Implementation for VNR-GA: Elastic Virtual Network Reconfiguration Algorithm Based on Genetic Metaheuristic

Cloud Computing Project Report

By

Nisarg S. Joshi (171CO226)
Arqum Shaikh (171CO241)
Manan Poddar (171CO221)
Yashas G. (171CO154)

Department of Computer Science and Engineering
National Institute of Technology Karnataka, Surathkal

March, 2021

Abstract

Cloud Computing offers elasticity and enhances resource utilisation. This is why its success strongly depends on the efficiency of the physical resource management. This project deals with dynamic resource reconfiguration to achieve high resource utilisation and to increase Cloud providers income. The paper proposes a new adaptive virtual network resource reconfiguration strategy named VNR-GA to handle dynamic users' needs and to adapt virtual resource allocation according to the applications' requirements. The proposed algorithm VNR-GA is based on Genetic metaheuristic and takes advantage of resources migration techniques to recompute the resource allocation of instantiated virtual networks. In order to optimally adapt the resource allocation according to customers' needs growth, the main idea behind the proposal is to sequentially generate populations of reconfiguration solutions that minimise both the migration and mapping cost and then select the best reconfiguration solution. We have implemented the VNR-GA algorithm discussed in the paper and obtained various evaluation metrics by simulating test cases.

Contents

A Introduction

A.1 Understanding the problem statement

Cloud Computing is a promising architecture and technology enabling a flexible deployment of scaling applications. It offers elasticity and enhances resource utilisation. Indeed, Cloud Providers (CPs) lease their substrate infrastructures ondemand to Cloud users who only pay for what they actually use. Consequently, end-users are freed from software and hardware constraints such as maintenance, updates, software license, etc. Thanks to virtualisation technology, elasticity and high resource utilisation can be easily achieved. Hence, many independent applications can be hosted in data centres (i.e., Software as a Service). Moreover, an application can be deployed in many geographical sites and makes use of a Virtual Network (VN), embedded in the Cloud's backbone (i.e., Substrate Network SN), to link all the geographical sites (i.e., Infrastructure as a Service). In this respect, an end user can install any routing protocol within the allocated VN and be responsible for network administration. In other words, since virtualisation technology offers isolation, an end-user can only manage his VN (i.e., instance) and cannot deteriorate the rest of VN s hosted in the SN .

To achieve high elasticity and improve resource utilisation within the Cloud infrastructure, efficient reconfiguration techniques are mandatory and need to be set up. Indeed, in such a dynamic environment, resource reconfiguration offers flexibility that allows to re-organise scarce resources and accommodate changing customers needs. To achieve both objectives, physical resource reconfiguration can be classified into two main groups. The first, preventive mechanisms that aim to "tidy up" the embedded virtual resources within the SN in order to balance the load of physical resources and thus optimise resource usage. The second, curative approaches that handle dynamic user requirements in terms of hardware resources (e.g., processing power, memory and bandwidth, etc.) and adjust allocated resources in the SN according to the new users' needs. Indeed, depending on the running applications requirements, a client may demand whether to expand or to shrink his allocated resources within the backbone network. As a consequence, CP must be able to immediately adjust the capacity of each virtual network by reconfiguring the resource allocation in the SN . Hence, the QoS required by the customer is guaranteed.

A.2 Previous work done / Related work

Few curative reconfiguration algorithms for virtual networks have been proposed in literature. We will hereafter summarise the prominent strategies as discussed in the paper.

In [3], the authors put forward an allocation algorithm handling incremental expansion and release of resource usage to support elasticity denoted by SecondNet. In the case of a bandwidth reservation increase between two virtual nodes, the algorithm proceeds as follows. First, they increase bandwidth reservation along the already allocated substrate path if the latter can support the new bandwidth requirement, else, they search for a new substrate path meeting

the new bandwidth requirement. Then, if such a path is not found, a virtual nodes reallocation is performed using a bipartite matching algorithm proposed [4] for virtual network embedding. However, whatever the frequency of bandwidth updates, the algorithm sequentially reallocates the virtual links which can lead to useless migrations. Moreover, both node and link expansion are resolved by migrating all the concerned nodes. Hence, SecondNet strategy may lead to a poor level of performance in terms of i) reconfiguration cost and reject rates of resource upgrade requests.

