NMRG P. Martinez-Julia, Ed.
Internet-Draft NICT
Intended status: Informational D. Lopez
Expires: January 10, 2022 TID
July 09, 2021

VEGA: A Genetic Method for Planning Virtual Network Embedding Configurations
draft-pedro-nmrg-vega-network-embedding-00

Abstract

The number of elements composing virtual networks (VN), together with the number of elements in the substrate networks (SN), makes unfeasible to obtain in a short time an optimum configuration for the assignation of elements from the VN to the SN. Here we present VEGA, standing for Virtual network Embedding method based on a Genetic Algorithm. It defines a particular strategy and heuristic to provide configurations for network embedding that are close to the optimum in a short time.

Status of This Memo

This Internet-Draft will expire on January 10, 2022.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Background 3
  3.1. Virtual Computer and Network Systems 3
  3.2. SDN and NFV 3
  3.3. Management and Control 4
  3.4. Virtual Network Embedding Problem 5
4. Genetic Method for Finding Network Embedding Configurations . 5
  4.1. Resolving the VNEP 5
  4.2. Using VEGA to Get a Configuration Close to the Optimum . 5
  4.3. Building an Embedding Configuration 7
  4.4. Basic Heuristic Function 8
  4.5. Incremental Heuristic Function 8
5. Relation to Other IETF/IRTF Initiatives 9
6. IANA Considerations 9
7. Security Considerations 10
8. Acknowledgements 10
9. References 10
  9.1. Normative References 10
  9.2. Informative References 10
Appendix A. Appendix 1 11
Authors' Addresses 11

1. Introduction

In this document we describe VEGA, which stands for Virtual network Embedding method based on a Genetic Algorithm. It, therefore, is a method for finding configurations for embedding the elements of a Virtual Network (VN) in the elements of a Substrate Network (SN) by using an algorithm following the Genetic Programming (GP) methodology. This is needed because the number of combinations of elements from the VN and SN is so big that it is unfeasible to obtain the optimum configuration in a short time.

This limitation is resolved by using some suboptimal method for searching the solution, as it is the case of an algorithm based on GP. However, a particular strategy and heuristic should be used to ensure that provided configurations are close to the optimum within a short time. Therefore, the time boundaries for the method presented here are low and the quality of the configurations is high. This quality is required for the correct operation of current and future networks, particularly those that must be adapted to dynamic needs.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Background

3.1. Virtual Computer and Network Systems

The continuous search for efficiency and cost reduction to get the most optimum exploitation of available resources (e.g. CPU power and electricity) has conducted current physical infrastructures to move towards virtualization infrastructures. Also, this trend enables end systems to be centralized and/or distributed, so that they are deployed to best accomplish customer requirements in terms of resources and qualities.

One of the key functional requirements imposed to computer and network virtualization is a high degree of flexibility and reliability. Both qualities are subject to the underlying technologies but, while the latter has been always enforced to computer and network systems, flexibility is a relatively new requirement, which would not have been imposed without the backing of virtualization and cloud technologies. Such flexibility is exploited by allowing management systems, such as presented in "Anticipating minimum resources needed to avoid service disruption of emergency support systems [ICIN-2018]," to configure the computer and network systems.

3.2. SDN and NFV

SDN and NFV are conceived to bring high degree of flexibility and conceptual centralization qualities to the network. On the one hand, with SDN, the network can be programmed to implement a dynamic behavior that changes its topology and overall qualities. Moreover, with NFV the functions that are typically provided by physical network equipment are now implemented as virtual appliances that can be deployed and linked together to provide customized network services. SDN and NFV complements to each other to actually implement the network aspect of the aforementioned virtual computer and network systems.

Although centralization can lead us to think on the single-point-of-failure concept, it is not the case for these technologies.

Conceptual centralization highly differs from centralized deployment. It brings all benefits from having a single point of decision but retaining the benefits from distributed systems. For instance, control decisions in SDN can be centralized while the mechanisms that enforce such decisions into the network (SDN controllers) can be implemented as highly distributed systems. The same approach can be applied to NFV. Network functions can be implemented in a central computing facility, but they can also take advantage of several replication and distribution techniques to achieve the properties of distributed systems. Nevertheless, NFV also allows the deployment of functions on top of distributed systems, so they benefit from both distribution alternatives at the same time.

3.3. Management and Control

The introduction of virtualization into the computer and network system landscape has increased the complexity of both underlying and overlying systems. On the one hand, virtualizing underlying systems adds extra functions that must be managed properly to ensure the correct operation of the whole system, which not just encompasses underlying elements but also the virtual elements running on top of them. Such functions are used to actually host the overlying virtual elements, so there is an indirect management operation that involves virtual systems. Moreover, such complexities are inherited by final systems that get virtualized and deployed on top of those virtualization infrastructures. A key to address them, from the architecture point of view, is to define a whole architecture, such as [ETSI-NFV-MANO], and complement it with required protocols, interfaces, and algorithms, such as the algorithm defined here, to ensure widespread of management techniques and qualities.

