects that there is another node sending the same message, it increases the period of broadcasting to reduce the total channel usage.
- Gossiping (Gos). A gossiping node forwards the received message to its neighbors with a given probability. Gossiping is easy to analyze mathematically, as neighboring nodes have minimal effect each other's operation.
- Density sensitive adaptive gossiping (AGos). In adaptive gossiping the probability of propagating the message depends on some condition. We used a density sensitive model, where the probability decreases as the number of neighbors gets higher.
- Aggressive flood (AgrF). In case of aggressive blind flood the node propagates each received message to its neighbors n times, with a certain waiting time between the repetitions. We used two different repetition amounts to analyze how the evolution can protect the system from aggressive attackers, trying to propagate their codes by sending messages in an aggressive manner.

The criteria for the choice of protocols in the initial generation were to include simple but versatile algorithms to see if evolution is able to improve them and to include a proven 'dangerous' algorithm to see if the evolution process can eliminate it by the use of strictly local metrics. Highly effective, but overly complex protocols were excluded from our experiments, as they are usually not good candidates for Genetic Programming.

Note that AgrF is a particularly dangerous protocol from the viewpoint of this stigmergic feed-forward evolution model, as it attempts to spread its seeds at the highest possible rate.

B. Selection

Every protocol generation is created from the previous locally available protocol generation and those non-local protocols that were discovered in the previous round. We used SUS (Stochastic Universal Sampling) with a score function that gives priority to better performing individuals [12]. SUS provides zero bias and minimum spread, meaning that the actual and expected probabilities of selecting an individual are equal and the range in the possible number of trials that an individual can achieve is minimal. SUS is a variant of the roulette wheel selection. The steps are as follows. (1) Order the individuals by their fitness score in non-increasing order. (2) Allocate slices on the wheel proportional to the fitness of the individuals. (3) Calculate the step-width. For example if we want to select n elements the step-with is (sum of fitness values)/n. (4) Choose a starting point on the wheel between 0 and step-width. The corresponding protocol is selected for the next generation. (5) Make n-1 step-width wide steps, and always select the protocol assigned to the given slice. Selecting an individual means making a copy of it and adding that copy to the new generation.

The new generation is then shuffled, individuals are coupled into pairs, and with a certain probability the crossover and mutation operators are applied. Crossover is applied on the pairs, and mutation on the instances.

C. Crossover and Mutation

A modified one-point crossover is used for combining two event handlers. Given that the protocol pair (A, B) is selected for crossover, the algorithm is the following:

- Choose an event handler from A randomly. If B has no such event handler, then return.
- Select a cutting point randomly in A's handler, and another point in B's handler. Cut the handlers along the cutting points, resulting in four fragments: A-head, A-tail, B-head, and B-tail.
- With 0.5 probability exchange the head and the tail fragment of the original handlers.
- Glue fragments together forming two new handlers, an (A-head, B-tail) and a (B-head, A-tail). To protect handlers from growing indefinitely, we limited the maximal size of event handlers; bodies above the limit were chunked.

We use constant parameter mutation, meaning that instead of modifying instructions in the event handler body, the mutation affects the constants, i.e. the runtime parameters of the algorithm. For example such a runtime parameter is the message propagation probability in the Gos. When a parameter with current value x is mutated, the new value is chosen from the $(0, 2x]$ range with a Gaussian distribution, favoring fine-tuning but also allowing larger changes.

D. Execution and Parallelism

We used a simulation with completely independent node entities, each running its own Virtual Machine executing GPDISS code and implementing our meta-protocol. The clocks of the nodes were not synchronized, and the VMs had slightly differing opcode execution times.

A new payload unit was generated periodically and nodes had the task of broadcasting it over the network.

E. Measurement Metrics

During simulations we collected various metrics about the protocol instances that appeared in the system. Recording of measurements happens whenever a protocol finishes its execution, either because it depleted its messaging quota or because the time limit expired.

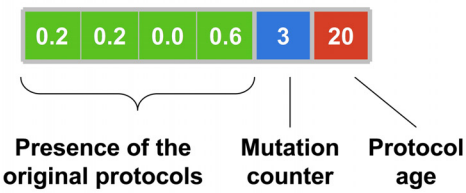


Figure 2: Protocol fingerprint for the case of 4 initial protocols. This protocol instance is the result of 20 evolution rounds, with 3 mutations. The original protocols are present in it in the amounts of 0.2, 0.2, 0, and 0.6.

The first metric, the protocol fingerprint is a vector, describing the genetic constitution, mutation count and the age of the protocol instance, as shown in figure 2. The first segment, the genetic constitution part, approximates the presence of the code parts of the initial protocols in the current instance. For example, a pure $0th$ initial protocolfor example APF, has a $[1, 0, 0, 0]$ genetic constitution part, while the mixture of the $0th$ and 1st initial protocols is represented as $[0.5, 0.5, 0, 0]$. The length of the constitution part equals to the number of initial protocols, and the sum of the constitution values is always 1. The second section of the fingerprint is the mutation counter. The third section is an age indicator, containing the local generation number at the node that created the instance.

