periodica polytechnica

Electrical Engineering
55/1-2 (2011) 65–70
doi: 10.3311/pp.ee.2011-1-2.07
web: http://www.pp.bme.hu/ee

RESEARCH ARTICLE

On-line genetic programming of multi-hop broadcast protocols in ad hoc networks

Endre Sándor Varga / *Vilmos* Simon / *Bernát* Wiandt

Received 2011-11-24

Abstract

From the family of ad hoc communication protocols the most challenging ones are those, that are designed to disseminate messages to all, or most of the nodes in the system. By their nature, these kinds of protocols use significant network resources, as the communication must involve a large fraction of the network nodes. Reducing the network load can be achieved by using the available local broadcast medium (radio channel), but it is not trivial how to select the set of nodes that should participate in the dissemination process. Previous attempts delivered algorithms that can provide reasonable performance and reliability but mostly for specific cases of ad hoc networks. In this paper a new way of tackling the broadcast problem is presented that takes no assumptions about the nature of the underlying network. Instead of using hand-optimizing protocols, we propose a framework for a self-optimizing and self-managing system inspired by natural selection and evolution. A generic distributed feed-forward performance evaluation criterion based on natural selection is presented along with an implementation of a virtual machine and a corresponding language for Genetic Programming to be used in tandem with the natural selection process.

Keywords

multihop broadcast · genetic programming · self-optimization

Endre Sándor Varga
Vilmos Simon
Bernát Wiandt
Department of Telecommunications, BME, H-1117 Budapest, Magyar tudósok krt. 2., Hungary

1 Introduction

A significant fraction of ad hoc networks (like opportunistic networks or Delay Tolerant Networks) uses multi-hop broadcast to disseminate a message to all (or most) of the nodes in the network. In contrast to ad-hoc routing protocols, the number of transmissions is significantly higher in a broadcast scenario, as the destination of a message is not a single node, but all of the nodes in the network. The optimization of such a protocol bears similarities with multicast protocols, but there is a distinguishing feature that presents interesting optimization possibilities: the availability of a local broadcast channel for every node. By exploiting local broadcast, the number of transmissions could be effectively reduced. The optimal solution is finding a minimal set of nodes so that the radio range of them cover all of the other nodes in the network. Finding such a set is not a trivial problem considering the distributed nature of the system. Mobility presents another challenge, as the optimal set of nodes will change over time.

A large amount of research is available on the optimization of broadcast protocols, and protocol design. Many of the surveys [1, 5, 8, 14] show that the performance of these protocols depends heavily on the parameters of the environment, such as the peculiarities of the mobility environment (including pattern and speed), density of nodes or traffic models. Some of the protocols also require the presence of specific devices (GPS, adjustable radio range) that may not be available to all nodes in a heterogeneous environment. The large number of protocols and their different "sweet-spots" make it hard to find a good compromise. Real systems also change, and mobility patterns are hard to model or predict, therefore even an initially sound decision could become under-optimal over time. A real versatile protocol should cover all of the possible cases in a changing environment, but this is a hard task in practice. There is evidence that the behavior of many of the efficient broadcast protocols could be emulated by simple heuristics [6]. Finding those heuristics are not trivial either, but an automated process may help finding good candidates. In our work we present a possible solution that uses natural selection to find good protocol candidates for a given system. As we demonstrated, an off-line, pre-deployment

optimization process – even if it is automated – is suboptimal, as the conditions will inevitably change sooner or later. Therefore, the natural selection and evolution of protocols must happen during the operation of the system. This way, we achieve an adaptive system that changes behavior reacting to changes.

To avoid the issue of choosing manually the best protocol for every given network, we investigated ways to achieve automatic protocol selection. Such kind of system should be able to run different protocols at the same time and be able to choose the most fit ones to the particular scenario. In our previous work, we introduced a natural selection based criterion for selecting a well-performing protocol locally [10]. We provide an overview of our selection criterion in Section 2. In this system a manually selected set of protocols was available to all of the nodes of the system, and nodes switched protocols according to their local environment. While this approach used natural selection as an optimization process, there was no real evolution implemented as no new protocols were introduced.

