Network Slice Placement in Wireless Mesh Networks

Alexander Seng*, Ulrich Trick*, Armin Lehmann*, and Bogdan Ghita†,
*Research Group for Telecommunication Networks, Frankfurt University of Applied Sciences, Frankfurt, Germany
†Centre for Security, Communications and Network Research, University of Plymouth, Plymouth, UK

***Abstract*—Network slicing is a key technology for 5G and future mobile networks. It enables the creation of different virtual networks on the same physical infrastructure. By applying network slicing to wireless mesh networks(WMNs), it's possible to provide communication infrastructure in a quick and less complicated way. This paper shows an approach based on an evolutionary algorithm connecting multiple WMN nodes to a slice. The first step is modeling the WMN as an undirected graph to get a data structure for the algorithm. The paper then shows the procedure of the different steps of the algorithm. It also shows at which number of nodes to connect the algorithm outperforms a brute force approach. The final results shown in this paper are how the time complexity increases when adding additional nodes to the slices and increasing the network size.**

***Index Terms*—Network Slicing, Wireless Mesh Networks, Evolutionary Algorithms, Optimization, Network Model**

I. Introduction

Network Slicing is a key feature of modern 5G networks that allows different services with varying requirements to operate on the same physical infrastructure. The network provides multiple independent virtual "slices" for each use case based on the physical network infrastructure. The concept has been proposed for 5G networks but cannot be transferred from 5G networks to WMNs without further research due to their different architectures. In a wireless mesh network, a slice consists of multiple nodes placed at different locations within the network. This is, in principle, shown in Figure 1. The physical nodes host virtual network functions (VNFs) and are connected through virtual links marked in the figure. If this network has no connections to other networks or the internet, the functionalities of the slices must be provided by the mesh network itself.

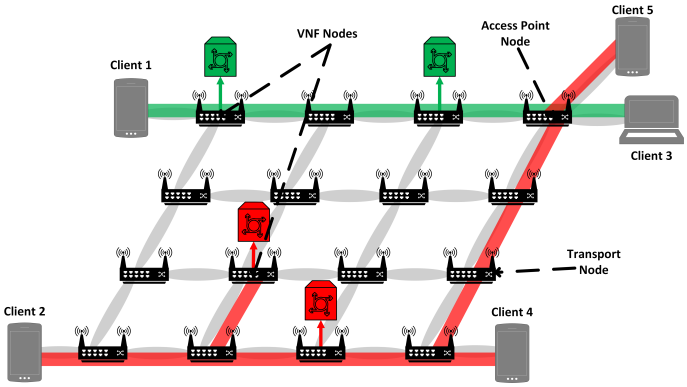

Fig. 1. Node types in the wireless mesh network

A wireless mesh network can restore communication in disaster areas. New nodes can join or leave, resulting in topology changes. This is relevant in natural disasters because nodes can be damaged. Widely available technology should be used, but communication must remain reliable and adaptable. Network slicing can provide virtual networks tailored to the needs of different groups of participants.

Wireless network links have limited capacity due to their shared media nature. Network slicing can help address this issue by enabling the creation of independent virtual networks. This paper proposes a method for creating and deploying network slices on a wireless mesh network. It uses an evolutionary algorithm to connect multiple nodes belonging to a slice and evaluates slice placement based on a predefined cost function. This paper focuses on determining how this method behaves with large networks and ever-larger slices. The paper is organized as follows: Related work is shown in Section II. Section III describes the mesh network model. Section IV introduces the node connection concept for network slicing. Experimental validation is described in Section V. The paper concludes in Section VI.

II. Related Work

The article in [1] suggested decreasing packet loss in wireless mesh networks by transmitting packets through various virtual interfaces. However, this approach lacks multitenancy, virtualization of network functions, and resource-sharing, which are necessary for a network-slicing concept. The papers, [2], [3] and [4] presented a wireless disaster network that uses network function virtualization (NFV) with increased resilience and better distribution of control data. However, they did not cover network slicing.

