Synthesizing Customized Network Protocols using Genetic Programming

Mohammad Roohitavaf, Ling Zhu, Sandeep Kulkarni, and Subir Biswas
Michigan State University
East Lansing, MI, USA

ABSTRACT

Given the advances in areas such as home automation, agricultural networks, smart cities, designers often need to design protocols that utilize the features of that network while dealing with its limitations. Utilizing standardized protocols for such networks may not be appropriate as they may not address limitations of the network such as heterogeneous nodes or limited capability of some nodes. While designing a customized protocol for that network would be desirable, it is extremely time-consuming unless we can automate the development of the required protocol. In this paper, we present NetSynth, a GP based mechanism to develop customized routing protocol for the given network. NetSynth lets us conveniently describe a network using an XML file, and it synthesizes a routing protocol that suits the input network by considering the characteristics specific to the given network. We show how NetSynth creates protocols that perform comparably to best-known protocols for the case where we want to broadcast a set of messages to all nodes in a grid. We also show how NetSynth helps us design protocols that provide a tradeoff between throughput and energy.

KEYWORDS

Genetic Programming, Wireless Networks, Time Synchronization, Radio Interference, Medium Access Control, Routing

ACM Reference Format:
Mohammad Roohitavaf, Ling Zhu, Sandeep Kulkarni, and Subir Biswas. 2018. Synthesizing Customized Network Protocols using Genetic Programming. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208272

1 INTRODUCTION

The quality of the communication over a network is directly affected by the protocol that the network nodes use to communicate with each other. Existing work on network protocols generally focuses on, among other things, general cases where nodes can dynamically join or leave the network, the topology can change and link characteristics may vary. This has several advantages. Most importantly, the protocols do not need to be redesigned and are easily deployable. However, this has reduced flexibility in terms of not being able to cater to the specific requirements of the network. For example, if some devices are battery powered and some devices are mains powered, it is desirable to have mains powered devices assist battery powered nodes to save energy by assisting them in routing, MAC, and other layers.

This issue is of a significant interest as we venture into new areas such as smart cities, agricultural networks, home networks and office automation. A key characteristic of such networks is that they consist of several devices that need to collaborate with each other. For example, in an agricultural network, nodes may consist of sensors that monitor moisture, humidity, and temperature to determine when sprinklers should be turned on. Home automation may consist of devices streaming audio/video, or security-related devices such as cameras, or alarms associated with the home intrusion. Each of these devices has different service requirements in terms of acceptable latency or jitter, desired throughput, etc. Some of these devices may be battery powered whereas some are connected to AC power. Thus, an application designer is tasked with deciding how these nodes should talk with each other to achieve their tasks under the given constraints.

A common approach in these cases is to ignore underlying system constraints and utilize a standardized protocol such as 802.11 or 802.15.4. The advantage of this approach is that it is the easiest to implement. However, this may introduce undesirable issues such as draining the battery of battery powered devices too quickly.

An alternate approach is to use non-standard specialized protocols such as S-MAC, Ariadne, DSR [8, 10, 14, 15] that are designed for optimizing a desired metric (e.g., energy usage). This approach increases the time to build a network, as the designer must optimize the parameters involved in such a protocol for the given network. This has the potential to optimize some known performance requirement (e.g., energy). However, it is not easily predictable whether the optimization of the given protocol will apply or scale to the specific given network.

These limitations can be overcome by the extreme approach of *handcrafting* a set of protocols at different protocol layers for a given network with specific properties and performance expectations. This approach has the potential to achieve the best tradeoff in terms of energy, throughput, etc. However, handcrafting such protocols manually is extremely time-consuming. Thus, if we want to handcraft a protocol that is targeted towards the given network, we must utilize automation to obtain the desired protocol. We focus on designing such handcrafted protocol with the help of Genetic Programming (GP).

We introduce our framework NetSynth as a way of designing and implementing communication network protocols by leveraging

[0]This work is supported in part by NSF XPS 1533802.

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208272

automated protocol synthesis via GP. NetSynth permits a network engineer to utilize the features and limitations of a given network to synthesize a routing protocol for that network. Our framework utilizes techniques from Genetic Algorithm (GA). Depending upon the context, it permits usage of single objective GP or multi-objective GP with NSGA-II [4].

NetSynth takes as input the underlying topology of the network. It allows one to model typically important requirements such as channel or node loss rate, amount of battery capacity (if any) at a given node, and whether AC power is available at that node. It also takes as input communication characteristics (senders, receivers, data rates) as well as whether the communication is targeted towards a specific node or is a broadcast communication. Additionally, as input, NetSynth takes the desired goals such as minimize power, reduce latency, and increase throughput.

The goal of NetSynth is to generate a protocol that identifies when and how the data should be transmitted. The *when* part focuses on the MAC layer, i.e., it identifies when each node should transmit its data, when it should listen for others and when it should be in sleep mode to save power. The *which* part focuses on how it should handle possibly lost messages. Finally, the *how* part focuses on routing issues. For example, it identifies how AC powered nodes could assist in saving power of battery powered nodes.

