Genetic improvement of TCP congestion avoidance

Alberto Carbognin[0000−0001−6987−0547],
Leonardo Lucio Custode[0000−0002−1652−1690], and
Giovanni Iacca[0000−0001−9723−1830]

University of Trento
Department of Information Engineering and Computer Science
Via Sommarive 9, 38123 Povo (TN), Italy
alberto.carbognin@studenti.unitn.it
{leonardo.custode,giovanni.iacca}@unitn.it

Abstract. The Transmission Control Protocol (TCP) protocol, i.e., one of the most used protocols over networks, has a crucial role on the functioning of the Internet. Its performance heavily relies on the management of the congestion window, which regulates the amount of packets that can be transmitted on the network. In this paper, we employ Genetic Programming (GP) for evolving novel congestion policies, encoded as C++ programs. We optimize the function that manages the size of the congestion window in a point-to-point WiFi scenario, by using the NS3 simulator. The results show that, in the protocols discovered by GP, the Additive-Increase-Multiplicative-Decrease principle is exploited differently than in traditional protocols, by using a more aggressive window increasing policy. More importantly, the evolved protocols show an improvement of the throughput of the network of about 5%.

Keywords: Genetic Programming · NS3 · TCP · Network protocols

1 Introduction

In the era of the Internet of Things (IoT), networked systems have become a crucial part of our everyday lives. Network protocols, which describe the interactions that can occur in a networked system, are traditionally modeled by means of automata, which require: a) complete knowledge about the environment, and b) strict assumptions on the interactions that can occur. In this scenario, several works proposed formal methods that, given a set of service specifications, perform automatic synthesis of the network protocols [1–4].

As an alternative to this approach, bio-inspired techniques can be used to evolve network protocols by simulating their behavior, i.e., without any need for formalizing all the protocol requirements. So, even though the computational budget required these approaches is higher than the one needed for formal methods, they have the advantage that there is no need for a complete knowledge of the environment. Thus, bio-inspired techniques allow to evolve protocols for scenarios that are hard to model analytically. Moreover, protocols discovered by means of bio-inspired approaches allow to perform continual learning and adaptation, which allows for: a) an improvement of the performance of the protocol over time; and b) adaptation to changing domains.

Among the various protocols at the bases of modern Internet, of the most important ones is the Transmission Control Protocol (TCP). The key element of TCP is the so-called congestion avoidance mechanism, which makes use of a congestion window to avoid overloading the link between the sender and the receiver. The size of the congestion window is traditionally managed by means of an Additive-Increase-Multiplicative-Decrease approach. However, it may be possible to adopt alternative, automatically generated congestion avoidance mechanisms.

In this paper, we apply Genetic Programming (GP) [5] for the automatic synthesis of a congestion window management protocol. We employ the NS3 simulator [6] to evaluate the effectiveness of the protocols evolved in a point-to-point WiFi scenario. In our numerical experiments, we observe that the evolved protocols are able to obtain approximately a 5% improvement in performance with respect to the corresponding baseline protocols.

The rest of the paper is structured as follows. In the next section, we present the background concepts on TCP. Then we make a brief overview of the related works in Section 3. In Section 4, we describe our methods. In Section 5, we present our experimental setup and numerical results. Finally, in Section 6 we conclude this work.

2 Background

The TCP is one of the most used communication protocols together, with the User Datagram Protocol (UDP) in the Transport Layer of the Internet Protocol Suite. TCP is well-known for its reliability rather than speed performance, indeed it is able to detect the loss of data packets, request missing segments, and guarantee that all the information is transmitted and delivered to the receiver. This behavior, however, reduces the available bandwidth; in fact, a potential issue that may arise by applying these reliability features is the congestion of the network. Besides the protocol implementation, network congestion can be caused by many factors, the most common being the low amount of bandwidth available from the channel and a not properly designed network infrastructure. TCP has the duty of preventing and mitigating network congestion by using ad-hoc strategies.

In order to achieve a stable and reliable connection between two hosts, it is required that the transmission is somewhat controlled at both ends. For instance, propagation delays due to the network infrastructure could affect negatively the overall throughput. Congestion control algorithms have been developed to avoid and recover from this kind of network degradation.