In [5], a resource-efficient approach is presented for service function chains (SFC), which relocates virtual network functions (VNFs) to different network parts to minimize resource consumption. The work in [6] focused on the optimal placement of network functions in a service chain using heuristic algorithms. [7] also presented an approach for optimizing VNF replacement to minimize the end-to-end delay of an SFC. Finally, [8] dealed with the placement of service function chains but did not consider the optimal connection between nodes or network slicing. The work in [9] proposed a management scheme that uses interference between wireless links for topology decisions. The proposed algorithm creates a path through the wireless network based on a weighted sum of

the interferences, aiming to minimize interference and create an optimized path. However, the term "sliced" used in the paper did not refer to network slicing as defined in Section I. Virtual network embedding (VNE) is the process of mapping a virtual network to a physical network. The resource limitations of WMNs make optimal placement and chaining of virtual network functions important. A survey by [10] discussed relevant approaches, but a generalized approach for Network Slicing was not included, nor are the special properties of WMNs addressed.

Evolutionary algorithms are commonly used for network optimization, such as in [11] and [12]. These works focused on multi-objective routing optimization without considering VNF placement or network slicing. Another related work in [13] used genetic algorithms for in-network processing in wireless sensor networks but did not handle multi-node connections or network slicing.

As we delved into the topic of network slicing in wireless mesh networks, it became apparent that the literature outlined in this chapter only scratches the surface of the various aspects that need to be considered. While virtualization, sharing network capacities, and WMNs are all crucial components, each one only provides a partial solution to the complex puzzle of network slicing. A more comprehensive approach is needed, one that considers all the various aspects of network slicing and how they interact with each other.

III. Modeling the Environment

This section proposes a model of the wireless mesh network and the network slices. The model is necessary for describing optimization approaches and for algorithmic processing. Wireless mesh networks have some architectural particularities that have to be considered for modelling the network infrastructure.

A. Model of the overall Network

It is possible to model the WMN as an undirected graph $G(V, E)$, with the nodes V and the edges E as it is shown in [9]. The graph vertices represent the mesh nodes of the WMN, and the edges indicate the links between the WMN elements. Figure 2 gives an example of a graph representation of a wireless mesh network.

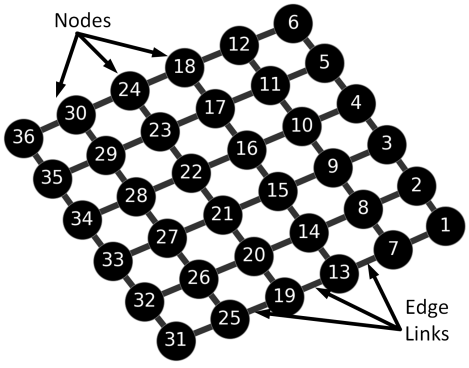

Fig. 2. Model of a wireless mesh network as a graph

Every link $e \in E$ has a maximum consumable bitrate $B(e)$, representing the link's maximum transmission speed. Because of the wireless nature of the links in a WMN, the bitrate is not a constant value, and it depends mainly on the distance between the nodes, the interference, and obstacles in the environment. Thus, interference on the radio link is represented by a reduction of the maximum available bitrate. In real networks, the devices reduce the modulation scheme when the link quality decreases. This also reduces the maximum available bitrate on a link. Also, in wireless transmissions, only one device can simultaneously transmit or receive at a channel. This leads to a decrease in the maximum consumable bitrate on the specific link. A model for a wireless mesh network has to consider these constraints. A possible assumption to this is to divide $B(e)$ with the number of identical channels N_{ch} used in the reception range of a node. Therefore $B(e)$ is calculated with the equation: $B(e) = \frac{B_{max}}{N_{ch}}$, where B_{max} describes the maximum possible bitrate on the link when there is no channel reuse. Any virtual link l on a wireless link e consumes a part $b_e(l)$ of the maximum available bitrate $B(e)$. As a result, the number of streams a link can supply that can work with a specific bitrate, e.g., 10 Mbit/s, is limited. The resulting constraint shows Equation 1. A virtual link l describes a communication relation between a start node, an end node, and intermediate virtual network functions if present. If the sum of all values from $b_e(l)$ exceeds $B(e)$, a link should be marked as unavailable.

