Application of genetic algorithm to load balancing in networks with a homogeneous traffic flow

Marek Bolanowski[1][0000−0003−4645−967X], Alicja Gerka[2][0000−0001−7406−8495], Andrzej Paszkiewicz[1][0000−0001−7573−3856], Maria Ganzha[3][0000−0001−7714−4844], and Marcin Paprzycki[3][0000−0002−8069−2152]

[1] Rzeszów University of Technology, Rzeszów, Poland
{marekb, andrzejp}@prz.edu.pl
[2] alicja.gerka@op.pl
[3] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
{maria.ganzha, paprzyck}@ibspan.waw.pl

Abstract. The concept of extended cloud requires efficient network infrastructure to support ecosystems reaching form the edge to the cloud(s). Standard network load balancing delivers static solutions that are insufficient for the extended clouds, where network loads change often. To address this issue, a genetic algorithm based load optimizer is proposed and implemented. Next, its performance is experimentally evaluated and it is shown that it outperforms other existing solutions.

Keywords: extended cloud · computer network · load balancing · SDN · routing · IoT · self-adapting networks.

1 Introduction

Today, typical "data processing systems" consist of "distributed data sources" and (a) "central cloud(s)". However, the advent of Internet of Things forces changes, since the "decision loop", from the "sensor(s)" to the cloud and back to the "actuator(s)", may take too long for (near-)real time applications. This problem materializes in, so called, *Extended Cloud* (EC). The EC encompasses highly heterogeneous hardware, which is very often managed using Software Defined Networking (SDN) architecture. One of the keys to effective management of SDN network is load balancing achieved, inter alia, by elimination of overloaded links, through dynamic adaptation of the routing policy. This requires an "arbitrator", which collects information about communication requests and dynamically manages routing [14]. This, in turn, allows balancing loads in the aggregation layer, connecting EC elements, including IoT and control devices [17].

Separately, note that classic load balancing algorithms, when applied to the SDN-based networks, are NP-hard [20]. Therefore, a genetic algorithm-based approach to SDN network load balancing (the SDNGALB algorithm), is being proposed [10, 5, 13, 12]. Note that optimization of weights, in MPLS and OSPF networks, is a separate research area. However, it is directly related to the topic

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32

discussed here and it is also NP-hard [7, 9]. Many works proposed heuristic algorithms to optimize the link weights in the OSFP protocol, to minimize the maximum load of the links [16, 15]. Overall, based on a comprehensive analysis of works related to use of standard, and metaheuristics-based, approaches to network load balancing, it can be stated that: (I) found solutions are focused mainly on use of static values of weights for communication links. This results in, temporarily optimal, but static, connection structure, and leaves an open research gap. (II) Special attention should be paid to the possibility of applying the developed solutions in the environment consisting of real network devices. To address found limitations, and to deliver solution applicable to real-world ECs, a genetic algorithm, with high implementation potential, is proposed.

2 Problem formulation and proposed solution

In what follows, computer network will be represented by a directed graph $G(N, E)$, where N is a set of nodes, representing network devices, and E is a set of edges representing network links. Each edge $e_{ij} \in E$ is assigned a weight w_{ij}, the modification of which will affect the current shaping of the routing policy. Moreover, the following assumptions have been made: (1) Communication channels, represented as $e_{ij} \in E$, have the same bandwidth; (2) $G\,(N, E)$ is a directed graph (capturing asymmetry of flows); (3) Network switches, routers and intermediary nodes are treated as "identical network devices" because, from the point of view of SDN network control, their distinction is irrelevant [18].

Network topology is represented by a graph adjacency matrix $G(N, E)$, denoted as M, with size $N \times N$. For a connection between two nodes i and j, value $e_{ij} = 1$ is assigned, while $e_{ij} = 0$ otherwise. Note that matrix M is not symmetric. To optimize the routing of flows, weights w_{ij} are assigned to the transmission channel. Here, w_{ij} are natural numbers from the range $(1, v)$, where v can take any value. Weight matrices W have size $N \times N$. The weighted adjacency matrix M_W is determined as the Hadamard product [19] of matrices M and W.

$$M_W = M \bullet W = \begin{pmatrix} w_{11} \cdot e_{11} & \dots & w_{1N} \cdot e_{1N} \\ \vdots & \ddots & \vdots \\ w_{N1} \cdot e_{N1} & \cdots & w_{NN} \cdot e_{NN} \end{pmatrix} \tag{1}$$