In parallel, virtual systems are empowered with additional, and widely exploited, functionality that must be managed correctly. It is the case of the dynamic adaptation of virtual resources to the specific needs of their operation environments, or even the composition of distributed elements across heterogeneous underlying infrastructures, and probably providers. Taking both complex functions into account, either separately or jointly, makes clear that management requirements have greatly surpassed the limits of humans, so automation has become essential to accomplish most common tasks. In addition, getting and analyzing telemetry [I-D.ietf-opsawg-ntf] gains a key relevancy in the management and control planes.

3.4. Virtual Network Embedding Problem

The Virtual Network Embedding Problem (VNEP) targets the construction of a configuration that assigns nodes from the SN (NSNs) to a VN. A configuration is, therefore, a map from VNF instances to NSNs. It must ensure that all nodes and links from the VN are embedded onto the SN. These solutions must ensure that the goals set by the tenants of both the VN and the SN are respected. However, it is well known that conventional methods to resolve the VNEP cannot be both subject to find the optimum and resolved in polynomial time, as discussed in "Automation and Multi-Objective Optimization of Virtual Network Embedding [IEEE-IM-2021]". The algorithm presented here provides a solution to the VNEP that is close to the optimum.

4. Genetic Method for Finding Network Embedding Configurations

4.1. Resolving the VNEP

In this section we describe VEGA, a method to get a solution to the VNEP that is close to the optimum. To do so, it incorporates a new algorithm, which has been designed using the GP methodology. It uses a particular heuristic and overall strategy to provide the resulting method with the ability to find a basic configuration and improve it, incrementally and iteratively, until some stop condition is met, generally after a number of iterations.

4.2. Using VEGA to Get a Configuration Close to the Optimum

By using an heuristically driven search approach, with most heuristics, an algorithm can be stuck into a local optimum. Although most of the time such kind of optimums are good enough for most VN embeddings, they can provide some disparate configurations. To prevent such situation, GP focuses on the iterative generation of configurations which have some degree of randomness from one iteration to the next, so disparate solutions are also considered, breaking any local optimum barrier.

VEGA exploits the benefits of such randomness by including the creation method of generating new solutions, as detailed below. This extends the search space beyond the path guided by the heuristic function and allows the algorithm to find better optimal configurations, closer to the global optimum. Moreover, the algorithm builds the configurations incrementally, following DP methodology, so the most relevant partial configurations are cached for allowing the find procedure to avoid re-calculating all of them when exploring different paths.

```
procedure VEGA
  C0 = MakeConf (Empty, F, S)
  Gi = [C0]
  while N > 0 do
    Gjm = Mutate (Gi, L)
    Gjb = Breed (Gi, L)
    Gjc = Create (Gi, L)
    Gj = (Gjm union Gjb union Gjc) - Gi
    SortByH (Gj)
    PickHomo (Gj, L)
    Gi = Gi union Gj
    N = N - 1
  end while
  SortByH (Gi)
  return First (Gi)
end procedure
```

Main algorithm of VEGA.

This algorithm relies on an inner algorithm, MakeConf, to build a basic configuration. As described below, it provides a configuration that accomplishes the overall objectives but, as it is greedy, it is most probably stuck in a local optimum. At first, VEGA begins building a basic configuration for the current set of VN elements (F) and SN elements (S), specifying that the base configuration is empty. This configuration is considered to be a local optimum. To go beyond it, VEGA uses three functions to generate new configurations to consider. They are Mutate, Breed, and Create.

Mutate gets each configuration from the input and changes some assignation of F to S randomly. Breed gets pairs of configurations from the input and creates a new configuration for each pair by including the odd or even assignations from the "parents" into the "child" configurations. Finally, Create builds new configurations by randomly assigning elements of F to elements of S. Each function generates, at most, the amount of new configurations specified (L).

All new and old configurations are unified together in a set and the previously considered configurations are removed from it. Then, the set is sorted by incremental value of the heuristic function (H) and a subset is picked by choosing L configurations homogeneously, although ensuring that the first (best configuration, according to H) is included. The new set is unified with the set of configurations previously generated and a new loop is done. After N iterations the loop exits. The resulting set of configurations is sorted by H and the first configuration (best) is returned.

4.3. Building an Embedding Configuration

A typical algorithm for implementing the inner function of finding associations of VN elements to SN elements, viz. configurations, would loop among all of them and generate all possible configurations. However, it is not just unpractical and generally unfeasible to be accomplished on polynomial time for bigger networks, but also inefficient because there would be a lot of configurations that are irrelevant and/or totally deviated from the objective. Instead, VEGA relies on a greedy version of such exploration algorithm. Although the search is not exhaustive, each iteration of this algorithm chooses the best alternative, so it quickly finds some local optimum.

```
procedure MakeConf (Ci, F, S)
  Cj = Ci
  C = Empty
  for all fi in F do
    for all si in S do
      C = C union [Cj union (fi, si)]
    end for
    SortByH (C)
    Cj = First (C)
  end for
  return Cj
end procedure
```

Algorithm for building a configuration, viz. a set of assignations of VN elements to SN elements.