In addition, we measured a set of local and global metrics for each protocol instance. In [17] we showed that the aggregate of certain local metrics could be very efficient approximators for the global ones, so we will not differentiate between local and global scores in the current analysis. The list of measured metrics is the following:

- GEN: Generation of a protocol, i.e. the value of the local evolution counter on the generating node. Note that as the system is asynchronous, new generations get produced by different rates at the different nodes
- SCORE: Score of the protocol instance, particularly, the number of useful messages minus the number of duplicates it generated.
- TIME_SLICE: Global time divided into epochs. The total simulation time was divided into 20 equal time segments, assigning the numbers $0..19$ to each.
- PROTOCOL_ID: Protocols were assigned identifiers in the order of their deaths. There were approximately $100,000$ observed protocols per experiment

It was a deliberate decision not to measure which protocol instances survive and which does not. This is because the actual performance of the system in the current round depends on the executed protocol instances and not on the surviving ones.

V. Experimental Evaluation

We executed several simulations to evaluate our protocol evolution model in the context of the multi-hop broadcast problem. First we describe the environment used in our experiments including the different scenarios used. Then, the results of the measurements using different mobility models are discussed. Finally, we illustrate the asynchrony of the system.

A. Simulation Environment

For the experiment we used a custom created discrete-event simulation engine, written in the Scala language [6], and our implementation of the GPDISS language and VM in Java. The compiler for the GPDISS was created with the ANTLR compiler generator.

Experiments were conducted on a laptop PC with $2x2$ GHz processor and 6 GByte memory, out of which 1 GByte was used for the simulation.

The general settings were the following:

- Mutation probability: 0.1
- Crossover probability: 0.2
- Node count: 500
- Simulation time: $7000s$
- New information message to be broadcasted over the network is generated every $20s$
- Maximum age of broadcasted payload: $20s$
- Maximum time budget for protocol instances: $7s$
- Messaging quota for a protocol: $5000byte$
- Average size of a payload message: $250byte$
- Average size of an instruction: $4byte$
- Channel bandwidth: $1Mb/s$
- Initial population: 5% AgrF, and equal proportions of the other three protocols.

B. Scenarios

The attributes common for all scenarios:

- Nodes move within a geographic area (800 x 1200 m) according to a mobility pattern. Nodes within $50m$ see each other, i.e. are neighbors. The movement of the nodes cause the connectivity graph to change over time
- The nodes of the system are independent entities, without any global knowledge or synchronization
- Initially, each node starts with a single, randomly selected protocol. This single protocol is the local initial population.

- A new payload message is injected in the system periodically. The overall goal is to broadcast the payload with the possible highest total coverage and lowest duplicate count before the payload expires.
- Nodes execute the decision inversion algorithm locally. By sending messages, protocols have the chance to spread over the network.

C. Mobility patterns

We used two highly different mobility patters for the evaluation: a simple random movement pattern (M1) and a scale-free group pattern (M2). The initial distribution of the nodes over the area was random in bothy cases.

1) M1: Random direction mobility pattern: In M1, nodes randomly choose a direction and distance and move with $1m/s$ speed until they reach the desired distance. After reaching the destination, the process starts over.

2) M2: Competing groups pattern: In M2, nodes have colors assigned. Each node tends to join nearby matching-color groups with a given probability. Groups, when a new member joins, relocate towards the new mass center. Nodes with different colors repel each other, and nodes with matching colors attract each other (until reaching a minimum distance), using a spring model. When the group becomes stable, i.e. no new member joins, the whole group tries to get nearer to a point of interest (top left corner of the area). Groups compete for this area.

We used 10 colors, $5m$ as a minimum distance between matching-color nodes, $30m$ as maximum distance from the group center, $30m$ as desired minimum distance between non-matching nodes, 0.5 probability for solitary nodes to join a nearby group, and 0.02 probability for a grouped node to leave its current group.

The dynamics of the model is shown in figures 3a and 3b, visualizing the two phases of the pattern: (i) group expansion and (ii) competition between groups. Note that even though the group expansion phase results in large movements and a high amount of topology changes, the second phase, the competition of groups for the point of interest even exaggerates that by causing massive-size local changes (e.g. the collision of two dense groups). Groups may temporarily even tear up due to a collision, controlled by to the spring model. Also note that some nodes do not join groups at all.

D. Aggressor transmission rates

We used two different AgrF repetition values in different simulation scenarios: $n = 1$ and $n = 3$. The 3-fold repetition makes the protocol more robust against transmission failures at the cost of significantly reduced performance.

E. Experiments using the M1 mobility pattern

The first set of experiments analyses the random movement based M1 mobility pattern. The goal to examine was twofold: (i) whether the evolution works, i.e. produces better scores than the situation without evolution, and (ii) to analyze how the genetic constitution of protocols changes with time. An interesting question here is whether AgrF, the aggressive protocol manages to survive or the evolution purges it. We analyzed the scores and the constitution of protocols with time.