A congested network can quickly result in very low performance. Traditional congestion control algorithms can be divided into two categories: end-to-end and network assisted. While in the former only information about the sender and the receiver is needed, in the latter metrics regarding the network infrastructure are used to take decisions [7].

The challenge, for end-to-end algorithms, is to use implicit signals to infer the congestion status of the network. For instance, for packet loss-based approaches, the objective is to increase throughput by exploiting the bandwidth. In general, if the sender does not receive back the acknowledgment from the receiver after a certain amount of time, the sender may "infer" that the packet is lost. On the other hand, delay-based approaches are better suited for networks that need high speed and flexibility [7], but also in this case calculating the exact transmission delay is tricky; other paths have been researched and some hybrid algorithms have been proposed such as [8].

3 Related works

Networks have now evolved into very complex systems, where one specific solution may be suitable for one network but ineffective in another one. For this reason, research has focused in solutions that make use of various Artificial Intelligence algorithms, including Evolutionary Computation and Machine Learning, to improve flexibility and performance of protocols. We briefly discuss some of these works below.

Two rather comprehensive surveys on the application of bio-inspired techniques to the evolution of network protocols can be found in [9, 10]. Most of the existing approaches focus on offline optimization. For instance, in [11], the authors employ the Particle Swarm Optimization algorithm for the routing path selection. In [12], the authors propose the ant routing method for optimizing routing models. In [13], the authors propose for the first time an EA to evolve protocols. After this work, several papers have tried to use an EA to evolve a variety of network protocols: MAC access protocols [14, 15], e.g. through Finite State Machines (FSMs) [16–18]; wireless protocols [19]; aggregation protocols [20–22]; and protocol adaptors [23].

Another line of work consists in using distributed EAs (including GP) to evolve some elements of the network, e.g. through distributed GP [24, 25] to evolve the nodes' parameters and functioning logics of WSNs, or through distributed optimization algorithms, including EAs and single-solution algorithms, such as simulated annealing, as in [26], and other optimization paradigms [27–29].

Finally, online learning approaches have been proposed, which allow the network elements to reconfigure at runtime. Su and Van Der Schaar, in [30], propose a learning approach in which each node, by observing the others' behavior, tries to predict the other nodes' reaction to its actions. STEM-Net [31] is a method that equips each node with an EA, that allows to reconfigure each layer of the node, depending on the current state. In [32], the authors propose an approach where protocols are formed as a combination of "fraglets". This concept is similar to those presented in [33, 34]. Another recent work on online optimization over networks has been presented in [35].

Other works have applied Machine Learning to predict congestion signals by available data. For instance, the Biaz algorithm is able to distinguish a wireless loss from a congestion loss [36]. Another algorithm is ZigZag [37], which is able to work with different networks infrastructures. The key advantage over common congestion control algorithms for wireless networks is the ability to take into consideration multiple parameters. In [38] a Bayesian detector has been developed and implemented by modifying the TCP New Reno algorithm; the experimental results reported that the model was able to infer the distribution of the round-trip time degradation caused by packet reordering and congestion. A critical point of these models is the difficulty in finding a suitable trade-off between network performance improvements and network resources consumption.

4 Method

In this work, we employ Genetic Programming (GP) [5] to evolve congestion control policies in the form of C++ programs. The function set is shown in Table 1, while the terminal set is shown in Table 2. The parameters used for the GP algorithm are shown in Table 3.

Note that, besides the selection, crossover, and mutation operators, another evolutionary operator is introduced: the Stagnation-Driven Extinction Protocol (SDEP) [39], which controls the extinction of the individuals in the evolutionary process. It makes use of the following hyperparameters:

- p_{sdep}: the extinction probability
- t_{sdep}: threshold used to control the individuals affected by extinction
- k_{sdep}: the number of stagnating generations that, once reached, triggers the operator

In this work, we employ a modified version of the Targeted extinction approach proposed in [39], where p_{sdep} is modified over time:

$$p^k_{sdep} = p^0_{sdep} + f_{sdep}(k) \tag{1}$$

where $f_{sdep}(k)$ is defined as:

$$f_{sdep}(k) = \frac{10\sqrt{k}}{1 + e^{-k}} \tag{2}$$

Moreover, instead of sorting the individuals by fitness, as done in [39], we employ a threshold-based approach to control the individuals affected by extinction. This allows us to reduce the computational complexity of the extinction protocol to a linear complexity. For this purpose, we make use of a threshold computed as follows:

$$\tau = F_{elite}(1 - t_{sdep}) \tag{3}$$

where F_{elite} is the fitness of the elite individual, and all the individuals that have fitness below τ are affected by extinction.