In [5], the authors deal with the problem of determining dynamic topology reconfiguration for service overlay networks with dynamic communication requirements. In this context, the authors define policies to reconfigure overlay topologies. A policy is defined as a sequence of basic overlay topologies used by an overlay network in response to a communication requirement. They are characterised by a cost function combining both the reconfiguration and occupancy cost. In order to find the optimal reconfiguration policy, the proposal proceeds as follows: Assuming that the range of all communication requirements values is within a predetermined set of communication patterns: i) small and ii) large size systems. First, for small size systems, the sequence of topology transitions (i.e., policy) is formulated as a continuous time Markov decision process. Each state in the Markov model corresponds to a couple of communication patterns with a feasible overlay. A transition in the Markov model corresponds to the cost of the reconfiguration. Then, the optimal reconfiguration policy is obtained by solving the Markov decision process. Secondly, for large size systems, the authors propose heuristic methods depending on the reconfiguration cost values: i) for low reconfiguration cost, the algorithm favours the reconfiguration (i.e., always-change strategy, ii) for high reconfiguration cost, the algorithm prohibits the reconfiguration (i.e., never change strategy) and iii) otherwise, a Cluster-Based policy is proposed where communication patterns, representing the aggregated communication requirements of all customers, are grouped into clusters. In fact, the main idea behind the aforementioned heuristics is to assign each communication pattern to the cluster with the least policy cost. Unfortunately, such an algorithm cannot be applied with a generic communication requirement. Indeed, the authors assume that all communication requirements must belong to a predefined set of communication patterns, which is not realistic and does not match the customer's requirements.

In [6], the authors propose a Virtual Network Topology reconfiguration algorithm based on an enhanced traffic estimation and denoted by VNT. It is worth noting that the proposal refers to a network topology constructed over an optical layer path. VNT proceeds reconfiguration into multiple stages and its objective consists in minimising the number of reconfigured optical layer paths. Indeed, VNT heuristic first checks the usage rate of all optical layer paths. Then, for each detected congested link, a new optical link is added between two end nodes extremities of the path containing the congested link. Each link with low utilisation rate is deleted from the topology. Unfortunately, the authors do not take advantage of node migration techniques to reconfigure the topology. Moreover, reconfiguration decision is taken based on traffic estimation which even if improved, still is prone to errors.

A.3 Motivation for using VNR-GA

In the concerned paper an elastic reconfiguration algorithm named VNR-GA has been proposed. Unlike [5], the algorithm is able to handle all kinds of communication requirements. Indeed, VNR-GA can be applied whatever the link requirements formulated by the owner of a virtual network (i.e., client). Moreover, unlike [3], this proposal simultaneously migrates all bandwidth upgraded virtual links in order to avoid the useless migrations and thus minimises the VN reconfiguration cost. Finally, unlike [6], VNR-GA makes use of node migration in order to enhance reconfiguration performances.

A.4 Overview of VNR-GA

The proposed algorithm is named Virtual Network Reconfiguration based on Genetic Algorithm VNR-GA. It proceeds as follows. First, the embedded VN concerned by the bandwidth expansion is divided into a set of star topologies, denoted by SC = {SCi}. Each SCi is formed by a central virtual node and virtual links that are directly attached and in which at least one requires more bandwidth. Then, VNR-GA randomly generates a large population of chromosomes. Based on the amount of required bandwidth and the residual substrate bandwidth (i.e., reconfiguration cost), VNR-GA selects the best set of feasible chromosomes and discards the rest. Note that a chromosome describes a feasible sequence SCi that will be sequentially migrated and this in the defined order to satisfy the new requirements. It is worth pointing out that a chromosome represents a potential solution of reconfiguration. Moreover, this chromosome is characterised by its fitness metric quantifying the reconfiguration cost. Note that a configuration cost takes into account both the number of triggered migrations and the embedding cost of the updated VN . Afterwards, to improve the quality of the reconfiguration solution, new populations of chromosomes are iteratively generated during a predefined number of iterations. Indeed, a population is generated by novel crossover and mutation mechanisms. In doing so, the population generated in the previous iteration is improved in terms of cost reconfiguration. At the end, the best chromosome (i.e., reconfiguration solution) obtained during all iterations is selected and then executed to satisfy the customer's demands.

To gauge the effectiveness of the proposal, the paper compares VNR-GA with the most prominent method found in literature denoted by SecondNet [3]. The simulation results show that VNR-GA reduces the rejection rate of i) VN s and ii) bandwidth upgrade requests. Besides, the proposal enhances the CP long-term revenue compared with SecondNet strategy.

B Implementation and Methodology

B.1 FORMULATION OF THE FLEXIBLE VN RECONFIGURATION PROBLEM

Now we will formulate the reconfiguration problem of the VN denoted by D in the substrate network (SN) denoted by G. To simulate real network conditions, we consider that all the physical resources (i.e., bandwidth, processing power and memory) are limited. Hence, G is able to host simultaneously only a finite number of VN requests. Consequently, an efficient reconfiguration strategy is required to enable the adjustment of substrate resources according to dynamic user's requirements.