By using natural selection instead of traditional feedback type optimization we achieved a scalable and distributed adaptive system. However our investigation showed that natural selection itself is not enough, as the diversity of the competing protocols depends entirely on the system operator who adds the protocols to the selection pool. To overcome this limitation we decided to add Genetic Programming (GP) capabilities to the system and so achieve full evolution. To enable GP – most particularity online GP – we had to design a specialized language and Virtual Machine (VM) that is expressive and simple enough to produce useful broadcast protocols. The description of our language can be found in Section 4.

2 Inverted decision by using natural selection

The essence of broadcast protocols is the approximation of a Minimal Connected Dominating Set (MCDS) [8, 12]. In practice, finding an MCDS is likely to be NP complete, but it does not matter too much, as in practice the graph could change faster than it is possible to discover the changes. As calculating an MCDS is impossible in reality, all of the broadcast protocols use some approximation based on simple heuristics and local knowledge. These heuristics differ in sophistication, from simple counter based and probabilistic methods to complex graph theoretic approximations [5, 9]. These heuristics are dependent on the environment, therefore it makes sense to choose a protocol according to the local situation.

If we exclude manual design, then the only possible way to select suitable protocols is to measure their performance locally. This leads to the problem of feedback:

- Only senders could reliably measure the real cost of successfully transmitting a message. Lost messages could not be seen by other nodes.
- Only receivers could reliably measure the number of duplicated messages.
- Receivers could measure only the *local* duplications, but not the total duplications.
- Collection of these measurements are possible only through message passing using the same channel as broadcast payloads.
- Measurement messages could get lost.

Our observations imply that implementing a centralized (even locally centralized) protocol selection criterion is not practical. Instead we proposed a feed-forward selection method using stigmergy and natural selection, that we introduced in [10]. The idea of natural selection is not new, in [2] the authors used a form of natural selection for parameter optimization, using explicit feedback from neighboring nodes. However, our approach differs in that it does not need any feedback mechanism and works with arbitrary broadcast protocols.

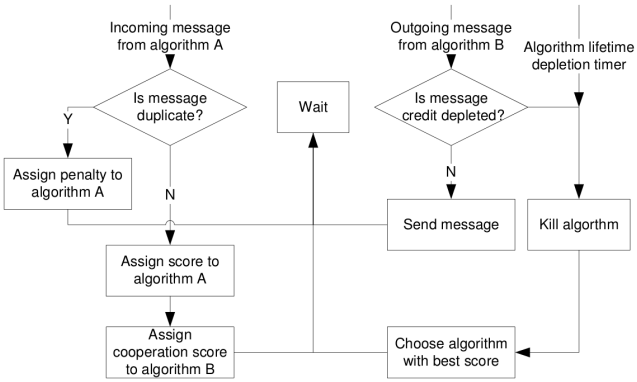

Fig. 1. Natural selection based protocol selection. Algorithm B is running on the node while a message from algorithm A arrives.

The proposed solution is based on the idea of decision-inversion. The naive way could be that a sender collects its performance metrics from the surrounding receivers and chooses the next protocol according to this measurement. As we explained earlier, this can not be efficiently implemented in practice. Instead of choosing locally, we implement the decision at the receivers, because they are in the best position to observe the performance of a protocol. To make this possible, the receivers must know the protocol that sent a given payload message. This is achieved by senders attaching the code of the sender protocol to every payload. This compound packet acts as a *virtual seed* where the nutritional part of the seed is the payload, and the genetic material is the code of the sender protocol.

Nodes collect seeds from surrounding nodes and assign scores to the protocol instances contained in them. Every payload that is useful to the receiver node means a score for the sender protocol. Every unnecessary message (duplicate) means a negative score to the sender protocol. This algorithm is summarized in Fig. 1.