$$\sum_{l \in L} b_e(l) \leq B(e) \tag{1}$$

Every WMN node has computing power C_n for providing virtual network functions (VNFs). The value for C_n describes how many percent of the maximum CPU power is still available for virtual network functions: $C_n \in [0\%, 100\%]$. A VNF f running on a node needs a part $c(f)$ of the node's computing power. The maximum number of VNFs a node can provide without performance degradation due to CPU overload results from the constraint of Equation 2:

$$\sum_{f \in F} c(f) \leq C_n \tag{2}$$

If the sum of $c(f)$ exceeds the value of C_n, a node cannot host any additional VNF. These constraints should mainly be considered when optimizing.

B. Wireless Channel Distribution

In standard mesh networks, each node has only one radio interface. This results in a reduced transmission rate due to the wireless medium's half-duplex characteristic, which allows only transmitting or receiving simultaneously. Additionally, other nodes on the same channel cannot transmit while another is transmitting inside their interference range. Multiple interfaces can reduce the problem, but each requires a different channel. The weighting function used in [14] did not consider the channel distribution, which can affect the maximum achievable bit rate. In contrast, this paper's weighting function

considers the frequency of channel reuse and adjusts the link weight accordingly. The calculation formula for the weighting W on a link e is as follows:

$$W = \begin{cases} e^{7*\frac{B*N_{ch}}{B_{max}}} & B \leq B_{max} \\ Inf. & B > B_{max} \end{cases}$$

Factor 7 results from the fact that W should have the value 1000 for $B = B_{max}$ or $B * N_{ch} = B_{max}$. The value 1000 was determined empirically. Since a path determination algorithm uses this weighting value to determine the paths, the channel distribution is thus also taken into account in the path determination. To verify the assumptions described here, various simulations were carried out using the NS3 network simulator. The behavior assumed here concerning the reduction in bit rates was confirmed.

C. Model of the slices in the Network

A network slice consists of the following components: Virtual network functions, a priority value to define different slice priorities, and the virtual links belonging to the slice. Derived from the virtual links and the VNFs, a slice is a subgraph $G_s(V_s, E_s) \in G(V, E)$ [9] of the WMN graph. V_s is the set of nodes that host the slice VNFs(V_f), forward the traffic of a slice flow (V_t) or work as a slice access point node (V_{ap}). Therefore, $V_s = V_f \cup V_t \cup V_{ap}$. E_s is the set of edges that connect the nodes of V_s.

IV. Node Connection Concept

In order to place a slice within the WMN, we need to identify three types of nodes: access point nodes (V_{ap}), VNF nodes (V_f), and transport nodes (V_t). Access point nodes cannot be changed, while VNF nodes can be moved through NFV. Transport nodes forward data traffic of one or more slices exclusively. Optimizing paths between access point nodes and placing VNFs on these paths is necessary, resulting in two optimization objectives:

- Placing the VNFs
- Connecting the access point nodes on the shortest path.

In networking, the Dijkstra algorithm [15] is useful for connecting two nodes. However, using it to connect multiple nodes is challenging. The approach of connecting access point nodes individually through the shortest path is insufficient, as it may not provide the shortest connection between all the nodes. This principle was proved using an example in Figure 3.

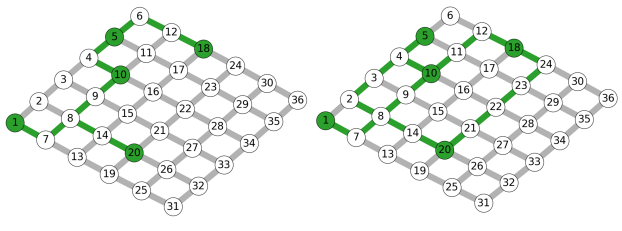

Fig. 3. Difference between shortest path calculations

Assume that the order in which Dijkstra's algorithm is applied to the nodes does not matter. Dijkstra's algorithm can be applied to nodes in any order, but the number of links used should be the same for different permutations. With the permutation (1, 18, 20, 5, 10), 19 links are used. With (1, 20, 10, 5, 18), only 11 links are used. This shows that the order matters and further optimization is necessary.