For vertex pairs (s, d), where $s, d \in N$, *homogeneous traffic flow* f specifies requests to transmit information, as represented by a flow matrix F_{sd}.

$$F_{sd} = \begin{pmatrix} p_{11} & \cdots & p_{1N} \\ \vdots & \ddots & \vdots \\ p_{N1} & \cdots & p_{NN} \end{pmatrix} \tag{2}$$

where: $s = d \rightarrow p_{sd} = 0$; $p_{sd} = m \cdot f$, $m \in \mathbb{N}$. Homogeneous flow f corresponds to the granularity of flows in the network, as is the case with queues, e.g. in the Ethernet network ($f = 64$kB). Total flow p_{sd}, is therefore defined as a multiple

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32

of the base flow f. When, a unit value f is assumed, then $p_{sd} = m$. For example, for channels 100 Mb/s and granulation $f = 64$kb, for any edge $\max m = 1563$. In what follows, such network will be named a *network with homogeneous flow structure*. The flow matrix can change over the life time of the network. The values of the elements of this matrix can also be predicted in advance [8]. The matrix L determines the current link load e_{ij} of the network topology. Here, it is assumed that it will be systematically modified, as a result of the modification of weights in the matrix W and the distribution of flows defined in the matrix F_{sd}. The path between the vertices s, d, for a given flow p_{sd}, will be determined using the Dijkstra algorithm [6]. However, other algorithms can also be used. Thus, the value for l_{ij} is determined as the sum of flows p_{sd} passing through the edge e_{ij}. Taking into account the need of dynamic control of link weights, the problem of network load balancing becomes: seeking a set of link weights W, for which the maximum number of flows passing through the "busiest edge" in the network has been minimized. Therefore, the problem can be formulated as:

$$\min\left(\max\left(l_{ij}\right)\right) \tag{3}$$

Note that, in the algorithm, the load matrix L is represented as a load vector $VL = [l_{11}, l_{12}, ..., l_{1N}, ..., l_{N1}, l_{N2}, ..., l_{NN}]^T$. As noted, the problem of balancing loads of links in the network is NP-hard. Hence, the SDN Genetic Algorithm Load Balancer (SDNGALB) is proposed. It is characterized by low computational complexity; allowing implementation of the balancing algorithm, and actual deployment in production systems.

2.1 SDNGALB algorithm description

Initially, values of elements w_{ij} are randomly populated, with natural numbers from the range $(1, v)$. In what follows, $\max v = 9$ was used. However, for very large networks, with high connectivity, a larger range of weights may be needed. However, this must be determined experimentally, or based on the designers' intuition. The following decisions outline the design of SDNGALB (Figure 1).

1. The chromosome is the weight list VW, of individual network links, obtained from matrix M_W, according to the rule: if for any $x, y \in (1, N)$, $w_{xy} \cdot e_{xy} = 0$ weight is omitted; if $w_{xy} \cdot e_{xy} > 0$ the weight is added to the list (the length of the chromosome is equal to the number of edges in the network graph).
2. The initial chromosomes are randomly generated, from range $(1, v)$. The size of the population is selected experimentally, and is denoted as n.
3. The fitness function (FF) is calculated using formula (3). The FF algorithm is presented, in the form of a pseudocode in Figure 1.
4. Individuals are ranked on the basis of the values of their fitness function.
5. Pairs of individuals, arranged according to the quality of adaptation, are crossed with each other using the standard one-point method – with the crossing point selected randomly.
6. Mutation occurs with the probability determined by the parameter mp, and consists of drawing a new value (from a specified range) of any gene in the chromosome.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32

7. Optimization is performed until one of the following stop condition occurs: (a) Stagnation parameter (sz) is reached, i.e. number of solutions, during which the obtained results do not improve; (b) The maximum number of generations (gs) is reached.