The contributions of the paper are as follows:

- We present NetSynth, a framework for designing a protocol for a given network which is specified in an XML document that specifies the following
  - Topology along with node and link characteristics
  - Traffic pattern to be either (1) broadcast communication where one node sends data to all other nodes, (2) unicast communication where we have pairs of senders and destinations.
  - Data arrival rate at the sender(s).
  - Objectives, which can be throughput, latency, or energy.
- NetSynth generates a protocol that identifies the MAC layer protocol (when a node should transmit, listen or sleep) and network layer protocol (how messages should be routed to reach the destination).
- We build a network simulator that evaluates protocols developed for NetSynth to analyze throughput, latency, and energy utilization. It can be used as a fitness function evaluator based on the objectives in the given input. When faced with more than one objective, we use NSGA-II during evolution.
- We demonstrate NetSynth with the case where we have broadcast communication and one node wants to send a set of packets to other nodes. For such a network, we show that NetSynth is able to synthesize the best-known protocol. Furthermore, while NetSynth cannot generate the best-known protocol in every execution, the median protocol identified in several runs has 85% throughput compared with the best-known protocol.
- In the context of unicast, we demonstrate the ability of NetSynth to develop protocols that require it to identify the shortest path as well as a path with low energy consumption. When faced with conflicting requirements, we show that NetSynth is able to provide a tradeoff between different objectives.
- We provide a graphical tool called Network Designer that lets the network designers conveniently define their network. This tool automatically generates the XML document needed by NetSynth.

Organization of the paper. The rest of the paper is organized as follows: In Section 2, we provide an overview of the previous work of using evolutionary techniques to synthesize/optimize network protocols. Section 3 provides an overview of NetSynth. Section 4 focuses on different types of protocols that we can synthesize with NetSynth. Section 5 introduces Network Designer too. Section 6 provides the results of some our experiments with NetSynth. Finally, Section 7 concludes the paper.

2 RELATED WORK

Several existing papers have explored evolutionary computation to improve network protocols. Alouf et al. in [1] used GA to adjust protocol parameters in response to the network dynamics. Specifically, they focus on the Epidemic Routing [1] where each node sends a copy of a received message to the other nodes based on a forwarding probability. The nodes estimate fitness objectives namely, delivery time and the number of copies, that are fed to the GA to evolve better forwarding probability. Similarly, Lewis et al. in [13] use GP to enhance IEEE802.11 protocol. Specifically, they explored GP to improve the behavior of the contention window by optimizing different parameters used in the protocol.

Evolving finite state machines to synthesize MAC protocols is studied in [7]. Specifically, in [7], a protocol is considered as a probabilistic state machine. Each genotype specifies the probabilities of transitions between different states. Results provided in [7] show how GA is able to evolve these probabilities to achieve a performance close to the well-known pure-ALOHA protocol. Authors have extended [7] in [6] toward Slotted ALOHA and CSMA logic. Like other papers mentioned in this section, [7] and [6] focus on evolving a set of transition probabilities for a state machine shared by all nodes. Comparing to these existing papers, our framework is not limited to parameter optimization. Instead, it allows us to modify the structure of an existing protocol to improve its performance, or even synthesize a new protocol from scratch.

3 OVERVIEW OF NETSYNTH

The goal of NetSynth is to provide the network designers with a general tool that lets them synthesize network protocols that suits their network the best in terms of communication performance and energy efficiency. Figure 1 shows the overall architecture of NetSynth. NetSynth permits the protocol designer conveniently describe the network and its different characteristics using a descriptor file as the input to the tool. This descriptor file is an XML file according to the `NetworkDescriptor.xsd` that comes with the framework.

Internally, NetSynth relies on GP to synthesize a customized protocol for the given network. Specifically, NetSynth uses either a single objective GP or a multi-objective GP with NSGA-II [4] algorithm to evolve protocols that aim to optimize design requirements. The network simulator plays the role of the fitness function.

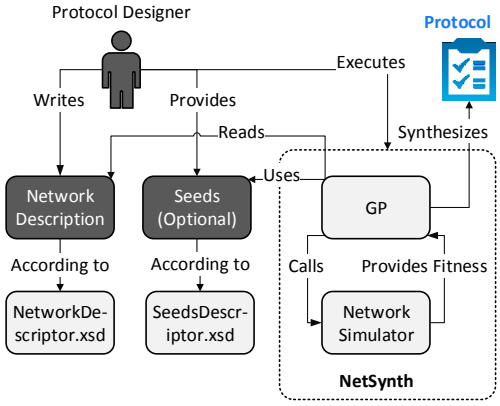


Figure 1: The overall architecture of NetSynth framework

Specifically, NetSynth simulates a candidate protocol on the given network, and uses the outputs of the simulation as the objectives achieved by the protocol. NetSynth comes with network simulator for broadcast and unicast communication pattern. Also, NetSynth is extensible in that it is straightforward to expand the framework by writing new simulators for other communication patterns and other instructions (e.g., sense-medium, backoff).

This paper focuses on using NetSynth to design protocols from scratch. However, if we model existing protocols as part of the initial protocols, it can be used to improve/combine existing protocols. Next, we discuss how different aspects of a network are modeled in NetSynth.