In the previous work [7], the author's have modelled SN as an undirected graph denoted by $G = (V(G), E(G))$ where $V(G)$ and $E(G)$ respectively represent the sets of physical routers and their connected links. Each substrate equipment, $w \in V(G)$, is characterised by its i) residual memory $M(w)$ and its ii) residual processing power $B(w)$. Likewise, each physical link, $e \in E(G)$, is typified by its i) capacity $C(e)$ and residual bandwidth $y(e)$.

As well, a VN request is presented as an undirected graph, denoted by $D = (V(D), E(D))$ where $V(D)$ and $E(D)$ are respectively the sets of virtual equipments and their virtual links. Each virtual router, $v \in V(D)$, is associated with the required memory $M(v)$ and processing power $B(v)$. Moreover, each virtual link $d \in E(D)$ is characterised by its bandwidth capacity $C(d)$.

The problem consists in reconfiguring the virtual graph topology D in order to satisfy the new user's requirements in terms of bandwidth. It is worth pointing out that the reconfiguration of D consists in migrating some of its allocated resources while respecting the following constraints.

Let $v \in V(D)$ be the migrating virtual node. The substrate node $w \in V(G)$ can host v only if it has sufficient available resources. Formally:

$$M(w) \geq M(v)$$
$$B(w) \geq B(v)$$

In this paper, our objective is to i) minimise the cost of reconfiguration and ii) maximise the CP's revenue. In fact, to reach the first objective we propose to minimise both the i) migration cost and the ii) embedding cost of the reconfigured elements in D.

Let $\varphi(D)$ denote the migration cost of the mapped D. φ(D) is defined as the weighted sum of migrated virtual nodes φ_n and links φ_e . Formally,

$$\varphi(D) = a \cdot \varphi_n(D) + b \cdot \varphi_e(D)$$

where $0 \leq a \leq 1$ and $0 \leq b \leq 1$ are respectively the weights of φ_n and φ_e . Note that $a + b$ must be equal to 1. We recall that i) φ_n is equal to the number of migrated virtual nodes and ii) φ_e is equal to the total allocated bandwidth of the virtual link e (i.e., the sum of allocated bandwidth over the physical path).

Let *ζ(D)* denote the embedding cost of *D*. As in [7], *ζ(D)* evaluates the amount of substrate bandwidth allocated for all virtual links belonging to *D*. It is worth noting that *ζ(D)* depends on the length of hosting substrate paths. Formally, *ζ(D)* is expressed as follows:

$$\zeta(D) = \sum_{d \epsilon D} \sum_{e \epsilon P(d)} C(d)$$

where *P(d)* denotes the substrate path embedding the virtual link *d* and *e* is a substrate link belonging to the path *P(d)*.

As detailed above, the optimisation problem consists in minimising reconfiguration cost. Formally:

$$minimise\ [C_{tot}(D) = \alpha \cdot \varphi(D) + \beta \cdot \zeta(D)]$$

where $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$ are the weights of φ and ζ respectively. Note that $\alpha + \beta$ must be equal to 1. It is straightforward to see that to optimise the revenue, we minimise the embedding cost of reconfigured virtual topologies. Indeed, by minimising the amount of allocated resources, the residual resources will be maximised. Hence more VN s can be accepted and more revenue can be generated.

Given a set of virtual link bandwidth pairs $RD=\{(d_i , C(d_i))\}1 \leq i \leq k$, where $d_i \in E(D)$. Our objective is to find the best reconfiguration of the corresponding embedded virtual graph *D* while satisfying the new links' requirement and minimising the reconfiguration cost.