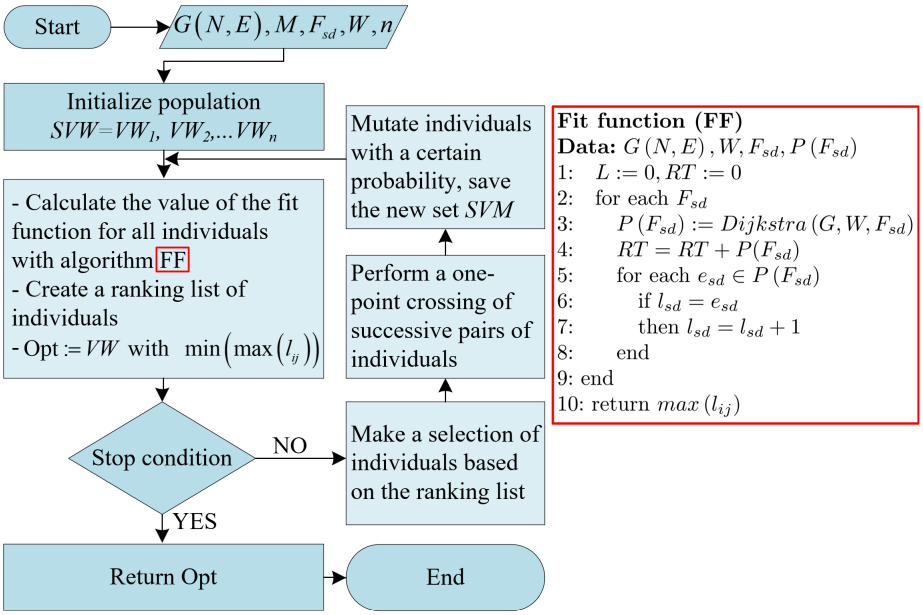

Fig. 1. Block diagram of the o SDNGALB algorithm and Fit Function (FF)

The Fitness Function (FF) algorithm requires a more detailed explanation. For each flow F_{sd}, the shortest path between the source node and the destination node is calculated (using the Dijkstra algorithm, and taking into account matrices M, W, M_W). Optimization of load distribution is achieved by manipulating the link weights in the matrix W (on the basis of which the matrix M_W is built) to eliminate network bottlenecks, by reducing the load on the most frequented edge in the network. For weights W, of chromosome VW, and flows F_{sd}, using the Dijkstra algorithm, the routing table RT is determined. It contains a number of rows equal to the number of non-zero matrix elements F_{sd} and each row contains a list of vertices to be traversed for a given flow p_{sd}, marked as $P(p_{sd})$. For example, a single row in the RT routing table, for a flow between 6 and 0 vertices $F_{6,0}$, might look like $P(F_{6,0}) \rightarrow e_{6,5}, e_{5,3}, e_{3,1}, e_{1,0}$, where $e_{u,d}$ denotes the edge connecting vertices u and d. Then, on the basis of RT, FF algorithm determines the maximum link load in the network $\max(l_{ij})$.

The results of operation of SDNGALB is the set of optimal link weights W, in the given network, that allows building a balanced routing table RT, which

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32

minimizes the maximum link load in the network. Note that this algorithm can be triggered periodically or when network flow related threshold values are exceeded. The source code of the algorithm is available at the website [3].

3 Experimental setup and results

The algorithm was implemented in Python and deployed in the environment consisting of, enterprise class, network devices. During experiments, a dedicated laboratory, configured for Internet of Things related research, was used [2].

The first set of tests was focused on the effectiveness of optimization. For flows in the network represented by graph $G(N, E)$, where $|N| = 10$ and $|E| = 39$, the values of the function (3) were compared before and after using the SDNGALB algorithm. Table 1 shows the mean value of $\max(l_{ij})$, before and after optimization, calculated as the average of 10 executions of the SDNGALB algorithm, for the defined network, for $|F_{sd}| = 20; 30; 40; 50; 100; 200$.

Table 1. Average effectiveness of optimization for different number of network flows.

Arithmetic mean of 10 executions	$\|F_{sd}\|$					
	20	***30***	***40***	***50***	***100***	***200***
$\overline{\max(l_{ij})}$ before optimisation	5,6	8,6	8.8	11,6	21,8	41,6
$\overline{\max(l_{ij})}$ after optimisation	2,8	4	4,8	5,8	11,6	22,4
Effectiveness of optimisation	50%	53%	45%	50%	47%	56%

As can be seen, application of SDNGALB resulted in a noticeable (about 50%) reduction of flows in the most heavily loaded links in the tested network.

The next set of experiments was performed to compare the time of reaching the solution using SDNGALB with the exact algorithm (BF), searching the entire solution space. Here, 100 simulations were performed for both algorithms for a network with 4 nodes and 5 edges, and 5 defined flows. The same optimization result, expressed by the value of the function (3), was obtained by both algorithms. For the genetic algorithm the mean the time was 0.0118065 seconds, while for the BF algorithm the time was 4448.077889 seconds.