1) Scores: Figure 4a shows the score chart of the $n = 1$, M1 mobility pattern setting without evolution. This setting serves a reference for comparison. The variance of the scores is very high, so a moving average line (light blue) was added to emphasize the trend. The average does not change significantly over time, it stays around the initial -20 value meaning that for example for 25 useful messages typically also 45 duplicates got produced.

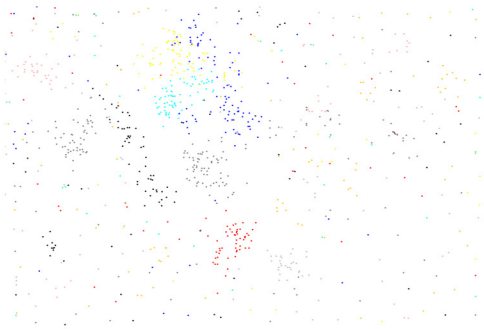
Figure 4b visualizes the same setting with evolution enabled. The variance of the scores is still high, but lower than in the no-evolution case. Also, a significant improvement in the scores can be observed; both the trend line and the minimal score rise, while the maximal achieved score stays constant. At the end of the simulation, the moving average is around -10, compared to the initial -20, which is a 50% improvement. Figure 4c displays the two previous charts in one figure. The gap between the two trends lines confirms that the evolution works.

The results produced by the more aggressive, $n = 3$ setting are summarized in figure 4d. The positive effect of the evolution is also confirmed here, with an even higher difference; because the no-evolution case produces a slightly decreasing trendline due to the more aggressive message propagation of the 5% AgrF instances. The trendline of the with-evolution case suggests that the mechanism starts producing good results after an initial struggling phase. This is because it takes time to get known with all protocols (initially, each node knows only one), and also to find a good blend. However, once a good combination is there, it is likely to produce further good offsprings.

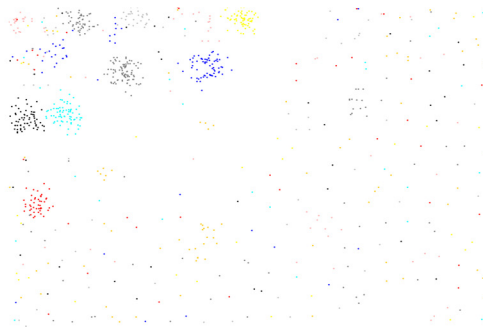
2) Genetic constitution: The next set of figures visualizes how the constitution of protocols changes with time. We quantized the constitution part of the fingerprint into 5 ranges: $0-20\%$, $20-40\%$, $40-60\%$, $60-80\%$ and $80-100\%$. For example, a $[0.1, 0.35, 0.55, 0]$ fingerprint constitution is quantized into a [“$0-20$”,”$20-40$”,”$40-60$”,”$0-20$”] signature. Then, these signatures were summarized by algorithm type and time slice.

Figure 5a displays the participation chart of the APF protocol over time. Columns represent the summary of a time slice. For example, in the time slice 10, APF was present in $0-20\%$ amount in 53% of the protocol population, in $20-40\%$ amount in 26% of the protocols, in $40-60\%$ in the 15% of the population, and in $60-80\%$ and $80-100\%$ amount in $3-3\%$ of the population, respectively. The curve suggests that APF is a surviving constituent; at the end of the simulation it was present in more than half of the protocols in at least 20%. On the other hand, APF is not a dominant part in the successful offsprings, only 3% of the final population contains APF in more than 80%, and only $3+5=8\%$ contains it in more than 60%.

The same chart for Gos is shown in figure 5b. Gos is a much more dominant survivor; in the first 10 time slices it manages to remain a $80\%+$ constituent in $35-45\%$ of the

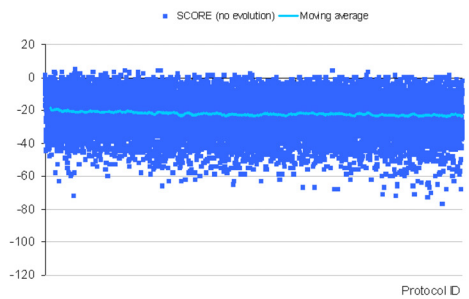


(a) M2 movement model: Group expansion phase.

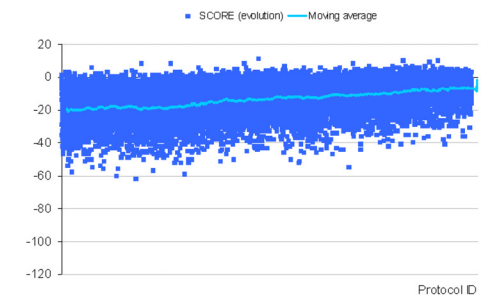


(b) M2 movement model: Groups competing to approach the point of interest in the top right corner.