The goal is to optimize the positioning of the slice in the WMN with minimal link usage and optimal placement of VNFs. The main aspect of this work should be the connection of the nodes. The recombination process of the algorithm can additionally replace the VNFs. This results in a multi-node shortest path problem, similar to the traveling salesman problem [16] or the vehicle routing problem [17]. In comparison, however, the problem mentioned here differs from this:

- Created paths can be used more than once.
- Certain nodes can be moved.
- No closed loop has to be created.

Such a problem belongs to the group of NP-hard problems. NP-hard problems cannot be efficiently solved using brute force methods that test all possibilities due to their steep complexity increase with size. Calculating all permutations to check the path length becomes impractical as the number of permutations increases rapidly based on the factorial function. However, the solution thus found is the optimal one, as all possible ones have been tested. The graphic in Figure 4 shows the rapid increase in time.

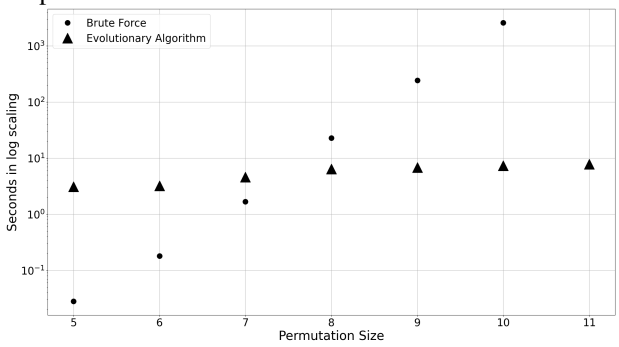

Fig. 4. Comparison between the brute force method and the evolutionary algorithm.

The processing time increases with every added node in the permutation. Therefore, the brute force method is usable with up to 8 nodes, after which another method is necessary. An evolutionary algorithm has a considerably shorter runtime and is used for larger permutations. With a permutation size of 10, the brute force approach takes 2579 seconds, while the evolutionary algorithm takes only 7.3 seconds.

Evolutionary algorithms use the principle of biological evolution for optimization [18]. The principle is based on the following steps: evaluation, selection, recombination, mutation, and re-insertion. The procedure shows Figure 5.

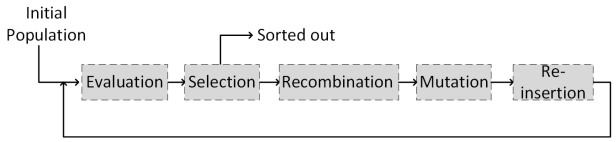

Fig. 5. Principle of Evolutionary Algorithms.

The evolutionary algorithm begins with a randomly generated population, which is evaluated and the best elements are selected. The remaining elements are sorted out and used for recombination to create new elements. The mutation prevents the algorithm from getting stuck in local optima. A termination criterion should be set to end the algorithm. For all of the steps shown in Figure 5, there are different possibilities for implementation. In the first step, the initial population of permutations is evaluated by the so-called fitness or evaluation function. Each element receives a tag with the corresponding evaluation value. The evaluation function evaluates the permutations based on two criteria:

- the length of the paths of a slice $\sum e \in E_s$,
- the average utilization of a node the path contains which is no AP node $\frac{\sum_{e \in E_n}(W(e))}{\sum_{e \in E_n}}$.

E_n is the set of edges a node n has. Based on these criteria, we can calculate the cost value for placing a slice in the mesh network with Equation 3.

$$cost = a * \sum_{e \in E_s} + \sum_{n \in V_t \cup V_f} b * \frac{\sum_{e \in E_n} W(e)}{\sum_{e \in E_n}} \quad (3)$$

The adjustable parameters a and b are used to modify optimization weighting. Tournament selection [19] is used to select the best elements from a population. Four randomly selected elements are evaluated using the function, and the best one is chosen. Only 25% of the best elements are used for recombination. The selection procedure is shown in Algorithm 1.1.