To reduce the number of messages sent out a cost must be assigned to transmission. As the selection of new algorithms

happen at the receivers it is impossible to implement explicit cost calculation without expensive control overhead. To avoid this, we adopted a stigmergic solution: by assigning a limited transmission budget to each protocol instance, the protocols are forced to make good use of 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 a new generation.

3 Genetic programming of protocols

Genetic algorithms are global search algorithms directed towards an optimal solution with the advantages of a random search. They eventually result in an optimal solution if one exists, by generating all of the possible solutions – however this could take an arbitrary long time. The algorithm evaluates each solution and ranks them using a fitness function. After the fitness of the solutions is evaluated the algorithm creates a new generation of them. The genetic operators include selection, crossover and mutation. Selection and crossover ensures the solutions are converging to an optimal solution and mutation gives the algorithm a random behavior, which is crucial for optimal results. Genetic Programming (GP) is a form of genetic algorithm, where solutions are programs composed of instructions in a particular programming language.

The problems present in ad hoc networks such as the unpredictability of the position or speed of the mobile node and the diversity of devices that can be present in such a network makes it an ideal target for genetic programming. Using this approach it is possible to freely blend the behavior of protocols based on distinct principles. Various protocols have distinct "sweet spots": they work best under different conditions. By increasing the diversity of the used protocols and enabling them to adapt freely to the current environmental conditions it is possible to always have a protocol (or a family of protocols) that works best in the current environment. This approach can be considered as an automated protocol design tool. In [6] the authors used machine learning algorithms to approximate the behavior of sophisticated broadcast algorithms and found that simple heuristics were able to reproduce them in 87 % of the time. This result indicates that in practice small, but powerful heuristics could provide good approximations. Such heuristics are usually hard to find by design, but a GP based system could find them by evolution.

The goal of introducing GP to our previous natural selection framework was to achieve greater diversity among the competing protocols. A simple natural selection scheme could result in a phenomenon known from Evolutionary Game Theory, namely, that the orbit of the system in the state space of protocol mixtures has no fix point but a limit cycle instead. This situation arises from the fact that many protocols are in a "rock-paper-scissors" relation of each other, periodically overthrowing each other in the network.

4 GP language for Multi-hop broadcast protocols

To be able to apply genetic programming to our protocols we have to express them in a particular programming language. Using a general language is problematic as genetic operators result in random modifications of the programs often causing syntactic errors. We have designed a language called GPDISS, with the purpose to describe multi-hop broadcast protocols, with the goal to optimize it for using genetic operators on it. In our language GP operators could be applied on the program code while still preserving syntactic correctness. This is crucial to reduce defective programs. The data structures and the instruction set is comprised of primitives, which are the basic building blocks of several well-known multi-hop broadcast protocols. The granularity of the instructions should by high in such a language in order to be "optimal" for genetic programming, resulting in a language, which is expressive enough to facilitate easy protocol implementation and the mixing of the behavior primitives through genetic operators.

Algorithm 1 Sample algorithm implementing a simple loop that executes five times

```
handler hello {
 1
 do
  int.inc
  int.dup
  5
  int.gt
 dowhile
}
```

One promising approach to GP languages is using an artificial chemistry [7], as they provide great resilience against random modifications. A good example of such a language is Fraglets [13], which was used to conduct experiments in protocol evolution in [15]. While these languages may have great possibilities, in our approach we followed a more conservative approach, and selected the well-known Push3 language as a base. The rationale is that evolved protocols are notoriously difficult to analyze for a human and artificial chemistries result in even more complicated systems.