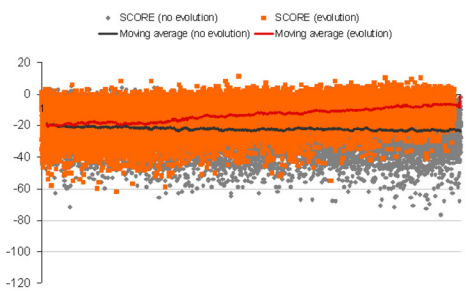
Figure 3: M2 mobility model



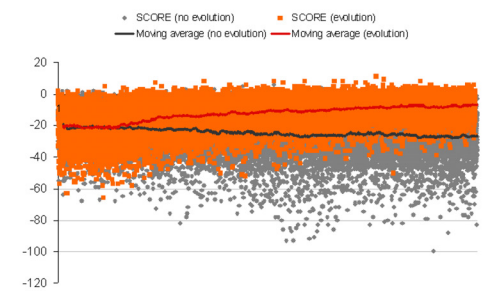
(a) Scores with M1, no evolution. 500 nodes, AgrF $n = 1$, M1



(b) Scores with M1, evolution on. 500 nodes, AgrF $n = 1$, M1



(c) Scores and moving averages with and without evolution. 500 nodes, AgrF $n = 1$, M1



(d) Scores and moving averages with and without evolution. 500 nodes, AgrF $n = 3$, M1

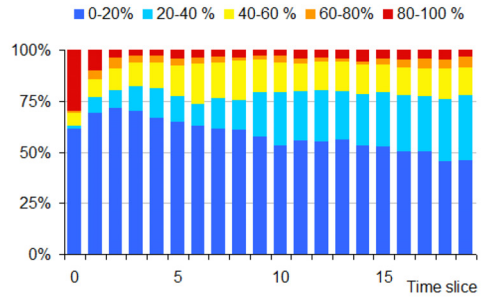
Figure 4: Scores using the M1 mobility pattern

population. At the end of the simulation, Gos still has a visible dominant presence (nearly 25% of the protocols is built from it in 80%+), but also starts blending with other protocols. Note that the presence of the protocol does not mean that it is present in an unmodified form; mutation i.e. fine tuning of constants is likely to increase the success of a protocol. This is exactly what happens to Gos, evolution optimizes the propagation probabilities to the actual neighbors.

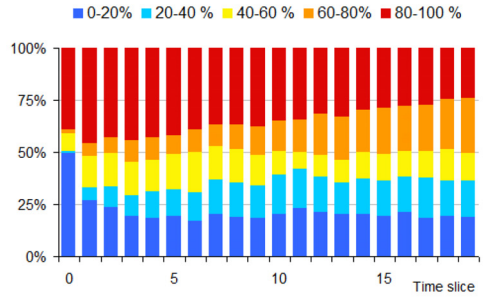
AGos offsprings, as shown in figure 5c, are less likely to be present in offsprings with time than the two previous protocols. There are a few successful crossover-generated offsprings, built partially from AGos, they are successful at certain locations but are not widely spread over the network.

The participation chart of AgrF in figure 5d, shows that in case of the M1 movement pattern this protocol is completely purged from the network in 10 time slices.

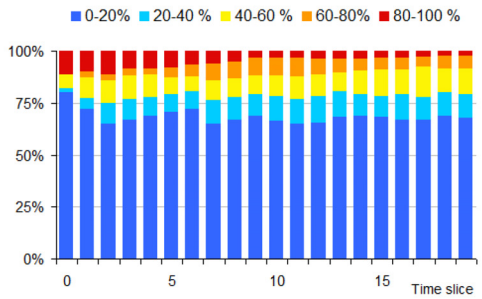
Figures 6a to 6d display the same charts for the $n = 3$ repetition case. Unexpectedly, AgrF here is not purged from the system, although suppressed quite fast. AgrF serves as a $20 - 40\%$ constituent in 3% of the protocols, constantly, in a non-increasing manner. Another difference is that the other three protocols manage to blend better with this setting, resulting in rarer 80%+ presence, and more $20 - 80\%$ presence. The higher ratio of blends means that the offsprings generated by crossover are successful.



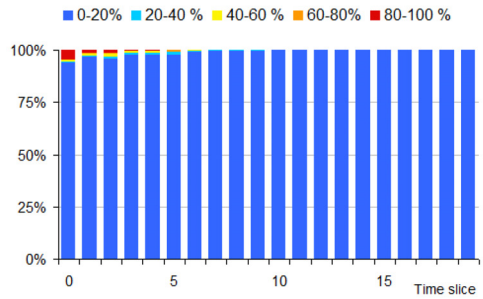
(a) Participation of APF offsprings in protocols over time. 500 nodes, AgrF $n = 1$, M1



(b) Participation of Gos offsprings in protocols over time. 500 nodes, AgrF $n = 1$, M1