Algorithm 1.1 Selection Procedure

Require: population, graph, slice
1: **for** i = 1, 2, ... *length*(population)/4 **do**
2: $j \leftarrow 0$
3: **while** j ¡ 4 **do**
4: $turn[j] \leftarrow population[random]$
5: $j++$
6: **end while**
7: **for** e = 1, 2, ... *length*(turn) **do**
8: $FitnessValue \leftarrow FitnessFunction(turn[e])$
9: $turnRes[e] \leftarrow [FitnessValue, turn[e]]$
10: **end for**
11: *sort*(turnRes)
12: $returnValue \leftarrow turnRes[0]$
13: **end for**
14: **return** returnValue

The elements processed by the algorithm contain nodes of the WMN. These can only take on discrete values. Discrete recombination is therefore used as the recombination scheme. These are then used to form new elements. The nodes are selected randomly, but all with the same probability [20]. This principle is shown in Figure 6.

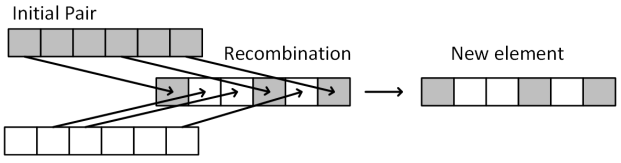

Fig. 6. Recombination principle

The approach doesn't remove original elements because newly created ones can be worse. So, two are added for every initial pair. The recombination procedure is shown in Algorithm 1.2. The input elements n and m are two chosen elements from the selection step. Both are equal in length, so the loop iterates up to the length of one of the elements. The random variable $var1$ is either zero or one. Therefore, in every iteration of the for loop, the new element gets a value from m or n. After this is done, the recombined element is returned.

Algorithm 1.2 Recombination Procedure

Require: n, m
1: **for** i = 1, 2, ... *length*(element1) **do**
2: $var1 \leftarrow randomInteger(0, 1)$
3: $newElement[i] \leftarrow n[i] * var1 + m[i] * (1 - var1)$
4: **end for**
5: **return** newElement

After the recombination part, every element has a small probability of mutation. This means that within an element, the entry for a node can change spontaneously with this probability. As mentioned in this section, this is done to avoid or overcome local optima. Algorithm 1.3 shows this procedure.

Algorithm 1.3 Mutation Procedure

Require: population, mutationRate
1: **for** i = 1, 2, ... *length*(population) **do**
2: **if** $random(0, 1) \leq mutationRate$ **then**
3: $k \leftarrow randomInteger(0, length(population[i]))$
4: $randNode \leftarrow randomInteger(0, Nodes)$
5: $population[i][k] \leftarrow randNode$
6: **end if**
7: **end for**

The loop iterates over every element in the population. The probability for mutation increases or decreases depending on which value the variable *mutationRate* is set. When the if-statement is fulfilled, a random index for the permutation and a random node for exchange is calculated. The former node on the specific index is then replaced. Afterwards, the newly created elements must be checked to see if they contain all the necessary nodes. This is shown in Algorithm 1.4.

Algorithm 1.4 Node check procedure

Require: population, sliceNodes
1: $containNode \leftarrow$ *False*
2: **for** i = 1, 2, ... *length*(population) **do**
3: **for** j = 1, 2, ... length(sliceNodes **do**
4: **if** $sliceNode[j]$ *in* $population[i]$ **then**
5: $containNode \leftarrow$ *True*
6: **else**
7: $containNode \leftarrow$ *False*
8: **return** $containNode$
9: **end if**
10: **end for**
11: **end for**
12: **return** $containNode$

The variable containNode is set to true if a permutation has a specific node. Otherwise, it is false. New elements are

evaluated, added, and the worst ones are removed to keep the population at 100 elements. This is repeated until the termination criterion is met, and the best one is stored in an array after each iteration. The variance is then calculated from this array, as shown in Figure 7.