Our language is based on the stack based PUSH(3.0) [11], which means that it has typed stacks for every type in the system. Instructions get their parameters from these stacks. This way we can avoid the use of variables, which results in a much cleaner and easier language. The code of a protocol is divided into event handlers, which are the base of genetic operators. For example a crossover operator is mixing the instructions of two event handlers, a mutation modifies one event handler. Each message type has an event handler, which are activated when the appropriate messages arrive. The semantic correctness of the programs can not be guaranteed after the use of genetic operators. If an instruction does not have an argument available on the stack, it can not be executed. These instructions default

Algorithm 2 Sample algorithm finding the 2-hop neighbors of the node

```
// Pushes a copy of the relation to the relation stack (for Step1)
relation.dup
// Pushes a copy of the relation to the relation stack (for Step2)
relation.dup

node.self // Pushes the identifier of the node  to the node stack

//Step1: Look up my direct neighbors
relation.filter_key eq // Filters the top element of the relation stack
by key, leaving only those rows whoose key column equals to the node

//Step2: Look up  neighbors  of the  neighbors
relation.join // Joins the two relations (filtered one and the original),
 so finding our neighbor's  neighbors in the original map
```

to a no-op instruction, which does nothing and execution goes on undisturbed. The implementation of a protocol is a list of the assembly-like instructions grouped into event handlers. This results in a quasi-linear [4] genetic programming language. We designed the language with an emphasis on easy extensibility: it is easy to make use of extra devices such as GPS receivers when designing protocols. Adding a new instruction is as easy as adding it to the grammar and writing its implementation. Table 1 shows some of the instructions in the language.

In the case of network protocols, performance is crucial to reduce latency. As it is demonstrated by many virtual machine implementations (Java Virtual Machine, Common Language Runtime, etc.) near real-time performance is achievable by using smart just-in-time (JIT) compilers. The only factor that causes non-deterministic latency using these runtime environments is the Garbage Collector that automatically manages deallocation of objects on the heap. Our language is stack based, so the popular Virtual Machines are good candidates as compile targets. Luckily, the GPDISS language has a very limited instruction set, and heap based operations are not present therefore avoiding most of the latency caused by garbage collection runs. Also, object-oriented features are missing, so the JIT compiler is free to inline methods as there are no polymorphic calls. In summary, the following features of GPDISS make it a good choice for low-latency applications:

- Low-level, assembly like instruction set that allow efficient and simple compilation of its instructions to machine code
- No dynamic memory management (no garbage collection involved)
- The very few operations that may need dynamic allocation could be simply implemented using an Arena Based Garbage Collection algorithm, very similar to that of the MySQL query compiler. This effectively ensures zero overhead deallocation in contrast to other (Copying, Mark-Sweep-Compact) deallocation approaches.
- No polymorphism is present, all of the methods are inlineable. This allows the instruction scheduler of the compiler to provide global reorderings to efficiently feed the CPU pipelines

In our current work we have not investigated the performance of the language in practice, as that would need a real compiler, which was not our goal at this point. In the future we plan to implement a compiler that targets the JVM runtime environment, and may also provide an implementation using the Low Level Virtual Machine (LLVM) framework that produces high quality native code for many architectures.

Algorithm 3 APF data handler in pseudo-code

```
On receive ( data : DataMessage ) :
 delta := 2
 if contains ( seen , data ) then
  period := period + delta
  // Increase broadcast period on duplicates
 else
  to_send := to_send + data
  seen := seen + data
```

In Algorithm 1 and Algorithm 2 we show some examples of the language in use. The first algorithm is a simple loop construct that executes five times, implemented by a counter on the integer stack. The second algorithm is a more complex one that demonstrates the relational instructions by calculating the 2-hop neighbors of the node, given the neighborhood graph represented as a set of relations.