Table 1. Non-terminals used, their corresponding C++ code, argument types and return types.

Non-terminal	C++ code	Argument types	Return type
`assignment`	`arg1 = arg2;`	*variable*, *exp*	body
`IfThenElse`	`if ( arg1 ){ arg2 };`	*condition*, *body*	*body*
`lt`	`( arg1 < arg2 )`	*condition*, *condition*	*condition*
`lte`	`( arg1 <= arg2 )`	*condition*, *condition*	*condition*
`gt`	`( arg1 > arg2 )`	*condition*, *condition*	*condition*
`gte`	`( arg1 >= arg2 )`	*condition*, *condition*	*condition*
`eq`	`( arg1 == arg2 )`	*condition*, *condition*	*condition*
`neq`	`( arg1 != arg2 )`	*condition*, *condition*	*condition*
`expression`	`arg1`	*body*	*body*
`mul`	`arg_1, ..., arg_n`	*body*	*body*
`sum`	`arg_1, ..., arg_n`	*body*	*body*
`sub`	`arg_1, ..., arg_n`	*body*	*body*
`div`	`arg_1, ..., arg_n`	*body*	*body*
`ReduceCwnd`	`ReduceCwnd(arg_1)`	*body*	*body*
`CongestionAvoidance`	`TcpLinuxCongestionAvoidance(arg_1, arg_2)`	*body*	*body*

Table 2. Terminals used, their corresponding C++ code and type.

Terminal	C++ code	Type
`cnt`	`arg1`	*body*
`segmentsAcked`	`arg1`	*body*
`tcb->m_cWnd`	`arg1`	*body*
`tcb->m_segmentSize`	`arg1`	*body*
`tcb->m_ssThresh`	`arg1`	*body*

Table 3. Parameter setting (Koza-style tableau) of the Genetic Programming algorithm.

Parameter	Value
Objective	Throughput
Function set	See Table 1
Terminal set	See Table 2
Population size	30
Number of generations	50
Max lines of code	100
Mutation	Operator flip, prob: 0.6 Switch branches, prob: 0.3 Switch expression, prob: 0.7 Truncate node, prob: 0.25 Max mutations: 10 mutations

4.1 Code simplification procedure

To simplify the evolved trees, we created a procedure that parses the rendered code and generates a more compact version of it. The procedure performs multiple tasks that can be summarized as follows:

– remove empty lines: loops through the code lines and removes the empty one after applying the function .strip;

- gathering variable names: loop through code lines and detects the declaration of variable int and float[1];
- remove if unused: creates an empty list of used variables, loops through the code lines to check if they are used in expression or `IfThenElse` condition blocks, delete the variables that are not inside the list of the used ones;
- clean empty "IfThenElse": loop through code lines and removes branches that are empty;
- simplify expression: loops through the code lines and detects the expression, if they only contains constant values they are simplified;
- compressing to "for" loop: loops through the code lines and detects the equal code lines, it then compress them inside a for loop.

5 Experimental results

To evaluate our method, we employ the NS3 simulator [40, 41]. The Network topology used in our experiments consists of two hosts connected through WiFi with an application data rate of 100 Mbps, a payload size of 1500 bytes and simulation time of 5 seconds. The position of the hosts is assumed to be fixed during the network simulation. The code we used for our experiments is available at https://carbogninalberto.github.io/pyGENP/.