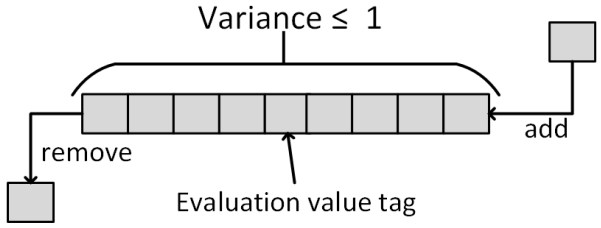

Fig. 7. Termination criterion scheme

The algorithm terminates if the variance is smaller than one for ten values in a row, returning the best element to that point. VNFs can be repositioned through recombination and mutation, allowing for optimization of their placement. Access point nodes are fixed and cannot be changed.

V. Test and Validation

We tested the evolutionary algorithm on a wireless mesh network graph, evaluating its optimization of network parameters for improved connectivity and coverage. The graph consisted of mesh routers as nodes and wireless links as edges. Results were analyzed to determine the algorithm's effectiveness.

A. Test Setup

After the algorithm was implemented, it was tested to ensure its effectiveness. The mutation rate parameter was adjusted with various values to determine the optimal value for the algorithm and the point where it diverges. To evaluate the algorithm's performance, the overall cost value for the Slice and the time required to place them in the network were calculated. The cost value was calculated using the Equation 3. The test was performed on a network with 100 nodes, equivalent to a 10x10 square architecture, similar to Figure 3. A slice comprising eleven access point nodes has been defined because these node types are not changeable. The number eleven has been chosen because, as shown in Figure 4, brute force cannot handle more than eleven nodes in a useful amount of time. The algorithm was executed 10000 times to generate sufficient values. The same slice was placed in the network for all 10000 iterations.

B. Result

Table I shows the results of the cost value calculations. Between the values for 2.5%, 5%, 7.5% and 10%, and 16% and 20%, there is no significant difference except for outliers. At rates of 25% and 33%, there is greater dispersion upwards and downwards. However, the number of outliers has also increased significantly. Besides mean and standard deviation(std), the table also shows the outliers. It can be seen that these seem to decrease again at higher rates. Therefore, the row *exceed* is also listed here. This describes the number of calculations where no result was found. If these are also added to the outliers, it becomes apparent how strongly the algorithm tends to diverge from this point on.

TABLE I
Results for the cost value evaluation.

	2.5%	5%	7,5%	10%	16%	20%	25%	33%
mean	306	308	311	312	317	319	322	328
std	15	15	16	16	18	18	20	22
outliers	11	9	17	35	136	143	21	46
exceed	0	0	0	0	0	0	624	6032

If only the cost values are considered, a mutation rate of 2.5% is the best value. However, the calculation time should also be taken into account. In theory, the calculation time should increase by increasing the mutation rate. This is because the algorithm is more likely to diverge. Also, the amount of outliers should increase. This is exactly the behaviour shown in Figure 8.

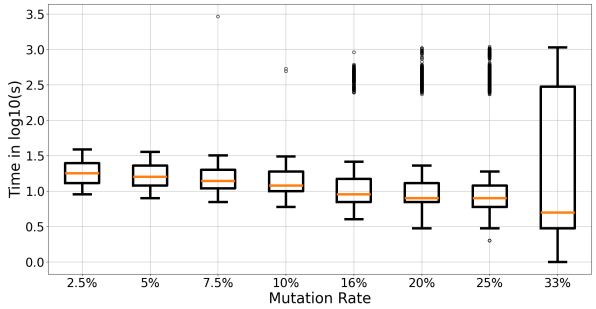

Fig. 8. Results of time measurement with different mutation rates

It is shown that the number of outliers increases with every increase in the mutation rate. It can also be seen that the algorithm starts to diverge at a rate of 33%. The detailed results from this calculation can be seen in Table II.