B.2 Proposed VNR-GA Algorithm and Implementation

VNR-GA proceeds as follows. As soon as the upgrade virtual network bandwidth demands $RD=\{(d_i , C(d_i))\}1\leq i\leq k$ arrive, VNR-GA builds the set of solution components denoted by $GD=\{SC_i\}$. In fact, SC_i is a star topology composed of a center virtual node with all its directly connected links in which at least one of them is in *RD* (i.e., requires more bandwidth). The implementation code for generating the initial graph and the *SC's* is as follows

```
def buildNetwork(N, linkCapacityR1, linkCapacityR2, resCapacityR1, resCapacityR2):
    adjMat = [[0 for x in range(N)] for y in range(N)]
    nodeArr = [(0,0) for x in range(N)] # (cpu, mem)
    for i in range(0,N):
        for j in range(i,N):
            val = random.uniform(0,1)
            if(val > 0.5):
                linkCapacity = random.randint(linkCapacityR1, linkCapacityR2)
                adjMat[i][j] = linkCapacity
                adjMat[j][i] = linkCapacity
        cpu = random.randint(resCapacityR1, resCapacityR2)
        mem = random.randint(resCapacityR1, resCapacityR2)
        nodeArr[i] = (cpu, mem)
    return {"nodeArr": nodeArr, "adjMat":adjMat}
```

```
def generateSolutionComponents(SN, VN, RD):
    n = len(VN["nodeArr"])
    numUpdates = len(RD) # An arr of the form (i,j, newCapacity)
    setOfCenters = set()
    GD = set()
    for i in range(0,numUpdates):
        setOfCenters.add(RD[i][0])
        setOfCenters.add(RD[i][1])
    for i in range(0,n):
        if(i in setOfCenters):
            GD.add(i) # Adding the node in the set GD

    return GD
```

Afterwards, with respect to *RD* and *GD*, VNR-GA yields the best reconfiguration solution that satisfies *RD* and minimises the reconfiguration cost. To do so, an initial population of feasible solutions (i.e., chromosomes) is put forward, then, after Nmax iterations, during which new populations of chromosomes are created thanks to crossover and mutation operators, an optimal solution of reconfiguration, Sb is selected. Finally, VNR-GA performs the reconfigurations of Sb to satisfy customer's demands (i.e., RD). Hereafter, we will describe the main VNR-GA components: i) Encoding and initial population, ii) Fitness evaluation, iii) Crossover and mutation of chromosomes and iv)Framework of VNR-GA.

B.2.1 Encoding and initial population

Based on aforementioned problem's formalisation, we opt for the natural encoding to represent the population's individuals. Indeed, each gene, *i*, of the chromosome denoted by S defines the identifier of the i^{th} migrated solution component. However, if the gene value corresponds to 0, it means that no solution component is migrated at this level. It is worth noting that a chromosome is expressed as follows: $S=[i_1, i_2, ..i_k, ..., i_l]$ where $k \in [0, |GD|]$. As a consequence, a chromosome contains a sequence of $\{SC_i\}s$ migrations triggered to reconfigure the VN topology. Moreover, the size of a chromosome corresponds to the maximum number of sequential migrations $\{SC_i\}s$. Note that the migrations of solution components are triggered sequentially and in the same order as defined in the chromosome. In other words, performing *S* is equivalent to move in order $SC_{i1}, SC_{i2}, ..., SC_{il}$. Indeed, within a chromosome, the reconfiguration of solution component strongly affects those of next ones. The code for getting the initial population is as follows

```
def perm_generator(seq):
    seen = set()
    length = len(seq)
    while True:
        perm = tuple(random.sample(seq, length))
        if perm not in seen:
            seen.add(perm)
            yield perm

def initialPopulation(GD, N):
    rand_perms = perm_generator(list(GD))
    P0 = [next(rand_perms) for _ in range(N)]
    return P0
```

B.2.2 Fitness Evaluation

The fitness of a chromosome is an indicator of the quality of the reconfiguration solution to satisfy the new demands of bandwidth. Since the migration is costly in terms of service interruption period. For example, in the previous work [8], the authors evaluated the average migration delay in testbed platform to 2ms. Hence, we aim to minimise the migration cost of the reconfiguration. Moreover, it is worth pointing out that the mapping cost of the VN affects the acceptance rate of VN s since it depends on the amount of allocated resources. Hence, we propose the following fitness metric:

$$Fitness = \frac{1}{Ctot(D)+\epsilon}$$

$where\ 0 < \epsilon <<<< 1$

B.2.3 Crossover and mutation of chromosomes

Given that VNR-GA is based on Genetic metaheuristic, its effectiveness strongly depends on some parameters such as size of population, crossover and mutation operators. Hereafter, we will detail crossover and mutation operators proposed for VNR-GA algorithm.

1) Crossover operator: Most crossover operators that are proposed in literature [9] (e.g., single point crossover, two points crossover and uniform crossover) are much more adapted to binary-encoded populations than natural-encoded populations. Based on [10], we adopt an improved crossover operator that is adapted to our reconfiguration problem. It is defined as follows. Given a pair of father chromosomes $S_1 = [i_1, i_2, \cdots, i_l]$ and $S_2 = [j_1, j_2, \cdots, j_l]$, we define the notion of similitude model as $Sim(S_1, S_2) = [k_1, \cdots, k_l] = S_1 \times S_2$. Note that $S_1 \times S_2 = Sim$ is a chromosome. Namely, if the i^{th} gene of S_1 (i.e., SC_i) is equal to the i^{th} gene of S_2 then Sim's i^{th} gene takes 1. Otherwise, it is set to 0.