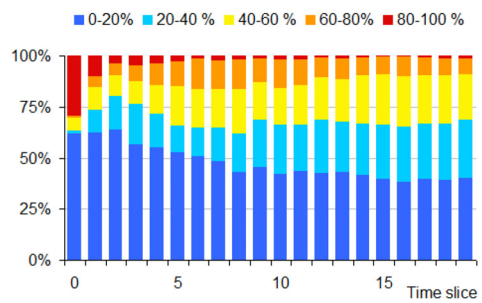


(c) Participation of AGos offsprings in protocols over time. 500 nodes, AgrF $n = 1$, M1

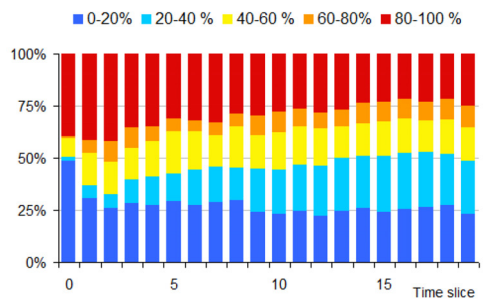


(d) Participation of AgrF offsprings in protocols over time. 500 nodes, AgrF $n = 1$, M1

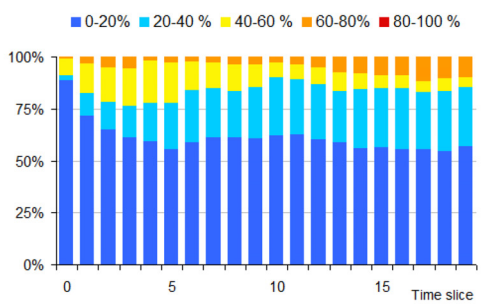
Figure 5: Genetic constitution of protocols using the M1 mobility pattern, $n = 1$



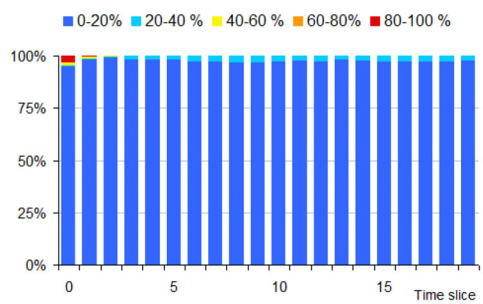
(a) Participation of APF offsprings in protocols over time. 500 nodes, AgrF $n = 3$, M1



(b) Participation of Gos offsprings in protocols over time. 500 nodes, AgrF n=3, M1



(c) Participation of AGos offsprings in protocols over time. 500 nodes, AgrF $n = 3$, M1



(d) Participation of AgrF offsprings in protocols over time. 500 nodes, AgrF $n = 3$, M1

Figure 6: Genetic constitution of protocols using the M1 mobility pattern, $n = 3$

F. Experiments using the M2 mobility pattern

The second set of experiments examines how the use of an intrinsically different mobility pattern changes the results of the evolution. Note that in this case the mobility is group based, meaning that once a node joins a group, a dense neighborhood is likely to be present for it for the remainder of the simulation. On the other hand, the competition between groups, hence collisions with other groups may highly influence the neighboring topology, also changing what kind of propagation strategies work well. Scores and the constitution charts were used for the evaluation, just like in the previous case.

1) Scores: Figure 7a shows the score curve of the $n = 1$ setting with and without evolution with the M2 mobility pattern. The main direction of the trend is the same as in the M1 case, however, the dynamics of the improvement is slightly different. The variance of the scores remains high with M2 throughout the simulation, beacsue M2 causes heavy but localized changes in the topology while groups compete for the area around the POI. The evolution is more successful with M2 than it was with M1, the moving average of the score ends at -6.5, meaning a 66% improvement compared to the non-evolution case.

The score chart of the $n = 3$, M2 case is shown in figure 7b. The improvement in means of average is also around 66% here. Again, in the non-evolving case, the aggressive flood produces a somewhat decreasing score curve and very high variance. Note that the worst score of the evolving population at the end of the simulation is basically the same as the average score of the non-evolving population, showing that the difference is really significant. Another interesting phenomenon observable in the figure is that the initial effect of the evolution is slightly negative, the score curve decreases in the beginning, before turning upwards.

Detailed analysis of the score pointed out that, as figure 7c demonstrates, the number of useful messages sent by the protocols remained the same throughout the experiment, while the number of duplicate messages declined with time. The same trend was visible in all experiments.

2) Genetic constitution: Constitution charts for the M2 case show different blending proportions than M1 did; the difference is especially articulated with the $n = 3$ setting.

With $n = 1$ (figure 8) an unexpected result is that AgrF does not get purged, instead, at the end of the simulation in 4% of the protocols it is still a 100% constituent. This is because AgrF can be a beneficial protocol in certain topologies when only a small portion of nodes use it.