3.1 Modeling MAC layer

The current version of NetSynth focuses on Time-Division Multiple Access (TDMA) based protocols. In these protocols, time is partitioned into frames and each frame consists of a fixed number of slots. Each slot contains an instruction that executes in that slot. NetSynth currently supports three instructions, transmit, listen and sleep. However, the implementation of NetSynth is generic enough to allow other types of protocols such as Carrier-Sense Multiple Access (CSMA) protocols by adding instructions such sense-if-medium-is-free, backoff-k-slots, etc.

Thus, MAC layer in NetSynth is modeled as shown in Figure 2. In this figure, the frame consists of five slots. The first slots of nodes 0, 1 and 2 are transmit, listen and sleep respectively. The size of the frame is identical for all nodes. However, the slot assignment of each node is potentially different. Since the frame is periodic in nature, each node will repeat its program after the completion of its frame. In other words, in Figure 2, slots 6 and 7 would be similar to slots 1 and 2 respectively.

Node A transmits successfully to its neighbor B if there is a slot where A is in transmit mode, B is in the listen mode and no other neighbor of B is in transmit mode. For example, in Figure 2, in the first slot, node 0 will successfully send a packet to node 1. However, in the second slot, there will be a collision at node 1 as both 0 and 2 are transmitting at the same time.

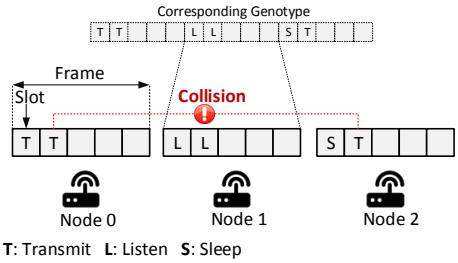


Figure 2: An example of a MAC layer protocol with frame size 5 and its corresponding genotype

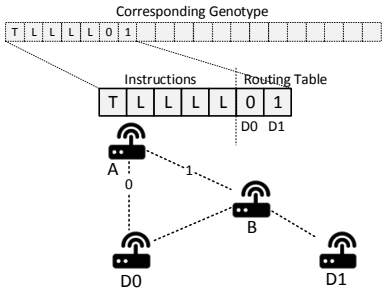


Figure 3: An example of a MAC and routing layer protocol with its corresponding genotype

3.2 Modeling Network Characteristics

Section 3.1 considers the case where messages may be lost due to a collision. In a wireless network, however, messages can be lost even under normal circumstances. In NetSynth, we allow one to model such losses by associating an interference probability with nodes and channels. If an interference probability is associated with node A, then it implies that messages sent to node A would be lost with given probability. Such a scenario occurs in ad-hoc networks where nodes are sharing the same frequency with other unrelated networks. In this case, with some probability, the other (unrelated) network may cause collision thereby causing loss of packets.

If an interference probability is associated with link $\langle A, B \rangle$ then messages sent over the link would be lost with the given probability. Thus, we can model effects caused by nodes that are far apart, are separated by a physical barrier causing an increase in loss of packets, asymmetric links, and so on. Finally, when both interference probabilities are specified, the effect is cumulative.

3.3 Modeling Routing Layer

One can hardcode the routing table based on the underlying topology. Alternatively, the routing table can be evolved using GP. In NetSynth, we follow the latter approach. Specifically, the genome consists of a routing table for each node. The routing table consists of one entry for each possible destination (cf. Figure 3). The entry for node t identifies the successor node to which the packet destined towards node t should be transmitted. For example, as shown in Figure 3, at node A, packet intended for node $D1$ should be forwarded to node B.

To ensure that the entry in the routing table is a neighbor of the current node, we include the *neighbor-number* in the routing table. For example, in Figure 3, for node A, its neighbor 0 is node $D0$, neighbor 1 is node B, and so on.

Note that keeping the routing table consistent in this fashion does not guarantee that messages would reach their destination. This property is evaluated by the fitness function which will capture, among other things, packets received by the destination.

4 USING NETSYNTH TO SYNTHESIZE NETWORK PROTOCOLS

In this section, we describe how NetSynth models and evaluates protocols for the given network. NetSynth enables synthesis of protocols where we have broadcast communication or (one or more) unicast communication. The former is essential for network management. Here, the node that initiates the broadcast intends to control other nodes in the network by changing their system parameters, changing how they coordinate with each other, etc. This area is also studied in the context of network reprogramming where the goal is to send the nodes in the network a new program (or a patch to an existing program) [9, 11, 12].

The latter captures a vast variety of protocols that is a generalization of the producer-consumer problem in the literature. In this case, the senders are producers that generate data and destinations are consumers to whom the data are targeted. As an example, senders may be cameras in a security system. They generate data to be analyzed by the destination, backend of the security system that determines if an alarm needs to be raised. Alternatively, senders could provide details of soil moisture and destinations are responsible for deciding when sprinkler should start.

4.1 Synthesizing Broadcast Protocols

In this section, we discuss how NetSynth enables the design of protocols targeted towards (multi-step) broadcast. In such a network, there is some node, called initiator, that has the data that it wants to send to all others node.