Next, the effectiveness of the proposed solution on networks with different topologies was explored. The speed of obtaining the solution (denoted as Ts, measured in seconds), for $\min(\max(l_{ij}))$ is reported. In the first phase, 1000 simulations were performed, for six different network topologies, in which the initial flows were randomly generated. The characteristics of the networks Net, used in the study, is presented in Table 2, where the connectivity parameter Cn should be understood as the percentage ratio of the number of edges in the tested network to the number of edges in the network with all possible connections.

For experiments reported in Table 2, flows were randomly generated for each individual simulation. The SDNGALB parameters were: mutation probability:

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32

Table 2. Parameters of networks used in the research and average time of reaching the solution by SDNGALB algorithm in seconds

Net	$n4e5$	$n5e11$	$n6e15$	$n10e39$	$n25e219$	$n50e872$
$\|N\|$	4	5	6	10	25	50
$\|E\|$	5	11	15	39	219	872
Cn	31%	44%	42%	39%	35%	35%
v	1-5	1-5	1-5	1-9	1-9	1-9
$\|F_{sd}\|$	5	10	15	20	45	100
Ts	0.012	0.021	0.034	0.093	1.091	10.264

$mp = 10\%$; population size: $n = 50$; generation size: $gs = 500$; stagnation: $sz = 100$. Here, as the network size increases, i.e. from 4 nodes and 5 connections to 50 nodes and 872 connections, the solution time remains within an acceptable range (10s max). Therefore, it can be stipulated that network optimization, based on the SDNGALB algorithm, can be deployed in production network systems.

The final set of experiments compared the performance of the SDNGALB algorithm with two alternative optimizers. The first one – Ant Colony Optimization Load Balancer (ACOLB [11]) uses ant colony optimization to determine optimal routes between nodes. The second one – Dijkstra's Shortest Path Algorithm (DSPA, [4]) is based on the identification of the shortest paths between given nodes, using Djkstra's algorithm, under the assumption that link weights are randomized (using values from a given range) and are not modified later. Here, DSPA did not optimize link weights and served only as a baseline for the the execution time length. Experiments were performed on the n10e39 network (with 10 nodes, 39 edges) with the assumption that $|F_{sd}| = 20$. The performance of the ACOLB algorithm was tested with different parameter values (number of ants: 1, 5, 10, 25, 50, 100, 500). For each combination of values for ACOLB, 100 simulations have been run, and the average results of execution time in second (Ex) and $\overline{\max(l_{ij})}$ are reported. The tested algorithms obtained the following results: SDNGLAB: Ex=0,09263 and $\overline{\max(l_{ij})}$=2,792; ACOLB: Ex=0,02229 and $\overline{\max(l_{ij})}$= 31,392, DSPA: Ex=0,00546 and $\overline{\max(l_{ij})}$=5,110. Therefore, it can be concluded that the proposed solution to optimization of link weights, in the network during routing, makes it possible to achieve nearly twice the lower maximum link load in the network, as compared to the performance of the DSPA algorithm. In contrast, the ACOLB does not deliver satisfactory optimization vis-a-vis the proposed solution.

4 Concluding remarks

In this work the need for efficient SDN network flow optimization has been addressed by means of the dedicated genetic algorithm. The overarching goal was to deliver efficient infrastructure for extended cloud infrastructures, where resources, typically realized as services and microservices, can be highly dispersed.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32

The discussed approach is fast, which should allow to quickly modify the routing table, in response to changing traffic patterns. Results obtained during tests are very encouraging, concerning both the speed and the quality of optimization. Additional details about the approach, including extensive literature review can be found in [1]. As part of further work, (1) scalability of the proposed algorithm will be tested for large, highly distributed, networks; (2) algorithm will be adapted to modify the physical topology of MESH networks; and, (3) possibility of automatic, adaptive tuning of optimizer parameters, using machine learning techniques will be explored.

Acknowledgements Work of Marek Bolanowski and Andrzej Paszkiewicz is financed by the Minister of Education and Science of the Republic of Poland within the "Regional Initiative of Excellence" program for years 2019–2023. Project number 027/RID/2018/19, amount granted 11 999 900 PLN. Work of Maria Ganzha and Marcin Paprzycki was funded in part by the European Commission, under the Horizon Europe project ASSIST-IoT, grant number 957258.