TABLE II
Results for the time value calculation

	2.5%	5%	7,5%	10%	16%	20%	25%	33%
mean	19	17	15	14	21	38	87	174
std	6	6	29	8	66	111	177	213
outliers	0	0	1	2	261	689	1701	65
exceed	0	0	0	0	0	0	624	6032

Here, the strong increase in outliers already starts at a mutation rate of 16%. The values for exceed are the same, as the time values were collected together with the evaluation values. As the table shows, the mean value for the time is at the lowest point with a mutation rate of 5%.

It is essential to test the behaviour as the network size and number of nodes increase. The results are shown in the figures 9 and 10. The absolute values of these results are not as interesting here as the increase in them. Figure 9 shows the processing time increases when the number of nodes a slice consists of increases. The time on the y-axis in Figure 9 is in a logarithmic scale. The nodes are increased from 10 to 100 in five node steps. The overall network has a size of 1024 nodes, which results in a 32x32 node architecture. Figure 10 shows the increase in the slice evaluation value. In both figures, it could be seen that the increase happens linearly.

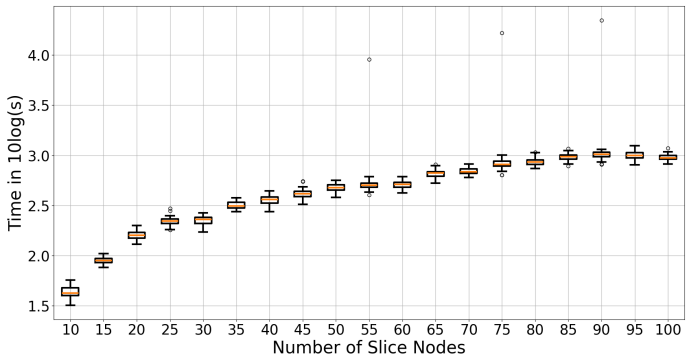

Fig. 9. Results of time measurement with increased number of nodes

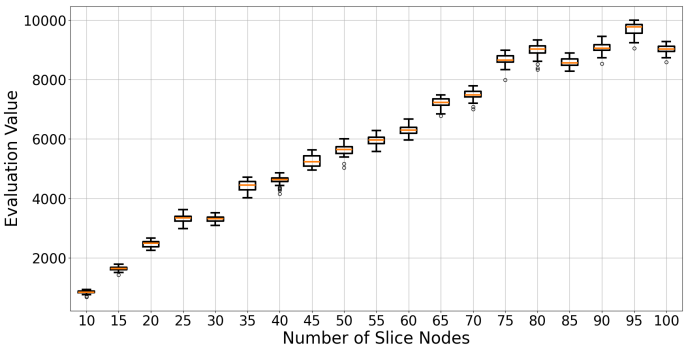

Fig. 10. Results of evaluation measurement with increased number of nodes

It can be concluded that this approach works even if the network or the number of nodes per slice is increased. A linear increase in processing time can be countered by, e.g. paralleling the calculations. An evolutionary algorithm is well suited for this [18] [21]. It is, therefore, possible to use this approach even in large setups with a feasible expenditure of time.

VI. Conclusion

This paper proposes a technique for optimizing network slice placement in wireless mesh networks. The network is modelled as a graph, and an evolutionary algorithm connects various slice nodes for efficient solutions. A ”brute force” approach to solve the network slice placement problem would take too much time to execute practically. The proposed algorithm is flexible and can solve the problem more efficiently, making it a practical solution. It can optimize the placement of network slices, leading to better resource utilization and improved network performance in various scenarios. Tests were conducted to determine the best mutation rate for an evolutionary algorithm. Results showed that a mutation rate of 10% was optimal for minimizing costs, although any rate between 2.5% and 10% could be used. Higher rates resulted in scattered results, and lower rates had a slightly increased calculation time. The second test showed that the proposed approach for network slice placement and optimization can effectively handle large networks with increasing slice nodes. Processing time increased linearly in networks with a maximum slice size of 100 nodes and 1024 nodes in total. The results suggest that the approach can be leveraged in practical scenarios to improve network performance.