The Hamming distance of father individuals S_1 and S_2 is defined as the number of genes corresponding to 1 in Sim, denoted by $H(S_1, S_2)$. The improved crossover operator depends on the Hamming distance H. It is defined as follows: If $H(S_1, S_2) \geq 2$, then the crossover operator performs a simple point crossover from the point $\alpha(S_1, S_2)$. Otherwise, the crossover is not performed. Note that $\alpha(S_1, S_2)$ is the effective distance between S_1 and S_2 and it is defined as the distance from the first 1 to the last 1 of the chromosome Sim. It has been proved in [10] that the above method generates new chromosomes which are distinct (i.e., no twins).

The implementation of sim and hamming distance is below -

```
def sim(s1,s2):
    n = len(s1)
    s = [0 for i in range(n)]
    for i in range(0,n):
        if(s1[i] == s2[i]):
            s[i] = 1
    return s

def hamming(sim):
    h = 0
    for i in range(0, len(sim)):
        if(sim[i] == 1):
            h+=1

    return h
```

The code for crossover operator is shown below -

```
def crossover(s1,s2):
    n = len(s1)
    s = sim(s1,s2)
    h = hamming(s)
    if(h >= 2):
        firstOne = -1
        secondOne = -1
        for i in range(0, len(s)):
            if(s[i] == 1):
                if(firstOne == -1):
                    firstOne = i
                secondOne = i
        alpha = secondOne - firstOne

        c1 = [0 for i in range(n)]
        c2 = [0 for i in range(n)]
        for i in range(n):
            if(i < alpha):
                c1[i] = s1[i]
                c2[i] = s2[i]
            else:
                c1[i] = s2[i]
                c2[i] = s1[i]

        return (c1, c2)
    return None
```

2) Mutation: An improved mutation operator is proposed to enhance the quality of generated chromosomes. The main idea behind this proposal is to replace worst existing chromosomes by new ones offering better fitness. To do so, genes (i.e., SCi) causing redundant links migration are discarded from the chromosome and replaced by the value 0. In other words, in one chromosome, we always favour SCs with disjoint links. Consequently, by removing useless link migrations, the cost of migration will be reduced and so will the reconfiguration cost. Note that the mutation process is launched every f iterations in order to prevent VNR-GA from falling into local extreme.

The code mutation is as follows -

```
def mutate(S, RD):
    nodeToRem = set()
    nodeToKeep = set()
    for i in range(0, len(RD)):
        if(RD[i][0] not in nodeToKeep):
            nodeToRem.add(RD[i][0])
        nodeToKeep.add(RD[i][1])

    newS = [S[i] for i in range(0, len(S))]
    for i in range(0, len(S)):
        if(S[i] in nodeToRem):
            newS[i] = 0

    return tuple(newS)
```

B.2.4 Framework Of VNR-GA

In this section, we will describe the framework of VNR-GA. VNR-GA generates N-sized initial populations using Roulette Wheel selection technique [9]. Indeed, the quality of each chromosome is evaluated thanks to the fitness function defined in B.2.2 . We propose to migrate the centre *v* of the star moving candidate SC_i to a less loaded area of the substrate network while respecting the bandwidth constraint. We recall that migrating a star component requires the embedding of all attached links while migrating the centre virtual node. To do so, we evaluate the quality of each substrate node, $w \in SN$ while considering its residual resources (i.e., *M (w)* and *B (w)* and those of its direct paths to the hosting nodes of v's neighbours.

Indeed, such chromosomes transformation aims to obtain new promising individuals that offer better fitness evaluation. We recall that every f iterations, a mutation transformation is performed to create a different generation of chromosome. Finally, at the end of N_{max} iterations,