Since the data are to be sent to all nodes, there is no need for a routing table. NetSynth utilizes diffusion as a way to get the data to all nodes in the network. Specifically, first, the initiator sends the data to its neighbors. They forward it to their neighbors and so on until all nodes receive that data.

4.1.1 Fitness evaluator for broadcast. For broadcast, in NetSynth, each node maintains information about packets its neighbors have received. This information could be based on the fact that the node has sent that packet before or based on the fact that it has acknowledged that packet before. Additionally, after a node concludes that all of its neighbors have received packet x then it can transmit packet $x + 1$.

Recall that when a node A sends a packet, it would be received by node B if (1) B is in the receive mode, (2) no other neighbor of B is is in transmit mode and (3) message is not lost due to probabilistic message loss at node B or link $\langle A, B\rangle$. Additionally, to know if neighbors have received a message, we use implicit acknowledgments. When a node transmits a message, it is used as an acknowledgment by its predecessors. For example, consider the case where a message is being routed from node A to B to C. When a message is transmitted by B to C, it is also received by A (if A were to be in listening mode at the time and there is no loss or collision) due to the nature of the wireless medium. Hence, node A will treat it as an acknowledgment from node B.

Since the objective in broadcast communication is throughput, the fitness evaluator provides the number of packets received by each node. They can be combined in different ways to compute the fitness of the given individual. One possibility is to include the sum of packets received by all nodes. An alternative is to keep track of the minimum number of packets received by any node. NetSynth lets the designer to specify this in the XML file.

4.1.2 Evolution of Broadcast Protocols. The GP process begins with an initial set of protocols. In experimental evaluations presented in this paper, we use only a single objective, the number of distinct packets received by all nodes. We discuss the motivation behind this in Section 6.

4.2 Synthesizing Unicast Protocols

In this section, we describe how NetSynth enables synthesis of a unicast protocol. We model the network to contain pairs (s_1, t_1), $(s_2, t_2), \cdots$. For each pair (s_i, t_i), there is a distribution function that identifies the arrival rate at node s_i. The network also identifies the topology and qualities of links/nodes.

The broad structure of a unicast protocol from sender s to destination t is as follows: Node s, first, obtains a new packet from its application. It forwards it to some other node in the network (as determined by the routing table), then to some other node and so on until it reaches t. One can instantiate it for a single sender-destination pair or multiple pairs. It is possible for a node to be part of multiple sender-destination pairs (as a sender or a destination). Next, we discuss, how we evaluate the fitness of a given protocol/individual that is generated as part of the GP process. Subsequently, we discuss how GP is used to evaluate the desired protocol.

4.2.1 Fitness Evaluator For Unicast Protocols . The arrival of packets at the network layer of s is determined by the application, i.e., it is part of characteristics of the given network. In our experiments presented in this paper, we assume that the packets are generated at a fixed rate defined in the input XML file. NetSynth lets other distributions (e.g., exponential).

When a new packet is available, it will be sent to other nodes when a transmit slot is available. Specifically, when the network layer at node s receives the packet, it sends it to one of its neighbors so that eventually node t will receive the packet. This is achieved with the help of a routing table.

The above discussion handles the case where there is only one packet that the sender node wants to transmit. When there are multiple packets to be sent, the sender must choose the packet ID that is to be sent. We achieve this as follows: first, we allow a node to maintain the list of packets it should send. Second, as in case of broadcast communication, when a node transmits a message, it is used as an acknowledgment by its predecessors. Third, NetSynth also provides a mechanism of explicit acknowledgment, whereby the node sends information about packets it has received from others. We envision that this would be achieved by including sequence numbers in the packets and including those sequence numbers in

the acknowledgments. Thus, acknowledgment to many packets can be included in one message.

In unicast communication, we allow a node to send multiple packets without requiring that each be acknowledgment immediately. Once again, we consider the case where packets are being routed from node A to B to C. In this case, node A will send up to W (window size) packets without receiving an explicit acknowledgment. It will continue to monitor implicit acknowledgments to determine which packets have been received. By maintaining such sliding window, node A can continue to transmit new packets until a loss of a packet forces it to retransmit a packet in the current window. In our implementation, we let the size W be fixed. The protocol designer can set this value in the input XML document. In a future implementation, it would be possible to make it be part of the genome so that it can be evolved to identify the best window size for the given network. One side effect of this is that when the destination node, say t, receives a message, it utilizes its transmit slots to provide acknowledgments to packets it has already received.

We perform this analysis for up to thr slots, where thr is a configurable parameter. After completion of thr slots, the fitness evaluator reports the power used by each node and the number of packets received by each node. In most of our experiments, we combine these values to identify the total power used and the total number of packets received. However, other combinations are also feasible based on application requirements.

4.2.2 Evoluation of Unicast Protocols. In our experiments, we use random protocols, i.e., protocols where each routing table entry is set to some neighboring node and each slot is randomly assigned to be transmit, listen or sleep. However, NetSynth is flexible in that we can initialize the initial population to consist of known protocols from the literature.

For each individual, we evaluate its fitness. To deal with probabilistic links, we compute the fitness function for each individual 10 times and take the median of these. This ensures that the fitness function does not fluctuate substantially due to probabilistic nature of communication. In the experiments in this paper, we combine the data provided by the fitness evaluator into two objectives: cumulative distinct packets received by all destinations and cumulative energy used by battery operated nodes. Subsequently, we use NSGA-II to evaluate the protocols in the next generation.