References

[1] Y. Deng and A. Nakao, “Mesh slicing: Improving robustness of a mesh network via multiple slices,” *Co-NEXT ’11: Conference on emerging Networking Experiments and Technologies*, 2011.

[2] A. Lehmann, A. Paguem Tchinda, and U. Trick, “Optimization of wireless disaster network through network virtualization,” *Proceedings of the Eleventh International Network Conference*, 2016.

[3] G. Frick, A. Paguem Tchinda, U. Trick, A. Lehmann, and B. Ghita, “Distributed nfv orchestration in a wmn-based disaster network,” *International Conference on Ubiquitous and Future Networks, ICUFN*, pp. 168–173, 2018.

[4] G. Frick, A. P. Tchinda, U. Trick, A. Lehmann, and B. Ghita, “Nfv resource advertisement and discovery protocol for a distributed nfv orchestration in a wmn-based disaster network,” in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–6, 2019.

[5] G. Miotto, M. C. Luizelli, W. L. C. Da Cordeiro, and L. P. Gaspary, “Adaptive placement & chaining of virtual network functions with nfv-pear,” *Journal of Internet Services and Applications*, vol. 10, no. 1, pp. 1–19, 2019.

[6] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, “Improve service chaining performance with optimized middlebox placement,” *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 560–573, 2017.

[7] C. Ouyang, Y. Wei, S. Leng, and Y. Chen, “Service chain performance optimization based on middlebox deployment,” in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pp. 1947–1952, 2017.

[8] F. Esposito, M. Mushtaq, M. Berno, G. Davoli, D. Borsatti, W. Cerroni, and M. Rossi, “Necklace: An architecture for distributed and robust service function chains with guarantees,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 152–166, 2021.

[9] N. An, Y. Kim, J. Park, D.-H. Kwon, and H. Lim, “Slice management for quality of service differentiation in wireless network slicing,” *Sensors*, vol. 19, no. 12, 2019.

[10] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, “Virtual network embedding: A survey,” *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[11] R. Murugeswari, S. Radhakrishnan, and D. Devaraj, “A multi-objective evolutionary algorithm based qos routing in wireless mesh networks,” *Applied Soft Computing*, vol. 40, pp. 517–525, 2016.

[12] C. A. Lozano-Garzon, M. Camelo, P. Vilà, and Y. Donoso, “A multi-objective routing algorithm for wireless mesh network in a smart cities environment,” *J. Networks*, vol. 10, no. 01, pp. 60–69, 2015.

[13] H. Afifi, K. Horbach, and H. Karl, “A genetic algorithm framework for solving wireless virtual network embedding,” in *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–6, 2019.

[14] A. Seng, U. Trick, A. Lehmann, and B. Ghita, “Path determination for network slicing in wireless mesh disaster networks,” in *Mobile Communication 2021 and IEEE*, pp. 1–6.

[15] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[16] K. L. Hoffman, M. Padberg, and G. Rinaldi, “Traveling salesman problem,” in *Encyclopedia of Operations Research and Management Science* (S. I. Gass and M. C. Fu, eds.), pp. 1573–1578, Boston, MA: Springer US, 2013.

[17] G. B. Dantzig and J. H. Ramser, “The truck dispatching problem,” *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[18] P. A. Vikhar, “Evolutionary algorithms: A critical review and its future prospects,” in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pp. 261–265, 2016.

[19] T. Blickle and L. Thiele, “A comparison of selection schemes used in evolutionary algorithms,” *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[20] H. Pohlheim, *Evolutionäre Algorithmen: Verfahren, Operatoren und Hinweise für die Praxis*. VDI-Buch, Berlin: Springer Berlin, softcover repr. of the hardcover 1. ed. 2000 ed., 2013.

[21] B. Olsson, D. Lundh, and A. Narayanan, *Biocomputing And Emergent Computation - Proceedings Of Bcec97*. World Scientific, 1997.