Adaptive Periodic Flood (APF) is a controlled flooding protocol, which achieves better performance than blind flood without the use of control messages. We give another demonstration of the language with an event handler of the APF protocol because it is easy to implement and provides an insight into the logic of the GPDISS language. In the example receive in Algorithm 3 and Algorithm 4 the event handler uses two lists storing data messages. The list on the top of the global stack is called "to_send" and contains the messages we are about to send when the timer we started earlier fires. The next list is the "seen" list and contains the data messages we have seen so far. This list is

Tab. 1. Instructions in the GPDISS language

Stack	Instruction	Description
*	dup, drop, swap, rotate, hold, release, ...	Common instructions available on all stacks. They are ideal for common stack handling tasks.
number	add, div, mult, random, ...	Simple floating point arithmetic and random number generation
bool	and, or, not, ...	Boolean logic 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.
timer	id, start_timer, ...	Timers can be used to schedule different tasks at different points in time.
-	if, else, endif, do, while, return, ...	Control flow constructs.

used to detect duplicate incoming messages and reduce the rate of periodic transmissions.

5 Simulation

To test that our language is able to provide a functional broadcast service, we simulated six scenarios that used two very basic protocols as a starting point, APF (explained in Section 4) and Blind Flood. We chose these because they are easy to implement and understand, so they simplify the validation of our language and virtual machine. The parameters of the simulation were:

Node count	100
Simulation area	100m × 100m
Mobility model	Random Direction Mobility
Transmission range	5m
Interference range	7m
Transmission speed	1Mb/s
Incoming traffic	New message in every 20s
Avg. payload	250 byte
Starting protocols	APF, Blind Flood

We simulated the system using no selection mechanism, natural selection, and using GP. In the natural selection scenario no genetic operators were used. The reproduction strategy simply picks the best protocol seen by the virtual machine (messages transfer the sending protocol along with them). In the GP scenario we used a simple mutation, which alters the number constants in the code according to the following formula: $mut(x) := x + gaussian() \cdot x$. Using this formula we can keep the magnitude of the constant, therefore it emulates a form of parameter optimization. For the GP scenario the reproduction strategy produced a new generation of protocols with SUS (Stochastic Universal Sampling) [3] selection. This selection algorithm provides zero bias and minimum spread. The new generation were then shuffled and for each pair the crossover and

Algorithm 4 APF data handler in GPDISS language

```
handler data {
// if a new message is arrived
// data between event handlers are
// shared through a global stack
number.popglobal
list.popglobal // to_send
list.popglobal // seen

data.dup
// making a copy of the data message

// true on the bool stack if  the data message
// is on the stack , false otherwise .
// Note : the lists are typed .
list.inlist

if
 // if we alrady saw this message
 number( 2 )
 number.add
 // increase period with delta
else
 // if the message is new
 data . dup
 list.additem // add message to seen
 list.swap // to_send , seen
 list.additem // add message to to_send
endif

number.pushglobal
list.pushglobal
list.pushglobal
}
```

Tab. 2. Results of 200 seconds of simulation, using different mixtures of initial protocols (NS=Natural Selection; GP=Natural Selection with Genetic Programming)

	APF 20% BF 80%			APF 80% BF 20%		
Selection strategy	None	NS	GP	None	NS	GP
Useful messages	1259	680	1504	5033	1942	595
Duplicates	32212940	6485	5945	4396	2046	1580

mutation strategy were applied. The mutation strategy was the parameter optimization that was described above.

Our crossover strategy for the scenario was a modified one-point crossover. It works on two protocols, *A* and *B*.

1 Choose an event handler from *A* randomly

2 If handler is present in *B*

(a) Two crossover points are selected
(b) Event handlers are cut along the crossover points giving two fragments – a head and tail for each handler
(c) With 1/2 probability we swap the head and tail we are about to attach.
(d) Handler fragments are glued together such that the event handler's tail chosen from *A* is attached to the event handlers head chosen from *B*.
(e) The same goes on for the tail from *B* and the head from *A*.

For safety reasons we had to limit the size of the event handlers. Ever growing event handlers can cause messages that cannot be transferred due to limitations in the virtual machines (each protocol from a generation has a transfer bound). All strategies were given a threshold value, a probability of their application.