References

1. https://arxiv.org/abs/2304.09313
2. Research stand IoE. https://zsz.prz.edu.pl/en/research-stand-ioe/about, accessed: 2023-01-02
3. SDNGALB source code. https://bolanowski.v.prz.edu.pl/download, accessed: 2023-01-02
4. Babayigit, B., Ulu, B.: Load Balancing on Software Defined Networks. In: 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT). pp. 1–4. IEEE, Ankara (Oct 2018). https://doi.org/10.1109/ISMSIT.2018.8567070
5. Chen, Y.T., Li, C.Y., Wang, K.: A Fast Converging Mechanism for Load Balancing among SDN Multiple Controllers. In: 2018 IEEE Symposium on Computers and Communications (ISCC). pp. 00682–00687. IEEE, Natal (Jun 2018). https://doi.org/10.1109/ISCC.2018.8538552
6. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (Dec 1959). https://doi.org/10.1007/BF01386390
7. Ericsson, M., Resende, M., Pardalos, P.: A Genetic Algorithm for the Weight Setting Problem in OSPF Routing. Journal of Combinatorial Optimization **6**(3), 299–333 (2002). https://doi.org/10.1023/A:1014852026591
8. Gao, K., Li, D., Chen, L., Geng, J., Gui, F., Cheng, Y., Gu, Y.: Predicting traffic demand matrix by considering inter-flow correlations. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 165–170 (2020). https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9163001
9. Jain, A., Chaudhari, N.S.: Genetic Algorithm for Optimizing Network Load Balance in MPLS Network. In: 2012 Fourth International Conference on Computational Intelligence and Communication Networks. pp. 122–126. IEEE, Mathura, Uttar Pradesh, India (Nov 2012). https://doi.org/10.1109/CICN.2012.119

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32

10. Jain, P., Sharma, S.K.: A systematic review of nature inspired load balancing algorithm in heterogeneous cloud computing environment. In: 2017 Conference on Information and Communication Technology (CICT). pp. 1–7. IEEE, Gwalior, India (Nov 2017). https://doi.org/10.1109/INFOCOMTECH.2017.8340645
11. Keskinturk, T., Yildirim, M.B., Barut, M.: An ant colony optimization algorithm for load balancing in parallel machines with sequence-dependent setup times. Computers & Operations Research **39**(6), 1225–1235 (Jun 2012). https://doi.org/10.1016/j.cor.2010.12.003
12. Li, G., Wang, X., Zhang, Z.: SDN-Based Load Balancing Scheme for Multi-Controller Deployment. IEEE Access **7**, 39612–39622 (2019). https://doi.org/10.1109/ACCESS.2019.2906683
13. Mahlab, U., Omiyi, P.E., Hundert, H., Wolbrum, Y., Elimelech, O., Aharon, I., Erlich, K.S.Z., Zarakovsky, S.: Entropy-based load-balancing for software-defined elastic optical networks. In: 2017 19th International Conference on Transparent Optical Networks (ICTON). pp. 1–4. IEEE, Girona, Spain (Jul 2017). https://doi.org/10.1109/ICTON.2017.8024847
14. Mazur, D., Paszkiewicz, A., Bolanowski, M., Budzik, G., Oleksy, M.: Analysis of possible SDN use in the rapid prototyping processas part of the Industry 4.0. Bulletin of the Polish Academy of Sciences: Technical Sciences **67**(1), 21–30 (2019). https://doi.org/10.24425/BPAS.2019.127334
15. Mohiuddin, M.A., Khan, S.A., Engelbrecht, A.P.: Fuzzy particle swarm optimization algorithms for the open shortest path first weight setting problem. Applied Intelligence **45**(3), 598–621 (Oct 2016). https://doi.org/10.1007/s10489-016-0776-0
16. Mulyana, E., Killat, U.: An Alternative Genetic Algorithm to Optimize OSPF Weights. Internet Traffic Engineering and Traffic Management pp. 186–192 (Jul 2002)
17. Paszkiewicz, A., Bolanowski, M., Budzik, G., Przeszłowski, L., Oleksy, M.: Process of Creating an Integrated Design and Manufacturing Environment as Part of the Structure of Industry 4.0. Processes **8**(9), 1019 (Aug 2020). https://doi.org/10.3390/pr8091019
18. Smiler. S, K.: OpenFlow cookbook. Quick answers to common problems, Packt Publishing, Birmingham Mumbai, 1. publ edn. (2015)
19. Styan, G.P.: Hadamard products and multivariate statistical analysis. Linear Algebra and its Applications **6**, 217–240 (1973). https://doi.org/10.1016/0024-3795(73)90023-2
20. Wang, H., Xu, H., Huang, L., Wang, J., Yang, X.: Load-balancing routing in software defined networks with multiple controllers. Computer Networks **141**, 82–91 (Aug 2018). https://doi.org/10.1016/j.comnet.2018.05.012

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36021-3_32