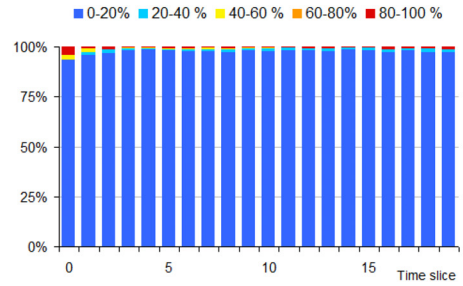


Figure 8: Participation of AgrF offsprings in protocols over time. 500 nodes, AgrF $n = 1$, M2

The charts of the $n = 3$ case are shown in figures 9a to 9d. Protocols here, with M2, blend more than they did with the M1 mobility pattern; $80\%+$ constituents are very rare, altogether 13%, at the end of the experiment, compared to the 32% of the same setting with M1.

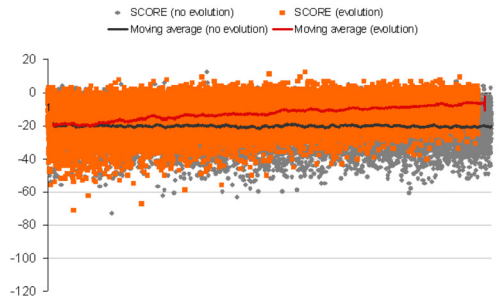
An interesting wave effect can be observed in the charts of Gos (figure 9b) and AGos (figure 9c), showing how the shift from the group expansion phase to the group competition phase influences the usefulness of these protocol as constituents. The shift occurs around slice $8 - 9$. With AgrF (figure 9d), the effect of the shift is also visible; AgrF becomes useful as a $20 - 40\%$ constituent when large and stable groups emerge (around slice 7), and manages to maintain and even improve this position until the end of the simulation. In the end, AgrF is present as $20 - 40\%$ constituent in 5% of the protocols.

G. Comparison with a homogeneous case

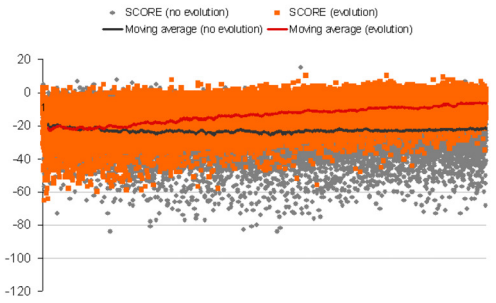
For comparison, we examined how the system (without evolution) would perform if only one non-aggressive protocol family was present, instead of the three tackled with in the experiments before. We kept the 5% AgrF presence for this setting too, as it would not make sense to compare a 100% homogeneous case with the previously discussed settings, where an aggressive, harmful protocol was present. Figure 10a visualizes the measured score values over time. The trend shows no improvement. This confirms that the comparisons used in the previous experiments, i.e. the use of a four-protocol setting, were valid. The use of several protocols did not introduce general bias or disturbance that is not present in a homogeneous case.

H. Execution metrics

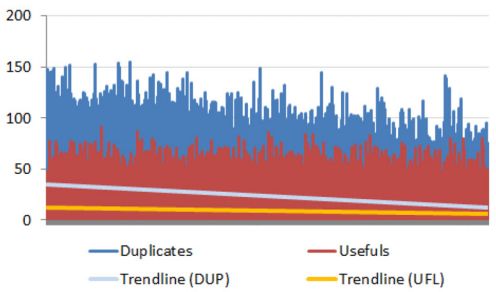
Finally, we demonstrate the asynchrony of the system. Figure 10b shows the age of each protocol (the value of the local evolution counter at the generator node) in the order of measurement, i.e. in the order of the protocols' death. The line-like shape suggests that the evolution speed, although not being synchronized explicitly, is more or less the same in most nodes. Deviations from the average line suggest a



(a) Scores and moving averages with and without evolution. 500 nodes, AgrF $n = 1$, M2

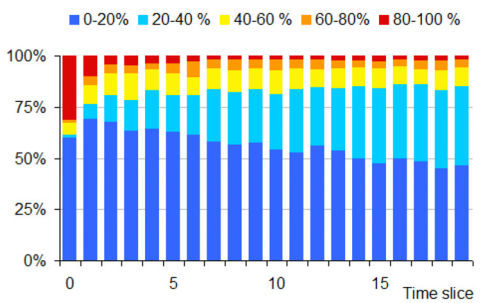


(b) Scores and moving averages with and without evolution. 500 nodes, AgrF $n = 3$, M2

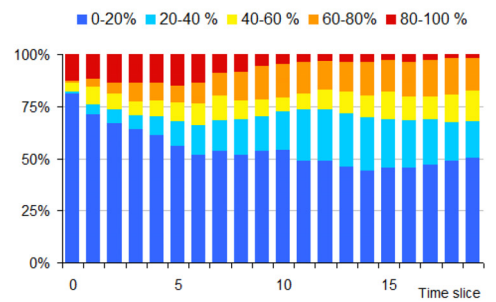


(c) Number of duplicates and useful messages over time. With evolution, 500 nodes, AgrF $n = 3$, M2