the best chromosome is selected and the reconfiguration of *D* is performed. The code for the above is as follows -

```
def bestReconfig(SN, VN, GD, RD, N, Nmax, f):
    P = initialPopulation(GD, N)
    print("Initial Population", P)
    for i in range(0,Nmax):
        for j in range(0, N):
            for k in range(j+1, N):
                c = crossover(P[j],P[k])
                if c is not None:
                    P.append(c[0])
                    P.append(c[1])
        P = populationSelection(P, N, VN, GD, RD)
        if(i%f == 0):
            for j in range(0,N):
                P.append(mutate(P[j], RD))
            P = populationSelection(P, N, VN, GD, RD)
    return populationSelection(P, 1, VN, GD, RD)[0]


def migrate(SN, resources, VN, Sb):
    possible = False

    for i in range(0, len(Sb)):
        reqCpu = VN["nodeArr"][Sb[i]][0]
        reqMem = VN["nodeArr"][Sb[i]][1]
        reqCapacity = 0
        for j in range(0, len(VN["nodeArr"])):
            reqCapacity += VN["adjMat"][Sb[i]][j]

        residualResIdx = 0
        minRes = math.inf
        for j in range(0, len(resources)):
            if(resources[j][0] > reqCpu and resources[j][1] > reqMem and resources[j][2] > reqCapacity):
                possible = True
                residualRes = math.sqrt(((resources[j][0]-reqCpu)**2) + ((resources[j][1]-reqMem)**2) + ((resources[j][2]-reqCapacity)**2))
                if residualRes < minRes:
                    minRes = residualRes
                    residualResIdx = j

        if(not possible):
            return possible
        resources[residualResIdx][0] -= reqCpu
        resources[residualResIdx][1] -= reqMem
        resources[residualResIdx][2] -= reqCapacity

    return True
```

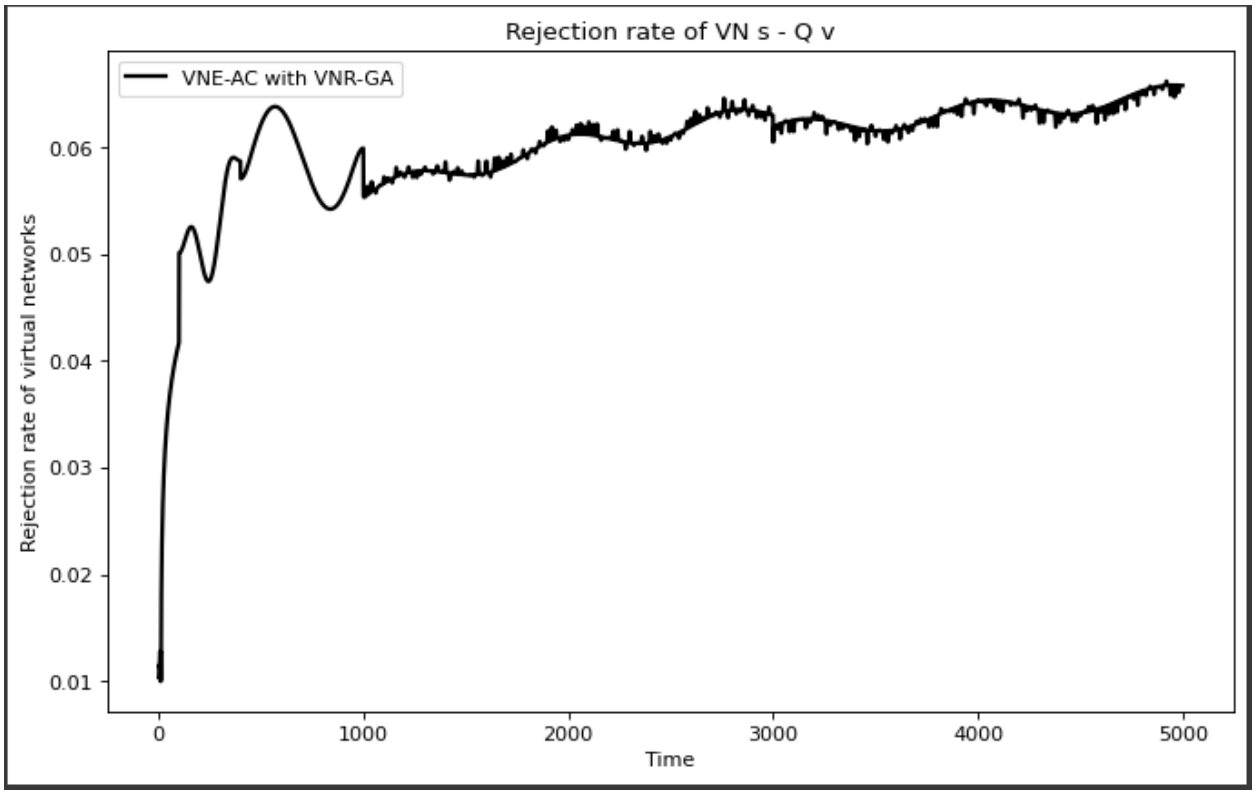
C Results And Conclusion

The work evaluates the performance of their algorithm by measuring and plotting certain metrics:

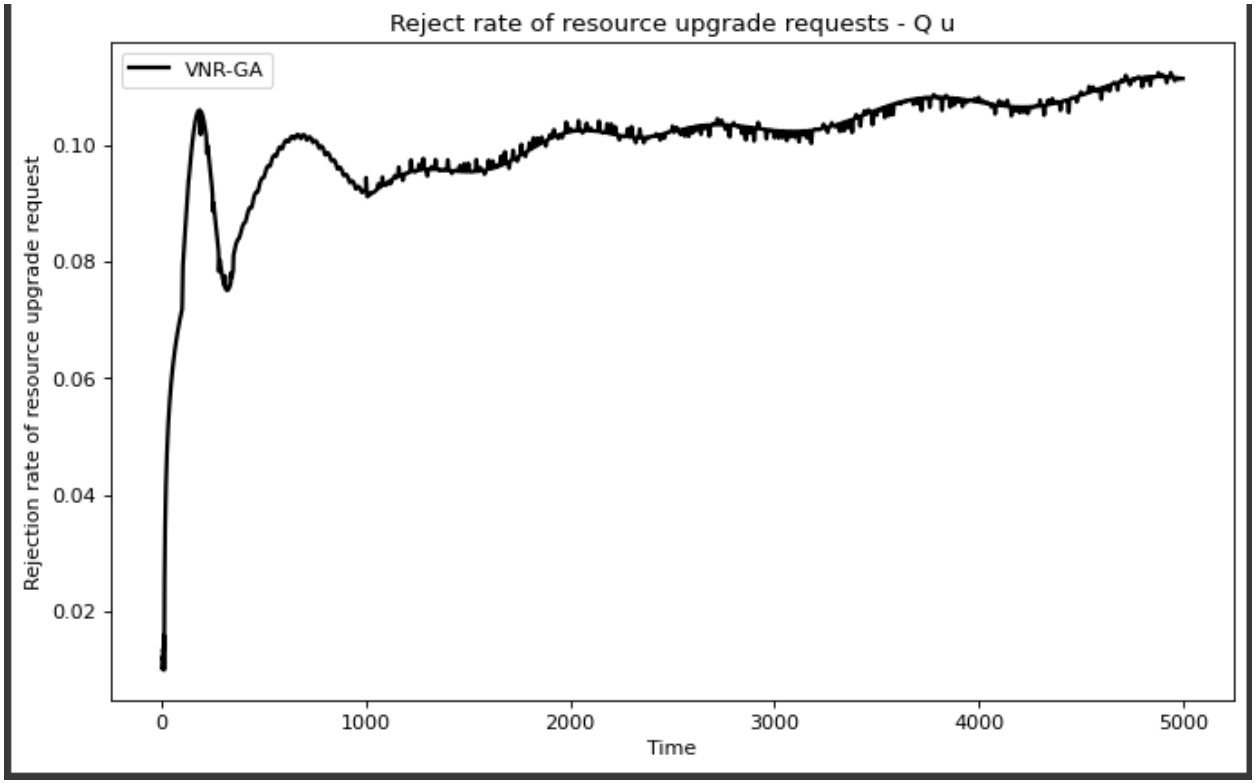
1. Q v : is the reject rate of VN requests.
2. Q u : is the reject rate of resource upgrade requests.
3. C tot : is the cost of reconfiguration

In our implementation also we have gathered these metrics and have generated these plots as follows:

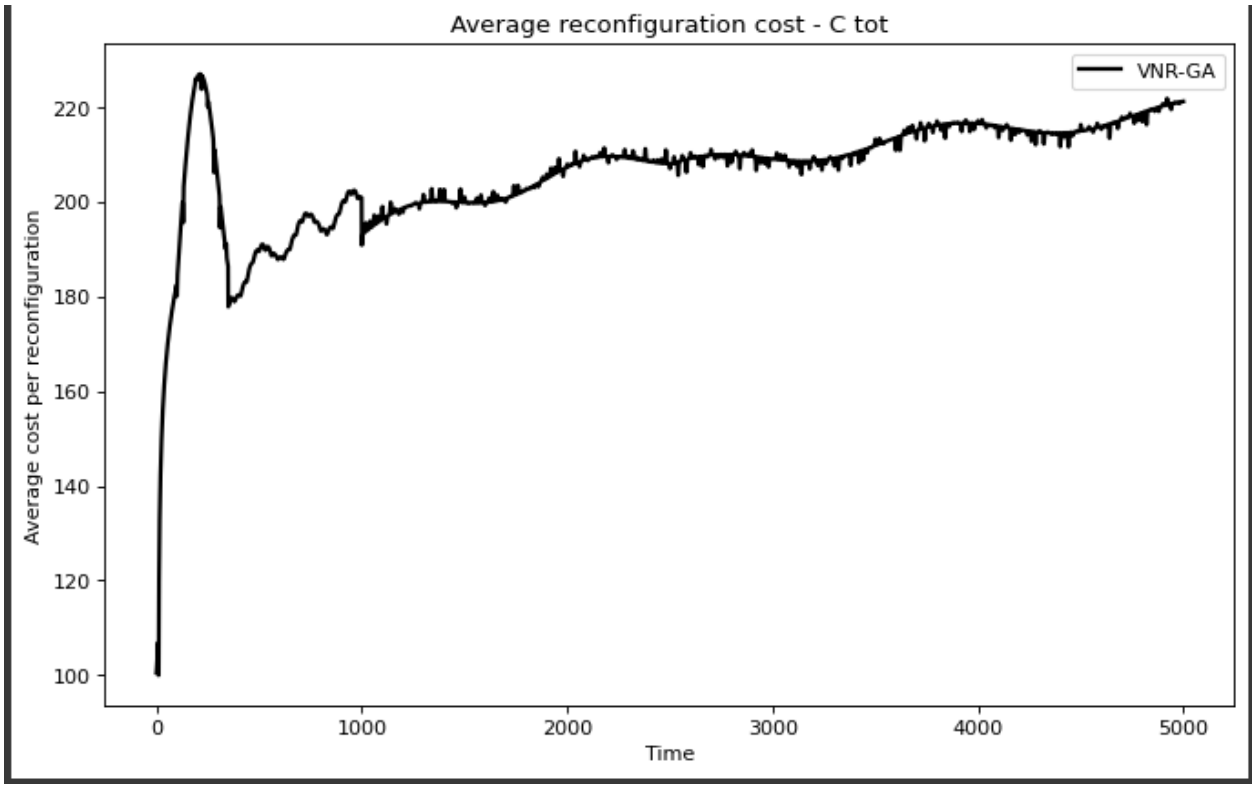
1.) Q v : is the reject rate of VN requests:



2.) Q u : is the reject rate of resource upgrade requests.



3.) C tot : is the cost of reconfiguration



D References

1. N. M. M. K. Chowdhury and R. Boutaba, “Network virtualization: State of the art and research challenges,” IEEE Communications Magazine, vol. 47, 2009.
2. I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, “VNR Algorithm: A Greedy Approach For Virtual Networks Reconfigurations,” in GLOBECOM. IEEE, 2011, pp. 1–6.
3. C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, “SecondNet: a data center network virtualization architecture with bandwidth guarantees,” in Proceedings of the 6th International COnference, ser. Co-NEXT ’10. New York, NY, USA: ACM, 2010, pp. 15:1–15:12.
4. H. Zha, X. He, C. Ding, H. Simon, and M. Gu, “Bipartite graph partitioning and data clustering,” in Proceedings of the tenth international conference on Information and knowledge management, ser. CIKM ’01. New York, NY, USA: ACM, 2001, pp. 25–32.
5. J. Fan and M. Ammar, “Dynamic topology configuration in service overlay networks: A study of reconfiguration policies,” in INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, 2006, pp. 1–12.
6. Y. Ohsita, T. Miyamura, S. Arakawa, S. Ata, E. Oki, S. Kohei, and M. Murata, “Gradually reconfiguring virtual network topologies based on estimated traffic matrices,” Networking, IEEE/ACM Transactions on, vol. 18, no. 1, pp. 177–189, 2010.
7. I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, “Adaptive-VNE: A flexible resource allocation for virtual network embedding algorithm,” IEEE GLOBECOM, 2012.
8. I. Fajjari, M. Ayari, O. Braham, G. Pujolle, and H. Zimmermann, “Towards an autonomic piloting virtual network architecture,” IFIP International Conference on New Technologies, Mobility and Security - NTMS, pp. 1–5, 2011.
9. D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, 1989.
10. Z. Qi-yi and C. Shu-chun, “An improved crossover operator of genetic algorithms,” in Computational Intelligence and Design, 2009. ISCID ’09. Second International Symposium on, vol. 2, 2009, pp. 82–86.
11. I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, “VNE-AC: Virtual Network Embedding Algorithm based on Ant Colony Metaheuristic,” IEEE ICC, 2011.
12. M. Chowdhury, F. Samuel, and R. Boutaba, “PolyViNE: policy-based virtual network embedding across multiple domains,” in Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures, ser. VISA ’10. New York, NY, USA: ACM, 2010, pp. 49–56.
13. V. V. Vazirani, Approximation algorithms. Springer, 2001.
14. N. Chowdhury, M. Rahman, and R. Boutaba, “Virtual network embedding with coordinated node and link mapping,” IEEE INFOCOM, pp. 783–791, 2009.