5 NETWORK DESIGNER TOOL

Although `NetworkDescriptor.xsd` allows us to flexibly define our networks, writing network descriptor files can be a tedious and error-prone task for larger networks. To solve this issue, we provide Network Designer tool. Network Designer is a graphical tool that allows us to define our network visually. The tool provides an area where we can add our battery/AC powered nodes. We can connect nodes by lines to specify network connections. When we have several components that need to be all connected to each other, we can use hubs to avoid connecting them one-by-one.

Figure 4 shows the interface of Network Designer with an example network. In this network, we have a node named *Base* that wants to send packets to two destinations $A0$ and $A1$. The network is created by connecting the *Base* to two subnetworks. The left subnetwork consists of nodes $B0$, $X0$, $Y0$, $Y1$, and $A0$. The right subnetwork consists of nodes $B1$, $X1$, $X2$, $Y2$, $Y3$, and $A2$. The left subnetwork, right subnetwork, and *Base* are connected via a hub component. Note that nodes with battery icon are battery powered nodes. Other nodes are AC powered nodes. In Section 6.2, we will see how NetSynth trades off the throughput and energy by evolving protocols for the left and right subnetworks of Figure 4.

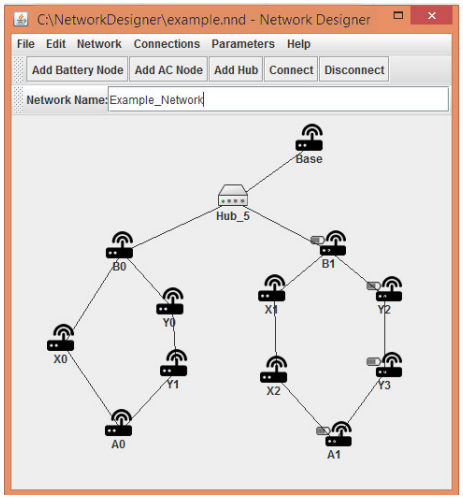


Figure 4: The interface of Network Designer

6 EXPERIMENTAL RESULTS

In this section, we present results with NetSynth for broadcast (Section 6.1) and unicast (Section 6.2). In all these experiments, we use GP (with single objective) or NSGA-II (with multi-objective) setting. We use bit flip mutation with mutation rate of $\frac{3}{\text{genome length}}$, and single-point crossover with probability 0.9.

6.1 Broadcast Results

In this section, we compare the effectiveness of NetSynth in generating protocols for broadcast in a rectangular grid. Here, the nodes in the network are arranged in a $m \times n$ rectangular grid where the initiator is in the upper-left corner (cf. Figure 5). For the network in Figure 5, we assume that each node can communicate with the neighboring nodes along the row or column.

The reason we consider this network is multi-fold. For one, this network is an abstraction of networks [2, 3, 5] deployed for monitoring intrusion in a physical deployment. Here, each node consists of a sensor that detects possible intrusion in the network. The nodes communicate with each other to ensure that any intruder is immediately detected and tracked. Another reason is to evaluate the effectiveness of NetSynth by comparing the generated protocols with best-known protocols that are designed by hand. Specifically, we first describe the best-known protocol, GridPr, for this network and compare it with protocols generated by NetSynth.

6.1.1 The Best-Known Protocol for Broadcast in Figure 5. The best-known protocol, GridPr, for the network in Figure 5 utilizes a frame size of 5 (this is independent of m and n as long as they are at least 3), i.e., each frame consists of 5 slots, 0..4. The initiator

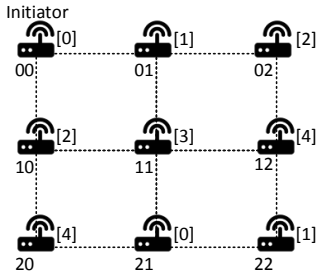


Figure 5: A 3×3 grid

transmits in slot 0 of every frame. The node on the right of the initiator (node 01) transmits in slot 1 and the node below the initiator (node 10) transmits in slot 2. This ensures that the initiator can receive (implicit) acknowledgment from its two neighbors without a collision. This pattern is repeated in the remaining network. In particular, a node transmits one slot after (in modulo 5 arithmetic) its left node and two slots after (in modulo 5 arithmetic) its top node. All other slots are for listening either for new messages from nodes closer to the initiator or for receiving (implicit) acknowledgments from nodes farther from the initiator. The slot assignment for the network in Figure 5 is shown in the figure with numbers in the brackets. Specifically, the node marked with $[i]$ in Figure 5 transmits at slot i and listens on all other slots.

The above protocol guarantees that each node can transmit one new packet for every frame. This is the best one can do if the frame size is 5, as a node cannot simultaneously transmit with its neighbors and internal nodes have four neighbors each. Although the goal of this discussion is not to argue whether such a protocol is *provably optimal*, to the best of our knowledge, it is the best-known protocol for such a network.