Table 2 shows the results of 200 seconds of measurements using different mixtures of protocols as a starting point. According to the measurements, there is no clear winner. The GP approach proved best in a relatively hostile (aggressive channel usage by Blind Flood) scenario where it outperformed all of the other approaches. In other scenarios it performed relatively poorly, although it was far from catastrophic – unlike the "No selection" scenario, that produced duplicates three magnitudes more than others. It is also clear that the GP scenarios maintained service in every case without rendering the channel unusable, which is quite remarkable regarding that the protocols were produced by an unguided process.

6 Conclusion and further work

In our article we introduced a language and a framework for dynamically adapting broadcast protocols for an unknown environment. We showed that such an approach could work in practice, and GP generated protocols using our natural selection method could maintain a multi-hop broadcast service without breaking it. We proved that such a system could in certain situations surpass simple hand-designed protocols, although we highlighted that this does not happen in every case. The practical applicability of this framework at this stage is not yet possible, several improvements must be done to make real-world use feasible.

In the future we will implement many of the more advanced broadcast protocols in our language and we will test them under an evolutionary scenario. We will also fine-tune the selection mechanism to increase the efficiency of the system and investigate further possibilities in the genetic recombination method, including heuristics to exclude most of the dysfunctional protocols.

References

1 **al Hanbali A, Ibrahim M, Simon V, Varga E, Carreras I**, *A survey of message delivery protocols in mobile ad hoc networks*, InterPerf (2009).

2 **Alouf S, Carreras I, Miorandi D, Neglia G**, *Embedding evolution in epidemic-style forwarding*, Proc. of IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2007), posted on October, 2007, DOI 10.1109/MOBHOC.2007.4428686 , (to appear in print).

3 **Blickle T, Thiele L**, *A comparison of selection schemes used in genetic algorithms*, 1995.

4 **Brameier M, Banzhaf W**, *Linear genetic programming*, Genetic Programming, vol. 9, Springer Netherlands, 2008.

5 **Cheng X, Huang X, Li D, zhu Du D**, *Polynomial-time approximation scheme for minimum connected dominating set in ad hoc wireless networks*, posted on 2003, DOI 10.1002/net.10097 , (to appear in print).

6 **Colagrosso M D**, *Intelligent broadcasting in mobile ad hoc networks: Three classes of adaptive protocols*, EURASIP Journal on Wireless Communications and Networking, posted on 2007, DOI 10.1155/2007/10216 , (to appear in print).

7 **Dittrich P, Ziegler J, Banzhaf W**, *Artificial Chemistries – A Review*, Artificial Life **7** (2001), no. 3, 225–275, DOI 10.1162/106454601753238636.

8 **Fei Dai J W**, *Performance analysis of broadcast protocols in ad hoc networks based on self-pruning*, 2004.

9 **Li H, Wu J**, *On calculating connected dominating set for efficient routing in ad hoc wireless networks*, Proc. of the Third International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DiaLM) (1999).

10 **Simon V, Berces M, E. Varga E, Bacsardi L**, *Natural selection of message forwarding algorithms in multihop wireless networks*, WiOpt, posted on 2009, DOI 10.1109/WIOPT.2009.5291633, (to appear in print).

11 **Lee Spector, Robinson A**, *Genetic programming and autoconstructive evolution with the push programming language. Genetic Programming and Evolvable Machines* **3** (2002), 7–40.

12 **Khuller S, Guha S**, *Approximation algorithms for connected dominating sets* (1996).

13 **Tschudin C**, *Fraglets - A Metabolistic Execution Model for Communication Protocols*, Proc. 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS) (July 2003).

14 **Brad Williams, Tracy Camp**, *Comparison of broadcasting techniques for mobile ad hoc networks*, 2002.

15 **Yamamoto L, Schreckling D, Meyer T**, *Self-replicating and self-modifying programs in fraglets*, Proc. of BIONETICS, posted on 2007, DOI 10.4108/ICST.BIONETICS2007.2446 , (to appear in print).