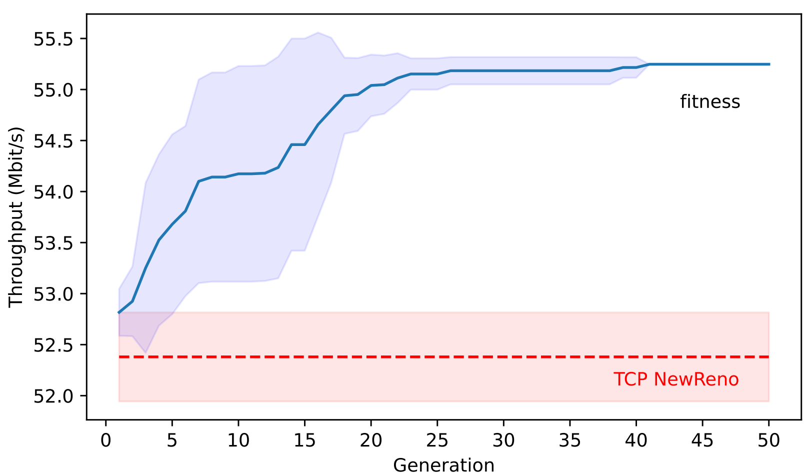

Fig. 1. Fitness trend (mean ± std. dev. across 10 runs) of the protocols evolved from TCP New Reno (blue) vs. the baseline throughput of TCP New Reno (red).

[1] The variables inside the expression are not detected

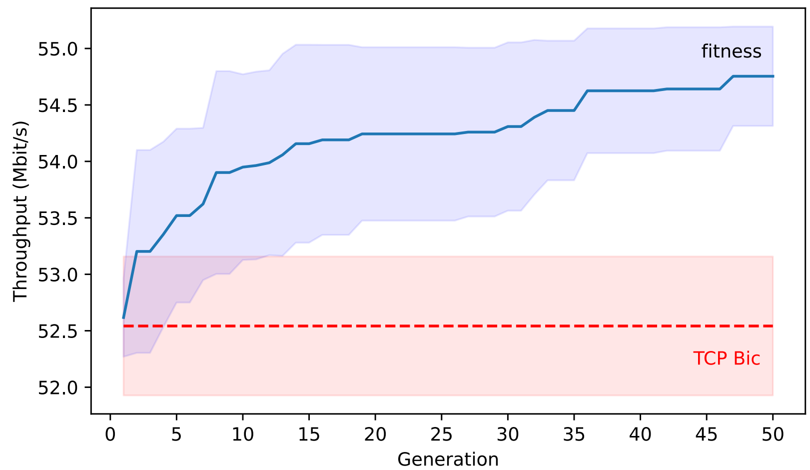

Fig. 2. Fitness trend (mean ± std. dev. across 10 runs) of the protocols evolved from TCP Bic (blue) vs. the baseline throughput of TCP Bic (red).

Table 4. Throughput of the best evolved TCP protocols in comparison with different congestion control algorithms from the literature (average across 10 runs), for different values of payload size.

Algorithm	**Payload size (bytes)**				
	250	1500	3000	7500	15000
TcpNewReno (ours)	14.05 ± 0.02	55.24 ± 0.04	53.73 ± 0.49	54.45 ± 0.50	55.30 ± 0.51
TcpNewReno	35.17 ± 0.26	52.38 ± 0.44	53.44 ± 0.46	54.70 ± 0.41	55.20 ± 0.41
TcpBic (ours)	35.44 ± 0.27	54.57 ± 0.02	55.39 ± 0.02	56.58 ± 0.02	57.74 ± 0.03
TcpBic	35.36 ± 0.25	52.54 ± 0.62	53.39 ± 0.54	54.86 ± 0.54	55.25 ± 0.56
TcpHybla	35.28 ± 0.21	52.58 ± 0.97	53.65 ± 0.53	54.65 ± 0.24	55.08 ± 0.30
TcpHighSpeed	35.30 ± 0.16	52.71 ± 0.60	53.45 ± 0.35	54.99 ± 0.67	55.02 ± 0.32
TcpHtcp	35.06 ± 0.26	52.53 ± 0.46	53.67 ± 0.61	55.13 ± 0.52	55.43 ± 0.43
TcpVegas	30.49 ± 3.06	53.89 ± 1.50	55.68 ± 0.17	55.96 ± 0.68	55.36 ± 0.62
TcpScalable	35.22 ± 0.34	52.27 ± 0.46	53.71 ± 0.66	54.69 ± 0.54	55.21 ± 0.45
TcpVeno	35.37 ± 0.24	52.53 ± 0.54	53.68 ± 0.38	54.67 ± 0.66	55.14 ± 0.46
TcpYeah	35.47 ± 0.23	52.58 ± 0.31	53.39 ± 0.82	54.72 ± 0.32	54.88 ± 0.39
TcpIllinois	35.16 ± 0.13	52.50 ± 0.48	53.79 ± 0.77	54.61 ± 0.47	55.08 ± 0.22
TcpWestwood	35.12 ± 0.27	52.77 ± 0.33	53.70 ± 0.50	54.74 ± 0.37	55.18 ± 0.52
TcpWestwoodPlus	35.17 ± 0.18	52.59 ± 0.38	53.78 ± 0.68	54.82 ± 0.37	55.30 ± 0.37
TcpLedbat	35.13 ± 0.16	52.45 ± 0.46	53.81 ± 0.48	54.78 ± 0.50	55.15 ± 0.46