We can also extend this protocol for cases where the frame size is larger. Specifically, if the frame size is 10, we can just extrapolate this algorithm so that each node transmits twice in each frame. However, if the frame size is between 6..9, it is unclear how the remaining frames could be used to speed up delivery of messages in a systematic fashion where the code of each node remains unchanged across frames. In other words, to the best of our knowledge, GridPr remains the best-known algorithm by allowing the nodes to either sleep or listen in the extra slots in each frame.

6.1.2 Analysis of GP in Broadcast. We repeated the experiments with GP 10 times and the analysis of each run is shown in Figure 6. Specifically, in Figure 6a, we show the 10 runs for the case where frame size is 5. As expected, the process of evolution causes the fitness function to increase. In 3 runs, GP was able to generate a protocol that almost as good as GridPr. In fact, in one of the runs (shown in solid thick blue), it produced the protocol that is identical to GridPr. Furthermore, the red dotted graph shows the median of relative throughputs of best individuals for different generations in 10 GP runs. From this, we find that after 300 generations, the median relative throughput of the best protocol generated by GP was about 85% of that of GridPr.

Figures 6b and 6c show the evolution when frame size is 6 and 7, respectively. Note that no matter what is the size of the frame, we run the experiments for the equal amount of time in terms of the number of slots that we execute. As discussed earlier, even in this case, GridPr remains the best protocol that was known to us. From these figures, we see that GP was able to find a protocol that was slightly better than GridPr. Another important observation from this is that the convergence is substantially faster with frame size of 6 and 7.

6.2 Unicast Results

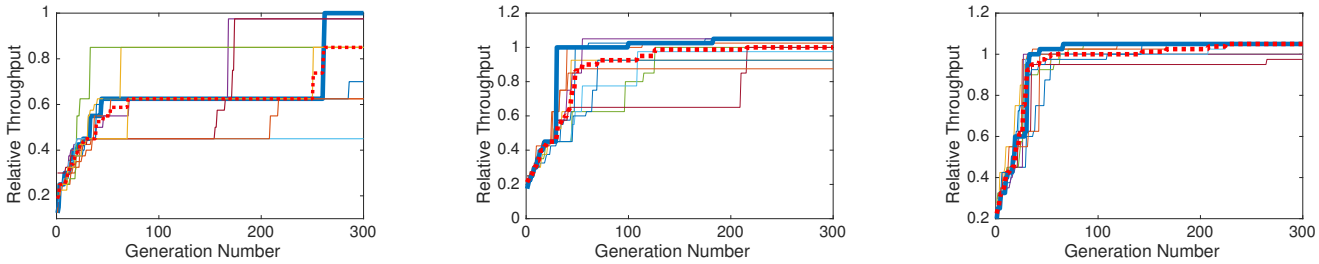
Our first set of experiments focuses on the ability of NetSynth to identify the shortest path based on the objectives. Towards this end, we considered the left subnetwork of the network shown in Figure 4, where the sender is node $B0$ and the destination is node $A0$. The data can be routed to $A0$ either via node $X0$ or via nodes $Y0$ and $Y1$. To facilitate the development of this protocol, each node maintains a routing table. Since $A0$ is the only destination, each node needs only one entry, namely for $A0$. The results for this evolution are in Figure 7a.

Our second set of experiments is to analyze the ability of NetSynth to develop a protocol with lower energy utilization. Towards this end, we considered the right subnetwork of the network shown in Figure 4. Here, node $B1$ has two choices: either route via $X1$ and $X2$ or via $Y2$ and $Y3$. Nodes $X1$ and $X2$ are AC-powered whereas $Y2$ and $Y3$ are battery powered. The results of this experiment are in Figure 7b.

Our third set of experiments is to analyze the ability of NetSynth to identify the tradeoff between energy conservation and throughput. To set up this evaluation, we again considered the right subnetwork of Figure 4. However, we change the interference (i.e. loss) probability of nodes $X1$ and $X2$ to be 0.5. Thus, there is a tradeoff between energy and throughput. Routing via $X1$ and $X2$ saves power whereas routing via $Y2$ and $Y3$ increases throughput. The results from this case are in Figure 7c.

6.2.1 Analysis of GP Results for Unicast. For unicast results, we focus on identifying the tradeoff identified by GP and how the evolution occurred over different generations. In our presentation, we normalize our results with respect to the best protocol found by GP in any run. Specifically, we normalize the throughput with respect to the maximum throughput found in any program, and we normalize the energy with respect to the maximum energy usage in any program considered by GP. Since the goal is to maximize the throughput and minimize the energy, the best solutions are those where throughput is 1 and energy is 0.

The evolution for the left subnetwork of Figure 4 is shown in Figure 7a. From the results in Figure 7a, we find that initially, in the first generation, the energy is really high whereas the throughput is low. In subsequent generations, throughput is maximized very quickly and reaches its maximum value. Energy, however, continues to reduce in subsequent generations; the least energy is obtained (by individual marked with A in Figure 7a) in generation 240 while preserving the maximum throughput. For this individual, GP has successfully found the correct shorter path (via $X0$) to route the packets to node $A0$. As we can see from Figure 7a, NetSynth also finds two other non-dominating solutions; one of these solutions (marked B in Figure 7a) is a trivial solution that lets a node sleep all the time thereby saving energy although the throughput is 0.