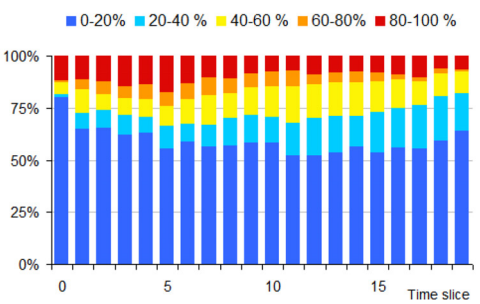
Figure 7: Scores using the M2 mobility pattern



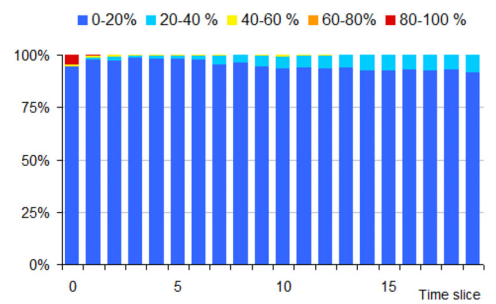
(a) Participation of APF offsprings in protocols over time. 500 nodes, AgrF $n = 3$, M2



(b) Participation of Gos offsprings in protocols over time. 500 nodes, AgrF $n = 3$, M2

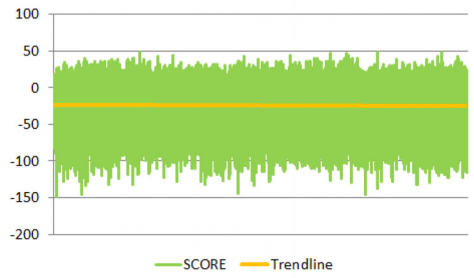


(c) Participation of AGos offsprings in protocols over time. 500 nodes, AgrF n=3, M2

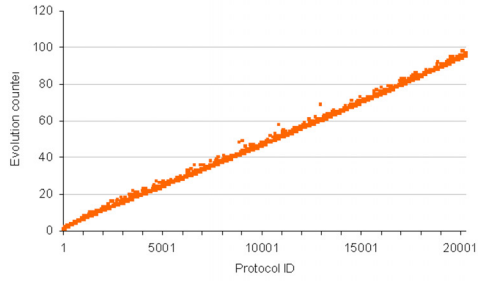


(d) Participation of AgrF offsprings in protocols over time. 500 nodes, AgrF $n = 3$, M2

Figure 9: Genetic constitution using the M2 mobility pattern, $n = 3$



(a) Score over time, homogeneous case (95% APF + 5% AgrF). 500 nodes, AgrF $n = 3$, M1



(b) Protocol age (local evolution counter) over time. AgrF $n = 3$, M2

Figure 10

local difference in the speed. Deviations typically occur in the positive direction, i.e. the rounds are executed faster at some nodes. This happens when a protocol empties its messaging budget before the time limit expires.

VI. Conclusions

Simulations confirmed that the autonomous, online protocol evolution model is a promising approach for using to optimize and self-adapt multi-hop broadcast networks. In our experiments, evolution produced $50-66\%$ gain in terms of achieved average score, and also significantly improved the worst-case score and the variance of scores.

Our model using evolution and natural selection was able to neutralize the negative effects of a malicious protocol present in the system. However, AgrF was not simply eliminated from the system, but instead, parts of its code got incorporated into good-performing offsprings in some cases. In one scenario, a small and non-increasing amount of pure AgrF instances persisted among the survivors, however they represented a small percentage.

The evolution resulted different survivor constitutions depending on the mobility pattern and on the set of initial protocols (rigorously, AgrF $n = 1$ and AgrF $n = 3$ are two different original protocols). The feed-forward selection mechanism was able to offer enough adaptivity to the changing requirements, as demonstrated with the M2 mobility pattern around the phase shift.

Our results affirm our belief, that the demands for the new forms of networking infrastructure can be effectively addressed by bio-inspired solutions. Our focus was on presenting an evolutionary framework for the family of multi-hop broadcast protocols in ad hoc networks, where it is usually impossible to find a single absolute candidate, as the optimal protocol choice always depends on the actual environment and application conditions. We introduced a novel idea in this field: instead of human engineered static protocols, autonomous evolutionary methods were applied to achieve dynamic emergence of new ones, driven by the current needs and environment of the communicating nodes. For this purpose we have introduced an evolutionary model built upon a low-overhead, feed-forward, fully distributed, stigmergy based natural selection mechanism, and a genetic programming language GPDISS incorporating some nonconventional concepts such and relation types and event handlers.

We showed that the proposed model of evolving protocols is applicable for the multi-hop broadcast problem in ad-hoc networks: with time, evolution results in better performance than that the initial, manually engineered, protocols could provide. The fitness function was defined so that it used only local and quasi-local input, resulting in a model that is applicable for fully distributed systems such as ad-hoc sensor networks. Also, the feed-forward nature of the evaluation and selection process eliminated most of the communication overhead needed for the calculation of fitness values. Additionally, the process was carried out in an online manner, that is, the evolution of protocols happened continuously during the normal operation of the system. This is a significant feature, as the process is able to continuously search for new and better protocols without interfering with the normal operation of the system.