First of all, from Figures 1 and 2 we can see that, for both TCP New Reno and Bic, the evolutionary process quickly outperforms the corresponding baseline values of throughput. On average, we can see that the evolved protocols achieve a 5% improvement on the baseline value of throughput in 50 generations. Moreover, we observe that the evolutionary process seems to stabilize faster (and more robustly across runs) in the case of TCP New Reno with respect to the case of TCP Bic.

In Table 4, we report the performance metrics for each algorithm available in the NS3 simulator. While our protocols have been evolved on a payload of 1500 bytes, we test them with different payload sizes, to understand whether the resulting protocols are biased towards the payload size used for the evolutionary process. We set as maximum payload size 15000 bytes, which is approximately 1/4 of the maximum theoretical payload size allowed by TCP (65535 bytes). By analyzing the results in the table, it seems reasonable to say that the evolved TCP New Reno protocol appears biased on the payload size used during the evolutionary process (1500 bytes); indeed, it performs comparably or worse than the original TCP New Reno for all the other payload sizes. On the other hand, while the evolved TCP Bic has less performance gain with respect to the original TCP Bic protocol, on average it performs better for all the payload sizes above 1500 bytes. Of note, the throughput reached by the evolved TCP Bic with a payload size of 15000 bytes is the highest among all the other compared congestion control protocols.

Listing 1.1 reports the code of one of the best evolved individuals obtained in the case of TCP New Reno; the solution sets the `segmentsAcked` variable to a fixed value of 175. It then calls the `ReduceCwnd` function that is updating the CWND as $CWND = max(\frac{CWND}{2}, segmentSize)$ and then calls the "TcpLinuxCongestionAvoidance" function. The interesting part of this protocol is the fact that this solution always sets the *segmentAcked* variable to a fixed value, thus removing the loss feedback that should be used by the TCP New Reno to signal possible congestion of the network. Moreover, it always reduces the congestion window before executing the congestion avoidance. The logic of this last code is to increase the congestion window by the segment size if the congestion window counter is equal or bigger than the number of segment sizes contained in the CWND, the variable w. Moreover, it always updates the counter by the `segmentsAcked` which is a static value. Then, it further updates the CWND if the congestion window counter is bigger than the variable w. Further investigations must be done to understand if in this case the static `segmentsAcked` is behaving if the network is congested; in the evolution environment, the simple network packets are lost according to the Friis propagation loss model [42, 43]. The algorithm may have exploited some specific properties of the simulated scenario; for this reason, future work should also include an analysis of the packet loss rates.

The solutions obtained were also very different from each other across runs. For instance, the one reported in Listing 1.2 shows a more complex logic even though, in terms of throughput, it achieves the same result as the one showed in Listing 1.1. This might indicate that the metric used to optimized the protocol may not be able to correctly discriminate solutions of different complexities.

In Listing 1.3, we report one of the best solutions obtained in the case of the TCP Bic protocol; the evolved logic in this case is a bit more complicated than the ones found in the case of TCP New Reno.

```
segmentsAcked = (int)175.271;
ReduceCwnd(tcb);
TcpLinuxCongestionAvoidance(tcb, segmentsAcked);
```

Listing 1.1. Best evolved individual for **TCP New Reno** with payload size 1500 bytes.

6 Conclusions and future work

Networks have become ubiquitous in our everyday lives. To increase the efficiency of such networks, it is crucial to efficiently manage the size of the TCP congestion window depending on the scenario at hand. In this paper, we propose a bio-inspired approach to the optimization of congestion control algorithms for a point-to-point WiFi scenario. As shown in Section 5, we were able to evolve protocols that increase the performance up to about 5% with respect to the baseline protocols from the literature. This result indicates that the proposed approach is a promising alternative for optimal protocol design.