(a) Frame size = 5 **(b) Frame size = 6** **(c) Frame size = 7**

Figure 6: Evolving protocol for the network shown in Figure 5. Each diagram shows ten different runs. The thick blue line is one of the runs that founds the best solution, and the dotted red line shows the median of 10 runs.

Another solution (marked C in Figure 7a) has a slightly lower energy than the solution that provides maximum throughput. However, its normalized throughput is only 0.1.

We also review the solutions with an integrated objective where energy and throughput are combined by the following formula[1]

$$weighted\_objective = w \times (1 - energy) + (1 - w) \times throughput$$

By changing the value of w in this formula, we can consider different scenarios where we consider different levels of importance for throughput and energy. We show the results of different weights in Figure 8. Specifically, we consider five different scenarios:

- $E >> T$: energy is significantly more important ($w = 0.9$)
- $E > T$: energy is more important ($w = 0.75$)
- $E = T$: energy and throughput are equally important ($w = 0.5$)
- $E < T$: throughput is more important ($w = 0.25$)
- $E << T$: throughput is significantly more important ($w = 0.1$)

From this figure, we observe that this integrated fitness function also continues to grow across generations. Most of the improvement occurs in the first few iterations and subsequently, there is a slow growth in the future generations.

Results for the second set of experiments is as shown in Figure 7b. Once again, in these experiments, the throughput gets optimized quickly, and energy continues to evolve over subsequent generations. Similar to the first set of experiments, we find non-dominated solutions as shown in Figure 7b. Of these, the solution marked A in Figure 7b transmits the packets via $X1$ and $X2$ thereby saving energy. Thus, for this case also, GP is successful to find the correct path to rout the packets. The results for combined objectives using the same formula for weighted average is shown in Figure 8b.

Our third set of experiments provide interesting results shown in Figure 7c. The results show that in every generation, we see solutions that provide a tradeoff between energy and throughput. For example, in generation 300, we have a solution (marked A in Figure 7c) that provides normalized throughput of 1 with energy 0.524. This solution corresponds to getting a higher throughput at the cost of energy. It routes messages via battery powered nodes (i.e. $Y2$ and $Y3$) with better links. On the other hand, another solution (marked B in Figure 7c) routes messages via AC powered nods (i.e., $X1$ and $X2$). It saves energy (normalized energy 0.45) but reduces throughput (normalized throughput 0.55).

Viewing this in terms of an integrated objective (cf. Figure 8c) solutions found by GP are biased towards scenarios where one objective is far more important than others. When $w = 0.9$ and $w = 0.1$, the solutions provide highest integrated objective.

7 CONCLUSION AND FUTURE WORK

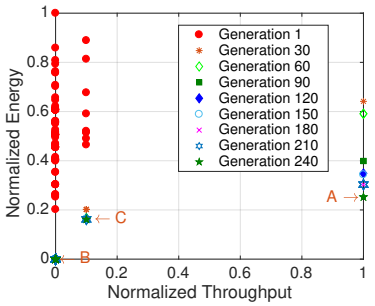
We presented NetSynth, a framework for synthesizing customized network protocols using GP. Our framework is motivated by the need to develop protocols that meet the requirements of the given network in terms of objectives such as energy and throughput while accounting for the differences in the network such as network connectivity, loss rate, availability of AC-power. It would be possible to design a network protocol specifically designed for that network only if the process can be substantially automated to reduce the overhead. NetSynth aims to achieve this with GP.

We demonstrated NetSynth for broadcast and unicast communication. For broadcast on a grid, we demonstrated that NetSynth was able to synthesize the optimized protocol. Moreover, the best median solution found by NetSynth had 85% throughput compared with the best-known protocol. We demonstrated the feasibility of NetSynth to synthesize protocol that provides the shortest path, reduces energy or provides a tradeoff between saving energy and increasing throughput.

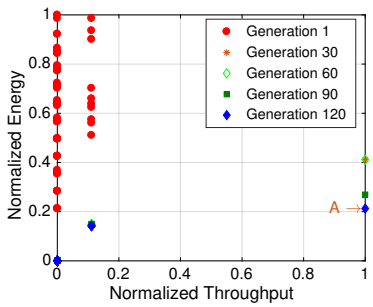
Our analysis was based on an initial description of random protocols. However, in NetSynth, one can set the initial population to consist of well-known protocols. This will allow one to identify best parameters for the given set of protocols or to combine concepts from different protocols. We also provided a graphical tool that lets the network designers graphically describe their networks and requirement to synthesize a protocol for it.