This algorithm works as follows. First, the provided base configuration is considered, although it can be the empty configuration. Then, a new set of configurations is generated by assigning each element of F to every element of S. This set is sorted by H and the first configuration (best) is chosen to be used for attaching the next assignation of F to S in the next iteration. This highly reduces the complexity of this algorithm, in terms of number of instructions. It does not provide the optimum, not even close local optimum, but it is oriented by H, so when combined with the overall procedure of VEGA, as described above, a configuration that is very close to, or equal to, the global optimum can be reached. After all elements of F have been assigned, the resulting configuration is returned.

4.4. Basic Heuristic Function

The heuristic used in the method is a key aspect since it determines how efficient and fast is the method iterating towards a solution that is as close as possible to the optimum. There are two definitions for the heuristic. The first, presented here, is in the general form, as follows:

```
function H (c)
  result = 0
  for all (_, si) in c do
    result = result + PathLenTo (si)
  end for
  for all sj in S do
    q = 0
    for all (_, sk) in c do
      if sk == sj then
        q = q + 1
      end if
    end for
    result = result + Z ^ q
  end for
  return result
end function
```

Iterative function to calculate the heuristic value of a configuration.

A value is assigned to a configuration in base of the placement of VNF instances, depending on the co-location of several instances on the same node of the SN and the length of the path from the gateway to the node of the SN where certain VNF instance has been placed. It is calculated by adding the length of all paths from the gateway to the node from S that is included in a configuration, and exponentiating a value (Z) to the amount of elements that are co-located in the same substrate node.

4.5. Incremental Heuristic Function

A new heuristic is derived form the basic heuristic to meet the requirements of the incremental strategy used by the algorithm. It, therefore, defines a value for a configuration by re-using the value obtained from a previous configuration. Therefore, the definition of the heuristic function is as follows:

```
function H (Cj)
  Ci = Cj - (fj, sj)
  result = H (Ci)
  q = 0
  for all (_, sk) in Ci do
    if sk == sj then
      q = q + 1
    end if
  end for
  result = result - Z ^ q
  result = result + PathLenTo (sj)
  r = 0
  for all (_, sk) in Cj do
    if sk == sj then
      r = r + 1
    end if
  end for
  result = result + Z ^ r
  return result
end function
```

Recursive function to calculate the heuristic value of a configuration.

It has the same considerations but relies on calculations already done and stored in a table, following the Dynamic Programming (DP) methodology. The addition of the cost derived from the new paths is straightforward, but the addition of the cost of co-locating VNF instances is not and must be calculated by subtracting first the previous value. Therefore, the function begins with the value of H for the previous configuration (Ci), which is equivalent to the new configuration (Cj) after removing the last element ((fj, sj)). Then, it calculates the amount of co-located elements in Ci and subtracts it to the result. Then, it acts as a single iteration in the previous definition of H to add the values of the new elements ((fj, sj)). Finally, it returns the result.

5. Relation to Other IETF/IRTF Initiatives

TBD.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

The main security consideration is tied to the source of the data provided to the algorithm for both knowing the functions forming the VN and the nodes forming the SN. It is up to the administration endpoint to ensure such information is addressed securely.

8. Acknowledgements

TBD.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

9.2. Informative References

[ETSI-NFV-MANO]
ETSI NFV GS NFV-MAN 001, "Network Functions Virtualisation (NFV); Management and Orchestration", 2014.

[I-D.ietf-opsawg-ntf]
Song, H., Qin, F., Martinez-Julia, P., Ciavaglia, L., and A. Wang, "Network Telemetry Framework", draft-ietf-opsawg-ntf-07 (work in progress), February 2021.

[ICIN-2018]
P. Martinez-Julia, V. P. Kafle, and H. Harai, "Anticipating minimum resources needed to avoid service disruption of emergency support systems, in Proceedings of the 21th ICIN Conference (Innovations in Clouds, Internet and Networks, ICIN 2018). Washington, DC, USA: IEEE, 2018, pp. 1--8", 2018.

[IEEE-IM-2021]
P. Martinez-Julia, V. P. Kafle, and H. Asaeda, "Automation and Multi-Objective Optimization of Virtual Network Embedding, in Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management", May 2021.

Appendix A. Appendix 1

TBD.

Authors' Addresses

Pedro Martinez-Julia (editor)
NICT
4-2-1, Nukui-Kitamachi, Koganei
Tokyo 184-8795
Japan

Phone: +81 42 327 7293
Email: pedro@nict.go.jp

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 82
Madrid 28006
Spain

Email: diego.r.lopez@telefonica.com