Future work may focus, among the other things, on: the modification of the fitness evaluation process, to take into account different payload sizes; the evolution of protocols that are able to work well with different packet loss models; the extension of the function set used in GP, in order to include loops and operators with arity greater than 2; the study of the GP parameter effect on the resulting protocols, e.g. through the irace [44] or the ParamILS [45] packages.

References

1. Saleh, Kassem and Probert, Robert: Automatic synthesis of protocol specifications from service specifications. In: International Phoenix Conference on Computers and Communications, New York, NY, USA, IEEE (1991) 615–621
2. Probert, Robert L. and Saleh, Kassem: Synthesis of communication protocols: survey and assessment. Transactions on Computers **40**(4) (1991) 468–476
3. Carchiolo, Vincenza and Faro, Alberto and Giordano, Daniela: Formal description techniques and automated protocol synthesis. Information and Software Technology **34**(8) (1992) 513–521
4. Saleh, Kassem: Synthesis of communications protocols: an annotated bibliography. SIGCOMM Computer Communication Review **26**(5) (1996) 40–59
5. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. Complex adaptive systems. MIT Press, Cambridge, Mass (1992)
6. Riley, G.F., Henderson, T.R.: The ns-3 network simulator. In: Modeling and tools for network simulation. Springer (2010) 15–34
7. Jiang, H., Li, Q., Jiang, Y., Shen, G., Sinnott, R., Tian, C., Xu, M.: When Machine Learning Meets Congestion Control: A Survey and Comparison. arXiv:2010.11397 [cs] (October 2020) arXiv: 2010.11397.
8. Tan, K., Song, J., Zhang, Q., Sridharan, M.: A Compound TCP Approach for High-Speed and Long Distance Networks. In: Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications. (April 2006) 1–12 ISSN: 0743-166X.

9. Nakano, Tadashi: Biologically inspired network systems: A review and future prospects. Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **41**(5) (2010) 630–643
10. Dressler, Falko and Akan, Ozgur B: A survey on bio-inspired networking. Computer Networks **54**(6) (2010) 881–900
11. Guo, Kai and Lv, Yang: Optimizing Routing Path Selection Method Particle Swarm Optimization. International Journal of Pattern Recognition and Artificial Intelligence **34**(12) (2020) 2059042
12. Zhang, Ximing and Li, Jin and Qiu, Rongfu and Mean, Tian-Shine and Jin, Fanzhu: Optimized Routing Model of Sensor Nodes in Internet of Things Network. Sensors and Materials **32**(8) (2020) 2801–2811
13. Khaled El-fakih and Hirozumi Yamaguchi and Gregor Bochmann: A Method and a Genetic Algorithm for Deriving Protocols for Distributed Applications with Minimum Communication Cost. In: International Conference on Parallel and Distributed Computing and Systems, Calgary, AB, Canada, IASTED (1999) 1–6
14. Lewis, Tim and Fanning, Neil and Clemo, Gary: Enhancing IEEE802.11 DCF using genetic programming. In: Vehicular Technology Conference. Volume 3., New York, NY, USA, IEEE (2006) 1261–1265
15. Roohitavaf, Mohammad and Zhu, Ling and Kulkarni, Sandeep and Biswas, Subir: Synthesizing customized network protocols using genetic programming. In: Genetic and Evolutionary Computation Conference Companion, New York, NY, USA, ACM (2018) 1616–1623
16. Sharples, Nicholas and Wakeman, Ian: Protocol construction using genetic search techniques. In: Workshops on Real-World Applications of Evolutionary Computation, Berlin, Heidelberg, Springer (2000) 235–246
17. Hajiaghajani, Faezeh and Biswas, Subir: Feasibility of Evolutionary Design for Multi-Access MAC Protocols. In: Global Communications Conference, New York, NY, USA, IEEE (2015) 1–7
18. Hajiaghajani, Faezeh and Biswas, Subir: MAC protocol design using evolvable state-machines. In: International Conference on Computer Communication and Networks, New York, NY, USA, IEEE (2015) 1–6
19. Tekken-Valapil, Vidhya and Kulkarni, Sandeep S: Derivation of Network Reprogramming Protocol with Z3 (2017)
20. Weise, Thomas and Geihs, Kurt and Baer, Philipp A: Genetic programming for proactive aggregation protocols. In: International Conference on Adaptive and Natural Computing Algorithms, Berlin, Heidelberg, Springer (2007) 167–173
21. Weise, Thomas and Zapf, Michael and Geihs, Kurt: Evolving proactive aggregation protocols. In: European Conference on Genetic Programming, Berlin, Heidelberg, Springer (2008) 254–265
22. Weise, Thomas and Tang, Ke: Evolving distributed algorithms with genetic programming. Transactions on Evolutionary Computation **16**(2) (2011) 242–265
23. Van Belle, Werner and Mens, Tom and D'Hondt, Theo: Using genetic programming to generate protocol adaptors for interprocess communication. In: International Conference on Evolvable Systems, Berlin, Heidelberg, Springer (2003) 422–433
24. Johnson, Derek M and Teredesai, Ankur M and Saltarelli, Robert T: Genetic programming in wireless sensor networks. In: European Conference on Genetic Programming, Berlin, Heidelberg, Springer (2005) 96–107
25. Valencia, Philip and Lindsay, Peter and Jurdak, Raja: Distributed genetic evolution in WSN. In: International Conference on Information Processing in Sensor Networks, New York, NY, USA, ACM/IEEE (2010) 13–23