There are several future extensions of NetSynth. For one, currently, we focused on TDMA channel access methods. One can extend NetSynth to synthesize other types of protocols such as CSMA. As another extension, we can consider GP with varying

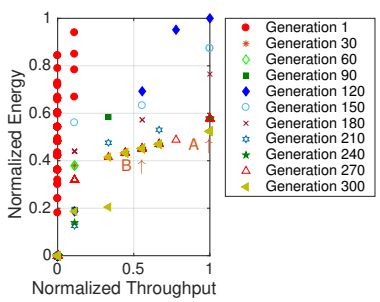
[1]We use 1 - energy since energy is to be minimized



(a) The left subnetwork of Figure 4

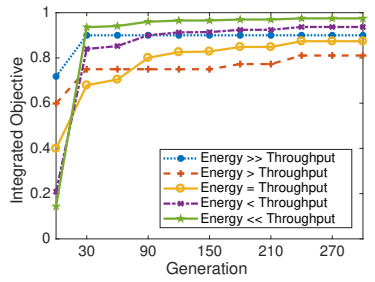


(b) The right subnetwork of Figure 4

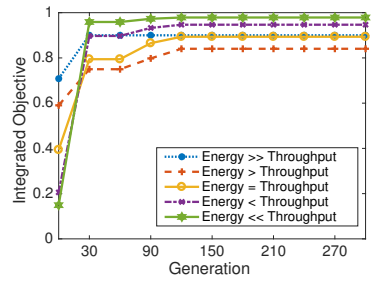


(c) The right subnetwork of Figure 4 when we have interference

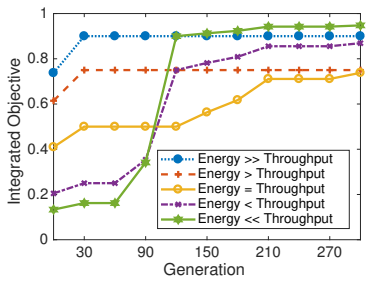
Figure 7: Objectives in different generations



(a) The left subnetwork of Figure 4



(b) The right subnetwork of Figure 4



(c) The right subnetwork of Figure 4 when we have interference

Figure 8: Integrated objective in different generations.

genome size. This lets us evolve the frame size as well as the instruction sequences, unlike the current NetSynth that assumes a fixed genome size given by the input file.

REFERENCES

[1] Alouf, S., Neglia, G., Carreras, I., Miorandi, D., and Fialho, Á. Fitting genetic algorithms to distributed on-line evolution of network protocols. *Computer Networks 54*, 18 (2010), 3402–3420.

[2] Arora, A. "exscal: A perspective on large scale wireless sensor networks". In *7th International Conference on Mobile Data Management (MDM 2006), Nara, Japan, May 9-13, 2006* (2006), IEEE Computer Society, p. 1.

[3] Arora, A., Dutta, P., Bapat, S., Kulathumani, V., Zhang, H., Naik, V., Mittal, V., Cao, H., Demirbas, M., Gouda, M., Choi, Y., Herman, T., Kulkarni, S., Arumugam, U., Nesterenko, M., Vora, A., and Miyashita, M. A line in the sand: a wireless sensor network for target detection, classification, and tracking. *Computer Networks 46*, 5 (2004), 605 – 634. Military Communications Systems and Technologies.

[4] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation 6*, 2 (2002), 182–197.

[5] Ertin, E., Arora, A., Ramnath, R., Naik, V., Bapat, S., Kulathumani, V., Sridharan, M., Zhang, H., Cao, H., and Nesterenko, M. Kansei: a testbed for sensing at scale. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks, IPSN 2006, Nashville, Tennessee, USA, April 19-21, 2006* (2006), J. A. Stankovic, P. B. Gibbons, S. B. Wicker, and J. A. Paradiso, Eds., ACM, pp. 399–406.

[6] Hajiaghajani, F., and Biswas, S. Feasibility of evolutionary design for multi-access mac protocols. In *Global Communications Conference (GLOBECOM), 2015 IEEE* (2015), IEEE, pp. 1–7.

[7] Hajiaghajani, F., and Biswas, S. Mac protocol design using evolvable state-machines. In *Computer Communication and Networks (ICCCN), 2015 24th International Conference on* (2015), IEEE, pp. 1–6.

[8] Hu, Y.-C., Perrig, A., and Johnson, D. B. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wirel. Netw. 11*, 1-2 (Jan. 2005), 21–38.

[9] Hui, J. W., and Culler, D. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems* (New York, NY, USA, 2004), SenSys '04, ACM, pp. 81–94.

[10] Johnson, D. B., and Maltz, D. A. *Dynamic Source Routing in Ad Hoc Wireless Networks.* Springer US, Boston, MA, 1996, pp. 153–181.

[11] Kulkarni, S. S., and Arumugam, M. Infuse: A TDMA based data dissemination protocol for sensor networks. *IJDSN 2*, 1 (2006), 55–78.

[12] Kulkarni, S. S., and Wang, L. Energy-efficient multihop reprogramming for sensor networks. *TOSN 5*, 2 (2009), 16:1–16:40.

[13] Lewis, T., Fanning, N., and Clemo, G. Enhancing ieee802. 11 dcf using genetic programming. In *Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd* (2006), vol. 3, IEEE, pp. 1261–1265.

[14] Royer, E. M., and Perkins, C. E. Multicast operation of the ad-hoc on-demand distance vector routing protocol. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking* (New York, NY, USA, 1999), MobiCom '99, ACM, pp. 207–218.

[15] Ye, W., Heidemann, J. S., and Estrin, D. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA, June 23-27, 2002* (2002), IEEE.