An important limitation of the model is that being based on a quasi-random search, it cannot provide any quality guarantee on the short term; for example, we cannot claim that the next generation of protocols will always improve the current one. While guarantees do not exist for the quality of protocol individuals, the overall performance of the system, especially for longer time windows, improves with high probability.

Online protocol evolution is a research subject that is in its infancy at this point. According to our knowledge, there have not been any initiative that resulted in a fully distributed, low-overhead, on-line, genetic programming based protocol evolution framework. Our ideas may be of interest to other researches for theoretic and practical reasons. It may provide insights to the solution of similar problems, especially in the area of ad-hoc, sensor and peer-to-peer networks. Theoretically, the idea is a fertile area for further research, and we hope that numerous interesting aspects and derivations will be introduced in the future.

References

[1] Ahmad Al Hanbali, Mouhamad Ibrahim, Vilmos Simon, Endre Varga, and Iacopo Carreras. A survey of message diffusion protocols in mobile ad hoc networks. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '08, pages 82:1–82:16, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[2] Sara Alouf, Iacopo Carreras, Daniele Miorandi, and Giovanni Neglia. Embedding evolution in epidemic-style forwarding. *2007 IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems*, pages 1–6, 2007.
[3] Markus F. Brameier and Wolfgang Banzhaf. *Linear Genetic Programming*. Genetic and Evolutionary Computation. Springer, New York, USA, 2007.
[4] Xiuzhen Cheng, Xiao Huang, Deying Li, and Ding zhu Du. Polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks. *Networks*, 42:2003, 2003.
[5] Michael D. Colagrosso. Intelligent broadcasting inmobile ad hoc networks: three classes of adaptive protocols. *EURASIP J. Wirel. Commun. Netw.*, 2007:25–25, January 2007.
[6] Vincent Cremet. *Foundations for SCALA: Semantics and Proof of Virtual Types*. PhD thesis, EPFL, Lausanne, Switzerland, 2006.
[7] Fei Dai and Jie Wu. Performance analysis of broadcast protocols in ad hoc networks based on self-pruning. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 15:1027–1040, 2004.
[8] Peter Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistries - a review. *Artif. Life*, 7:225–275, June 2001.
[9] Rajiv Gandhi, Arunesh Mishra, and Srinivasan Parthasarathy. Minimizing broadcast latency and redundancy in ad hoc networks. *IEEE/ACM Trans. Netw.*, 16:840–851, August 2008.
[10] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20:374–387, April 1998.
[11] Yamamoto Lidia, Schreckling Daniel, and Meyer Thomas. Self-replicating and self-modifying programs in fraglets. BIONETICS, Budapest, Hungary, 2007.
[12] K.F. Man, K.S. Tang, and S. Kwong. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics*, pages 519–534, 1996.
[13] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 151–162, New York, NY, USA, 1999. ACM.
[14] Vilmos Simon, Márton Bérces, Endre Varga, and László Bacsárdi. Natural selection of message forwarding algorithms in multihop wireless networks. In *Proceedings of the 7th international conference on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, WiOPT'09, pages 16–22, Piscataway, NJ, USA, 2009. IEEE Press.
[15] Lee Spector and Alan Robinson. Genetic programming and auto-constructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3:7–40, March 2002.
[16] Christian F. Tschudin. Fraglets - a metabolistic execution model for communication protocols. In *AINS'03*, AINS'02, Menlo Park, USA, 2003.
[17] E. S. Varga, B. Wiandt, B. K. Benko, and V. Simon. *Biologically Inspired Networking and Sensing: Algorithms and Architectures*. IGI Books, 2012.
[18] Brad Williams and Tracy Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '02, pages 194–205, New York, NY, USA, 2002. ACM.
[19] Jie Wu and Hailan Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, DIALM '99, pages 7–14, New York, NY, USA, 1999. ACM.

E. S. Varga is a PhD candidate at the Budapest University of Technology and Economics. He received his MSc degree in computer science in 2006. His main interests are discrete-event simulations and complex systems.

B. Wiandt is a PhD student at the Budapest University of Technology and Economics. He received his BSc degree in computer science in 2010. His main interests are self-organizing networks and genetic algorithms.

B. K. Benkő was a research fellow at Budapest University of Technology and Economics. She received her MSc degree in computer science in 2003. Her research interests include data mining, machine learning, and distributed and self-organizing systems.

V. Simon is an associate professor at Budapest University of Technology and Economics. He received his MSc degree in 2003 and PhD degree in 2009. His research interests include self-organizing and adaptive networks, evolution of communication protocols, opportunistic and delay-tolerant networks and mobility management in 3G- 4G mobile systems.