26. Iacca, Giovanni: Distributed optimization in wireless sensor networks: an island-model framework. Soft Computing **17**(12) (2013) 2257–2277
27. Wang, S., Li, C.: Distributed robust optimization in networked system. IEEE Transactions on Cybernetics **47**(8) (2017) 2321–2333
28. Ning, B., Han, Q., Zuo, Z.: Distributed optimization of multiagent systems with preserved network connectivity. IEEE Transactions on Cybernetics **49**(11) (2019) 3980–3990
29. Wang, D., Yin, J., Wang, W.: Distributed randomized gradient-free optimization protocol of multiagent systems over weight-unbalanced digraphs. IEEE Transactions on Cybernetics **51**(1) (2021) 473–482
30. Su, Yi and Van Der Schaar, Mihaela: Dynamic conjectures in random access networks using bio-inspired learning. Journal on Selected Areas in Communications **28**(4) (2010) 587–601
31. Aloi, Gianluca and Bedogni, Luca and Felice, Marco Di and Loscri, Valeria and Molinaro, Antonella and Natalizio, Enrico and Pace, Pasquale and Ruggeri, Giuseppe and Trotta, Angelo and Zema, Nicola Roberto: STEM-Net: an evolutionary network architecture for smart and sustainable cities. Transactions on Emerging Telecommunications Technologies **25**(1) (2014) 21–40
32. Yamamoto, Lidia and Schreckling, Daniel and Meyer, Thomas: Self-replicating and self-modifying programs in fraglets. In: Workshop on Bio-Inspired Models of Network, Information and Computing Systems, New York, NY, USA, IEEE (2007) 159–167
33. Tschudin, Chr and Yamamoto, Lidia: Self-evolving network software. Praxis der Informationsverarbeitung und Kommunikation **28**(4) (2005) 206–210
34. Miorandi, Daniele and Yamamoto, Lidia: Evolutionary and embryogenic approaches to autonomic systems. In: International Conference on Performance Evaluation Methodologies and Tools, New York, NY, USA, ACM (2008) 1–12
35. Yaman, A., Iacca, G.: Distributed embodied evolution over networks. Applied Soft Computing **101** (2021) 106993
36. Biaz, S., Vaidya, N.: Discriminating congestion losses from wireless losses using inter-arrival times at the receiver. In: Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No.PR00122). (March 1999) 10–17
37. Cen, S., Cosman, P.C., Voelker, G.M.: End-to-end differentiation of congestion and wireless losses. IEEE/ACM Trans. Netw. **11**(5) (2003) 703–717
38. I'onseca, N., Crovella, M.: Bayesian packet loss detection for TCP. In: Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Volume 3., Miami, FL, USA, IEEE (2005) 1826–1837
39. Ye, G.Z., Kang, D.K.: Extended Evolutionary Algorithms with Stagnation-Based Extinction Protocol. Applied Sciences **11**(8) (April 2021) 3461
40. Kurkowski, S., Camp, T., Colagrosso, M.: Manet simulation studies: the incredibles. SIGMOBILE Mob. Comput. Commun. Rev. **9**(4) (2005) 50–61
41. Stojmenovic, I.: Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions. IEEE Communications Magazine **46**(12) (2008) 102–107
42. Friis, H.T.: A note on a simple transmission formula. Proceedings of the IRE **34**(5) (1946) 254–256
43. Stoffers, M., Riley, G.: Comparing the ns-3 propagation models. In: 2012 IEEE 20th international symposium on modeling, analysis and simulation of computer and telecommunication systems, IEEE (2012) 61–67

44. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3** (2016) 43–58
45. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36** (2009) 267–306

```
if (tcb->m_segmentSize > tcb->m_ssThresh) {
    tcb->m_ssThresh = (int)1.0;
    ReduceCwnd(tcb);
} else {
    tcb->m_ssThresh = (int)35.63;
    tcb->m_ssThresh = (int)2706.002;
}
tcb->m_segmentSize = (int)(85.733 - (56.436) - (tcb->m_ssThresh) - (92.142) - (70.956) - (6.654));
float hnPsxuBCtPVMMYBm = (float)(36.2 - (8.073) - (16.935) - (78.417) - (21.996) - (tcb->m_segmentSize));
if (tcb->m_segmentSize >= segmentsAcked) {
    segmentsAcked = (int)(hnPsxuBCtPVMMYBm * (68.195) * (tcb->m_cWnd) * (tcb->m_ssThresh) * (9.219) * (96.226) * (85.611) * (31.971) * (18.886));
    ReduceCwnd(tcb);
} else {
    segmentsAcked = (int)(72.07 - (tcb->m_ssThresh) - (segmentsAcked));
}
TcpLinuxCongestionAvoidance(tcb, segmentsAcked);
if (tcb->m_cWnd <= hnPsxuBCtPVMMYBm) {
    hnPsxuBCtPVMMYBm = (float)(92.554 - (74.251) - (81.969) - (27.667) - (segmentsAcked) - (54.344) - (64.616) - (12.799));
    segmentsAcked = SlowStart(tcb, segmentsAcked);
    tcb->m_cWnd = (int)(41.965 + (69.396) + (26.746) + (30.182) + (tcb->m_ssThresh) + (12.114) + (tcb->m_ssThresh));
} else {
    hnPsxuBCtPVMMYBm = (float)(tcb->m_segmentSize * 3643.109);
    tcb->m_cWnd = (int)-34.453;
}

for (int i = 0; i < 2; i++) {
    TcpLinuxCongestionAvoidance(tcb, segmentsAcked);
}
```

Listing 1.2. Another best evolved individual for **TCP New Reno** with payload size 1500 bytes.

```
if (cnt != segmentsAcked) {
    tcb->m_segmentSize = (int)(13.704 * (53.117) *
    (74.527) * (segmentsAcked) * (61.69) * (17.898));
    if (m_cWndCnt > cnt) {
        tcb->m_cWnd += tcb->m_segmentSize;
        m_cWndCnt = 0;
    }
} else {
    tcb->m_segmentSize = (int)-352.836;
    tcb->m_ssThresh = (int)1.133;
}
int MkycqeZLOKenojJc = (int)1.549;
cnt = (int)(94.326 * (89.844) * (7.283) * (47.081) * (
    tcb->m_ssThresh) * (94.293) * (segmentsAcked) *
    (72.366) * (25.407));
ReduceCwnd(tcb);
if (tcb->m_ssThresh > cnt) {
    tcb->m_cWnd = (int)(segmentsAcked + (68.779) +
    (83.102) + (85.846) + (cnt) + (9.069));
    if (m_cWndCnt > cnt) {
        tcb->m_cWnd += tcb->m_segmentSize;
        m_cWndCnt = 0;
    }
    MkycqeZLOKenojJc = (int)(23.82 - (79.771) -
    (14.523) - (27.086) - (65.009) - (0.513) - (49.232)
     - (tcb->m_ssThresh));
} else {
    tcb->m_cWnd = (int)(52.023 - (70.074) - (19.636) -
    (tcb->m_ssThresh) - (47.417) - (55.579) - (
    MkycqeZLOKenojJc));
}
ReduceCwnd(tcb);
```

Listing 1.3. Best evolved individual for **TCP Bic** with payload size 1500 bytes.